电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

博士学位论文

DOCTORAL DISSERTATION

论文题目 云中多维资源与分布式流水线并行训练的优化调度技术

学科专业	**软件工程**
学　　号	**202011090909**
作者姓名	**周光耀**
指导老师	**田文洪 and Rajkumar Buyya　教授**
学　　院	**信息与软件工程学院**

分类号 _____ 密级 公开

UDC[注1] _____

学 位 论 文

云中多维资源与分布式流水线并行训练的优化调度技术

（题名和副题名）

周光耀

（作者姓名）

指导老师 **田文洪 教授**

电子科技大学 成都

指导老师 **Rajkumar Buyya 教授**

University of Melbourne 墨尔本

（姓名、职称、单位名称）

博士 学科专业 **软件工程**

提交论文日期 _____ 论文答辩日期 _____

学位授予单位和日期 **电子科技大学 年 月**

答辩委员会主席 **郑纬民**

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

Optimization Scheduling Technologies for Multi-dimensional Resources and Distributed Pipeline Parallel Training in Cloud Computing

A Doctoral Dissertation Submitted to
University of Electronic Science and Technology of China

Discipline:	**Software Engineering**
Author:	**Guangyao Zhou**
Student ID:	**202011090909**
Supervisor:	**Prof. Wenhong Tian and Rajkumar Buyya**
School:	**School of Information and Software Engineering**

ABSTRACT

In the era of rapid development of the Internet, Cloud computing, as a key field of the digital economy, has received broad attention. Cloud computing is an important paradigm in current distributed computing systems. Relying on various fundamental mathematical branches such as discrete mathematics, operations research and stochastic processes, resource scheduling management of large-scale distributed systems have always been one of the key and difficult points in the field of computer science, which is necessary to simultaneously consider multi-dimensional resources and multiple optimization objectives, and the optimization modeling and solving theory still needs to be developed. In addition, the scale of artificial intelligence models, mainly represented by deep learning, is becoming increasingly large, and their parallel training relies on the development of Cloud system architectures and resource scheduling technologies. Most resource scheduling problems in distributed systems belong to NP-Hard problems, and the research of the algorithm design and optimization theory lag behind industrial applications. Various existing algorithms can be used to obtain feasible scheduling schemes for specific scenarios, but there is significant room for improvement in their computational complexity and optimality. In order to explore methods or strategies to enhance the resource management capabilities of distributed computing systems such as Cloud computing, this dissertation selects five sets of scheduling scenarios (whose problems all belong to NP hard problems) in Cloud environments, and conducts discussions on the design of their new optimization algorithm architecture and algorithm theory. The main research work and conclusions are as follows:

(1) To address the scheduling problem of independent task sets in Cloud nodes considering single-dimensional resources, the dissertation proposes a multi-route local search algorithm (MSRA) series using heuristic algorithms as search routes (HLSA). The dissertation models the problem of minimizing makespan, proposes several MSRA algorithms, and proves the upper limitation approximation ratios of the proposed algorithms are $\frac{5}{4}$ in the classical problem minimizing the maximum makespan in parallel machines, which improves $\frac{1}{12}$ compared to $\frac{4}{3}$ of longest processing time alogirthm. The experimental results show that: For homogeneous systems, MSRA in large-scale scenarios reduces the average makespan by $\geq 8.56\%$; that in small-scale scenarios has the probability 73.56% of obtaining the theoretical optimal solution, which is 1.49 times that of the best baseline algorithm.

For heterogeneous systems, MSRA reduces the average makespan by $\geq 9.47\%$.

(2) To solve the scheduling problem of independent virtual machine sets in Cloud nodes considering multi-dimensional resources, the dissertation proposes a growable genetic algorithms (GGA) series with additional growth route. The dissertation formulates the problem of minimizing the maximum utilization of resources in each dimensions and minimizing system's energy consumption in heterogeneous nodes, and proposes instantiation algorithms GHW-NSGA II and GHW-MOEA/D using HLSA as the growth route. The GGA series reconstructs the genetic algorithm architecture, allowing various algorithms to serve as its growth route, thus improving the convergence speed of the genetic solution process and the optimality of convergence solutions. Simulation data sets and public data sets drive experiments to validate the advantages of GGA: the convergence speed of GGA can reach 10 times that of the baseline algorithms of evolutionary algorithms (NSGA II and MOEA/D) in large-scale experimental scenarios.

(3) Aiming at the scheduling problem of parallel training workflow of deep model using equal Microbatch's data partitioning-based pipeline parallelism in GPU Cloud server nodes, the dissertation derives the analytic formulas of theoretical cost model (not only simultaneously considering the GPU computing time and network communication time, but also taking into account the nonlinear relationship between them and the data amount), proposes improved multi-dimensional dichotomy (IMD) and IMD-based cross-search algorithm (CSIMD). This dissertation proves that the theoretical optimal error of the IMD can approach 0, and proves IMD's computational complexity is a linear growth function much lower than the best baseline algorithms including dynamic programming and recursive algorithm. Parallel training experiments in a realistic distributed environment show that: the average training speeds obtained by CSIMD in CNN-related networks training are respectively $2.0\times$ and $2.5\times$ of baseline strategies GPipe-R and GPipe-E; and that in transformer-related networks training are respectively $1.5\times$ and $1.6\times$.

(4) The dissertation designs a novel parallel training architecture for deep learning, i.e., unequal Microbatches-based pipeline parallelism (UMPIPE). UMPIPE allows different processes of the neural network to select different Microbatch sizes, introducing better training schemes for feasible solutions. In order to solve the optimization scheme of UMPIPE, the dissertation proposed a dual-chromosome genetic algorithm series (DGAP). To tackle the difficulty of calculating the training time corresponding to UMPIPE training scheme, the dissertation further proposes a matrix operation-based two-level accelerated

improvement strategy to simultaneously calculate the end training time corresponding to multiple individuals and multiple Microbatches of DGAP. Theoretical analysis proves the optimality of UMPIPE architecture, and proves that the convergence of dual-chromosome strategy is far superior to that of single-chromosome for solving UMPIPE. The experiments of training GPT1 and VGG16 in realistic environment show that, the speeds of UMPIPE's training scheme are increase by 13.89% and 14.36% respectively compared with the optimal training scheme under the baseline architecture GPipe.

(5) The dissertation formulates the system model of the hierarchical Cloud computing with multiple subsystems (HCCMS) for the diversity of task requests and diversity optimization objectives. For the joint optimization problem with multiple subproblems, the dissertation proposes a novel perspective of regarding the scheduling algorithms as the schedulable resources, and designs the scheduling framework to select the scheduling algorithms. In order to instantiate the algorithm selector series, the dissertation proposes the deep learning-based algorithm selector (DLS) and the deep reinforcement learning-based algorithm selector (DRLS). Compared with the best results among the baseline strategies, DLS reduces the weighted cost of system by 18.8% in the scenarios with the stable parameter range, and DRLS reduces the weighted cost of system by 11.5% in the dynamic scenarios with varying ranges of parameters.

This dissertation proposes multiple families of optimization scheduling algorithms for scene element changes at different progressive levels, involving various types of algorithms such as heuristic, local search, meta-heuristic, and machine learning. This dissertation expands the adaptability of Cloud systems and algorithm systems from application breadth and theoretical depth: the research progression from single dimensional resources to multi-dimensional resources has improved the adaptability to changes in resources' dimensions; the research progression from independent task sets to associated workflow sets has improved adaptability to changes in tasks' correlations; the research progression from single objective to multi-objective has improved the adaptability to changes in the number of optimization objectives; the research progression from single-center with single-layer to multi-centers with multi-layers and multi-subsystems has improved the joint utility in system hierarchy and scheduling scenarios.

Keywords: Cloud Environment, Scheduling of Multi-Dimensional Resources, Pipeline Parallelism, Growable Genetic Algorithm, Algorithm Selector

Contents

List of Figures

List of Tables

Chapter 1 Exordium

1.1 Background and Motivation

Thanks to the development of basic disciplines such as physics, mathematics and computer science, mankind has entered the Internet era, followed by the era of big data and artificial intelligence [1,2]. Distributed computing system is a key system architecture paradigm, which can support the requirements of big data and artificial intelligence for the comprehensive performance of computing systems [3]. The emerging trends of Industry 4.0 and 5G have significantly increased the number of tasks that Internet based computing systems need to process in real time, thus putting forward higher requirements for the comprehensive performance (including flexibility, reliability, dynamics, etc.) of large-scale distributed computing systems [4].

As a paradigm for providing high-performance computing services in a pay as you go manner, cloud computing ① effectively supporting increasingly complex software systems and computational programs. The current cloud computing systems have a stable structural paradigm, such as Software as a Service (SaaS), software-as-a-service), Platform-as-a-Service (PaaS) and Infrastructure as a Service (IaaS), infrastructure-as-a-service), and it can meet the diverse needs of a wide range of users [5]. Cloud computing has numerous users worldwide, providing powerful support for the operation of society and has become an indispensable component of today's industrial big data era. Until now, cloud computing is not only serving commercial systems, but also related to social layout.

Cloud computing, as an elastic, reliable, and dynamic service provider, provides multi-dimensional resources on the basis of devices such as CPU (central processing unit), RAM (random access memory), GPU (graphics processing unit), DS (disk storage), and BW (band width) [4,6,7] ②. Time generally refers to the entire service lifespan

① Cloud computing refers to a distributed computing paradigm, typically also used to represent the entire cloud computing system. A cloud system represents an organic whole constructed according to the cloud computing paradigm, consisting of several interdependent components with specific functions, including hardware devices (CPU, GPU, disk, motherboard, Ethernet cable, router, etc.), application software, operating system software, etc; Cloud scheduling is the abbreviation for cloud computing resource scheduling

② In cloud computing, multidimensional resources represent the accumulated capabilities of different types of component devices (such as CPU, RAM, DS, BW, etc.) that can support cloud computing services in time and space, and time and space can also be considered as resources. Multi dimensional resource scheduling means that in the process of cloud computing resource scheduling, it is necessary to consider the overall optimization scheduling of each server node containing multiple different dimensions of resources (different types of devices) - the optimization objective is related to the operation of resources in multiple dimensions. Single dimensional resource representation only requires resources from one dimension during scheduling - the optimization objective is only

of a cloud computing platform, while space refers to the actual physical location where cloud computing physical devices are placed. From the perspective of the composition of the entire space-time element, ”time” and ”space” are also two key resources of cloud computing. The various electrical components of cloud computing devices work in time and space driven by electrical energy, forming the true resource collection of cloud computing. Therefore, the essential resource provided by cloud computing is actually the effective conversion of electrical energy (energy, time, and space) per unit time and unit space. However, the deployment capability and resource scheduling capability of cloud computing systems still lag behind the growth of Internet data volume. The increasing task requests and data transmission make the scale of cloud computing systems gradually expand on the basis of hardware device deployment. With the expansion of cloud computing user groups and facilities, a large number of user requests and reservation tasks pose challenges to resource scheduling in cloud computing. Inappropriate resource scheduling schemes can lead to excessive energy consumption, long task running times, and excessive system burden, thereby reducing service quality, shortening the service life of equipment and components, and increasing carbon dioxide emissions. The scale of cloud computing systems is still expanding, and research on cloud computing resource management has an impact on the development of cloud computing related industries in industry [3,8]. At present, multi-stage methods [9], virtual machine migration [10], queuing models [11], service migration [12], workload migration [13], application migration [14], task migration [15], and scheduling algorithm based schedulers are commonly used strategies for optimizing cloud computing resource management. Among them, the scheduler based on resource scheduling algorithm is one of the key solutions to cloud computing resource management problems, and also the foundation of most other methods. However, factors such as time cycle, resource status, and environment can affect the optimization objectives and key parameters of resource scheduling, leading to a significant increase in the complexity of the solving process. Therefore, the research on cloud computing resource scheduling has always been a hot and difficult topic in the era of big data, which also affects the positioning and development of cloud computing in society. Independent task requests, virtual machine requests, and workflows are common processing objects in cloud computing. In specific scenarios, it is necessary to develop corresponding resource scheduling plans based on the characteristics of the processed object and optimization objectives. In

related to resources from one dimension

addition, with the application of AI (artificial intelligence) technology, mainly based on deep learning, to solve increasingly complex practical problems including machine vision and natural language processing, the scale of deep learning models is becoming larger and larger, and the training cycle of deep learning models is also changing towards complexity and length. There is no system, pattern or architecture that can significantly improve the training cycle from basic components. Often, iteratively training a complex large-scale deep learning model can take several hours, days, weeks, or months, which means extremely high time and resource costs for the rapidly developing information age. Even though cloud computing systems can support distributed iterative training and improve training speed to some extent, their training speed still cannot meet the requirements of practical applications and needs to be greatly optimized.

Most resource scheduling problems in cloud computing environments belong to NP (Non Deterministic Polynomial) polynomial complexity non deterministic problems Difficult problem [1,4], currently there is no polynomial time complexity method that can ensure global optimization of resource scheduling. Some heuristic algorithms, meta-heuristic algorithms, random algorithms, and machine learning algorithms are used to seek better approximate solutions. Meta heuristic algorithms, deep reinforcement learning, and other algorithms can to some extent solve the problem of variable optimization objectives and scenarios. However, as the complexity of the problem increases, these algorithms require a significant amount of computing resources and time. However, algorithms with lower time complexity such as LPT (longest processing time), FCFS (first come first serve), RR (round robin), greedy, and randomized algorithms have significantly lower optimization performance than other algorithms with higher time complexity. Therefore, they can only solve resource scheduling problems in scenarios where solution optimization requirements are not high. At present, an important issue facing resource scheduling is how to balance the optimization and complexity of algorithms, which also determines the upper limit of the energy conversion rate of the entire cloud system. In addition, the increase in algorithm complexity will reduce the practical value of algorithm deployment and application, and cause resource bottlenecks and downtime risks to the operation process of cloud systems. In addition to cloud computing, other distributed computing systems, such as grid computing [16], peer to peer computing [17], utility computing [18], edge computing [19], fog computing [20], also face resource management problems. The algorithms that can be used for resource scheduling problems in certain distributed systems often have the po-

tential to be applied to other distributed systems, and can even be applied to optimization problems in other industries or fields, such as airport comprehensive scheduling [21], assembly scheduling, public transportation scheduling [22], etc. Therefore, research on cloud computing resource scheduling algorithms is not only beneficial for the development of cloud computing and resource conservation, but also contributes to the development of other fields or industries. The architecture design and theoretical exploration of optimization scheduling algorithms are not only beneficial for exploring specific tools for practical engineering applications, but also for exploring computer algorithms and their related fundamental theories.

1.2 Research Trends and Development Status

Considering that this dissertation aims to explore methods or strategies to enhance the resource management capabilities of distributed computing systems such as cloud computing in a wide range of scenarios, and to promote the expansion of scheduling related optimization algorithms and their theories, the main research content of this dissertation involves the design of optimization algorithm architectures in multiple scheduling scenarios of cloud computing, exploration of algorithm theories, workflow task architecture design, and cloud system architecture construction. Therefore, this section will review the research trends and development status of the following three aspects: cloud computing development, optimization problems in cloud computing resource scheduling (referred to as cloud scheduling), and optimization algorithms in cloud scheduling. Given that this dissertation mainly focuses on the design and research of optimization algorithms, this section focuses on reviewing optimization algorithms in cloud scheduling.

1.2.1 Development of Cloud Computing

With the advent of Web 2.0 [23] in the era of network information, the flow of network information presents the characteristics of big data, high speed, and multifunctionality. Amazon, Google, IBM, Microsoft, Yahoo, and others have made early attempts to build cloud computing platforms. The literature [23] gives an overview of cloud computing: ”Now is the era of Web 2.0, the era of collaborative production of Internet data, and cloud computing can become its platform.” In the literature [23], Fox and others believe that cloud computing will be possible due to extensive broadband Internet access, provider capabilities, appropriate billing models, and the development of efficient virtualization

technology. At the same time, experts began to discuss the relationship and differences between cloud computing and grid computing. Foster et al. compared cloud computing and grid computing from multiple aspects such as business models, architecture, resource management, programming models, application models, and security models. Foster et al. [16] added a definition to cloud computing: large-scale distributed computing paradigm driven by economies of scale, in which abstract, virtualized, dynamically scalable, and manageable computing power, storage, platforms, and service pools are delivered to customers on demand through the Internet. In Foster's definition of cloud computing, the difference between cloud computing and traditional distributed computing paradigms is that it has scalability, can be encapsulated as an abstract entity that provides different levels of services to customers, is driven by economies of scale, and has dynamic services. In the literature [24], Klems et al. believed that cloud computing is an emerging trend of providing scalable and reliable services on the Internet as a computing tool. Armtrust et al. [25] gave the definition of cloud computing: "cloud computing refers to applications delivered through the Internet as services, as well as hardware and system software in data centers that provide these services", and demonstrated the advantages of public cloud over private data centers. Armbruct et al. [25] believe that the characteristics of public clouds include: cloud systems provide on-demand computing resources without time and space limitations, cloud users do not need to make prior commitments, cloud users can pay for the use of computing resources in the short term according to their needs, large-scale data centers bring economies of scale, and resource virtualization technology can simplify operational operations and improve system utilization. The above characteristics are not possessed or generally not possessed by traditional data centers. Buyya et al. [26] discussed cloud computing as a new information technology platform and mentioned that with significant advances in information and communication technology, one day "computing" will become the fifth "utility".

In cloud computing, various models or services such as virtual machines, cloud disks, and big data cloud platforms have been introduced and developed [27,28]. At present, in addition to cloud computing, other new distributed computing system architectures or models that are independent of, dependent on, or collaborative with cloud computing platforms have been proposed, including Edge computing [19], fog computing [20], no service computing [29], micro service [30]. In addition, with the development of deep learning technology and the growth of model scale, GPU cluster cloud systems are also one of the

most common distributed systems currently available [31,32].

1.2.2 Optimization Problems in Cloud Scheduling

In distributed systems, scheduling problems are usually NP-hard [1,4,33,34]. Some of the mainstreams in Cloud scheduling focus on objectives including minimizing energy consumption [35–37], minimizing makespan [38–40], minimizing delay time (or delayed services) [41–43], reducing response time [12,35,44], maximizing the degree of load balancing [38,45,46], increasing reliability [12,35,41], increasing the utilization of resources [47–49], maximizing the profit of providers [38,39,50], maximizing task completion ratio [12,51,52], minimizing Service Level Agreement (SLA) Violation [12,47,53], maximizing throughput [37,54,55], and multi-objectives [36,37,39].

There are multiple ways to classify optimization problems, among which the two main classification methods are based on the characteristics of the optimization objective function and the feasible solution space.

According to the characteristics of the optimization objective function, there are also multiple classification methods. According to the number of optimization objectives, it can be divided into single objective optimization and multi-objective optimization. The corresponding algorithms can also be divided into single objective optimization algorithms and multi-objective optimization algorithms. Some multi-objective optimization algorithms include: non dominated sorting genetic algorithms (NSGA), multi-objective evolutionary algorithms based on decomposition (MOEA/D), etc. According to the characteristics of the optimization objective function, it can be divided into linear optimization problems and nonlinear optimization problems. The linear optimization problem refers to the optimization objective function being linearly related to all optimization independent variables. Linear programming is often used to solve such problems. In the actual optimization process, the relationship between the optimization objective function and the optimization independent variable also includes: the optimization objective is a nonlinear function relationship about the optimization independent variable, the optimization objective is an implicit mapping relationship about the optimization independent variable, the optimization objective is a time-varying non stable mapping relationship, and the optimization objective is dynamically influenced by the parameters of other non optimization independent variables, etc. These optimization problems belong to nonlinear optimization problems in a broad sense. For nonlinear optimization problems, the optimization

objective value is difficult to show a significant monotonic increase or decrease trend in a certain mapping dimension of the feasible solution space. The commonly used algorithms for solving nonlinear optimization problems are metaheuristic algorithms, including genetic algorithms, particle swarm optimization algorithms, simulated annealing algorithms, etc.

The classification based on the characteristics of feasible solution space also includes various classification methods. According to the discrete and continuous characteristics of the feasible solution space, it can be divided into continuous solution space optimization problems and discrete optimization problems. A special case of continuous solution space optimization problem is convex optimization problem. Discrete optimization problems include integer programming (0-1 knapsack problem, traveling salesman problem). According to the complexity (or measure) of the feasible solution space, it can be divided into problems where the feasible solution space grows in polynomial form and problems where the feasible solution space grows in exponential form. The problem where the feasible solution space grows as a polynomial function can be expressed as having two polynomial functions p_1 and p_2 such that the complexity c of the solution space satisfies $p_1\ lec\ lep_2$. The property of the problem where the feasible solution space grows exponentially is that there is no polynomial function p such that $c\ lep$ always holds. This problem can be expressed as the existence of two exponential functions e_1 and e_2 such that the complexity of the solution space always satisfies $e_1\ lec\ lee_2$.

In existing distributed computing systems with cloud computing as one of the main modes, several common optimization scenarios include scheduling scenarios that consider single dimensional resources [56,57], scheduling scenarios that consider multi-dimensional resources [58], scheduling scenarios that consider task correlation in workflows [4,34], and joint scheduling scenarios that consider multiple optimization problems for multi region subsystems [48]. With the development of machine learning related technologies and the expansion of model scale, a common workflow scheduling scenario is the parallel training workflow of machine learning models deployed on distributed machine learning platforms (such as GPU server clusters). In cloud scheduling, it includes both single objective optimization problems and multi-objective optimization problems; Most of them are discrete optimization problems and nonlinear optimization problems (some of which can be approximated as 0-1 integer programming problems); Most of them are NP hard problems, so they are also problems where the feasible solution space grows exponentially. Given

the difficulty of NP hard problems and the massive scale of cloud systems, small improvements to the optimization solution of any cloud scheduling problem will bring significant benefits to optimization algorithm theory, cloud systems, and related industry fields.

1.2.3 Optimization Algorithm in Cloud Scheduling

1.3 Scheduling and Algorithms in Cloud

Referring to existing studies of Cloud scheduling and for the sake of comprehensive discussion, we can establish a universal formulation for scheduling problems. It can be assumed that the number of indivisible tasks is M, the number of server nodes is N, and each server node has D dimensional resources (such as CPU load, GPU load, RAM, bandwidth, disk storage, etc.). Then, the i-th task can be represented by a parameter matrix $V_i = \{v_{ijk}\}_{N\times D}$ where $1 \leq i \leq M$, $1 \leq j \leq N$, $1 \leq k \leq D$, and v_{ijk} indicates the capacity or space or time requirement for j-th dimensional resource when the i-th task is allocated to the j-th service node. The set of parameters of tasks $\langle V_1, V_2, \ldots V_M \rangle$ is set as $V = \{v_{ijk}\}_{M\times N\times D}$. The parameters of server nodes can be set as $L = \{l_{jk}\}_{N\times D}$, where l_{jk} means the load status of the k-th dimensional resource in the j-th server node. Using a matrix $X = \{x_{ij}\}_{M\times N}$ to represent the allocation solution of mapping "Tasks $\rightarrow$ Resources" and a matrix $S = \{s_i\}_M$ to represent the start time of tasks, then a scheduling scheme can be expressed by the combination of X and S, marked as $\langle X, S \rangle$. Wherein, $x_{ij} \in \{0, 1\}$ and $\sum_{j=1}^{N} x_{ij} = 1$, which means the indivisible task can be allocated to only one node. $x_{ij} = 1$ means the i-th task is allocated to the j-th node. Limiting S can generate the execution order between tasks. For example, setting $s_{i_1} \geq e_{i_2}$ (where e_i is the end time of the i-th task) equals that the i_1-th task must begin after the finish of the i_2-th task. Thus, the matrix S is sufficient to include the execution order of the task.

A optimization result of scheduling is a mapping from the solution $\langle X, S \rangle$ and the parameters of tasks V and server nodes L. Thus, the optimization objective can be set as

$$\min \omega = \omega(X, S, V, L) \tag{1-1}$$

where ω is a function with respect of X, S, V and L. Multi-objectives can be represented

by multiple functions of ω as

$$\min \omega = \begin{cases} \omega_1 (X, S, V, L) \\ \omega_2 (X, S, V, L) \\ \cdots \end{cases} \tag{1-2}$$

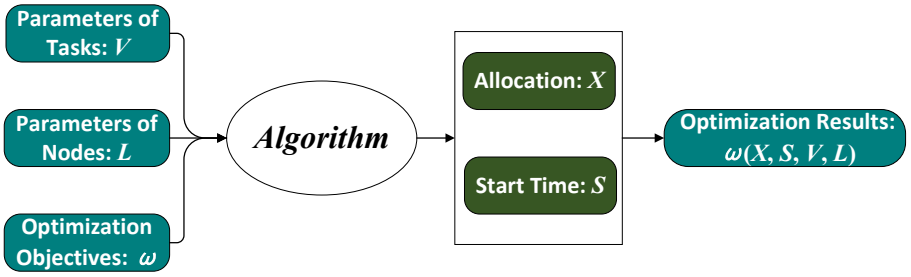


Figure 1-1 A Diagram of Scheduling Algorithm to Generate the Scheme $\langle X, S \rangle$.

With the formulation of the scheduling problem, a scheduling algorithm is an integration of mappers from (V, L, ω) to the scheduling scheme $\langle X, S \rangle$. It can be set an algorithm as Al and its solution can be expressed by

$$Al (V, L, \omega) = \langle X, S \rangle \tag{1-3}$$

Thus, a process of using an algorithm to solve the optimization solutions can be shown as Fig. 1-1. From Fig. 1-1, two key factors for scheduling are production and evaluation of schemes. In solving scheduling schemes, the evaluation for the performance of an optimized solution, i.e. the process of obtaining $\omega (X, S, V, L)$ or its equivalent evaluation functions, is crucial. Some simple optimization objectives in ideal scenarios can be directly calculated. However, for some complex optimization objectives, this function $\omega (X, S, V, L)$ may not have explicit expressions. E.g., for minimizing energy consumption cannot be represented by elementary functions generally so that the expression of $\omega (X, S, V, L)$ is implicit. For some optimization objectives with explicit expressions in ideal scenarios, it may be also difficult to directly calculate the optimization results when in some highly stochastic system processes. The different mapping processes of Eq. 1-3 will correspond to different categories of algorithms.

When considering dynamic scheduling, a diagram of its process over time can be seen in Fig. 1-2. The scheduling scheme at a time t is responsible for meeting the scheduling requirements at the current time, but will also be related to the status of server nodes at subsequent times. It indicates that when making scheduling decisions at time t, it is necessary

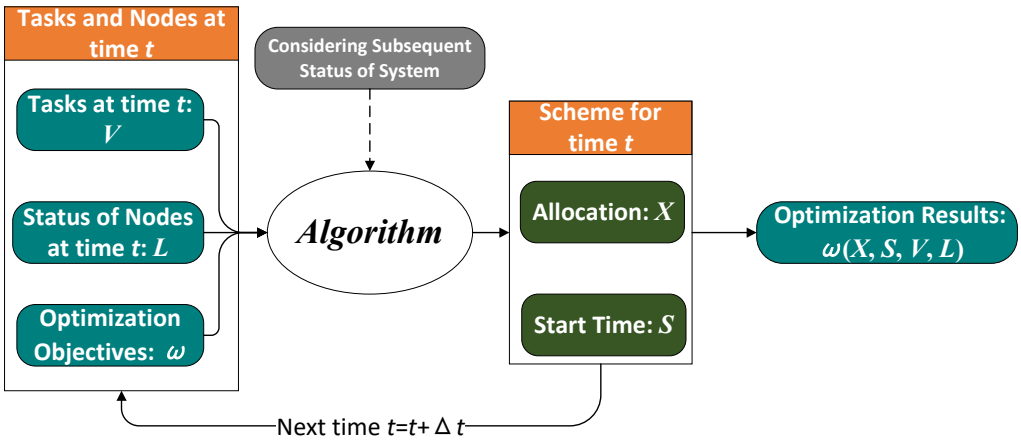


Figure 1-2 A Diagram for Continuous Dynamic Scheduling Process over Time *t*.

to consider the subsequent changes in the system. This also puts forward requirements for evaluating the quality of scheduling schemes, which shows the significance of a predictor.

Generally, algorithms for Cloud scheduling contain six categories: Dynamic Programming(DP), Probability algorithm (Random), Heuristic method, Meta-Heuristic algorithm, Hybrid algorithms and Machine Learning. From the properties of these algorithms, except for ML, other algorithms do not have the ability to predict system states. In this dissertation, we regard dynamic programming, randomization, heuristic method, meta-heuristic algorithm, and hybrid algorithm as classic approaches. In order to analyze the future direction of Cloud scheduling and discuss the potential application of DRL, we will review the current scheduling algorithms of Cloud.

Heuristic is an algorithm to solve an optimization problem based on intuitionistic or empirical construction. Due to their lower complexity, heuristic algorithms are prevalent in some scenarios with a clear evaluation function requiring rapidity but not requiring high optimization results. Additionally, the worst-case of heuristic algorithms is generally predictable hence with a lower risk of improper allocation.

In existing research, [59] applied the Jacobi Best-response Algorithm (JBA) to minimize cost in Multi-Broker Mobile Cloud Computing Networks and proved theoretical results demonstrating the existence of disagreement points and convergence of Jacobi Best-response Algorithm of the brokers to disagreement points. [56] proposed an adapting Johnson's model-based algorithm with 2-competitive to minimize the makespan of multiple MapReduce jobs and proved its performance in theory. [60] proposed Peak Efficiency Aware Scheduling (PEAS) to optimize the energy consumption and QoS in the on-line virtual machine allocation and reallocation of Cloud. Dynamic Bipartition-First-

Fit (BFF), a $(1 + \frac{g-2}{k} - \frac{g-1}{k^2})$ competitive algorithm based on First-Fit algorithm, was proposed and its performance was proved theoretically by [57]. The work [61] proposed a QoS-Aware Distributed Algorithm based on first-come-first-improve (FCFI) and all-come-then-improve (ACTI) algorithms to reduce computation time and energy consumption of Industrial IoT-Edge-Cloud Computing Environments. ECOTS (energy consumption optimization cloud task scheduling algorithm), with low time and space complexity, took into account multiple key factors such as task resource requirements, server power efficiency model and performance degradation in order to reduce energy consumption of Cloud [62]. Longest Loaded Interval First algorithm (LLIF), a 2-approximation algorithm with theoretical proof of its performance, was proposed by [15] to minimize the energy consumption of VM reservations in the Cloud.

Other common heuristic methods are Johnson's model, FF (first fit), BF (best fit), RR (round-robin), FFD (first fit decreasing), BFD (best fit decreasing), Jacobi Best-response Algorithm [59] and their variants.

From existing research, heuristic algorithms mainly focus on the single-objective optimization including minimizing makespan, minimizing energy consumption and load balancing. However, there are several defects of heuristics as follows.

(1) For the scenarios using heuristic, some major objects (such as the time, energy or load) are often assumed to be given or easily calculated. For complex scenarios where the optimization objective is implicit with respect to solutions, heuristic algorithms often fail to generate a feasible solution.
(2) A heuristic algorithm is often designed for one or few specific scenarios. When only one element in the scenario changes, the algorithm may need to be redesigned.
(3) Heuristics are usually only suitable for the single-objective problems.
(4) Moreover, the solution of heuristic algorithms can usually be further optimized.

In skeleton, meta-heuristic, the combination of heuristic and randomization [10], includes Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC), Genetic Algorithm (GA), Firefly Algorithm (FA), etc.

ACO imitates ant colony to search for food as a search route. [58] proposed OEMACS combining OEM (order exchange and migration) local search techniques and ACO to resolve energy consumption of VMs deployment in Cloud computing, which significantly reduced the energy consumption and improved the effectiveness of different resources compared with conventional heuristic and other evolutionary-based approaches. [63] pro-

posed two ant colony-based optimization algorithms (TACO) to address VM scheduling and routing in multi-tenant Cloud data centers aiming at improving the utilization of energy in Cloud computing. [64] proposed an alternative meta-heuristic technique based on the Ant Lion Optimizer Algorithm (MALO) to resolve multi-objective optimization of Cloud computing, which performed better in load balancing and makespan compared with GA, MSDE, PSO, WOA, MSA and ALO.

GA imitates the process of natural evolution as a search route of the algorithm. Proposed by [65], NSGA-II occupies better convergence and optimal solution and has become one of the benchmarks using the fast non-dominated sorting algorithm, introducing elite strategy and using congestion-congestion comparison operator. [66] improve the search strategy based on NSGA-II to reduce the energy consumption, response time, load imbalance and makespan in Cloud computing. NSGA-III utilizes reference points with preferable distribution as a novel search route to maintain the diversity of the population to improve the optimization results of GA [67,68]. [69] applied NSGA-III to optimize the execution time and energy consumption of IoT-enabled Cloud-edge computing. MOGA [70] and MOEAs [9] improved the search route strategies based on NSGA-II and were utilized to settle Cloud scheduling.

The studies of Firefly algorithm include FA [46] and FIMPSOA [55]. That of PSO include MOPSO [71], TSPSO [72] and HAPSO [73]. Other meta-heuristic algorithms include Multi-objective Whale Optimization Algorithm (MWOA) [74], nature-inspired Chaotic Squirrel Search Algorithm (CSSA) [75], etc.

From existing research, meta-heuristic algorithms with searchability for the solution can address more complex optimization problems not only for single-objective problems but also for multi-objective problems. They are applied to solve optimizing cost, energy consumption, makespan, running time and resource utilization. Meta-heuristic algorithms are more applicable than heuristic algorithms but at the expense of computational complexity and randomness. However, although these optimization objectives in meta-heuristics include some complex objects (such as energy, Qos and cost), their calculations have been simplified with some ideal assumptions far from reality [63,69,76].

Meta-heuristic and other search algorithms are based on the specific search route, whose diagram can be seen in Fig. 1-3. They use the search route to adjust the current solutions to generate new solutions, evaluate the performance (such as fitness) of newly generated solutions according to the optimization objectives-based evaluation functions,

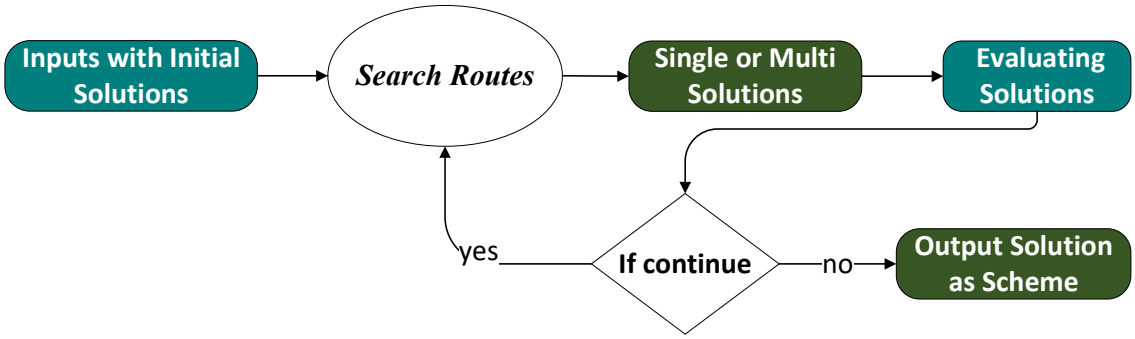


Figure 1-3 A Diagram of Search-based Algorithms.

and then determine whether to proceed to the next search based on the evaluation. The two key factors in Fig. 1-3 are the search route and evaluation of solutions. The search route needs to generate better solutions. However, there are several inevitable defects of meta-heuristic as follows.

(1) For scenarios where the solution can be directly evaluated, the convergence of the meta-heuristic cannot be guaranteed due to the presence of randomness. The randomness of the meta-heuristic increases redundant computations.

(2) As the search space increases, the required search iterations must also increase accordingly, subsequently producing more redundant solutions.

(3) When it is difficult to evaluate the quality of a solution, the search route will lose its direction, and the search algorithm will degenerate into pure randomization. When $\omega(X, S, V, L)$ is implicit, the meta-heuristic and other search algorithms themselves do not provide a method for evaluating solutions.

(4) Meta-heuristic also does not provide a way to predict system status.

The first and second defects will limit the optimality of the meta-heuristic for its feasible scenarios. The third and fourth defects, which also appear in heuristic, will cause the algorithm unable to be used in some real-world complex scenarios.

The optimization algorithm based on machine learning mainly establishes a mapper that can generate optimized solutions, driven by data or instances, so that the mapper can learn the generation mapping of more optimized solutions. Machine learning based optimization algorithms mainly include deep learning (DL) and reinforcement learning (RL). In addition, other types of machine learning methods such as KNN (K-nearest neighbor, K-nearest neighbor algorithm) [77], imitation learning [2], SVM (support vector machine) [78], etc. have also been applied to cloud scheduling, but these algorithms usually perform certain classification tasks (or system state recognition tasks) without directly participating

in the generation of optimization solutions. QEEC [49] is an efficient and energy-saving cloud computing task scheduling framework based on Q-learning, which uses Q-table to represent the probability of decision makers' actions. PCRA [79] implements feedback control through resource allocation prediction of Q-value prediction model in RL through feedback control Operations under different system states. DeepRM uses a neural network with six layers of Convolutional Neural Network (CNN) to describe successful decision mappers based on Deep Neural Network (DNN) in image processing. The central cluster, waiting queue, and backlog queue constitute the environment state represented by the image. Other cloud scheduling algorithms based on Deep Reinforcement Learning (DRL) include HCDRL [80], DT [81] using GPT, etc.

Machine learning based optimization algorithms (mainly reinforcement learning) generally have good modeling ability, strong adaptability to optimization objectives, and experience memory ability (through experience replay, optimization solutions can be generated directly without searching the solution space). However, as machine learning is a training based optimization solution generator, its main problem is that it cannot ensure the optimality and stability of the optimization results before making optimization decisions. That is, when any factor in the optimization scenario is disturbed, the optimization effect of machine learning may significantly decrease, and even unacceptable optimization decision solutions may be generated in some scenarios. This makes it have the following shortcomings.

(1) Low sample utilization and long training time: requiring an extremely large number of training samples. And the training cost is high, and its applicable scenarios are limited by sample data. For optimization problems with exponentially increasing complexity of feasible solution space, the sample size for training reinforcement learning also needs to be exponentially increased, and the exponential increase in sample size often requires a higher demand than the exponential complexity of feasible solution space. The unexplainability of the training process poses a challenge to theoretical deduction based on mathematical techniques. The modeling and theoretical derivation of high-dimensional continuous state spaces require the development of mathematics.
(2) Poor optimization: The final optimization performance is often not good enough. In scenarios where it is easy to evaluate the quality of optimized solutions, the optimization effect of reinforcement learning is far inferior to most metaheuristic algorithms and some heuristic algorithms designed specifically for scenarios.

(3) Poor adaptability to changes in scenarios and environments: retraining is required to meet new optimization objectives. For the same optimization objective with different parameter inputs, it still lacks scalability. The same reinforcement learning may even struggle to adapt to an increase in input formats.

Some other classic algorithms used in Cloud scheduling mainly contain DP, Random algorithms, and hybrid algorithms (combining two or more algorithms). Among them, hybrid algorithms are also widely used in solving complex scheduling problems in Cloud computing. Hybrid algorithms can combine the advantages of multiple algorithms to produce better solutions. In terms of search algorithms, a single algorithm has an inherent local convergence solution and the solution of the hybrid algorithm needs to satisfy the convergence conditions of multiple algorithms simultaneously [82]. Therefore, the convergence solution of the hybrid algorithm is usually better than the corresponding single algorithm. PSO-ACS [83], mingled with PSO and ACO, applied PSO to find the optimal solution of task scheduling and ACO to find the best migration path of VMs on PMs. FACO [28], a hybrid fuzzy ant colony optimization algorithm, exploited a fuzzy module dedicated to pheromone evaluation to improve the performance of ACO by optimizing the search route of ACO. Hybrid Genetic-Gravitational Search Algorithm (HG-GSA) [84] based on gravitational search algorithm for searching the best position of the particle consequently optimizing the search route of GA. FMPSO (modified PSO + fuzzy theory) [85] used crossover and mutation operators surmounting the local optimum of PSO and applied a fuzzy inference system for fitness calculations. SFLA-GA algorithm (shuffled frog leaping algorithm + GA) [86] took advantage of the two algorithms to transmit information among groups hence the optimal search route. GHW-NSGA II [87] leveraged heuristic-based search algorithm as an extra search route of NSGA II to optimize the utilization of multi-dimensional resources, which improved the convergence speed and optimality on the basis of GA. SPSO-GA [88] combined Self-adaptive Particle Swarm Optimization algorithm with Genetic Algorithm operators to reduce the energy consumption of the scenario offloading DNN layers Cloud-Edge environment. On the basis of SPSO-GA, PSO-GA-G [89] added a greedy algorithm to optimize computation offloading. The combination of multiple meta-heuristics is beneficial for improving the overall convergence speed, hence improving search efficiency. LPT-One and BFD-One [82] used heuristic algorithms to act as the search routes and combined multiple heuristic-based search routes to improve the approximation of minimizing makespan.

Other hybrid algorithms in Cloud scheduling, include ABC-SA integrating the functionality of simulated annealing (SA) into artificial bee colony [90], SFGA (a hybrid Shuffled Frog Leaping Algorithm and Genetic Algorithm) [91], TSDQ-hybrid meta-heuristic algorithms based on Dynamic dispatch Queues [92], etc. These algorithms demonstrated the flexibility, superiority, adaptability and mobility of hybrid algorithms and simultaneously manifested the unlimited possibilities and significance of research hybrid structurally.

Similar to metaheuristic algorithms, hybrid algorithms are also suitable for various multi-objective problems, and due to the joint use of multiple algorithms, their local optima are often better than those of a single algorithm. However, hybrid algorithms based on multi restart or metaheuristic algorithms cannot exceed the scenarios where the basic algorithm is applicable. In addition, due to the use of multiple algorithms, their computational complexity will be higher than that of a single algorithm at the same number of iterations. At this time, it is necessary to ensure that the performance of a single search of the hybrid algorithm is higher than that of a single algorithm under the same computational consumption.

1.4 Research Content and Key Issues

From the review of existing research work, it can be seen that most resource scheduling problems in distributed computing systems belong to NP hard problems, where optimization algorithms are the key components for solving these problems. Currently, various optimization scheduling algorithms (heuristic algorithms, metaheuristic algorithms, machine learning algorithms, and hybrid algorithms) are facing their own inherent defects and challenges (due to the types and forms of algorithms themselves). At the theoretical level of algorithms, the common core of these defects and challenges is how to obtain more optimized feasible solutions with less computational resources (time, space, equipment, etc.) consumption. At the application scenario level, there are various types of resource scheduling problems in cloud environments. One of the common problems facing the industry is how to improve the adaptability of cloud computing systems to changes in various elements of cloud resource scheduling scenarios, in order to enhance the overall resource management capabilities of distributed computing systems. Merely studying a certain type of algorithm and a certain scenario is no longer sufficient to address the issues and challenges mentioned above. This also presents a new challenge that must be addressed (difficult but necessary): comprehensive research on multi class algorithms and

multi scheduling scenarios (including system environment and optimization problem modeling, system architecture and algorithm design, theoretical analysis, experimental evaluation) Given the above professional consensus, this dissertation takes ”exploring methods or strategies that can enhance the resource management capabilities of distributed computing systems, including cloud computing, and other industries to a certain extent and within a certain scope” as the starting point, and focuses on ”modeling and optimizing algorithm architecture design and theoretical analysis of cloud computing resource scheduling problems”. It attempts to explore and discuss changes in resource dimensions, task correlation, task granularity, and multi subsystem hierarchy.

1.4.1 Research Contents

Due to the difficulty in enumerating all possible scheduling scenarios in cloud environments, this dissertation selects five sets of resource scheduling scenarios (considering the starting point and focus of this dissertation), corresponding to the five research contents:

(1) Optimization scheduling of independent task sets in cloud server nodes with single dimensional resources: Research modeling of single dimensional resource scheduling problems, study and design optimization algorithms suitable for homogeneous and heterogeneous single dimensional resource scheduling problems, and explore their theoretical approximation ratios.

(2) Optimization scheduling of independent virtual machine sets in cloud server nodes with multi-dimensional resources: Study modeling of multi-dimensional resource scheduling problems, research and design optimization algorithms suitable for complex single objective optimization and multi-objective optimization problems, improve algorithm convergence speed and optimization of convergence solutions.

(3) Optimization scheduling of pipeline parallel deep learning model training workflow based on equal microbatch data partitioning in GPU cloud server nodes: Study the joint optimization problem modeling of pipeline model partitioning and microbatch processing data partitioning for deep learning model parallel training workflow in GPU cloud distributed systems, research and design corresponding optimization algorithms, and explore the theoretical approximation ratio and computational complexity of the algorithms.

(4) Design and Optimization Scheduling of a New Parallel Training Architecture for Deep Learning Models in GPU Cloud Server Nodes: Study a new parallel training

workflow architecture for deep learning models in distributed systems based on unequal data partitioning, research and design optimization algorithms for training schemes under the new architecture, and conduct theoretical analysis of the architecture and algorithm.

(5) Joint optimization scheduling of diverse task requests and diverse optimization objectives in multi-level and multi subsystem cloud computing systems: Exploring and researching a new system architecture pattern for high-performance cloud resource management, and studying scheduling algorithms and strategies for dealing with multiple subsystems, time-varying optimization objectives, and dynamic resources.

1.4.2 Key Issues

Corresponding to the above scenarios and research content, this dissertation attempts to explore the following five key issues:

(1) How to construct a mathematical model for one-dimensional resource optimization scheduling problems, analyze and improve the approximation ratio of optimization scheduling algorithms (such as heuristic algorithms and local search algorithms): Currently, the approximation ratio of some commonly used algorithms still needs to be improved. The approximation ratio of an algorithm is one of the important indicators for measuring its deterministic performance. In large-scale cloud computing systems, the uncertainty of algorithms can lead to unpredictable situations. Therefore, an algorithm with good approximation ratio may improve the overall optimization and stability of cloud computing systems. In addition, the exploration of algorithm approximation ratios has also influenced the development of algorithm theory. At present, the mathematical theoretical modeling of resource scheduling problems has always been an urgent problem to be solved in both academic and practical engineering fields. In addition, research on optimizing and scheduling single dimensional resources is also the foundation of multidimensional resource scheduling research (multidimensional resource scheduling problems are often transformed into multiple single dimensional resource scheduling problems).

(2) How to model multidimensional resource scheduling problems and improve the convergence performance of effective Pareto solutions: In practical cloud computing resource scheduling scenarios, it is necessary to consider the collaborative operation of multiple resources. For massive cloud computing task requests, the operational status between different resources is interdependent. The bottleneck of single dimensional resources will also affect the use of other dimensional resources. How to analyze the coupling

relationship between different resources and allocate multidimensional resources reasonably? How to establish a multi-objective optimization problem for multi-dimensional resource scheduling and find the Pareto boundary for multi-dimensional resource scheduling? These are important issues that urgently need to be addressed.

(3) How to build a theoretical loss model for parallel training workflow of deep learning models with equal data partitioning, analyze and improve the optimal performance of optimization algorithm theory: In the existing theoretical research and engineering fields, deep learning model training tasks are a type of task that occupies the main computing resources. If this is used as a practical scenario, it is necessary to perform feature modeling and quantitative analysis on deep training tasks, and construct the relationship between the parameter elements of the training task and the required resources. The research on accelerating parallel training workflow will be related to the efficient operation and maintenance of widely deployed GPU cloud servers.

(4) How to design a new architecture for parallel training of deep learning models based on unequal data partitioning in distributed systems, and quickly solve optimization training schemes within an effective time range: Existing distributed parallel training architectures are limited to equal data partitioning, which means that in each device, all processes including computation and communication in each layer of the network have the same number of microbatches. In real-world scenarios, the time required for computation and communication is not necessarily proportional to the amount of data, and the optimal number of partitions for each layer of the network may not be the same. Continuing to maintain equal partitioning will result in resource waste. Therefore, it is necessary to introduce unequal data partitioning. However, due to uneven data partitioning, it will greatly increase the difficulty of modeling theoretical loss models, which will result in the inability to solve the optimization training scheme for the new architecture within an acceptable time. Therefore, it is necessary to establish and improve theoretical models and optimization algorithms for new architectures while advancing the theory.

(5) How to build a new system architecture pattern for cloud computing and solve the dynamic resource scheduling problem of multiple subsystems considering time-varying optimization objectives: The existing flat deployment method limits the resource management capability of cloud computing platforms and affects the efficient use of all resources. How to build a new architecture model for cloud computing systems is related to the long-term development of cloud computing in the future. In addition, in actual engi-

neering application environments, the demand for cloud computing systems often dynamically changes with time and space switching, which increases the difficulty of applying resource scheduling algorithms in practical scenarios. It can be confirmed that currently no algorithm can outperform other algorithms in all resource scheduling optimization scenarios. Is it to search for this algorithm (which can outperform other algorithms in all scenarios) or to fully utilize existing algorithms? If we choose to make full use of existing algorithms, how should we utilize them? This is a question that scholars have been exploring and discussing.

1.5 Organization of this Dissertation

Chapter 1 mainly introduces the research background, significance, existing research trends, and development status of this dissertation, as well as the research content, key issues, relevant plans or methods of this dissertation.

Chapter 2 models a distributed computing system considering single dimensional resources and its minimum completion time optimization scheduling problem. It introduces a series of multi-path search algorithms based on heuristic algorithms as search paths to solve single objective optimization problems in homogeneous systems and dual objective optimization problems in heterogeneous systems. For the minimum completion time problem in isomorphic systems, the approximation ratio and theoretical proof of multiple sets of multi-path algorithms are introduced. Based on simulation experiments, the optimization performance of the multi-path search algorithm series in homogeneous and heterogeneous distributed systems considering single dimensional resources has been verified.

Chapter 3 models a distributed computing system considering multi-dimensional resources and its resource utilization and energy consumption optimization scheduling problems. It introduces a series of growable genetic algorithms based on heuristic search paths as growth paths to solve multi-objective optimization problems and single objective optimization problems (with complex expressions) in heterogeneous systems. Based on simulation experiments and real dataset driven experiments, the optimization and convergence performance of the scalable genetic algorithm series in heterogeneous distributed systems considering multi-dimensional resources were verified.

Chapter 4 models the optimization scheduling problem of minimizing total training time for deep learning equal data partitioning pipeline parallel training workflow consid-

ering task correlation in GPU cloud systems. It introduces a cross search algorithm based on multi-dimensional improved binary method to solve the joint optimization problem of deep neural network layer partitioning and data partitioning under a single optimization objective (with analytical expressions). Theoretical proofs have been provided for the approximation ratio and global optimality of the multidimensional improved binary method. The parallel training experiment based on deep neural networks in a real GPU distributed cluster environment verified the fast performance and optimization performance of the cross search algorithm based on multi-dimensional improved binary method.

In Chapter 5, the architecture design and system modeling of unequal data partitioning for deep learning parallel training tasks with task correlation in GPU cloud systems are presented. The recursive formula for training time of unequal data partitioning workflow is derived, and a second-order improved acceleration multi chromosome genetic algorithm based on matrix form recursive formula is introduced to solve single objective optimization problems without analytical expressions (with complex recursive expressions). Theoretical analysis and proof were conducted on the optimization of the new parallel training workflow architecture and the optimization convergence of the multi chromosome genetic algorithm. Based on experiments in simulated datasets and real environments, the fast performance of the new parallel training workflow architecture and the optimization and fast performance of the two-level improved multi chromosome genetic algorithm were verified.

Chapter 6 models a distributed computing system with multiple layers and subsystems and its multi subproblem joint optimization scheduling problem (variable multiple optimization objectives). The resource scheduling algorithm is regarded as schedulable resources, and the scheduling algorithm selector architecture is introduced. Correspondingly, the deep learning based algorithm selector and the reinforcement learning based algorithm selector are introduced. Based on dynamic simulation experiments, the optimization performance of algorithm selectors in resource management of complex dynamic systems has been verified.

Chapter 7 summarizes the work of the entire dissertation and provide prospects for future work.

Chapter 2 Single-dimensional Resource Scheduling based on Multi-route Search Algorithms

Cloud computing, as a large-scale distributed computing system dynamically providing elastic services, is designed to meet the requirement of delivering computing services to users as subscription-oriented services. In general, the problems of resource scheduling in Cloud computing like minimizing makespan are usually NP-Hard problems. Various common algorithms including heuristic, meta-heuristic and machine learning are applied in resource scheduling of Cloud computing to obtain the solutions, which however are still probable and imperative to be optimized. Through innovatively applying heuristic algorithms namely LPT (Longest Processing Time) and BFD (Best Fit Decreasing) as the basic search routes and integrating these with neighborhood search algorithm namely OneStep, this chapter proposes multi-search-routes-based algorithms containing LPT-OneStep, BFD-OneStep and their combinations for the sake of enhancing theoretical performance and improving solutions of scheduling schemes especially for problems of minimizing makespan for homogeneous and heterogeneous resources. Theoretical derivations prove that the proposed algorithms possess better theoretical approximation ratios for $P||C_{max}$. Extensive experiments on simulation environment demonstrate the proposed algorithms outperform than corresponding compared algorithms for minimizing makespan problems in both homogenous resources and heterogeneous resources, which validates the superiority of the proposed algorithms.

2.1 Introduction

The emerging trend of Industry 4.0 and 5G significantly enhances the number of tasks that the Internet-based computing systems need to process in real-time. The increase in data size proposes a demand for a large-scale distributed system such as Cloud computing which can provide flexible, reliable and dynamic services to users [4]. Cloud computing, also as a policy to provision high-performance computation services in a pay-as-you-go manner, has supported increasingly complex software systems and computing programs substantially, which has indicated the indispensability of Cloud computing. Currently, many IT companies, such as Amazon, Microsoft, Google, Alibaba and IBM [93–96], have established relatively mature Cloud computing systems. These systems usually have the

stable structures such as Software-as-a-Service (SaaS), Platform-as-a-Service(PaaS) and Infrastructure-as-a-Service (IaaS), as well as can flexibly provide services to meet numerous requests from users [5,97].

Resource scheduling is defined by [5] as to find an "optimal" mapping "Tasks → Resources" to meet one or several given objectives. With the enlargement of the Cloud's user groups and facilities, extensive requests and reservations of users are challenging the resource scheduling of Cloud computing. Additionally, inappropriate resource scheduling will cause the waste of users' time, decrease QoS (quality of service), increase energy consumption, and increase carbon dioxide emissions. As the Cloud computing system is still expanding its scale and development of multitudinous industries depends on the reasonable operation of Cloud computing, therefore the research on its resource management is a prominent issue from the birth of Cloud computing to nowadays which will also affect the orientation and prospect of Cloud computing in the society [3,8].

One of the keys to address resource management of Cloud computing is the scheduler lying on the platform layer based on the resource scheduling algorithm. Cloud architecture and resource scheduling process are shown in Fig 2-1. The users operate the clients in the application layer to submit task requests to the Cloud center through the high-speed networks of the connection layer; The Cloud center on the platform layer collects tasks, generates scheduling schemes leveraging scheduling algorithms, and allocates tasks to server nodes including VMs and PMs on the resource layer and fabric layer; The server nodes then provide corresponding services to users. In Cloud computing, the scheduling algorithm is a crucial component affecting the quality of resource management. As the resource scheduling problem of Cloud computing is commonly an NP-Hard problem without a feasible method in polynomial time to ensure the global optimization of resource scheduling unless $P = NP$ [1,4,33], hence it is usually settled by heuristic, meta-heuristic, randomization and machine learning algorithms [98]. Some existing algorithms like Ant colony algorithm [63], NSGA II algorithm [99] and deep reinforcement learning algorithm [2,100] have achieved excellent performance in the research results. However, most of them with quite high computational complexity are unstable with randomness, which makes the worst case of scheduling results unpredictable with inevitable risk. For Cloud computing systems in the realistic scenario, some algorithms with determinacy and rapidity such as LPT (Longest Processing Time), FCFS (First Come First Server), RR (Round-Robin), BFD (Best Fit Decreasing), and SPT (Shortest Processing Time) are still

practical, whose results nevertheless require to be optimized yet [2, 101].

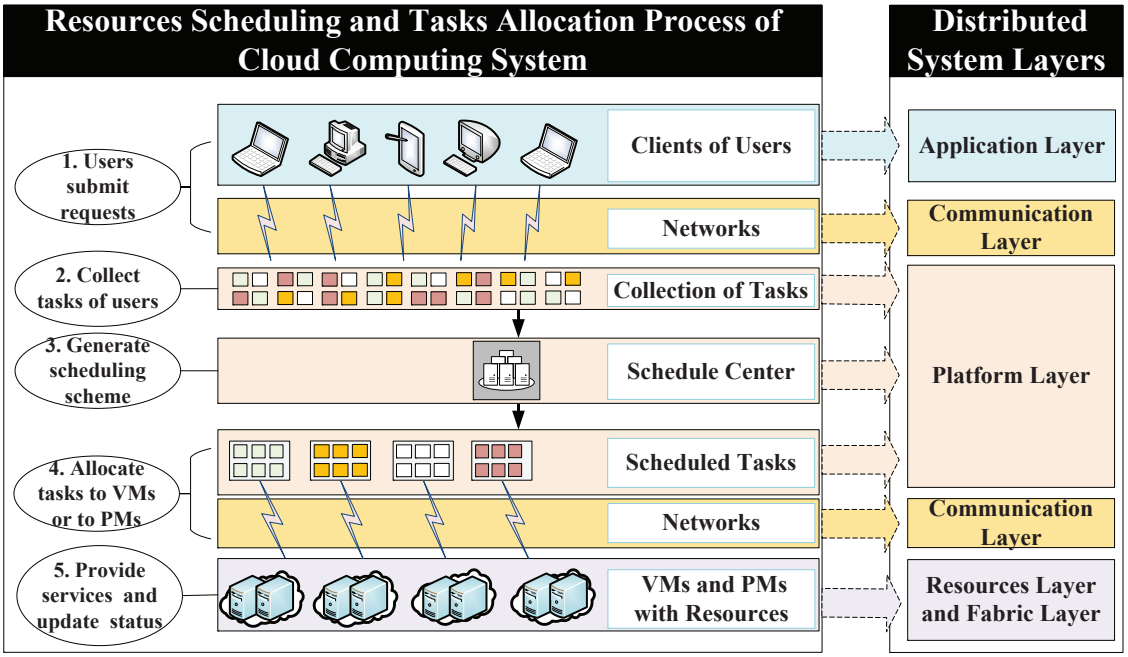


Figure 2-1 Cloud Architecture with Resources Scheduling Process.

Considering iterative optimization properties of the search algorithm as well as the low complexity and analyzable approximation ratio of heuristic, we apply heuristic algorithms as search routes in the general local search algorithm based on the neighbors of dual resources and propose heuristic-based local search algorithms. For the heuristic-based search algorithm, the selection of the heuristic will directly affect its convergence and optimization performance. Hence, leveraging LPT and BFD as its search routes in light of the better theoretical approximation ratios of LPT and BFD than other heuristic algorithms such as RR, FCFS and SPT to the optimization problems studied in this chapter. Based on the heuristic-based search algorithm, we propose multi-search-routes-based local search algorithms combining the One-Step search route (or *K*-Step search route) and heuristic-based search routes for the instances of minimizing makespan for homogeneous and heterogeneous resources. According to different combinations of search routes, we obtain various multi-search-routes-based algorithms containing LPT-One, BFD-One, LPT-BFD-One, etc, where the combination of multi-search routes is conducive to jumping out of the local optimum of the single search route to improve the performance of algorithms. As the proof of the theoretical approximation ratio of the search algorithm is rare in the previous research, this chapter theoretically proves the approximation ratios of several multi-routes local search algorithms for the problem of minimizing makespan in

homogenous resources (i.e. the scheduling problem of minimizing makespan on parallel machines abbreviated as $P||C_{max}$), which breaks the dilemma that the search algorithm rarely has approximation ratio proof in previous research. And then, this chapter provides experimental results for minimizing makespan of both homogenous and heterogeneous resources to validate the performance of proposed algorithms.

For $P||C_{max}$, the approximation ratios of existing algorithms are as that $Ar_{LPT} \leq \frac{4}{3} - \frac{1}{3M}$, $Ar_{LPT-REV} \leq \frac{4}{3} - \frac{1}{3(M-1)}$ [102,103], $Ar_{Multifit} \leq \frac{72}{61} + 2^{-k}$ where k is the number of attempts to find the smallest number of machines (by binary search) [33]. A principle as PTAS (Polynomial-Time Approximation Scheme) revealed that the guarantee of $Ar \leq 1+\varepsilon$ required complexity of $\mathrm{O}\left((\mathrm{N}/\varepsilon)^{1/\varepsilon^2}/\mathrm{M}\right)$ and no FPTAS (Fully Polynomial-Time Approximation Scheme) exists for $P||C_{max}$ unless $P = NP$ [33]. Additionally, few existing studies proved the approximation ratio of local search algorithms. The approximation ratios of our proposed LPT-One and BFD-One are proved as $\frac{5}{4} - \frac{1}{4M}$ and that of LPT-K and BFD-K are not greater than $1 + \frac{M-1}{(3+K)M}$ for $P||C_{max}$ where M is the number of resources, which reveals the increasing K can ameliorate the upper limit of the approximation ratio even to 1.

The main contributions of this chapter are as follows:

(1) Proposing a new framework of local search algorithms based on the neighbors of dual resources and heuristic-based search routes: this chapter proposes a framework of the local search algorithm based on the neighbors of dual resources. With neighbors of dual resources, this chapter defines various heuristic-based neighborhoods including neighborhoods of LPT, MLPT, BFD and Best-BFD, which correspond to heuristic-based search routes as LPTS, MLPTS, BFDS and BestBFDS respectively. Different from existing algorithms, heuristic-based search routes endowing heuristic algorithms with a novel role can significantly reduce the computational complexity of the local search algorithm and is advantageous to the theoretical derivation of the approximation ratio.

(2) Proposing multi-routes-based search algorithms: to further improve the performance of the scheduling algorithm, this chapter proposes the multi-routes-based local search algorithms to resolve minimizing makspan of homogeneous and heterogeneous resources in Cloud based on the combination of heuristic-based search routes and OneStep (K-Step) search route. Different combinations obtain various multi-routes algorithms containing dual routes search algorithms such as LPT-OneStep search (LPT-

One, LPTO), BFD-OneStep search (BFD-One, BFDO) and MLPT-One search, as well as triple routes search algorithm as LPT-BFD-One search, which outperform existing algorithms.

(3) Providing theoretical proofs of approximation ratios for multi-routes-based search algorithms: in theory, it's not easy to prove the approximation ratio of search algorithms. This chapter gives theoretical proofs of the approximation ratios for multi-routes-based search algorithms as that $Ar_{LPTO}, Ar_{BFDO} \leq \frac{5}{4} - \frac{1}{4M}$ and $Ar_{LPT-K}, Ar_{BFD-K} \leq 1 + \frac{M-1}{(3+K)M}$ for $P||C_{max}$, which are better than existing algorithms and have significance for the theoretical exploration of search algorithms.

(4) Simulation experiments: this chapter executes experiments with abundant instances to verify the performance of the proposed algorithms in both homogeneous and heterogeneous resources.

The remainder of this chapter is organized as follows. We review the related works in Section 2.2. The problem formulation and general local search algorithm are presented in Section 2.3. Based on various basic search routes, we propose the multi-search-routes-based algorithms and provide theoretical proof of approximation ratios of several specific algorithms in Section 2.4. The experiment results and discussion are given in Section 2.5. Finally, we conclude this chapter in Section 2.6.

2.2 Related work

According to the focus of this chapter, we review related work from two aspects, i.e. types of scheduling algorithms and the assumption of system model, to reflect the relationship between this chapter and existing research.

2.2.1 Reviews of Scheduling Algorithms

To optimize the utilization of resources, energy consumption, response time, makespan, etc, multi-phase method [6,9], virtual machine migration [10,63], queuing model [1], joint optimization [95] and resource scheduling algorithm are frequent strategies used in Cloud computing, where scheduling algorithm is a critical component attracting scholars. Current scheduling algorithms in Cloud computing include local search algorithm, heuristic algorithm, meta-heuristic algorithm, randomization, machine learning algorithm and hybrid algorithm.

Machine Learning in scheduling algorithms mainly contains three types that deep

learning (DL) such as DREP [104] and DLSC [44], reinforcement leaning (RL) such as QEEC [49], unified reinforcement learning (URL) [105], adaptive reinforcement learning (ARL) [106] and ADEC [53], as well as Deep reinforcement learning (DRL) such as DRM_Plus [2], A3C RL [107], MDRL [108], DPM [109], DQTS [110] and DQN [100].

A local search algorithm is to select the neighbor solution according to a strategy by comparing the current solution with the neighbor solution, where neighborhood structure and neighborhood selection (search route) are the basic components. In Cloud computing, some local search algorithms, including Neighborhood Search (NS) [111], Correlation-Aware Heuristic Search (CAHS) [112], IBGSS [113], Crow Search [114], Dynamic Grouping Integrated Neighboring Search (DGINS) [115] and Tabu Search [116], are applied to solve the problems of resource management.

The meta-heuristic algorithm is a combination of local search algorithm and randomization containing Ant Colony Optimization (ACO), Genetic Algorithm (GA), FireFly, Particle Swarm Optimization (PSO), etc [98].

ACO imitates ant colony to search for food as a search route. Liu et al. [58] proposed OEMACS combining OEM local search techniques and ACO to resolve VMs deployment in Cloud computing, which reduced the energy consumption and improved the effectiveness of different resources compared with conventional heuristic and other evolutionary-based approaches. Chakravarthy et al. [63] proposed two ant colony-based algorithms (TACO) to address VM scheduling and routing in multi-tenant Cloud data centers aiming at improving the utilization of energy in Cloud computing.

GA imitates the process of natural evolution as the search route of the local search algorithm. Reference [66, 99] improve the search strategy based on NSGA-II to reduce the energy consumption, response time, load imbalance and makespan in Cloud computing. Xu et al. [69] applied NSGA-III to optimize the execution time and energy consumption of IoT-enabled Cloud-edge computing. MOGA proposed by H. Jiang et al. [70] and MOEAs proposed by P. Cong et al. [9] improved the search route strategies based on NSGA-II and were utilized to settle resource scheduling in Cloud.

FireFly including FA [46] and FIMPSOA [55], PSO including MOPSO [71], APDPSO [117], and TSPSO [72], are other meta-heuristic algorithms applied in scheduling of Cloud.

For each instance I of a scheduling problem, assuming the solution of an algorithm Al as $Al(I)$ and the theory optimal solution as $OPT(I)$, if $\exists\tau$ (a function) for $\forall I$ **s.t.** $Al(I) \leq$

$\tau(|I|) \cdot OPT(I)$ (or $Al(I) \geq \tau(|I|) \cdot OPT(I)$) and the running time of Al is bounded by a fixed polynomial in $|I|$, then Al is defined as an approximation algorithm with approximation ratio τ [118]. The approximation ratio is a momentous index to evaluate the performance of an algorithm.

The heuristic algorithm is a type of algorithm to solve an optimization problem based on intuitionistic or empirical construction. Currently, some heuristic algorithms in the scheduling of Cloud computing have given approximation ratios and other types of algorithms can rarely obtain the theoretical approximation ratio generally. For the justification of the huge scale of tasks in Cloud computing, higher computational complexity and randomness of meta-heuristic and machine learning, as well as timeliness requirements for processing tasks, heuristic algorithms are still widely applied in practical Cloud.

Zhang et al. [119] adopted Lagrange Relaxation based Aggregated Cost (LARAC) to reduce the energy consumption of Mobile Cloud computing. Dynamic Bipartition-First-Fit (BFF), a $(1 + \frac{g-2}{k} - \frac{g-1}{k^2})$ competitive algorithm based on First-Fit algorithm, was proposed and its performance was proved theoretically by Tian et al. [57]. Hong et al. [61] proposed QoS-Aware Distributed Algorithm based on first-come-first-improve (FCFI) and all-come-then-improve (ACTI) algorithms to reduce computation time and energy consumption of Industrial IoT-Edge-Cloud Computing. Tian et al. [15] proposed Longest Loaded Interval First Algorithm (LLIF), a 2-approximation algorithm with theoretical proof, to minimize the energy consumption of VMs reservations in the Cloud. Other algorithms such as RR (Round-Robin) algorithm, greedy, BFD (Best Fit Decreasing), LPT (Longest Processing Time), and Jacobi Best-response Algorithm are frequent algorithms in realistic and have likewise become baselines in existing research [59].

The hybrid of two or more algorithms is also a strategy to improve the search route of the local search algorithm. Kumar et al. [120] proposed a hybrid algorithm called HGA-ACO combining GA and ACO to solve task allocation and ensure QoS parameters in the Cloud environment. Compared with the single search route of GA and ACO, HGA-ACO achieved better performance using the optimization solution of GA as the initial state of ACO [120]. Yang et al. [121] proposed a hybrid meta-heuristic called DAAGA combining GA and improved ACO by taking the solution of ACO as the seed of GA to address Cloud service composition and the optimization problem. NN-DNSGA-II combining NSGA-II with neural networks [122], PSO-ACS applying PSO to find the optimal solution of task scheduling and applying ACO to find the best migration path of VMs on PMs [83], and

FACO exploiting fuzzy module dedicated to pheromone evaluation [28], are also used in Cloud.

2.2.2 Review of System Model

In Cloud Computing, the scheduling scenarios can be divided into dynamic scheduling and static scheduling [4, 10].

Dynamic scheduling, usually applied in online scheduling, is generally regarded as more consistent with realistic Cloud computing where tasks usually vary with time and are unknown at the initial scheduling time. In the dynamic scheduling, the tasks are usually predicted with the random process [2, 49] or machine learning [109] based on historical data to support the subsequent scheduling using an optimization algorithm, which makes dynamic scheduling rely on two aspects that the prediction models and optimization algorithms. Yuanjun et al. [9] focused on the hybrid tasks in Cloud and proposed multi-phase integrated scheduling with six representative multi-objective evolutionary algorithms. In [9], the orders of tasks are independent without processing constraints among different orders, each task whose processing process is non-preemptive is conducted continuously without interruption, and the resource is free for another step immediately once a processing step is finished. Ding et al. [49] focused on dynamic task scheduling on VMs for energy-efficient cloud computing and proposed a framework QEEC based on Q-learning and M/M/S queueing system. In [49], the energy is modeled as related to tasks' running times, each task is treated as integral which cannot be further split into smaller tasks, a task is conducted by only one VM, and each VM can only execute one task at any time.

Due to the uncertainty of prediction, it's difficult to obtain the theoretical performance of the solution in dynamic scheduling, whose frequently used algorithms are metaheuristic algorithms [58, 98] and machine learning algorithms [2, 49, 109]. Additionally, after getting the prediction of tasks or the real-time status of resources, dynamic scheduling can be converted to static scheduling by using a static scheduling algorithm as the optimization algorithm of dynamic scheduling. Therefore, there is still a lot of research focusing on static scheduling to explore the theory of scheduling algorithms.

In static scheduling that is usually leveraged in reservation, tasks and resources as known before scheduling and the algorithms aim at finding a scheme to allocate tasks to resources. Tian et al. [123] focused on the load balance of VMs reservations in data centers and proposed "Prepartition" to prepare migration in advance and set process time bound

for each VM on a PM. In [123], the system model was modeled as all data are deterministic unless otherwise specified, time is formatted in slotted windows and there is no priority between VMs, which means the orders of VMs don't change their processing times. Zhang et al. [117] focused on the static load balancing in Cloud computing and proposed a novel adaptive Pbest discrete PSO (APDPSO). In [117], only the computation and bandwidth resources are considered, the network topology of hosts is deterministic, the unit data transmitting cost between each host pair is fairly unchanged, the constraints of resources and the required resources of tasks or VMs are known, and the amount of data to be transferred between federates is unchanged. Ghalami et al. [33] focused on the scheduling jobs on parallel identical machines to minimize makespan ($P||C_{max}$) and developed sequential approximation algorithms with various approximation guarantees. In modeling of [33], the processing times of jobs were available for processing at time zero, a job cannot be preempted once assigned to a processor for execution, and each machine cannot process more than one job at a time.

In this chapter, we continue some assumptions about Cloud computing system modeling referring to reviewed literature to formulate the static scheduling of minimizing makespan in homogeneous and heterogeneous resources, which makes the improvement of the theoretical performance of our proposed algorithm in the minimizing makespan problem to be useful for resource scheduling of Cloud computing. According to the review of types of scheduling algorithms, it can be seen that our proposed multi-routes search algorithms using heuristic algorithms as search routes are distinct from the previous algorithms. For multi-routes search algorithms with a similar principle to hybrid algorithms, different search routes can jump out the inherent local optimum of a single search route resulting in optimization of the solution. Application of heuristic algorithm as search route enables the theoretical proof of approximate ratio of the search algorithm, which has not been carried out for search algorithms in the previous studies.

2.3 Cloud Systems and Optimization Problems Formulations considering Single-dimensional Resources

In this section, we model the universal scheduling problem of Cloud computing and present objectives for minimizing makespan both for homogeneous and heterogeneous resources. And then, we establish the framework of a general local search algorithm based on the adjustment of tasks between two resources. The notations used in this chapter are

presented in Table 2-1, where the given parameters are known without the influence of scheme and scheme-related variables are the opposite.

Table 2-1 Notations and Descriptions

Notation	Description	Nature
i	Index of task	Object
j	Index of resource	
T_i	The task with index i	
R_j	The resource with index j	
$\mathbf{card}(S)$	The cardinal of set S	
N	Number of tasks	Given
M	Number of resources	
E_{ij}	General element of task T_i when executed in R_j	
ET_{ij}	The processing time of T_i when executed in R_j	
$MaxA_j$	General Upper limit of resource R_j	
TS_j	Set of tasks in resources R_j	Scheme-related
A_j	General aspect of resource R_j	
ω	General aspect of problem	
RT_j	The total processing time of resource R_j	
P_i	The parameter set of task T_i	
KS	The set of TS_j where $KS = \{TS_1, \ldots, TS_M\}$	

2.3.1 Models of Minimizing Makespan in Cloud Computing

The resource in the problems of this chapter refers to a physical machine or virtual machine that can process tasks with some specific component such as CPU, RAM, DS, etc, where task refers to a request from the user. In this chapter, we mainly focus on the static scheduling and leverage the processing time to express a task that $T_i = \{\langle ET_{ij} \rangle\} = \langle ET_{i1}, ET_{i2}, \ldots, ET_{iM} \rangle$. For minimizing makespan, the features of resources and tasks are as follows referring to existing research.

(1) The set of tasks $\{T_1, T_2, \ldots, T_N\}$ is deterministic [123];

(2) All tasks are independent and preemptive without precedence constraints for the order of tasks [123];

(3) Each task is treated as an integral task and cannot be further split into smaller tasks [49];

(4) Each task can be fully fulfilled by one and only one resource (usually virtual machine) [49];

(5) When $A_j \leq MaxA_j$, the processing capacity of the resource remains unchanged, which means the parameter P_i of each task is fixed and unaffected by the status of the resource.

(6) The total processing time of all resources starts from 0;

(7) Each resource (i.e. VM or PM) is either idle or processing only one task [33,49];

(8) Once a task is finished, the occupied resource is free for another task immediately ignoring switching time [9];

(9) The processing time ET_{ij} is fixed without affection of the resource's status or the execution order of tasks;

(10) And the number of available resources is invariant.

Assuming the mapping between general element of task and general aspect of resources as function $h : E \rightarrow A$, the mapping between general aspect of resources and optimization objective as function $f : A \rightarrow \omega$, and taking $P_i = \langle E_{i1}, E_{i2}, \ldots, E_{iM} \rangle$, the universal resource scheduling problem of Cloud computing can be written as

$$\min \omega = f(A_1, A_2, A_3, \ldots, A_M) \tag{2-1}$$

where $A_j = h_j\left(E_{ij}|_{T_i \in TS_j}\right)$ subject to $A_j \leq MaxA_j$. In Cloud computing, the general element of task can be taken as processing time, volume, energy consumption, bandwidth, storage request and other elements of tasks, in addition the general aspect of resource can be taken as degree of load balance [9,46], makespan [46,110], energy consumption [63,108], cost [59,71], delay ratio [9], utilization of resource [46,49], throughput [44,55], SLA Violation [53,71], etc.

If $\forall l \neq k$ **s.t.** $h_l(x) = h_k(x)$, $MaxA_l = MaxA_k$ and $E_{il} = E_{ik}$, the resources are homogeneous, otherwise the resources are heterogeneous. In this chapter, we consider homogeneous resources when studying the theoretical approximation ratio of the algorithms, while we also study the scheduling problems of heterogeneous resources to approach to real Cloud scenarios.

Solution of minimizing makespan is a way to reduce the working time of resources, where less working time may bring multifaceted benefits such as reduction of energy consumption, increase of resource utilization, prolong of devices' lifespan, improvement of processing capacity, etc. For problem of minimizing makespan, the decisive factors are time-related parameters including ET_{ij} and RT_j assuming time is slotted in the unit of time. Thus, we need to use ET_{ij} and RT_j to replace E_{ij} and A_j in problem then substitute the parameter set of tasks as $P_i = \langle ET_{i1}, ET_{i2}, \ldots, ET_{iM} \rangle$ where $ET_{ij} \in \mathbb{R}^+$ means the processing

time of task T_i when it on the resource R_j.

According to the features of tasks and resources, the total processing time of R_j equals to the sum of processing time of all tasks executed on R_j that

$$RT_j = \sum_{T_i \in TS_j} ET_{ij}. \tag{2-2}$$

Along with minimizing makespan, the sum of the running time of all resources is considerable simultaneously. Therefore, two objectives that are minimizing the total running time and minimizing the makespan require considerations shown as

$$\begin{cases} \min \omega_{total-time} = \min \left(\sum_{j=1}^{M} \sum_{i=1}^{N} x_{ij} ET_{ij} \right) \\ \min \omega_{makespan} = \min \left(\max_{j=1,2,\ldots,M} \left(\sum_{i=1}^{N} x_{ij} ET_{ij} \right) \right) \end{cases}. \tag{2-3}$$

For heterogeneous resources, Eq.2-3 is a bi-objective problem, and for homogeneous resources, total running time is a constant where Eq.2-3 can degenerate into a single objective optimization problem. The constraints of Eq.2-3 are subject to

$$\sum_{j=1}^{M} x_{ij} = 1, \forall i \in \{1, \ldots, N\}, \tag{2-4a}$$

$$x_{ij} \in \{0, 1\}, \forall i \in \{1, \ldots, N\}, \forall j \in \{1, \ldots, M\}, \tag{2-4b}$$

$$A_j \leq MaxA_j, \forall j \in \{1, \ldots, M\}, \tag{2-4c}$$

where Eq.2-4a and Eq.2-4b are the common constraints of zero-one integer programming, as well as Eq.2-4c is corresponding to the fifth features of system to ensure the parameters of tasks and resources unchanged.

Table 2-2 Problems Executed in Experiments.

Sign	Description of Problem
P_1	Minimizing makespan for homogenous resources
P_2	Minimizing makespan and total running time for heterogenous resources

In Eq.2-3, the optimization solution of $\{x_{ij}\}$ necessarily and sufficiently occupies a

unique set of KS, hence KS can express the solution corresponding to $\{x_{ij}\}$. When KS is determined, other scheme-related parameters in Table 2-1 can be calculated according to given parameters. Other scheduling problems can be formulated similar to these problems based on the universal mode of Eq.2-1. Then, the formulated problems considered in this chapter are in Table 2-2.

2.4 Algorithm Design: Multi-Route Search Algorithm

2.4.1 General Local Search Algorithm

Although the main problem studied in this chapter is minimizing makespan, our proposed algorithms have generality for other optimization problems, such as load balancing, minimizing energy consumption, etc. Therefore, we still use the general parameters E_{ij} and A_j to present subsequent methodologies and theoretical proofs. When solving the makespan-related problems, they only need to be replaced by ET_{ij} and RT_j.

For scheduling problems, the local search algorithm is a considerable method, whose strategy is to search the optimal state by local neighborhoods [111, 112]. In this chapter, neighbors of dual resources is described as only two resources have different sets of tasks in two scheduling solutions, which can be defined by mathematical formulas as: assuming two solutions $KS_1 = \{TS_1, TS_2, \ldots, TS_M\}$ and $KS_2 = \{TS'_1, TS'_2, \ldots, TS'_M\}$, if $\exists j_1 \neq j_2 \in \{1, 2, \ldots, M\}$ **s.t.** $TS_{j_1} \neq TS'_{j_1}$, $TS_{j_2} \neq TS'_{j_2}$ and $TS_k = TS'_k$ for $\forall k \in \{1, 2, \ldots, M\} - \{j_1, j_2\}$, then KS_1 and KS_2 are neighbors of dual resources.

General Local Search Algorithm based on the neighbors of dual resources for the universal scheduling problem, utilizes a specified route to search for a better state in the neighbors of the current state until non-existent neighbors are better than the current state as Algorithm 2-1. General Local Search Algorithm based on the Neighbors of Dual Resources has four essential fundamentals as follows which vary with scenarios:

(1) The specified local search route which be substituted by various route strategies;
(2) The neighbor selection criteria consistent with $f: A \rightarrow \omega$;
(3) The initial state with initialization policy that determines which local optimal cluster to search in;
(4) The specific order strategies.

In these four essential fundamentals, the specified local search route directly influences the performance of the solution and is also a widely studied topic. This chapter

Algorithm 2-1 General Local Search Algorithm based on the Neighbors of Dual Resources

Input : $\{T_1, T_2, \ldots, T_N\}$ and $\{R_1, R_2, \ldots, R_M\}$
Output: solution $KS = \{TS_1, TS_2, \ldots, TS_M\}$

Initially Allocate tasks to resources with **confirmed or random initialization policy** and gain the general aspect of resources by $A_j = h_j\left(E_{ij}|_{T_i \in TS_j}\right)$
while Exist_adjustment **do**
    Exist_adjustment = False
    Sort resources by their value A_j
    for j_1 *in resources of a* ***specific order strategies*** **do**
        $or\_j = j_1$, $\beta = f(A_1, A_2, \ldots, A_M)$
        for j_2 *in* $[0, j_1)$ **do**
            Use **specified local search route** (one or more of LPTS, BFDS, K-Step etc.) to adjust tasks belonging to TS_{j_1} and TS_{j_2} then gain TS'_{j_1} and TS'_{j_2} subject to $A_j \leq MaxA_j$ for $\forall j \in \{1, 2, \ldots, M\}$
            if $\beta > f\left(A_1, \ldots, A'_{j_2}, \ldots, A'_{j_1}, \ldots, A_M\right)$ **then**
                $pre\_j = j_2$ and Exist_adjustment = True
                $\beta = f\left(A_1, \ldots, A'_{j_2}, \ldots, A'_{j_1}, \ldots, A_M\right)$
        if Exist_adjustment **then**
            Update resources of or_j and pre_j based on the **specified local search route** as $TS_{or\_j} = TS'_{or\_j}$, $TS_{pre\_j} = TS'_{pre\_j}$
            Break (for repeat of j_1)

focuses on this to propose multi-search-routes-based algorithms.

2.4.2 Specified basic Local Search Route

It can be seen from the flow of Algorithm 2-1 that any algorithm able to readjust the tasks of two resources can be applied as its **specified local search route** and the property of its convergence solution is affected by this route. This property of Algorithm 2-1 not only allows a heuristic algorithm to be a search route but also allows multiple search routes to be used simultaneously to jump out of the convergence points of a single search route. Based on the definition of different neighborhoods, we will present several specified local search routes as basic routes including *K*-Step search, LPT search and BFD search. Substituting these algorithms into Algorithm 2-1 as **specified local search route**, we can get various local search algorithms.

2.4.2.1 *K*-Step Search Route

On the basis of neighborhood of dual resources, neighbors of *K*-Step can be defined as: assuming KS and KS' of two states of solutions satisfy the neighborhood of dual resources, if $\mathbf{card}\left(TS_{j_1} - TS_{j_1} \cap TS'_{j_1}\right) \leq K$ and $\mathbf{card}\left(TS_{j_2} - TS_{j_2} \cap TS'_{j_2}\right) \leq K$, then KS and KS' are mutually neighbors of *K*-Step. The *K*-Step search route based on neighbors of *K*-Step is Algorithm 2-2.

Algorithm 2-2 *K*-step search route

Input : tasks set TS_{j_1} and TS_{j_2} of R_{j_1} and R_{j_2}
Output: TS'_{j_1} and TS'_{j_2}
1 Find tasks set $B_{j_1} \subset TS_{j_1}$ and tasks set $B_{j_2} \subset TS_{j_2}$ **s.t.** $f\left(A'_{j_1}, A'_{j_2}\right) < f(A_{j_1}, A_{j_2})$
where $A'_j = h_j\left(E_{ij}|_{T_i \in TS'_j}\right)$, $0 < \max\left(\mathbf{card}(B_{j_1}), \mathbf{card}(B_{j_2})\right) \leq K$,
$TS'_{j_1} = TS_{j_1} - B_{j_1} + B_{j_2}$ and $TS'_{j_2} = TS_{j_2} + B_{j_1} - B_{j_2}$

With the adaptability of objectives, *K*-Step search can improve the current state of the solution. When $K = 1$, we can obtain One-Step search, which will be applied in our multi-route search subsequently as a special case of *K*-Step search. Generally, the computational complexity of finding the tasks sets B_{j_1} and B_{j_2} in One-Step search is $(card(TS_{j_1}) + card(TS_{j_2}) + card(TS_{j_1})card(TS_{j_2}))$. In our proposed One-Step search, we use a sort-based algorithm to optimize the process of finding task sets, which can reduce the computational complexity of finding the tasks sets to $((card(TS_{j_1}) + card(TS_{j_2}))\log(card(TS_{j_1}) + card(TS_{j_2})))$. And then, we obtain the algorithm of One-Step search route based on sort algorithm as Algorithm 2-3.

Substitution of *K*-Step search route to the **specified local search route** of Algorithm 2-1 (i.e. General Local Search Algorithm) can obtain *K*-Step Search Algorithm. This chapter mainly applies One-Step to improve other search routes such as LPTS and BFDS presented subsequently.

For the sake of subsequently demonstrating proofs of approximate ratio, we can present the property of *K*-Step in the Property 1 according to the definition of *K*-Step neighbors, which also applies to One-Step when $K = 1$.

Property 1 (*K*-Step) Assume $KS = \{TS_1, \ldots, TS_M\}$ is the convergence solution of *K*-Step Search Algorithm. For $\forall j_1 \neq j_2$, $\forall B_\alpha \subset T_{j_1}$, $\forall B_\beta \subset T_{j_2}$, if $0 < \max\left(\boldsymbol{card}(B_\alpha), \boldsymbol{card}(B_\beta)\right) \leq K$, then: $f\left(A'_{j_1}, A'_{j_2}\right) \geq f(A_{j_1}, A_{j_2})$ where $A'_j = h_j\left(E_{ij}|_{T_i \in TS'_j}\right)$,

Algorithm 2-3 One-step search route

Input : tasks set $TS = TS_{j_1} \cup TS_{j_2}$ of R_{j_1} and R_{j_2} where it can be set as $A_{j_1} \geq A_{j_2}$
Output: TS'_{j_1} and TS'_{j_2}

Set$\gamma_{j_1} = \frac{\sum_{T_i \in TS} E_{ij_1}}{M}$, $C_{j_1} = A_{j_1} - \gamma_{j_1}$
$B_1 = \arg\min_{\{T_i\}} \left(|E_{ij_1} - C_{j_1}|_{T_i \in TS_{j_1}}, |E_{ij_2} + C_{j_1}|_{T_i \in TS_{j_2}} \right)$
$vl_1 = \min \left(|E_{ij_1} - C_{j_1}|_{T_i \in TS_{j_1}}, |E_{ij_2} + C_{j_1}|_{T_i \in TS_{j_2}} \right)$
Sort $\left\{ E_{ij_1}|_{T_i \in TS_{j_1}}, E_{ij_2} + C_{j_1}|_{T_i \in TS_{j_2}} \right\} \rightarrow \{G_{\eta_1}, G_{\eta_2}, \dots\}$
$B_2 = \arg\min_{\{T_{\eta_i}, T_{\eta_i+1}\}} \left| G_{\eta_i} - G_{\eta_{i+1}} \right|$, $vl_2 = \min \left| G_{\eta_i} - G_{\eta_{i+1}} \right|$ where $\{T_{\eta_i}, T_{\eta_{i+1}}\} \not\subset TS_{j_1}$ and $\{T_{\eta_i}, T_{\eta_{i+1}}\} \cap TS_{j_1} \neq \emptyset$
if $B_1 = \emptyset$ *and* $B_2 = \emptyset$ **then**
    Return Exist_adjustment = False
else
    if $vl_1 \leq vl_2$ **then**
        Update by $TS'_{j_1} = TS_{j_1} - B_1$, $TS'_{j_2} = TS_{j_2} - B_1$
    else
        Update by $TS'_{j_1} = TS_{j_1} \cup B_2 - TS_{j_1} \cap B_2$, $TS'_{j_2} = TS_{j_2} \cup B_2 - TS_{j_2} \cap B_2$
    Return TS'_{j_1}, TS'_{j_2} and Exist_adjustment = True

$TS'_{j_1} = TS_{j_1} - B_\alpha + B_\beta$ and $TS'_{j_2} = TS_{j_2} + B_\alpha - B_\beta$.

2.4.2.2 LPT Search Route and Modified LPT Search Route

LPT (Longest Processing Time) algorithm is proposed to solve minimizing makespan of parallel machines $(P||C_{max})$ [33]. Currently, LPT algorithm has approximation ratio $Ar_{LPT} \leq \frac{4}{3} - \frac{1}{3M}$ where $M \geq 2$ and LPT-REV has approximate ratio $Ar_{LPT-REV} \leq \frac{4}{3} - \frac{1}{3(M-1)}$ where $M \geq 3$ in $P||C_{max}$ [102, 103]. Considering the merit of LPT, this chapter applies LPT as the search route shown as Algorithm 2-4 by adjusting the classic LPT algorithm so that it can be applied to Algorithm 2-1 as the search route. Based on the characteristics of LPT algorithm, neighborhoods of LPT-route can be defined as: assuming KS' and KS are the neighbors of dual resources and $TS'_{j_1} \cup TS'_{j_2} = TS_{j_1} \cup TS_{j_2} = \{T_{\alpha_1}, T_{\alpha_2}, \dots\}$ where $E_{\alpha_i} \geq E_{\alpha_{i+1}}$, if $T_{\alpha_i} \in \arg\min_{TS'} \left(\sum_{T_{\alpha_k} \in TS'_1} E_{\alpha_k}, \sum_{T_{\alpha_k} \in TS'_2} E_{\alpha_k} \right)$ where $k < i$ for $\forall T_{\alpha_i} \in TS'_{j_1} \cup TS'_{j_2}$ then KS' is an LPT route-based neighbor of KS, while by contraries KS may not be that of KS'.

Substitution of LPT search route to the **specified local search route** of Algorithm 2-1 can obtain LPT Search Algorithm (LPTS) that can adapt to various objectives corre-

Algorithm 2-4 LPT search route (based on LPT)

Input : tasks set $TS = TS_{j_1} \cup TS_{j_2}$ of R_{j_1} and R_{j_2}
Output: TS'_{j_1} and TS'_{j_2}

```
Mark_{j1} = 0, Mark_{j2} = 0, TS'_{j1} = ∅ and TS'_{j2} = ∅
for T_α in TS from largest to smallest do
    if Mark_{j1} ≤ Mark_{j2} then
        Mark_{j1} += E_{αj1} and TS'_{j1} += {T_α}
    else
        Mark_{j2} += E_{αj2} and TS'_{j2} += {T_α}
```

sponding to balance. LPTS inherits the approximation ratio $Ar_{LPTS} \leq \frac{4}{3} - \frac{1}{3M}$ for $P||C_{max}$ of LPT algorithm, and has better solutions in statistics, which means the solution of LPTS is not inferior to that of LPT algorithm.

Similarly for the demonstration of subsequent proofs, we present a property of LPTS as Property 2 according to the definition of LPT route-based neighbor.

Property 2 (LPTS) $KS = \{TS_1, \ldots, TS_M\}$ is the convergence solution of LPTS. For $\forall j_1 \neq j_2$, assuming $TS_{j_1} \cup TS_{j_2} = \{T_{\alpha_1}, T_{\alpha_2}, \ldots\}$ where $E_{\alpha_i j_1} \geq E_{\alpha_{i+1} j_1}$, if $TS'_{j_1} \cup TS'_{j_2} = TS_{j_1} \cup TS_{j_2}$ and $T_{\alpha_i} \in \arg\min_{TS'} \left(\sum_{T_{\alpha_k} \in TS'_{j_1}} E_{\alpha_k j_1}, \sum_{T_{\alpha_k} \in TS'_{j_2}} E_{\alpha_k j_2} \right)_{k<i}$ for $\forall T_{\alpha_i} \in \{T_{\alpha_1}, T_{\alpha_2}, \ldots\}$, then: $f\left(A'_{j_1}, A'_{j_2}\right) \geq f(A_{j_1}, A_{j_2})$.

However, the LPT algorithm is originally intended to resolve the problems in homogenous resources. For heterogeneous resources, a Modified LPT search (MLPT) algorithm seen in Algorithm 2-5 can be applied to address the problem of minimizing makespan. The Modified LPT search route considers the difference in the processing time or volume of a task between different resources and preferentially puts the task with the largest difference into a specific resource. In this way with adaptability for heterogeneous resources, the local search algorithm using MLPT route can obtain an optimized solution.

Algorithm 2-5 is a version convenient for comprehension. In realistic program of algorithm, we can use array operations on GPU to accelerate Algorithm 2-5. For the tasks set $TS = TS_{j_1} \cup TS_{j_2}$ of two resources R_{j_1} and R_{j_2}, we can assume the set sorted in ascending order according to the value of $E_{ij_1} - E_{ij_2}$ in TS as $\{T_{\alpha_1}, T_{\alpha_2}, \ldots, T_{\alpha_n}\}$. Then, the operation of Algorithm 2-5 equals to finding an index ζ in $\{0, 1, 2, \ldots, n\}$ **s.t.** $\sum_{i=0}^{\zeta} E_{\alpha_i j_1}$ and $\sum_{i=\zeta+1}^{n+1} E_{\alpha_i j_2}$ as close as possible assuming $E_{\alpha_0 j} = E_{\alpha_{n+1} j} = 0$ where $j \in \{j_1, j_2\}$. And $\sum_{i=0}^{\zeta} E_{\alpha_i j_1} \approx \sum_{i=\zeta+1}^{n+1} E_{\alpha_i j_2}$ is equivalent to $\sum_{i=0}^{\zeta} E_{\alpha_i j_1} + \sum_{i=0}^{\zeta} E_{\alpha_i j_2} \approx$

Algorithm 2-5 Modified LPT search route for heterogenous resources (Modification of LPT route)

Input : tasks set $TS = TS_{j_1} \cup TS_{j_2}$ of R_{j_1} and R_{j_2}
Output: TS'_{j_1} and TS'_{j_2}

$Mark_{j_1} = 0$, $Mark_{j_2} = 0$, $TS'_{j_1} = \emptyset$ and $TS'_{j_2} = \emptyset$
while $TS \neq \emptyset$ **do**
  if $Mark_{j_1} \leq Mark_{j_2}$ **then**
    $c = j_1$, $b = j_2$
  else
    $c = j_2$, $b = j_1$
  Find $T_{\alpha_l} \in TS$ **s.t.** $T_{\alpha_l} = \arg\min_{T_i \in TS} (E_{ic} - E_{ib})$ to obtain a set of $\{T_{\alpha_1}, T_{\alpha_2}, \ldots, T_{\alpha_s}\}$
  if $s \geq 2$ **then**
    Choose T_α **s.t.** $T_\alpha = \arg\max_{1 \leq l \leq s} E_{\alpha_l c}$
  $Mark_c += E_{\alpha c}$, $TS'_c += \{T_\alpha\}$ and $TS -= \{T_\alpha\}$

$\sum_{i=\zeta+1}^{n+1} E_{\alpha_i j_2} + \sum_{i=0}^{\zeta} E_{\alpha_i j_2} \rightarrow \sum_{i=0}^{\zeta} (E_{\alpha_i j_1} + E_{\alpha_i j_2}) \approx \sum_{i=0}^{n+1} E_{\alpha_i j_2} = Sum_{j_2}$. Setting $S_{j_1} = \{E_{\alpha_0 j_1}, E_{\alpha_1 j_1}, \ldots, E_{\alpha_n j_1}\}$ and $S_{j_2} = \{E_{\alpha_0 j_2}, E_{\alpha_1 j_2}, \ldots, E_{\alpha_n j_2}\}$, therefore, we can use the GPU-based program to quickly calculate a new array as $|cusum(S_{j_1} + S_{j_2}) - Sum_{j_2}|$ where $cusum(S)$ means the cumulative sum of S, choose the index at its minimum as ζ, and update $TS'_{j_1} = \left\{T_{\alpha_1}, T_{\alpha_2}, \ldots, T_{\alpha_\zeta}\right\}$ and $TS'_{j_2} = TS - TS'_{j_1}$ where if $\zeta = 0$ then $TS'_{j_1} = \emptyset$. Using this idea, we can quickly adjust the tasks of two resources with GPU operation-based program, and even quickly adjust the tasks of multiple resources simultaneously. Similarly, other search routes can also be accelerated by GPU operation. As this is not the focus of this chapter, we do not expand the explanation.

2.4.2.3 BFD Search Route

BFD (Best Fit Decreasing) is usually used to solve the bin packing problem and also applies to problems related to minimizing makespan. A search route based on BFD is Algorithm 2-6 and the neighborhood using the BFD search route can be called BFD route-based neighborhood similar to the definition of LPT route-based neighborhood. Using BFD search route to replace the **specified local search route** in Algorithm 2-1 obtains BFD search (BFDS) algorithm. Similar to the relationship between LPTS and LPT, BFDS, with a better statistical performance, inherits the approximation ratio of BFD. A property of BFDS is Property 3 according to its definition.

Algorithm 2-6 BFD search route (based on BFD)

Input : tasks set $TS = TS_{j_1} \cup TS_{j_2}$ of R_{j_1} and R_{j_2}
Output: TS'_{j_1} and TS'_{j_2}

set $\gamma_j = \frac{\sum_{T_i \in TS} E_{ij}}{M}$ where $j \in \{j_1, j_2\}$
$Mark_{j_1} = 0$, $Mark_{j_2} = 0$, $TS'_{j_1} = \emptyset$, $TS'_{j_2} = \emptyset$
for T_α *in TS from largest to smallest* **do**
  if $\left|Mark_{j_1} + E_{ij_1} - \gamma_{j_1}\right| < \left|Mark_{j_2} + E_{ij_2} - \gamma_{j_2}\right|$ **then**
    $TS'_{j_1} += T_\alpha$, $Mark_{j_1} += E_{ij_1}$
  else
    $TS'_{j_2} += T_\alpha$, $Mark_{j_2} += E_{ij_2}$

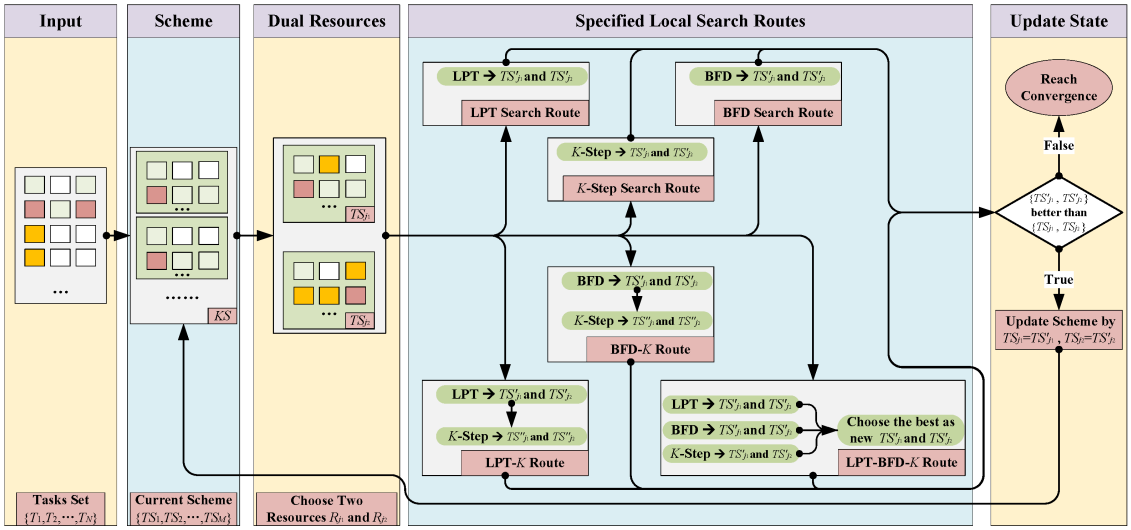


Figure 2-2 Flowchart of Local Search Algorithms based on the Neighbors of Dual Resources with Various Search Routes including LPT, BFD, *K*-Step and their Combinations.

Property 3 (BFDS) $KS = \{TS_1, \ldots, TS_M\}$ is the convergence solution of BFDS. For $\forall j_1 \neq j_2$, assuming $TS_{j_1} \cup TS_{j_2} = \{T_{\alpha_1}, T_{\alpha_2}, \ldots\}$ where $E_{\alpha_i j_1} \geq E_{\alpha_{i+1} j_1}$, if $TS'_{j_1} \cup TS'_{j_2} = TS_{j_1} \cup TS_{j_2}$ and $T_{\alpha_i} \in \arg\min_{TS'} \left(\left|Q_{ij_1} + E_{\alpha_i j_1} - \gamma_{j_1}\right|, \left|Q_{ij_2} + E_{\alpha_i j_2} - \gamma_{j_2}\right|\right)$ for $\forall T_{\alpha_i} \in TS'_{j_1} \cup TS'_{j_2}$ where $Q_{ij_1} = \sum_{T_{\alpha_k} \in TS'_{j_1}, k<i} E_{\alpha_k j_1}$ and $Q_{ij_2} = \sum_{T_{\alpha_k} \in TS'_{j_2}, k<i} E_{\alpha_k j_2}$, then: $f\left(A'_{j_1}, A'_{j_2}\right) \geq f(A_{j_1}, A_{j_2})$.

An improved strategy of the BFD search route, which is called Best-BFD search route in Algorithm 2-7, is to record schemes and select the best scheme as the final solution.

Algorithm 2-7 Best BFD search route

Input : tasks set $TS = TS_{j_1} \cup TS_{j_2}$ of R_{j_1} and R_{j_2}
Output: TS'_{j_1} and TS'_{j_2}

set $\gamma_{j_1} = \frac{\sum_{T_i \in TS} E_{ij_1}}{M}$, $k = 0$, $Mark_{j_1} = 0$, $TS^{(0)}_{j_1} = \emptyset$
for T_α *in TS from largest to smallest* **do**
  if $Mark_{j_1} + E_{ij_1} \leq \gamma_{j_1}$ **then**
    $TS^{(0)}_{j_1} += T_\alpha$ and $Mark_{j_1} += E_{ij_1}$
  else
    $k++$
    $L_k = \left| Mark_{j_1} + E_{ij_1} - \gamma_{j_1} \right|$ and $TS^{(k)}_{j_1} = TS^{(0)}_{j_1}$
$L_0 = \left| Mark_{j_1} - \gamma_{j_1} \right|$
$TS'_{j_1} = \arg\min_{TS^{(\alpha)}_{j_1}} (L_0, L_1, \ldots, L_k)$, $TS'_{j_2} = TS - TS'_{j_1}$

2.4.3 Combination of Multi-routes and the Flowchart

As mentioned above, the specified local search route of Algorithm 2-1 can be replaced by multiple search routes simultaneously. Therefore, the proposed search routes can be arbitrarily combined as the search routes of Algorithm 2-1. This chapter presents three combinations of them including LPT-One, BFD-One and LPT-BFD-One regarding LPT route, One-Step route and BFD route as the basic routes. These algorithms using various search routes can be uniformly called multi-search-routes-based algorithms and their operation processes corresponding to the specified search route in Algorithm 2-1 are as follows:

(1) **LPT-One** : $(TS_{j_1}, TS_{j_2}) \rightarrow \left(TS'_{j_1}, TS'_{j_2}\right)$ by LPTS and $\left(TS'_{j_1}, TS'_{j_2}\right) \rightarrow \left(TS''_{j_1}, TS''_{j_2}\right)$ by OneS; use (TS''_{j_1}, TS''_{j_2}) as the the neighbor of (TS_j, TS_k) in Algorithm 2-1.

(2) **BFD-One**: $(TS_{j_1}, TS_{j_2}) \rightarrow \left(TS'_{j_1}, TS'_{j_2}\right)$ by BFDS and $\left(TS'_{j_1}, TS'_{j_2}\right) \rightarrow \left(TS''_{j_1}, TS''_{j_2}\right)$ by OneS; use (TS''_{j_1}, TS''_{j_2}) as the neighbor of (TS_{j_1}, TS_{j_2}) in Algorithm 2-1.

(3) **LPT-BFD-One**: Use LPTS, BFDS and OneS, then choose the best solution as the neighbor of (TS_{j_1}, TS_{j_2}).

Based on the above description, we can draw the flow charts of our proposed algorithms as Fig 2-2 to intuitively demonstrate how the LPT, BFD, One-Step and their combinations act as search routes of Algorithm 2-1. From Fig 2-2, we can see again the flexibility of Algorithm 2-1 that it allows various algorithms as its search routes. In the framework, we only need to change the search routes to obtain specific search algorithms with different convergence properties.

As LPT and BFD have increased performance and One-Step has a descending performance with the increasing number of tasks, therefore LPT-OneStep search (LPT-One) and BFD-OneStep search (BFD-One) occupy better approximation ratios respectively than LPTS and BFDS. Multi-routes search has various combination patterns of search routes where one is the combination with repeated iterations of search routes such as LPT_route+OneStep_route, as well as the other is the combination with repeated iterations of search algorithms such as LPT_algorithm+OneStep_algorithm. Theoretically, the two patterns have the same theoretical approximation ratio. In this chapter, LPT-One and BFD-One appertain to that of search routes.

2.4.3.1 Example Demonstration of Algorithms Flows

For the sake of the presentation of multi-search-routes-based algorithms, we provide several examples to demonstrate the flows of algorithms.

Firstly for the problems of minimizing makespan for homogeneous resources, we select an instance with eight tasks $\{T_1, T_2, \ldots, T_8\}$ and three resources $\{R_1, R_2, R_3\}$ where the processing time of tasks can be set as $ET_1 = 2, ET_2 = 3, ET_3 = 4, ET_4 = 5, ET_5 = 6, ET_6 = 7, ET_7 = 8, ET_8 = 9$. Using LPT algorithm can obtain the solution as $TS_1 = \{T_8, T_3, T_2\}$, $TS_2 = \{T_7, T_4, T_1\}$ and $TS_3 = \{T_6, T_5\}$ where the maximum makespan is $RT_1 = 16$. Using LPT search algorithm cannot improve the solution, however using LPT-One can improve it. Taking the result of LPT algorithm as the initial state of LPT-One, the solution of scheduling is $TS_1 = \{T_6, T_3, T_2\}$, $TS_2 = \{T_7, T_4, T_1\}$ and $TS_3 = \{T_8, T_5\}$ where the maximum makespan is $RT_2 = RT_3 = 15$. When assuming $TS_1 = \{T_8, T_7, T_6, T_5, T_4, T_3, T_2, T_1\}$, $TS_2 = TS_3 = \emptyset$ as the initial state, the algorithm flow of LPT-One is as follows:

- 1-th iteration: Using LPT search route to adjust TS_1 and TS_2 can gain $TS_1 = \{T_8, T_5, T_4, T_1\}$ and $TS_2 = \{T_7, T_6, T_3, T_2\}$. Presently, OneStep search cannot adjust TS_1 and TS_2.
- 2-th iteration: Using LPT search route to adjust TS_1 and TS_3 can gain $TS_1 = \{T_8, T_1\}$ and $TS_3 = \{T_5, T_4\}$. Presently, OneStep search cannot adjust TS_1 and TS_3.
- 3-th iteration: Using LPT search route to adjust TS_1 and TS_2 can gain $TS_1 = \{T_8, T_3, T_2\} \rightarrow RT_1$ and $TS_2 = \{T_7, T_6, T_1\}$. Presently, OneStep search cannot adjust TS_1 and TS_2.
- 4-th iteration: Using LPT search route to adjust TS_2 and TS_3 can gain $TS_2 = \{T_7, T_4, T_1\}$ and $TS_3 = \{T_6, T_5\}$. Using OneStep to adjust present TS_2 and TS_3 can gain $TS_2 =$

$\{T_6, T_4, T_1\}$ and $TS_3 = \{T_7, T_5\}$.

- 5-th iteration: Using LPT search route to adjust TS_1 and TS_1 can gain $TS_1 = \{T_8, T_3, T_1\}$ and $TS_2 = \{T_6, T_4, T_2\}$.
- Currently, LPT-One achieves convergence, the scheme is $TS_1 = \{T_8, T_3, T_1\}$, $TS_2 = \{T_6, T_4, T_2\}$, $TS_3 = \{T_7, T_5\}$ and the maximum makespan is $R_1 = R_2 = 15$.

Secondly for the problem of minimizing makespan and minimizing total running time for heterogeneous resources, we demonstrate an instance with six tasks $\{T_1, T_2, \ldots, T_6\}$ and three resources $\{R_1, R_2, R_3\}$ where the processing time of tasks can be set as $ET_1 = \langle 3, 6, 7\rangle$, $ET_2 = \langle 8, 4, 6\rangle$, $ET_3 = \langle 9, 11, 28\rangle$, $ET_4 = \langle 7, 9, 7\rangle$, $ET_5 = \langle 8, 5, 9\rangle$ and $ET_6 = \langle 9, 4, 8\rangle$ where $ET_1 = \langle 3, 6, 7\rangle$ means the prossing time of task T_1 is 3, 6 or 7 when executed on R_1, R_2 or R_3 correspondingly. Although LPT is not suitable for this scenario, the strategy of it can be utilized to gain a solution as $TS_1 = \{T_3, T_6\}$, $TS_2 = \{T_4, T_1\}$ and $TS_3 = \{T_5, T_2\}$ where the maximum makespan is $RT_1 = 18$ and the total running time is $TR = 48$. A strategy of SPT (shortest processing time) can gain a solution as $TS_1 = \{T_1, T_5\}$, $TS_2 = \{T_2, T_6\}$ and $TS_3 = \{T_4, T_3\}$ where maximum makespan is $RT_3 = 35$. Taking $TS_1 = \{T_6, T_5, T_4, T_3, T_2, T_1\}$, $TS_2 = TS_3 = \emptyset$ as the initial state, we show algorithm flow of MLPT-One as follows.

- 1-th iteration: Using MLPT search route to adjust TS_1 and TS_2 can gain $TS_1 = \{T_1, T_3, T_4\}$ and $TS_2 = \{T_6, T_2, T_5\}$. Presently, OneStep search cannot adjust TS_1 and TS_2.
- 2-th iteration: Using MLPT search route to adjust TS_1 and TS_3 can gain $TS_1 = \{T_3\}$ and $TS_3 = \{T_4, T_1\}$. Using OneStep search route to adjust present TS_1 and TS_3 can gain $TS_1 = \{T_3, T_1\}$ and $TS_3 = \{T_4\}$.
- Currently, MLPT-One achieves convergence, the scheme is $TS_1 = \{T_3, T_1\}$, $TS_2 = \{T_6, T_2, T_5\}$, $TS_3 = \{T_4\}$ and the maximum makespan is $R_2 = 13$ and the total running time is $TR = 32$.

2.5 Theoretical Analysis and Proof

In consideration of the complexity and difficulty to prove the approximate ratio of heterogeneous resources, we only demonstrate the proofs of approximation ratio for homogeneous resources i.e. mainly focusing on $P||C_{max}$. Since the given general upper limits $MaxA_j$ of resources are the same in homogenous resources, which means the constraint is only related to the maximum aspects $\max\{A_1, A_2, \ldots, A_M\}$, so we don't need to consider

the constraint of Eq.2-3 in the proof.

As LPT-One and BFD-One are originated from the heuristic algorithms which have theoretical approximation ratios, hence their approximation ratios can be proved by referring to their original heuristic algorithms. Considering $E_{i1} = E_{i2} = \cdots = E_{iM}$ for $\forall i$ in homogeneous resources, we donate $G_i = E_{i1} = E_{i2} = \cdots = E_{iM}$ to represent the general elements of task T_i for the sake of demonstration of proofs. Next, we present the approximation ratios of several algorithms and their proofs.

Theorem 2.1 $Ar_{LPTO} = Ar_{LO} \leq \frac{5}{4} - \frac{1}{4M}$ for $P||C_{max}$.

Proof: According to the convergence condition, the convergence solution of LPT-One simultaneously obeys Property 1 and Property 2. Suppose the set of instances that do not satisfy Theorem 2.1 is H, the instance $I = \{T_1, T_2, \ldots, T_n\} \in H$ has minimum numbers of tasks, and $G_1 \geq G_2 \geq \cdots \geq G_n$, which means $Ar_{LO}(x|\forall x \in H) > \frac{5}{4} - \frac{1}{4M}$ and $\boldsymbol{card}(I) = \min \boldsymbol{card}(x|\forall x \in H)$.

Then, $T_n \in \arg\max_{TS_j}(A_j)$, otherwise $\exists I' = I - \{T_n\}$ **s.t.** $Ar_{LO}(I') > \frac{5}{4} - \frac{1}{4M}$ hence $I' \in H$ and $\boldsymbol{card}(I') < \boldsymbol{card}(I)$, which is contradicted to $\boldsymbol{card}(I) = \min \boldsymbol{card}(x|\forall x \in H)$.

Assuming $T_n \in TS_k$, when LPTS converges, $A_k - G_n \leq A_j$ for $\forall j \neq k$ from Property 2. Thus, $A_k \leq \frac{1}{M}\sum_{i=1}^{n} G_i + \frac{M-1}{M}G_n \leq OPT(I) + \frac{M-1}{M}G_n$ where $OPT(I)$ is the theoretical optimization.

As $Ar_{LO}(I) > \frac{5}{4} - \frac{1}{4M}$, so $\frac{5}{4} - \frac{1}{4M} < 1 + \frac{M-1}{M}\frac{G_n}{OPT(I)}$. $\therefore OPT(I) < 4E_n$. $\because G_n \leq G_i$ for $\forall i = 1, 2, \ldots, n$, $\therefore \boldsymbol{card}(TS'_j) \leq 3$ for $\forall j = 1, 2, \ldots, M$ where $KS' = \{TS'_1, TS'_2, \ldots, TS'_M\}$ is the theoretical optimization scheme corresponding to $OPT(I)$.

It can be assumed that $n = 3M$.

Assuming $\boldsymbol{card}(TS_j) = 3$ for $\forall j = 1, 2, \ldots, M$ of instance I, $TS_k \cup TS_j$ can be set as $\{T_{\alpha_1}, T_{\alpha_2}, T_{\alpha_3}, T_{\alpha_4}, T_{\alpha_5}, T_n\}$ where $\forall j \neq k$ and $G_{\alpha_1} \geq G_{\alpha_2} \geq \cdots \geq G_{\alpha_5} \geq G_n$. Then, there are two situations for TS_k and TS_j considering $T_n \in TS_k$ that as:

$$\begin{cases} T_{\alpha_1} \in TS_k \\ T_{\alpha_2}, T_{\alpha_3} \in TS_j \end{cases} \quad or \quad \begin{cases} T_{\alpha_1} \in TS_j \\ T_{\alpha_2}, T_{\alpha_3} \in TS_k \end{cases} \tag{2-5}$$

If $T_{\alpha_1} \in TS_k$, there are two states that $T_{\alpha_4} \in TS_k$ or $T_{\alpha_5} \in TS_k$. When $T_{\alpha_4} \in TS_k$, the relationships of $\{T_{\alpha_1}, T_{\alpha_2}, T_{\alpha_3}, T_{\alpha_4}, T_{\alpha_5}, T_n\}$ based on Property 2 are:

$$\begin{cases} G_{\alpha_1} + G_{\alpha_4} \leq G_{\alpha_2} + G_{\alpha_3} + G_{\alpha_5} \\ G_{\alpha_1} + G_{\alpha_4} + G_n \geq G_{\alpha_2} + G_{\alpha_3} + G_{\alpha_5} \\ G_{\alpha_1} + G_n \leq G_{\alpha_2} + G_{\alpha_3} \end{cases} \tag{2-6}$$

$\because I \in H, \therefore \exists j \neq k$ **s.t.**:

$$\begin{cases} \frac{G_{\alpha_1}+G_{\alpha_4}+G_n}{G_{\alpha_1}+G_{\alpha_2}} > \frac{9}{8} \\ G_{\alpha_1} + G_{\alpha_2} \geq \frac{\sum_{i=1}^{5} G_{\alpha_i}+G_n}{2} \geq OPT(I) \end{cases} \tag{2-7}$$

derived from Property 1. Substitution Eq.2-7 into Eq.2-6 obtains:

$$\frac{53}{24}G_{\alpha_1} + \frac{45}{24}G_{\alpha_2} < \frac{\sum_{i=1}^{5} G_{\alpha_i} + G_n}{2} \leq OPT(I) \tag{2-8}$$

Eq.2-8 contradicts $\frac{53}{24}G_{\alpha_1} + \frac{45}{24}G_{\alpha_2} > G_{\alpha_1} + G_{\alpha_2} \geq OPT(I)$.

When $T_{\alpha_5} \in TS_k$, the relationships are as follows based on Property 1 and Property 2.

$$\begin{cases} G_{\alpha_1} + G_{\alpha_5} \geq G_{\alpha_2} + G_{\alpha_3} + G_{\alpha_4} \\ G_{\alpha_5} + G_{\alpha_6} > \frac{1}{8}G_{\alpha_1} + \frac{9}{8}G_{\alpha_2} \\ G_{\alpha_1} + G_{\alpha_2} \geq \frac{\sum_{i=1}^{5} G_{\alpha_i}+G_n}{2} \geq OPT(I) \end{cases} \tag{2-9}$$

Simplification of Eq.2-9 can obtain $4OPT(I) > \frac{35}{8}G_{\alpha_1} + \frac{27}{8}G_{\alpha_2}$. However, $\because \frac{1}{8}G_{\alpha_1} + \frac{9}{8}G_{\alpha_2} < G_{\alpha_5} + G_{\alpha_6} \leq 2G_{\alpha_3}$ and $G_{\alpha_1} \geq G_{\alpha_2} + G_{\alpha_3}, \therefore 3G_{\alpha_1} > 5G_{\alpha_2}. \therefore \frac{35}{8}G_{\alpha_1} + \frac{27}{8}G_{\alpha_2} > 4\left(G_{\alpha_1} + G_{\alpha_2}\right) \geq 4OPT(I)$, which is constricted to $4OPT(I) > \frac{35}{8}G_{\alpha_1} + \frac{27}{8}G_{\alpha_2}$.

If $T_{\alpha_1} \in TS_j$, $\{T_{\alpha_1}, T_{\alpha_2}, T_{\alpha_3}, T_{\alpha_4}, T_{\alpha_5}, T_n\}$ satisfies that:

$$\begin{cases} G_{\alpha_1} + G_{\alpha_4} \leq G_{\alpha_2} + G_{\alpha_3} \\ G_{\alpha_1} + G_{\alpha_4} + G_{\alpha_5} \geq G_{\alpha_2} + G_{\alpha_3} \\ \frac{G_{\alpha_2}+G_{\alpha_3}+G_n}{G_{\alpha_1}+G_{\alpha_2}} > \frac{9}{8} \\ G_{\alpha_1} + G_{\alpha_2} \geq \frac{\sum_{i=1}^{5} G_{\alpha_i}+G_n}{2} \geq OPT(I) \end{cases} \tag{2-10}$$

Reduction of Eq.2-10 can also obtain Eq.2-8, which is constricted to $\frac{53}{24}G_{\alpha_1} + \frac{45}{24}G_{\alpha_2} > OPT(I)$.

Assuming $\exists \boldsymbol{card}(TS_j) \neq 3$, then $\exists \boldsymbol{card}(TS_j) = 2$ and $\exists \boldsymbol{card}(TS_l) \geq 4$. Thus, it can be suppose $\boldsymbol{card}(TS_l) = 4$ and $TS_l \cap TS_j = \{T_{\alpha_1}, T_{\alpha_2}, \ldots, T_{\alpha_6}\}$ where $G_{\alpha_1} \geq G_{\alpha_2} \geq \cdots \geq G_{\alpha_6}$. Four probable cases are $TS_j = \{T_{\alpha_1}, T_{\alpha_4}\}$, $TS_j = \{T_{\alpha_1}, T_{\alpha_5}\}$, $TS_j = \{T_{\alpha_1}, T_{\alpha_6}\}$ or

$TS_j = \{T_{\alpha_2}, T_{\alpha_3}\}$, where One-Step search can reach the optimum in either case.

Using the same derivation method combining with mathematical induction, it can be proved that $\nexists I$ **s.t.** $Ar_{LO}(I) > \frac{5}{4} - \frac{1}{4M}$, which means $H = \emptyset$. Therefore, Theorem 2.1 is true.

■

From proof of Theorem 2.1, it can be observed that One-Step search can improve and optimize the solution when $\mathbf{card}(TS'_j) \leq 3, \forall j = 1, 2, \ldots, M$. If the processing time of tasks are as Eq.2-11 with $3M$ tasks and M resources, then the allocation result of LPT algorithm is as Eq.2-12 where $\epsilon \in [0, \frac{M}{2})$ and the first row $TS_1 = \{3M-1, M, M-\epsilon\}$ means the resource R_1 has three tasks with processing time as $3M-1$, M and $M-\epsilon$.

$$\begin{cases} G_1 = 3M - 1; \\ G_2 = 2M - 1; \\ G_i = 2M - \left\lceil \frac{i}{2} \right\rceil, i = 2j+1, 1 \leq j \leq M-1; \\ G_{2M+1} = M; \\ G_i = 2M - \left\lceil \frac{i}{2} \right\rceil + \epsilon, i = 2j, 2 \leq j \leq M; \\ G_i = M - \epsilon, 2M+2 \leq i \leq 3M \end{cases} \tag{2-11}$$

$$\begin{cases} TS_1 = \{3M-1, M, M-\epsilon\}; \\ TS_2 = \{2M-1, M+0+\epsilon, M-\epsilon\}; \\ TS_3 = \{2M-2, M+1+\epsilon, M-\epsilon\}; \\ TS_4 = \{2M-3, M+2+\epsilon, M-\epsilon\}; \\ \vdots \\ TS_M = \{2M-(M-1), M+(M-2)+\epsilon, M\} \end{cases} \tag{2-12}$$

If using the allocation result of LPT as initial allocation or coincidentally using Eq.2-12 as the initial allocation, Eq.2-12 will be the local optimum of LPT-One, where the approximate ratio is $Ar_{LO}(I^*) = \frac{5M-1-\epsilon}{4M}$ and $\lim_{\epsilon \to 0} Ar_{LO}(I^*) = \frac{5M-1}{4M}$. Thus, the instance of Eq.2-11 with initial allocation as Eq.2-12 is a worst-case of LPT-One when the number of resources is M.

Theorem 2.2 $Ar_{BFDO} = Ar_{BO} \leq \frac{5}{4} - \frac{1}{4M}$ for $P||C_{max}$.

Proof: For BFD-One, assuming $Ar_{BO}(x|\forall x \in H) > \frac{5}{4} - \frac{1}{4M}$, $I \in H$ and $\boldsymbol{card}(I) = \min \boldsymbol{card}(x|\forall x \in H)$. Then, the same conclusion as that in LPT-One is $OPT(I) < 4G_n$ for

Table 2-3 Summary of Proposed Algorithms and their corresponding Problems evaluated in Subsequent Experiments.

Ascription	Algorithm	Category	Description	Problems
Single route	LPTS	Local Search	LPT-Search algorithm	$P_1 + P_2$
	MLPTS	Local Search	LPT-Search algorithm	P_2
	BFDS	Local Search	BFD-Search algorithm	P_1
	OneS	Local Search	OneStep-Search algorithm	P_1
	BestBFDS	Local Search	BestBFD-Search algorithm	P_1
Dual routes	LPT-One	Local Search	LPT-OneStep Search algorithm	$P_1 + P_2$
	MLPT-One	Local Search	MLPT-OneStep Search algorithm	P_2
	BFD-One	Local Search	BFD-OneStep Search algorithm	P_1
Triple routes	LPT-BFD-One	Local Search	Using the LPT, BFD and OneStep as search routes	P_1

Table 2-4 Comparison Algorithms evaluated in Experiments.

Algorithm	Category	Description	Problems
Random	Randomization	Randomly allocating tasks to resources	P_1
Greedy	Greedy	Scheduling with greed priory	P_1
RR	Heuristic	Round Robin algorithm	P_1
LPT	Heuristic	Longest Processing Time algorithm	P_1
BFD	Heuristic	Best Fit Decreasing algorithm	P_1
GA-Random	Meta-Heuristic	Genetic algorithm using random initialized state	P_1
PSO	Meta-Heuristic	Particle Swarm Optimization algorithm	$P_1 + P_2$
ACO	Meta-Heuristic	Ant Colony Optimization algorithm	P_2
GA-MinMin	Hybrid	Genetic algorithm using MinMin initialized state	$P_1 + P_2$
PSO-GA	Hybrid	Using the output of PSO as the input of GA	$P_1 + P_2$
ACO-GA	Hybrid	Using the output of ACO as the input of GA	P_2

BFDS based on Property 3. Analogous to LPT-One algorithm, it can be proved that $H = \emptyset$ based on Property 1 and Property 3 beneficial from combination of OneStep and BFDS. ■

From the similarities in proofs of Theorem 2.1 and Theorem 2.2, a novel theorem to improve the approximation ratio based on *K*-Step search can be gained as:

Theorem 2.3 $Ar_{LPTK}, Ar_{BFDK} \leq 1 + \frac{M-1}{(3+K)M}$ in $P||C_{max}$.

Proof: For LPT-*K*, similar to the proof of Theorem 2.1, assuming $Ar_{LPT-K}(x|\forall x \in H) > 1 + \frac{1}{3+K} - \frac{1}{(3+K)M}, I \in H$ and $\boldsymbol{card}(I) = \min \boldsymbol{card}(x|\forall x \in H)$. Then, a conclusion of I analogous to that in LPT-One search algorithm is $OPT(I) < (3+K)G_n$ according to Property 2. Therefore, $\boldsymbol{card}(TS'_j) \leq 2 + K, \forall j = 1, 2, \ldots, M$ where

$KS' = \{TS'_1, TS'_2, \ldots, TS'_M\}$ is the theoretical optimization scheme corresponding to $OPT(I)$. Based on the same principle of Property 1, K-Step search can optimize the solution when $\boldsymbol{card}(TS'_j) \leq 2 + K$. Therefore, $H = \emptyset$ from mathematical induction.

The proof process also works for BFD-K algorithm. ■

Theorem 2.3 reveals an avenue of to optimize the approximate ratio to infinitely approach to 1 by increasing K of K-Step search. For K-Step Search algorithm, the limit of the approximate ratio is 2 in theory which cannot be promoted with increasing K because of the existence of a counterexample as Eq.2-13, where $M\epsilon = \delta$.

$$\underbrace{\underbrace{\{\epsilon, \epsilon, \ldots, \epsilon\}}_{M+K}, \underbrace{\{\epsilon, \epsilon, \ldots, \epsilon\}}_{M+K}, \ldots, \underbrace{\{\epsilon, \epsilon, \ldots, \epsilon\}}_{M+K}}_{M-1}, \{\delta, \delta\} \tag{2-13}$$

For $\forall K$, $\exists M \gg K$ **s.t.** Eq.2-13 is a local optimum where K-Step search algorithm cannot adjust tasks of each resource. Presently in Eq.2-13, $Ar_{K-Step} \leq \frac{2M\delta}{(M+1)\delta+(M-1)K\epsilon} \rightarrow 2$. However, combination of LPTS (or BFDS) and K-Step has a dominant performance to surmount the limitation of counterexamples on each search route according to Theorem 2.3.

The computational complexity of LPT-One (Cc_{LO}) consists of two parts. One is the number of iterations which can be deduced as $\mathrm{O}(M)$ and the other is the complexity of each iteration which is about $\mathrm{O}(MN)$ for $P||C_{max}$. Therefore, the computational complexity of LPT-One is $Cc_{LO} = \mathrm{O}\left(M^2N\right)$. BFD-One also satisfies this property. We will also verify Cc_{LO} through experiments in the next section. As the approximation ratios of existing algorithms are as that $Ar_{LPT} \leq \frac{4}{3} - \frac{1}{3M}$, $Ar_{LPT-REV} \leq \frac{4}{3} - \frac{1}{3(M-1)}$ [102, 103] and $Ar_{Multifit} \leq \frac{72}{61} + 2^{-k}$ [33], the improvement and proof of approximate ratios of LPT-One and BFD-One still occupy theoretical significance, as well as the LPT-K and BFD-K, which provides a method to approach the upper bound 1.

2.6 Experimental Results and Analysis

2.6.1 Problems and Simulated Environment

Conduction of multi groups of experiments to the comprehensive evaluation of our proposed algorithms is essential. Therefore, we carried out experiments of problems shown in Table 2-2 for minimizing makespan under static scheduling for homogeneous and heterogeneous resources respectively.

For the problem of minimizing makespan in experiments, we simulate the Cloud

environment as that each resource (especially VMs) can process only one task simultaneously, tasks are independent mutually and each task only has one working procedure, where the total processing time of a resource equals to the sum of the processing time of the tasks on this resource.

In consideration of that verification of algorithms' performance especially statistical performance requires abundant instances, we establish the simulation environment through randomly generating tasks, which is conducive to producing adequate instances and observing the performance of different algorithms under the same instance. The parameters of tasks are generated by a given uniform distribution and their specified parameters of the generation will be described in each instance. As the variation in the number of tasks or resources has an impact on the performance of the algorithm, we fix the number of resources or tasks and observe the trend of algorithms' performance with the number of the other.

2.6.2 Compared Baselines and Evaluation Indexes

To assist the evaluation of the proposed algorithms for problems in Table 2-2, a variety of commonly used algorithms are regarded as baselines, including some random, greedy, heuristic, meta-heuristic and hybrid algorithms. The details of the proposed algorithms and the comparison algorithms in the experiments are in Table 2-3 and Table 2-4 respectively.

Assuming the set of algorithms participating in evaluation is $\{X_1, X_2, \ldots, X_p\}$, we conduct random experiments with 100 instances for each group of (M, N) donated as $\left\{I_1^{(M,N)}, I_2^{(M,N)}, \ldots, I_{100}^{(M,N)}\right\}$. Then, we donate the makespan obtained by algorithm X_k under the instance $I_l^{(M,N)}$ as $Y\left(X_k, I_l^{(M,N)}\right)$, the total running time as $Z\left(X_k, I_l^{(M,N)}\right)$, and the theoretical optimal makespan of $I_l^{(M,N)}$ as $OPT\left(I_l^{(M,N)}\right)$. Then, we can obtain several statistical indexes as Eq.2-14 where λ_1 is the average makespan, λ_2 is the ratio between the average makespan and the least makespan (abbreviated as AM/LAM), λ_3 is probabilities achieving the least makespan (PALM), λ_4 is probabilities achieving the theoretical optimization (PATO), λ_5 is maximum approximate ratio, λ_6 is average of total running time, λ_7 is the ratio between the average total running time and the least total running time (AT/LAT), and $\lambda_1^{X_k}\Big|_{(M,N)}$ means the index λ_1 of algorithm X_k in the scenario of (M, N).

$$\begin{cases} \lambda_1^{X_k}\Big|_{(M,N)} = \dfrac{\sum_{l=1}^{100} Y\left(X_k, I_l^{(M,N)}\right)}{100}; \lambda_2^{X_k}\Big|_{(M,N)} = \dfrac{\lambda_1^{X_k}\Big|_{(M,N)}}{\min\limits_{q=1}^{p} \lambda_1^{X_q}\Big|_{(M,N)}}; \\ \lambda_3^{X_k}\Big|_{(M,N)} = \dfrac{\sum\limits_{l=1}^{100}\left(Y\left(X_k, I_l^{(M,N)}\right) == \min\limits_{q=1}^{p} Y\left(X_q, I_l^{(M,N)}\right)\right)}{100}; \\ \lambda_4^{X_k}\Big|_{(M,N)} = \dfrac{\sum\limits_{l=1}^{100}\left(Y\left(X_k, I_l^{(M,N)}\right) == OPT\left(I_l^{(M,N)}\right)\right)}{100}; \\ \lambda_5^{X_k}\Big|_{(M,N)} = \max\limits_{l=1}^{100}\left(\dfrac{Y\left(X_k, I_l^{(M,N)}\right)}{OPT\left(I_l^{(M,N)}\right)}\right); \\ \lambda_6^{X_k}\Big|_{(M,N)} = \dfrac{\sum_{l=1}^{100} Z\left(X_k, I_l^{(M,N)}\right)}{100}; \lambda_7^{X_k}\Big|_{(M,N)} = \dfrac{\lambda_6^{X_k}\Big|_{(M,N)}}{\min\limits_{q=1}^{p} \lambda_6^{X_q}\Big|_{(M,N)}}. \end{cases} \tag{2-14}$$

Except for the above indexes, we will also apply the iterative process of makespan and Pareto scatter to comprehensively evaluate the performance of our proposed algorithms.

Then, simulation experiments are launched in a desktop computer with configurations as follows:

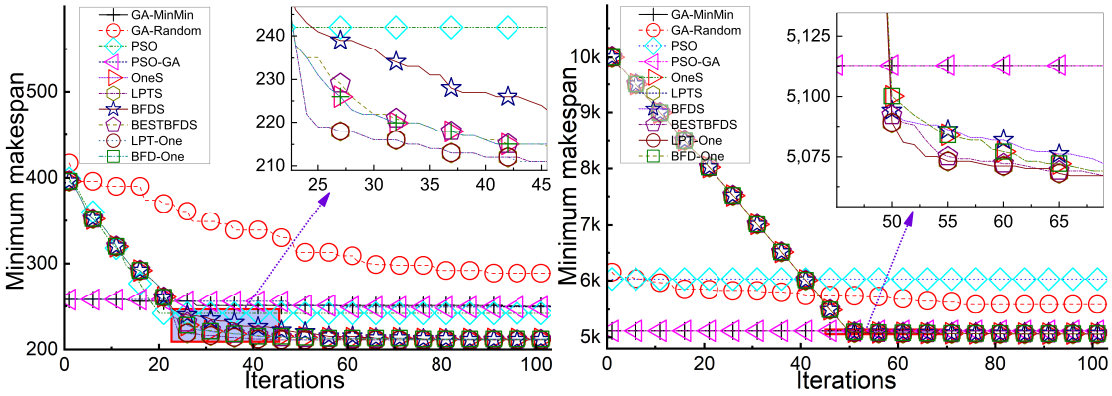
- CPU: Intel(R) Core(TM) i5-8400 CPU @ 2.8GHZ;
- SSD: KINGSTON SA400S37 240GB;
- GPU: NVIDIA GeForce GTX 1060 6GB;
- Program version: Python 3.6;

2.6.3 Result and Discussion

2.6.3.1 Minimizing Makespan for Homogeneous Resources

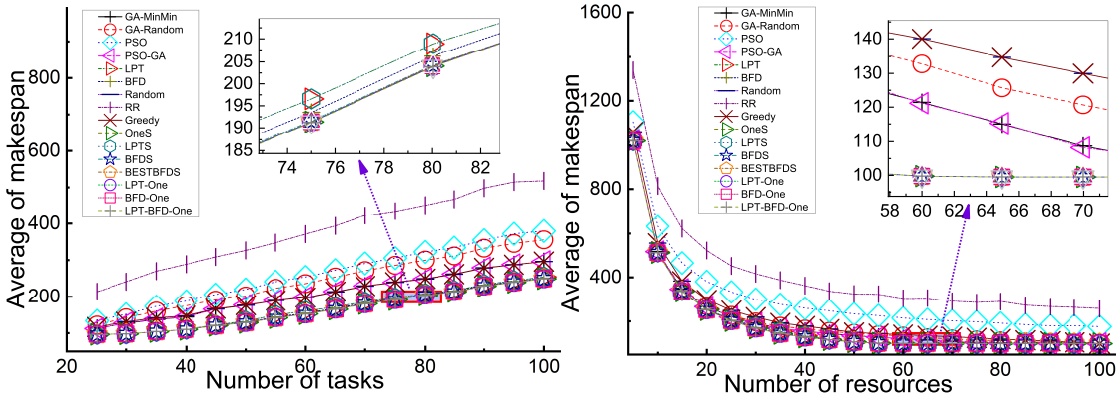
Firstly for minimizing makespan of homogenous resources ($P||C_{max}$), we carry out extensive experiments to observe the iterative processes. Since a large number of experiments can obtain similar conclusions, we only plot the iterative process of the two instances $(M,N) = (50,200)$ and $(M,N) = (100,10000)$ in Fig 2-3 for each algorithm with the property of searching solution in Table 2-4. In Fig 2-3, we choose the minimum makespan of all iterations before the current iteration as the value of the current. As iterations reaching convergence of the proposed algorithms are far less than 100, we replenish them to 100 iterations by their convergence values. Fig 2-3 shows that the proposed algorithms take about 25 iterations for instance of $(M,N) = (50,200)$ and 50 iterations

for instance of $(M, N) = (100, 10000)$ to reach an optimized state and their maskespans of converging are evidently smaller than compared algorithms, which points out that the proposed algorithms, including OneS, LPTS, BFDS, BestBFDS, LPT-One and BFD-One, can reach a better state close to convergence with iterations about the half number of resources, and reach converges by less than 100 iterations. In Fig 2-3, LPT-One possesses the fast convergence speed followed by LPTS and BFD-One.



(a) $M = 50, N = 200$ (b) $M = 100, N = 10000$

Figure 2-3 Iterative processes of makespan with 100 iterations for the problem of minimizing makespan for homogeneous resources.

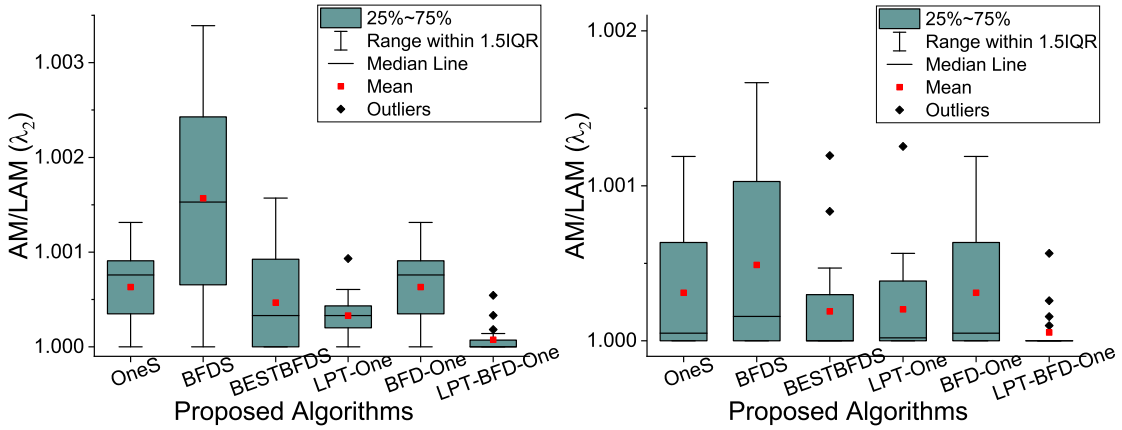


(a) $M = 20, N \in [25, 100]$ (b) $M \in [5, 100], N = 100$

Figure 2-4 The average makespans (λ_1) under each (M, N) with 100 instances respectively for problem of minimizing makespan for homogeneous resources.

To further investigate the statistical performance of proposed algorithms for $P||C_{max}$, we fix the number of resources or tasks, and randomly generate 100 instances for each combination (M, N) respectively where $ET_{ij} \in [1, 100]$ (unit of time). Similarly, we plot the average makespans (λ_1) of the 100 instances under each (M, N) for homogeneous re-

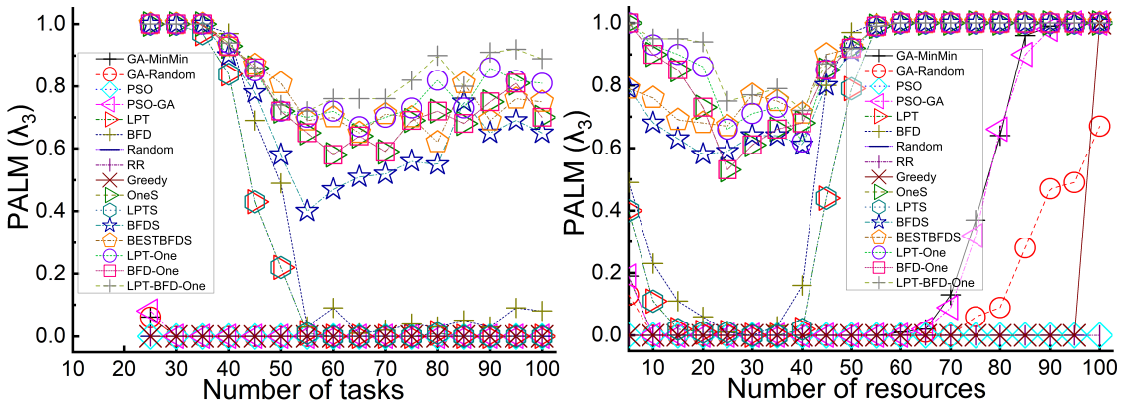
sources as Fig 2-4. For the sake of quantitative observation, we provide the numerical tables of Fig 2-4 in Table 2-6 and Table 2-7 of 2.8. As shown in Fig 2-4, our proposed OneS, BFDS, BestBFDS, LPT-One, BFD-One and LPT-BFD-One achieve the lowest and almost coincident makespans, followed by LPTS, LPT, BFD. The makespans of other algorithms are significantly higher than those of our proposed algorithms. The results of Fig 2-4 roughly shows that our proposed algorithms in Table 2-3 are better than the baselines in Table 2-4. Since the differences between our proposed OneS, BFDS, BestBFDS, LPT-One, BFD-One and LPT-BFD-One are far smaller than the whole ordinate span, it is difficult to distinguish which is better than others using Fig 2-4. Therefore, we calculate the ratio between the average makespan and the least average makespan (AM/LAM, λ_2) and plot the box chart of OneS, BFDS, BestBFDS, LPT-One, BFD-One and LPT-BFD-One in Fig 2-5. Its numerical tables are presented in Table 2-8 and Table 2-9 of 2.8. As Fig 2-5, LPT-BFD-One has the lowest AM/LAM followed by LPT-One, BestBFDS and BFD-One. This is because LPT-BFD-One uses three search routes and needs to meet the convergence conditions of them simultaneously, which makes the performance of LPT-BFD-One better than the dual routes algorithms and single routes algorithms. LPT-One and BFD-One outperform LPTS, BFDS and OneS for similar reasons. Additionally, LPTS is better than LPT and BFDS is better than BFD also demonstrate the search algorithm with heuristic algorithm as the search route is better than corresponding heuristic algorithm.



(a) $M = 20, N \in [25, 100]$ (b) $M \in [5, 100], N = 100$

Figure 2-5 The box chart of ratio between average makespan and the least average makespan (AM/LAM, λ_2) for our proposed algorithms corresponding to the experiments of Fig 2-4.

Fig 2-4 and Fig 2-5 verify the performance of our proposed algorithms from the



(a) $M = 20, N \in [25, 100]$ (b) $M \in [5, 100], N = 100$

Figure 2-6 The probabilities achieving the least makespan (PALM, λ_3) corresponding to the experiments of Fig 2-4.

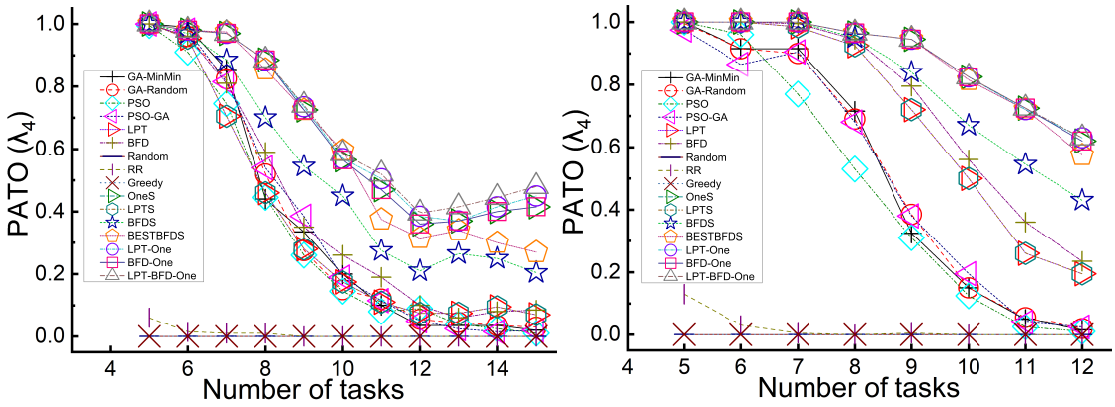
perspective of the average value of makespan. In addition to considering the average performance in practical applications, we usually consider the probability of an algorithm obtaining the best optimization solution. Therefore, we plot the probabilities achieving the least makespan (PALM, λ_3) under each (M, N) in Fig 2-6. Consistent with the conclusion form Fig 2-4 and Fig 2-5, LPT-BFD-One has the highest probabilities to obtain the least makespan in Fig 2-6 also followed by LPT-One and BestBFDS. On the whole, the probability of LPT-BFD-One achieving the least makespan remains above 70%, that of LPT-BFD-One remains above 65%, and BestBFDS above 60%.

This group of experiments not only demonstrate our proposed multi-search-routes-based algorithms outperform than baseline, but also demonstrate increasing the types of search routes can improve the optimization solution.

Table 2-5 The parameter ξ and evaluation index R^2 to fit the average computational complexities $Cc_{LO} \approx \xi N$ of LPT-One for $P||C_{max}$.

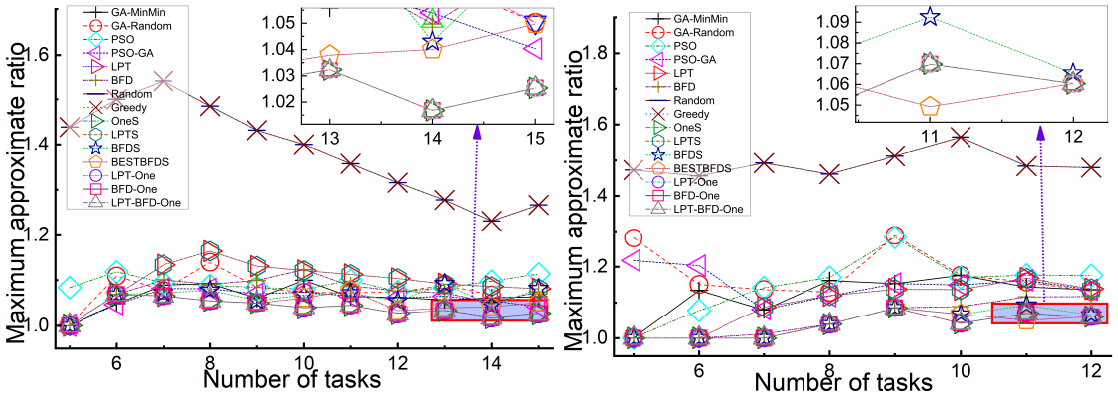
M	2	3	4	5	6	7	8	9	10	11	12
ξ	15.0	28.7	42.5	56.9	72.2	88.3	105.1	122.6	141.1	161.3	181.9
R^2	0.996	0.997	0.997	0.997	0.995	0.998	0.998	0.997	0.997	0.997	0.997
M	13	14	15	16	17	18	19	20	30	40	50
ξ	203.7	226.2	248.6	274.9	300.9	327.4	353.9	381.5	730.4	1193	1760
R^2	0.996	0.995	0.998	0.992	0.991	0.988	0.996	0.998	0.992	0.981	0.965

To further evaluate the performance of our proposed algorithms, we compare the solution of the proposed algorithms with the theoretical optimal solution obtained by the



(a) $M = 3, N \in [5, 15]$ (b) $M = 4, N \in [5, 12]$

Figure 2-7 The probabilities achieving the theoretical optimal makespan (PATO, λ_4) under each (M, N) with 100 instances respectively for problem of minimizing makespan for homogeneous resources.



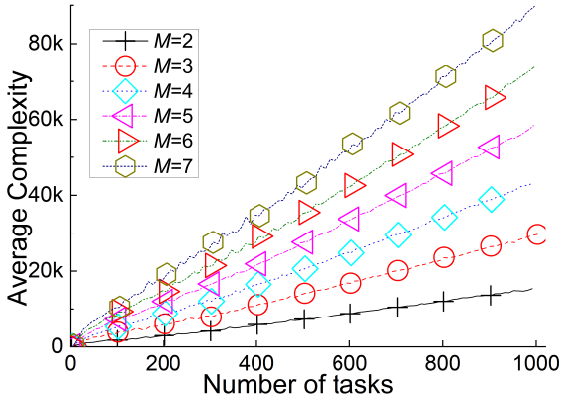
(a) $M = 3, N \in [5, 15]$ (b) $M = 4, N \in [5, 12]$

Figure 2-8 Maximum approximation ratios of makespan (λ_5) corresponding to the experiments of Fig 2-7.

enumerative algorithm. Considering the computation complexities of the enumerative algorithm are too large for instances with more resources, we only present the results of 3 resources and 4 resources. Then, we plot the probabilities achieving the theoretical optimal makespan (PATO, λ_4) in Fig 2-7, and plot the maximum approximation ratio of makespan (λ_5) in Fig 2-8.

From Fig 2-7, LPT-One, BFD-One and their combination LPT-BFD-One occupy higher probabilities to achieve the theoretical optimal makespan than other algorithms under all the combination of (M, N) in Fig 2-7. Concurrently, they keep the approximation ratio closest to 1 better than other algorithms from Fig 2-8. The results of Fig 2-7 and

Fig 2-8 verifies our proofs to some extent. From Fig 2-7(a), when the number of tasks is 4 times the number of resources, LPT-One, BFD-One and LPT-BFD-One obtain their lowest PATO about 40%, however the PATO of other algorithms such as GA, LPTS, PSO-GA etc are less than 10%, which shows that our proposed multi-routes algorithms have made significant improvement in the probability to achieve the theoretical optimum for NP-Hard problem. Fig 2-8 shows that the approximate ratios of our proposed LPT-One, BFD-One and LPT-BFD-One have been lower than 1.1 in experiments, which verifies the stability of these algorithms and can provides a reliable scheme for the task allocation or resource scheduling in realistic.



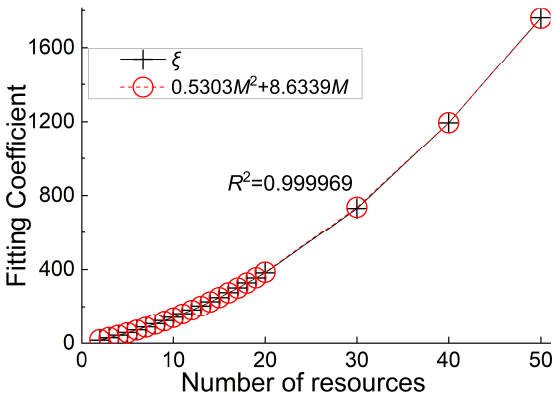


Figure 2-9 Average Complexities of LPTO for $P||C_{max}$Figure 2-10 The Relationship between ξ and M.

With above evaluation from several aspects, we continue to verify the calculational complexity. As our proposed multi-route algorithms are based on the general local search algorithm of Algorithm 2-1, which makes the complexity of these algorithms similar, so we choose to analyze the calculation complexity of LPT-One, whose complexity has been deduced as $Cc_{LO} = \mathrm{O}\left(M^2N\right)$. Similar to the indexes of λ_1 to λ_7, we record the calculational complexity of LPT-One in each $I_l^{(M,N)}$ and calculate its average complexities in 100 instances i.e. $\left\{I_1^{(M,N)}, I_2^{(M,N)}, \ldots, I_{100}^{(M,N)}\right\}$. Then, we plot the average complexities of LPT-One in Fig 2-9 under the scenarios of $(M \in \{2,3,\ldots,7\}, N \in [M,1000])$. From Fig 2-9, the complexity is approximately proportional to the number of tasks for each group of experiment. Thus, we assume the complexity $Cc_{LO} \approx \xi N$ and utilize linear regression to fit the complexities of more groups of experiments. The parameter ξ and the evaluating indexes *R-Square* are as Table 2-5. From Table 2-5, positive scale function $Cc_{LO} = \xi N$ can fit the computational complexity of LPTO well, whose evaluating indexes of R^2 are almost about 0.99. Furthermore, as the coefficient ξ increases monotonically with respect to the number of resources, we leverage quadratic polynomial regression to fit the

relation between ξ and M, and then gain the expression as $\xi \approx 0.5303M^2 + 8.6339M$ with evaluation index $R^2 = 0.999969$ shown in Fig 2-10. Fig 2-10 means it is reliable to use $0.5303M^2 + 8.6339M$ to fit ξ. Therefore, we can obtain the complexity $Cc_{LO} \approx (0.5303M^2 + 8.6339M)N$ which is identical to $Cc_{LO} = \mathrm{O}\left(M^2N\right) + \varphi(M,N)$. Specifically, the coefficient 0.5303 of M^2 is consistent with the appearance in Fig 2-3 that LPTO can achieve the convergence through iterations with about half number of resources. Using similar experimental process can derive similar conclusion for other proposed algorithms. This group of experiments shows that the average computational complexity of our proposed LPT-One belongs to quadratic polynomial.

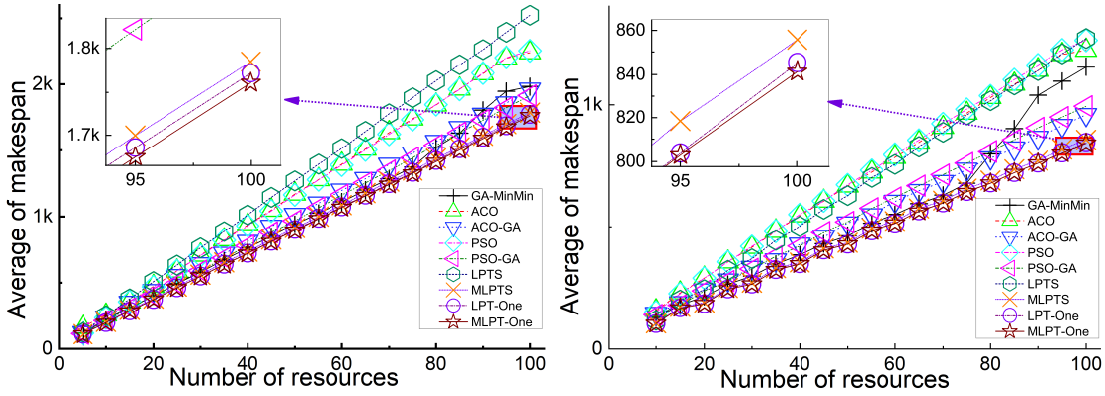
In summary, this section verifies our proposed algorithms perform well in homogeneous resources from several aspects: convergence, optimality and computational complexity.

2.6.3.2 Minimizing Makespan and Total Running Time for Heterogeneous Resources

The above experiments have demonstrated the advantages of the proposed multi-search-routes-based algorithms for minimizing the makespan of homogenous resources. Following, we execute experiments to observe that in heterogeneous resources. As LPT is designed to resolve $P||C_{max}$ for homogeneous resources, the LPT search cannot adapt to the problem of minimizing makespan for heterogeneous resources. However, LPT search can be modified by policy seen in Algorithm 2-5. Combining with OneStep Search, MLPT-One is also applied to solve problems of minimizing makespan and total running time for heterogeneous resources. In addition, the algorithms of Greedy, RR and Random don't have advantages to solve the problem of minimizing makespan according to the results of the above experiments. Thus, without losing representativeness, we only choose several meta-heuristic algorithms, i.e. GA, ACO, PSO and their combinations, as the baselines of the experiments for heterogeneous resources.

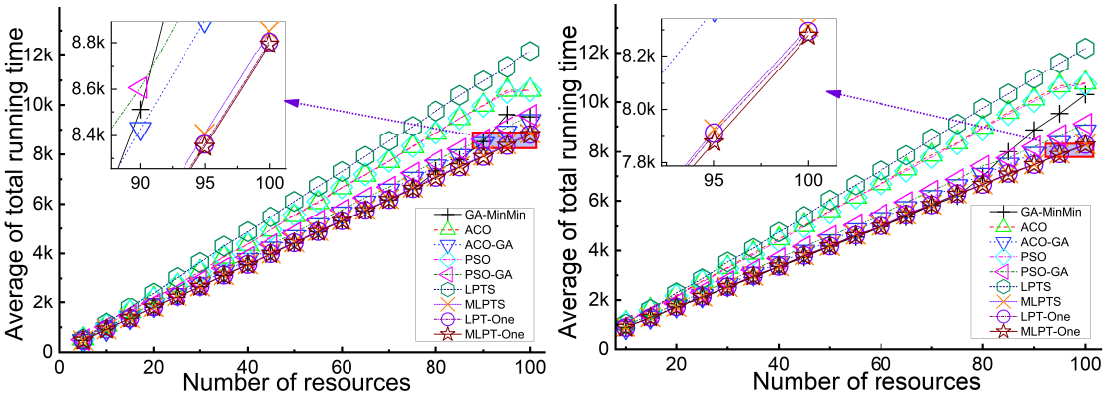
Similarly, extensive experiments can gain the same conclusion, so we only present two groups of experiments that $(M = 5, N \in [5, 100])$ and $(M = 10, N \in [10, 100])$, where each different combination of (M, N) also has 100 random instances generated by simulation systems and the processing time of each task on any resource is a random integer as $ET_{ij} \in [75, 150]$ (unit of time). Then, we plot the average of makespan (λ_1) in Fig 2-11 and the average of total running time (λ_5) in Fig 2-12 respectively. Table 2-10 and Table 2-11 of 2.8 provide the numerical value of Fig 2-11. As shown in Fig 2-11 and Fig 2-12,

MLPT-One obtains the lower average makespan and lower total running time than other algorithms followed by LPT-One and MLPTS.



(a) $M = 5, N \in [5, 100]$ (b) $M = 10, N \in [10, 100]$

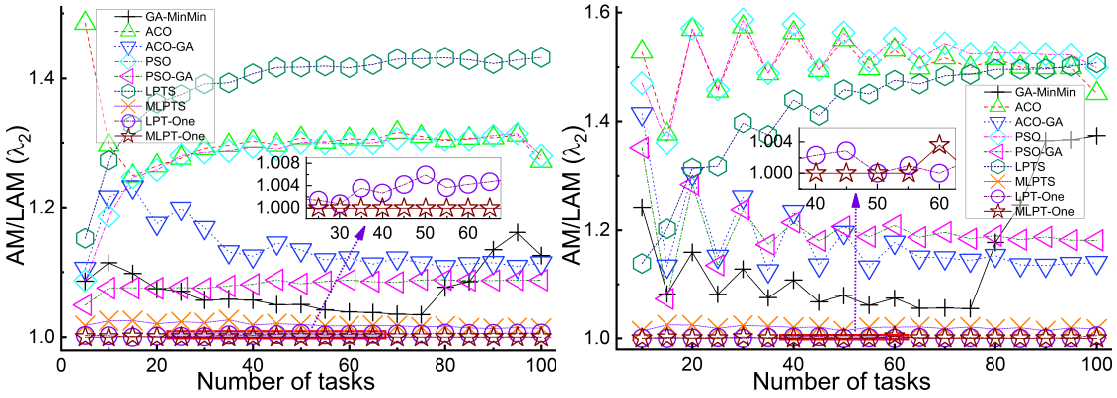
Figure 2-11 The average makespans (λ_1) under each (M, N) with 100 instances respectively for the problem of minimizing makespan and total running time for heterogenous resources.



(a) $M = 5, N \in [5, 100]$ (b) $M = 10, N \in [10, 100]$

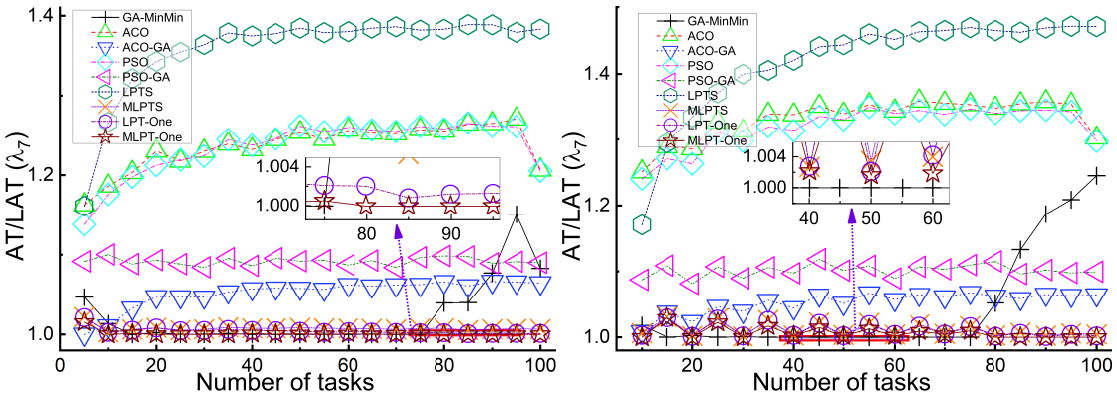
Figure 2-12 The average of total running time (λ_6) under each (M, N) with 100 instances respectively corresponding to the experiments of Fig 2-11.

For the sake of more clear observation for the results, we plot the AM/LAM (λ_2) and AT/LAT (λ_7) in Fig 2-13 and Fig 2-14 respectively. To clearly observe the performance of MLPT-One, LPT-One and LPTS, we also plot their box charts in Fig 2-15 and Fig 2-16. It can be clearly seen from Fig 2-13 and Fig 2-14 that MLPT-One, LPT-One, and MLPTS are obviously superior to other algorithms. LPTS has the highest average makespan and average total running time, which illustrates again that LPTS is not suitable for heterogeneous resources. However, combining LPTS with One-Step, LPT-One greatly improves the per-



(a) $M = 5, N \in [5, 100]$ (b) $M = 10, N \in [10, 100]$

Figure 2-13 The ratio between average makespan and the least average makespan (AM/LAM, λ_2) corresponding to the experiments of Fig 2-11.

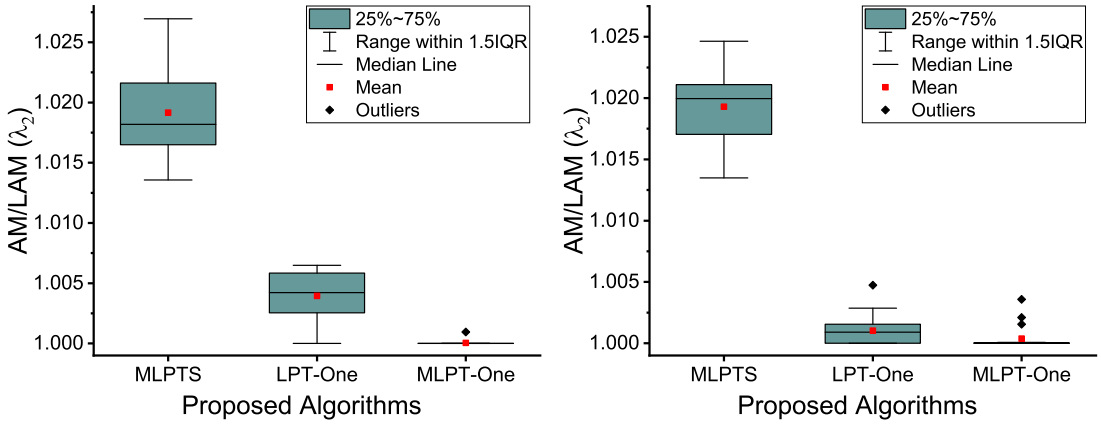


(a) $M = 5, N \in [5, 100]$ (b) $M = 10, N \in [10, 100]$

Figure 2-14 The ratio between average total running time and the least average total running time (AT/LAT, λ_7) corresponding to the experiments of Fig 2-11.

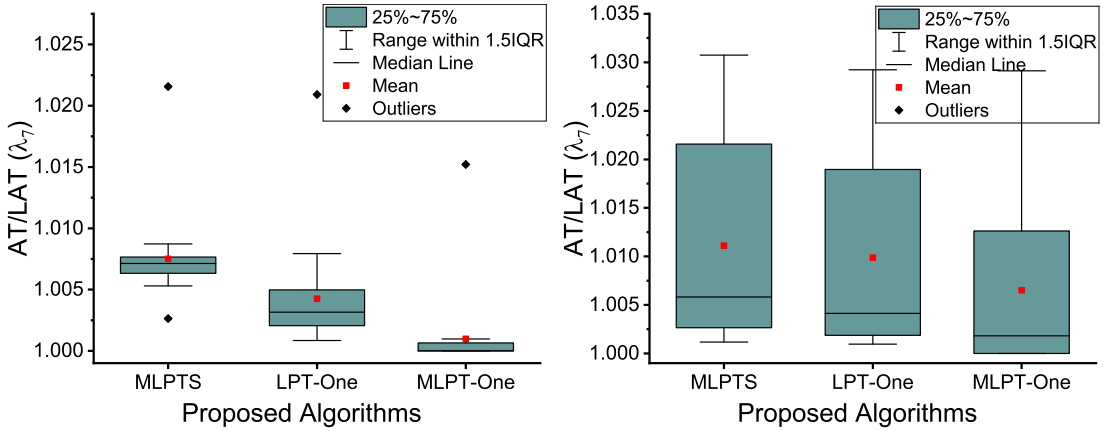
formance, which is because One-Step adds a convergence condition to ensure optimization. The comparison between LPTS and LPT-One also demonstrates that multi-routes can make the algorithm adapt to its originally unsuited scene. From Fig 2-15 and Fig 2-16, MLPT-One outperforms LPT-One and MLPTS, which is because MLPT-One improves the LPT neighborhood compared to LPT-One and adds a search route One-Step compared to MLPTS. Additionally, the observation, that LPT-One performs better than MLPTS in scenarios of heterogeneous resources, confirms the combination of multi-search-routes like LPT-One is more effective than modification of the single algorithm like MLPT.

Pareto Scatter is usually used to evaluate the solution of multi-objective problems [110]. Furthermore, we execute four instances to demonstrate the Pareto Scatter of total running time and makespan as Fig 2-17. From Fig 2-17, the solution of MLPT-One



(a) $M = 5, N \in [5, 100]$ (b) $M = 10, N \in [10, 100]$

Figure 2-15 The box chart of ratio between average makespan and the least average makespan (AM/LAM, λ_2) corresponding to the experiments of Fig 2-11.



(a) $M = 5, N \in [5, 100]$ (b) $M = 10, N \in [10, 100]$

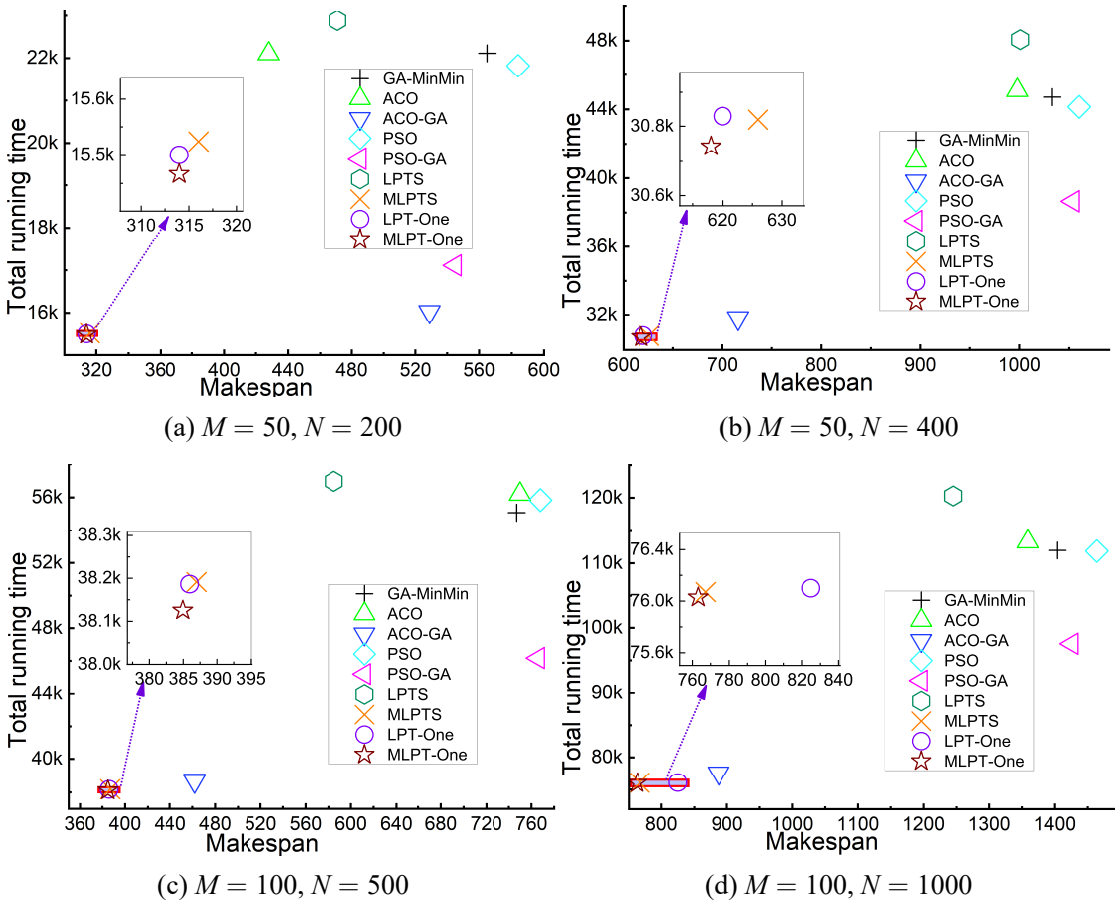
Figure 2-16 The box chart of ratio between average total running time and the least average total running time (AT/LAT, λ_7) corresponding to the experiments of Fig 2-11.

satisfies Pareto Optimality better than compared algorithms, which gets benefits from the assistance of One-Step and shows again the advantages of multi-search-routes.

Overall, these experimental results validate the feasibility and superiority of using multi routes-based algorithms to address problems of heterogeneous resources.

2.6.4 Summary

In multi groups of experiments with abundant simulation instances, the proposed algorithms based on multi-search routes outperform the compared baselines. LPT-One,



(a) $M = 50, N = 200$ (b) $M = 50, N = 400$

(c) $M = 100, N = 500$ (d) $M = 100, N = 1000$

Figure 2-17 Pareto scatter of makespan and total running time for heterogeneous resources.

BFD-One and LPT-BFD-One achieve higher probabilities to obtain the best solutions and with lower approximation ratios of the worst cases for minimizing the makespan of homogenous resources. LPT-One and its modified algorithm MLPT-One achieve better solutions for minimizing makespan and total running time of heterogeneous resources than compared baselines.

2.7 Summary of this Chapter

In this chapter, we propose local search algorithms, LPT-Search, BFD-Search, and OneStep-Search, using heuristic algorithms LPT and BFD as basic search routes to solve resource scheduling problems in Cloud computing. Based on the basic search routes, we also propose multi-search-routes-based algorithms combining various search routes including LPT-One, BFD-One and LPT-BFD-One.

By theoretical deductions, we prove the approximation ratios of LPT-One and BFD-

One as $\frac{5}{4} - \frac{1}{4M}$ as well as that of LPT-K- and BFD-K as $1 + \frac{M-1}{(3+K)M}$, which are better than approximation ratios of LPT, LPT-REV and other existing algorithms for $P||C_{max}$. Moreover, in extensive simulation experiments for minimizing makespan for homogenous and heterogeneous resources, the proposed algorithms based on multi-search-routes outperform the compared algorithms with observations of various indexes, which demonstrates the fact that the proposed algorithms can achieve better solutions in fewer iterations also with better optimization results.

In addition to improving the theoretical approximation ratio of the algorithm, the dominant meaning of proposed algorithms is that they demonstrate the significant potential of applying heuristic algorithms as the search routes of search algorithms and combining different search routes to increase the theoretical analyzability and comprehensive performance of algorithms. Along this research direction as part of future work, we plan to apply the search route to other algorithms such as meta-heuristic algorithms and machine learning algorithms, to explore more search routes-based algorithms and combinations of multi-routes to optimize the performance of scheduling algorithms for more objectives and complex scenarios in Cloud computing. We will also explore whether LPT-K can improve the existing PTAS. In theory, it is also a meaningful work to explore and prove the theoretical approximation ratio of MLPT-One and other search algorithms in heterogeneous resources.

2.8 Appendix: Numerical Table of Experimental Results

The appendix provides numerical tables of several groups of experimental results for the sake of quantitative observation. Among them, Table 2-6 corresponds to Fig 2-4(a), Table 2-7 to Fig 2-4(b), Table 2-8 to Fig 2-6(a), Table 2-9 to Fig 2-6(b), Table 2-10 to Fig 2-11(a), and Table 2-10 to Fig 2-11(b).

Table 2-6 The average makespans (λ_1) under ($M = 20, N \in [25, 100]$) respectively for problem of minimizing makespan for homogeneous resources corresponding to Fig 2-4(a).

Algorithm	$(M = 20, N = ?)$															
	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
GA-MinMin	112.88	127.14	137.23	144.08	165.58	176.17	189.05	197.35	213.49	229.1	239.38	248.63	261.66	278.75	287.59	296.06
GA-Random	121.8	141.73	160.45	174.89	190.9	202.07	223.18	236.06	253.94	271.48	285.63	300.08	311.43	332.68	345.53	355.74
PSO	133.74	151.49	173.22	187.89	206	219.97	241.13	254.25	272.91	291.03	305.69	322.66	334.42	354.76	372.12	379.61
PSO-GA	112.27	127.99	138.52	143.88	165.77	176.71	189.38	197.97	214.15	229.34	239.55	248.57	261.78	279.1	287.88	296.34
LPT	96.57	97.74	103.94	110.09	122.36	133.35	149.65	160.41	172.13	185.84	196.64	208.98	217.09	233.51	244.03	254.17
BFD	96.57	97.74	103.9	109.73	120.92	130.24	144.14	154.49	167.86	182.57	193.6	206.17	215.27	231.43	241.41	250.88
Random	116.7	131.12	141.92	147.03	166.57	176.45	189.06	197.55	213.66	229.38	239.84	249.2	261.86	278.86	287.62	296.06
RR	214	241.35	270.22	288.8	308.6	326.64	347.72	371.4	393.7	422.43	434.78	451.16	467.48	498.04	514.97	517.94
Greedy	116.7	131.12	141.92	147.03	166.57	176.45	189.06	197.55	213.66	229.38	239.84	249.2	261.86	278.86	287.62	296.06
OneS	96.57	97.74	103.9	109.77	120.43	128.93	141.65	152.46	165.23	180.28	191.46	203.96	213.22	229.26	239.56	249.21
LPTS	96.57	97.74	103.94	110.09	122.36	133.35	149.65	160.41	172.13	185.84	196.64	208.98	217.09	233.51	244.03	254.17
BFDS	96.57	97.74	103.9	109.86	120.59	129.23	142.02	152.73	165.45	180.55	191.67	204.17	213.11	229.41	239.73	249.29
BESTBFDS	96.57	97.74	103.9	109.76	**120.39**	**128.83**	**141.54**	152.33	165.23	180.16	191.47	204.09	213.09	229.31	239.61	249.3
LPT-One	96.57	97.74	103.9	**109.75**	120.44	128.95	141.6	152.3	165.17	180.15	191.39	203.86	213.15	229.14	239.55	249.05
BFD-One	96.57	97.74	103.9	109.77	120.43	128.93	141.65	152.46	165.23	180.28	191.46	203.96	213.22	229.26	239.56	249.21
LPT-BFD-One	96.57	97.74	103.9	**109.75**	120.43	128.9	141.56	**152.26**	**165.07**	**180.09**	**191.3**	**203.77**	**213.08**	**229.09**	**239.45**	**248.99**

Table 2-7 The average makespans (λ_1) under ($M \in [5, 100], N = 100$) respectively for problem of minimizing makespan for homogeneous resources corresponding to Fig 2-4(b).

Algorithm	(M =?, N = 100)																			
	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
GA-MinMin	1020.51	517.49	345.3	266.26	219.58	190.86	167.83	154.67	144.39	135.39	127.6	121.39	115	108.65	104.07	101.4	99.6	99.44	99.4	99.34
GA-Random	1020.91	518.89	348.38	270.42	226.49	197.16	177.36	164.65	154.37	145.47	138.81	132.77	125.67	120.7	115.83	111.55	106.23	104.92	103.46	101.38
PSO	1103.69	632.81	465.56	381.12	333.5	300.31	277.1	259.12	247.29	233.53	225.48	214.75	208.39	202.43	197.42	191.35	185.22	179.5	180.04	176.26
PSO-GA	1020.7	517.51	345.74	265.61	220.02	190.86	167.77	154.26	143.48	134.14	128.81	121.16	115.27	108.18	104.97	101.29	99.89	99.44	99.4	99.34
LPT	1020.03	514.55	338.65	255.78	206.45	174.94	153.57	134.49	117.58	106.84	101.23	99.75	99.52	99.53	99.5	99.21	99.48	99.42	99.4	99.34
BFD	1019.75	513.36	337.16	253.2	203.66	170.57	146.42	129.39	116.18	106.41	101.22	99.75	99.52	99.53	99.5	99.21	99.48	99.42	99.4	99.34
Random	1058.54	556.29	382.83	297.61	247.92	218.16	192.61	178.43	163.9	149.41	144.34	140.05	134.67	129.94	125.2	119.93	114.49	109.74	105.21	99.34
RR	1339.12	814.32	622.35	522.41	451.99	416.89	387.49	362.53	337.23	324.51	323.08	304.46	305.03	291.22	285.05	289.45	272.86	266.72	263.18	257.43
Greedy	1058.54	556.29	382.83	297.61	247.92	218.16	192.61	178.43	163.9	149.41	144.34	140.05	134.67	129.94	125.2	119.93	114.49	109.74	105.21	99.34
OneS	1019.12	512.22	335.51	251.27	201.96	168.48	144.28	127.68	115.99	106.47	101.23	99.75	99.52	99.53	99.5	99.21	99.48	99.42	99.4	99.34
LPTS	1020.03	514.55	338.65	255.78	206.45	174.94	153.57	134.49	117.58	106.84	101.23	99.75	99.52	99.53	99.5	99.21	99.48	99.42	99.4	99.34
BFDS	1019.34	512.45	335.75	251.44	201.93	168.54	144.37	127.68	116.07	106.47	101.23	99.75	99.52	99.53	99.5	99.21	99.48	99.42	99.4	99.34
BESTBFDS	1019.33	512.37	335.69	251.36	201.8	168.33	144.15	127.59	115.94	106.46	101.23	99.75	99.52	99.53	99.5	99.21	99.48	99.42	99.4	99.34
LPT-One	1019.12	512.19	335.46	251.14	201.81	168.38	144.19	127.75	115.99	106.47	101.23	99.75	99.52	99.53	99.5	99.21	99.48	99.42	99.4	99.34
BFD-One	1019.12	512.22	335.51	251.27	201.96	168.48	144.28	127.68	115.99	106.47	101.23	99.75	99.52	99.53	99.5	99.21	99.48	99.42	99.4	99.34
LPT-BFD-One	1019.12	512.17	335.41	251.06	201.72	168.32	144.13	127.61	115.97	106.47	101.23	99.75	99.52	99.53	99.5	99.21	99.48	99.42	99.4	99.34

Table 2-8 The probabilities achieving the least makespan (PALM, λ_3) under ($M = 20, N \in [25, 100]$) corresponding to Fig 2-6(a).

Algorithm	$(M = 20, N = ?)$															
	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
GA-MinMin	0.06	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GA-Random	0.06	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PSO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PSO-GA	0.08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LPT	1	1	0.97	0.84	0.43	0.22	0.01	0	0	0	0	0.01	0	0	0	0
BFD	1	1	1	0.97	0.69	0.49	0.03	0.09	0.01	0.02	0.04	0.03	0.05	0.03	0.09	0.08
Random	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Greedy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
OneS	1	1	1	0.93	0.86	0.72	0.65	0.58	0.64	0.59	0.69	0.72	0.68	0.75	0.81	0.7
LPTS	1	1	0.97	0.84	0.43	0.22	0.01	0	0	0	0	0.01	0	0	0	0
BFDS	1	1	1	0.9	0.78	0.58	0.4	0.47	0.51	0.52	0.56	0.55	0.75	0.65	0.69	0.65
BESTBFDS	1	1	1	0.93	**0.87**	**0.81**	0.69	0.7	0.65	0.71	0.7	0.62	**0.81**	0.69	0.76	0.75
LPT-One	1	1	1	0.94	0.85	0.72	0.7	0.72	0.67	0.7	0.73	0.82	0.73	0.86	0.82	0.81
BFD-One	1	1	1	0.93	0.86	0.72	0.65	0.58	0.64	0.59	0.69	0.72	0.68	0.75	0.81	0.7
LPT-BFD-One	1	1	1	**0.94**	0.86	0.74	**0.72**	**0.76**	**0.76**	**0.76**	**0.82**	**0.9**	0.8	**0.91**	**0.92**	**0.89**

Table 2-9 The probabilities achieving the least makespan (PALM, λ_3) under ($M \in [5, 100], N = 100$) corresponding to Fig 2-6(b).

Algorithm	(M =?, $N = 100$)																			
	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
GA-MinMin	0.19	0	0	0	0	0	0	0	0	0	0	0.01	0.02	0.13	0.37	0.64	0.96	0.99	1	1
GA-Random	0.13	0	0	0	0	0	0	0	0	0	0	0	0.01	0	0.06	0.09	0.28	0.47	0.49	0.67
PSO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PSO-GA	0.19	0	0	0	0	0	0	0	0	0	0	0	0.02	0.09	0.32	0.66	0.9	0.98	1	1
LPT	0.4	0.11	0.02	0.01	0.01	0	0	0.02	0.44	0.79	0.99	1	1	1	1	1	1	1	1	1
BFD	0.49	0.23	0.11	0.06	0.01	0	0.03	0.16	0.8	0.97	1	1	1	1	1	1	1	1	1	1
Random	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
RR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Greedy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
OneS	1	0.9	0.85	0.73	0.53	0.61	0.66	0.68	0.85	0.92	0.99	1	1	1	1	1	1	1	1	1
LPTS	0.4	0.11	0.02	0.01	0.01	0	0	0.02	0.44	0.79	0.99	1	1	1	1	1	1	1	1	1
BFDS	0.79	0.68	0.63	0.58	0.59	0.64	0.64	0.61	0.8	0.91	0.99	1	1	1	1	1	1	1	1	1
BESTBFDS	0.79	0.76	0.69	0.68	0.67	**0.77**	0.75	0.71	**0.9**	0.92	0.99	1	1	1	1	1	1	1	1	1
LPT-One	1	0.93	0.9	0.86	0.66	0.71	0.73	0.62	0.85	0.92	0.99	1	1	1	1	1	1	1	1	1
BFD-One	1	0.9	0.85	0.73	0.53	0.61	0.66	0.68	0.85	0.92	0.99	1	1	1	1	1	1	1	1	1
LPT-BFD-One	1	**0.95**	**0.95**	**0.94**	**0.75**	**0.77**	**0.79**	**0.72**	0.87	0.92	0.99	1	1	1	1	1	1	1	1	1

Table 2-10 The average makespans (λ_1) under ($M = 5, N \in [5, 100]$) for the problem of minimizing makespan and total running time for heterogenous resources corresponding to Fig 2-11(a).

Algorithm	GA-MinMin	ACO	ACO-GA	PSO	PSO-GA	LPTS	MLPTS	LPT-One	MLPT-One
5	117.84	161.15	119.98	117.94	113.91	125.07	110.47	108.83	**108.52**
10	219.31	255.22	239.33	233.47	211.6	250.54	201.79	197.25	**196.78**
15	312.98	356.05	351.24	355.58	306.7	378.67	292.44	**285.11**	285.38
20	400.75	471.78	438.52	469.28	401.04	507.89	380.8	373.68	**373.05**
25	492.35	587.32	551.56	589.25	495.03	632.57	471.07	460.76	**460.1**
30	579.63	708.25	641	704.85	589.47	763.06	560.65	548.81	**548.37**
35	673.7	824.91	719.98	819.53	686.52	886.49	653.54	638.66	**636.39**
40	761.01	937.54	809.71	931.28	778.08	1012.89	733.4	721.7	**719.75**
45	849.18	1048.36	925.95	1043.32	879.73	1145.9	825.42	811.97	**808.57**
50	940.75	1170.94	1015.95	1164.48	968.62	1269.27	912.96	900.77	**895.45**
55	1025.15	1279.52	1100.66	1278.34	1067.58	1395.87	1000.95	987.18	**983.58**
60	1110.59	1396.08	1199.26	1388.09	1162.41	1514.34	1087.58	1073.44	**1068.9**
65	1200.43	1508.76	1286.25	1503.23	1254.79	1643.58	1177.65	1161.8	**1156.36**
70	1286.3	1635	1393.58	1623.01	1346.48	1776.3	1264.44	1248.65	**1242.17**
75	1373.36	1738.17	1478.67	1732.56	1443.56	1898.79	1348.22	1333.51	**1327.12**
80	1522.28	1843.21	1567.81	1845.77	1538.95	2025.61	1438.53	1423.11	**1414.35**
85	1630.2	1963.73	1671.91	1961.08	1632.92	2146.93	1523.81	1511.08	**1502.4**
90	1804.21	2079.12	1772.32	2084.7	1728.17	2262.41	1611.54	1599.37	**1589.96**
95	1946.1	2198.04	1857.29	2202.08	1822.28	2394.08	1700.25	1686.32	**1675.46**
100	1981.98	2239.36	1961.87	2254.51	1914.42	2522.67	1785.03	1771.97	**1760.73**

($M = 5, N = ?$)

Table 2-11 The average makespans (λ_1) under ($M = 10, N \in [10, 100]$) for the problem of minimizing makespan and total running time for heterogenous resources corresponding to Fig 2-11(b).

Algorithm		GA-MinMin	ACO	ACO-GA	PSO	PSO-GA	LPTS	MLPTS	LPT-One	MLPT-One
$(M = 10, N = ?)$	10	128	157.49	145.75	151.54	139.26	117.37	104.46	103.1	**103.07**
	15	180.91	230.22	189.55	227.48	179.35	201.09	171.35	**167.23**	167.49
	20	215.24	291.32	242.09	291.66	238.4	243.97	190.19	185.77	**185.69**
	25	265.84	357.9	283.12	358.61	279.05	323.92	250.33	246.21	**245.83**
	30	299.84	418.38	335.25	421.97	329.04	371.35	272.11	**265.83**	266.39
	35	349.35	483.68	365.69	484.02	381.41	446.87	332.33	325.56	**325.09**
	40	386.74	545.67	430.93	551.18	423.88	502.63	356.09	349.92	**349.12**
	45	432.62	605.84	460.54	606.26	478.56	572.29	413.74	406.67	**405.51**
	50	465.11	668.23	515.39	673.83	520.06	628.53	437.01	**431.03**	**431.03**
	55	514.53	726.04	548.94	733.02	576.15	703.49	494.72	485.49	**485.02**
	60	551.09	785.96	604.27	796.62	620.45	757.35	523.62	**513.19**	515.03
	65	594.86	844.83	648.52	854.66	669.23	828.22	575.76	**564.84**	563.86
	70	630.84	906.64	686.45	922.34	714.86	886.76	608.05	597.84	**597.56**
	75	679.4	965.35	737.21	981.89	762.94	957.87	657.65	644.9	**644.27**
	80	800.08	1030.2	783.7	1037.46	808.76	1016.81	690.98	680.25	**679.64**
	85	901.24	1084.82	821.48	1103.5	855.07	1081.83	737.29	**723.19**	723.21
	90	1039.54	1142.18	865.15	1161.15	904.97	1140.58	772.6	**762.14**	762.19
	95	1096.9	1203.78	914.17	1222.68	948.75	1206.22	818.46	803.87	**803.08**
	100	1155.25	1221.15	959.23	1260.96	993.74	1268.47	855.65	845.3	**841.31**

Table 2-12 The average makespans (λ_1) under ($M = 10, N \in [10, 100]$) for the problem of minimizing makespan and total running time for heterogenous resources corresponding to Fig 2-11(b).

Algorithm	(M=20,N=?)							
	30	40	50	60	70	80	90	100
GA-MinMin	127.14	144.08	176.17	197.35	229.1	248.63	278.75	296.06
GA-Random	141.73	174.89	202.07	236.06	271.48	300.08	332.68	355.74
PSO	151.49	187.89	219.97	254.25	291.03	322.66	354.76	379.61
PSO-GA	127.99	143.88	176.71	197.97	229.34	248.57	279.1	296.34
LPT	97.74	110.09	133.35	160.41	185.84	208.98	233.51	254.17
BFD	97.74	109.73	130.24	154.49	182.57	206.17	231.43	250.88
Random	131.12	147.03	176.45	197.55	229.38	249.2	278.86	296.06
RR	241.35	288.8	326.64	371.4	422.43	451.16	498.04	517.94
Greedy	131.12	147.03	176.45	197.55	229.38	249.2	278.86	296.06
OneS	97.74	109.77	128.93	152.46	180.28	203.96	229.26	249.21
LPTS	97.74	110.09	133.35	160.41	185.84	208.98	233.51	254.17
BFDS	97.74	109.86	129.23	152.73	180.55	204.17	229.41	249.29
BESTBFDS	97.74	109.76	**128.83**	152.33	180.16	204.09	229.31	249.3
LPT-One	97.74	**109.75**	128.95	152.3	180.15	203.86	229.14	249.05
BFD-One	97.74	109.77	128.93	152.46	180.28	203.96	229.26	249.21
LPT-BFD-One	97.74	**109.75**	128.9	**152.26**	**180.09**	**203.77**	**229.09**	**248.99**

Chapter 3 Multi-dimensional Resource Scheduling based on Growable Genetic Algorithms

Multi-Dimensional Resources Scheduling Problem (MDRSP, usually a multi-objective optimization problem) has attracted focus in the management of large-scale cloud computing systems as the collaborative operation of various devices in the cloud affects resource utilization and energy consumption. Effective management of the cloud requires a higher performance method to solve MDRSP. Considering the complex coupling between multi-dimensional resources and focusing on virtual machines allocation, we propose GGA-HLSA-RW (GHW, a novel family of genetic algorithms) to optimize the utilization and energy consumption of the cloud. In GGA-HLSA-RW, we add a growth stage to the genetic algorithm and construct a Growable Genetic Algorithm (GGA) using the Heuristic-based Local Search Algorithm (HLSA) with Random multi-Weights (RW) as the growth route. Based on the GHW, we propose GHW-NSGA II and GHW-MOEA/D by applying the sorting strategies and population regeneration mechanism of NSGA II and MOEA/D. To evaluate the performance of GHW, we carry out extensive experiments on the simulation dataset and AzureTraceforPacking2020 for the problems of minimizing the maximum utilization rate of resources for each dimension and minimizing total energy consumption. Experiment results demonstrate the advantages of growth strategy and dimensionality reduction strategy of GHW, as well as validate the applicability and optimality of GHW in realistic cloud computing. The experiments also demonstrate our proposed GHW-NSGA II and GHW-MOEA/D have better convergence rates and optimality than state-of-the-art NSGA II and MOEA/D.

3.1 Introduction

The emerging trend of IoT and mobile communication accelerates the growth of Internet data urgently demanding large-scale software systems [25]. As a successful distributed computing paradigm, cloud computing is playing an important role in various industries. Cloud computing interconnects extensive heterogeneous server nodes to flexibly provide services for various requests from users including computing requests, storage requests, cache requests and mixed requests [124]. With services almost covering all industries, cloud computing has blossomed into an indispensable impetus in the new network

era of IoT currently [2].

The main work units of cloud computing are the components integrated by various micro-circuits including CPU (Central Processing Unit), RAM (Random Access Memory), DS (Disk Storage), GPU (Graphics Processing Unit), BW (BandWidth), etc [4, 110, 125]. Constantly expanding requests from users extraordinarily enhance the difficulty and burden to manage resources of the cloud. Some factors noteworthy in resource management comprise heterogeneity of resources, timeliness of response, operation cost, quality of service, etc. These factors are complex and eventually result in the inferior utilization rate of resources and exceedingly massive energy consumption in realistic cloud computing systems.

In practice, the resource bottleneck in any dimension will limit the operating status of the cloud computing system, and then affects the quality of services. Additionally, the inferior utilization rate of resources usually accompanied by low energy conversion efficiency will also cause excessive CO_2 emissions. Therefore, targeting the preservation of social resources, the research on effective multi-dimensional resources scheduling in heterogeneous nodes of the cloud has become a hotspot.

Resource scheduling in cloud computing is defined by [5] as to find an "optimal" mapping "Tasks $\rightarrow$ Resources" to meet one or several given objectives. Multi-Dimensional Resource Scheduling Problem (MDRSP) in the heterogeneous nodes of cloud computing, as a multi-objectives problem involving resources in different dimensions, is an NP-hard problem and far more complex than single-objective resource scheduling. Some heuristic algorithms, such as LPT (Longest Processing Time First), FCFS (First Come First Serve) and BFD (Best Fit Decreasing) [33, 126], are inappropriate to solve MDRSP. Therefore, meta-heuristic is a common type of algorithm in the existing research on MDRSP such as the modified binary pigeon-inspired algorithm [127], IBSMA (Improved Developed-Slime-Mould-Algorithm) [128], force-directed search [129], NSGA II (Non-dominated Sorting Genetic Algorithm II) [99]. For large-scale cloud computing with ultra-high energy consumption, small-scale improvements to these algorithms will bring considerable significance [130]. Thus, better methods are still necessary especially to obtain the Pareto boundary, where the Pareto boundary is frequently used to evaluate multi-objective optimization algorithms [9, 50].

Focusing on the optimization of resource utilization and energy consumption in cloud computing, this chapter considers the Virtual Machine (VM) allocation scenario in hetero-

geneous nodes with Multi-Dimensional Resources (MDRs). In this scenario, we formulate two types of MDRSPs that are minimizing the maximum utilization rate of each dimension of resources and minimizing energy consumption of the total system. Aiming to solve these problems, we apply the concept of stages to divide the classical genetic algorithm into four stages namely initialization stage, infancy stage, mature stage and genetic stage. Based on these four stages of GA (Genetic Algorithm), we add a growth stage for each individual and propose the Growable Genetic Algorithm (GGA) leveraging the Heuristic-based Local Search Algorithm (HLSA) as its growth route with Random multi-Weight-based dimensionality reduction (RW), which can be called GGA-HLSA-RW (abbreviated as GHW). In solving MDRSP, GHW selects the better part of the individual in each generation to generate the offspring, uses RW to reduce the dimensionality of MDRSP to Multi Single–Objective Problems (MSOPs), utilizes HLSA to gain the solution of these dimensionality-reduced MSOPs as the next individual and then updates the solution set of MDRSP. GHW can be regarded as a family of algorithms that allows the combination of various optimization strategies. To further improve the convergence rate and optimality of GHW, we proposed GHW-NSGA II and GHW-MOEA/D applying the sorting strategies and population regeneration mechanism of NSGA II and MOEA/D (Multi-Objective Evolutionary Algorithm based on Decomposition). Extensive experiments on simulation dataset and experiments driven by the AzureTraceforPacking2020 [131] demonstrate the advantages of our proposed algorithms, where AzureTraceforPacking2020 is a popular public VMs traces representing part of the workload on Microsoft's Azure Compute and is provided by Microsoft Azure for VM allocation. In the scenarios studied in this chapter, several multi-objective optimization quality indicators, including hypervolume-over-time and the average probability of finding the theoretically optimal Pareto solutions, show that our proposed GHW family of algorithms has a much better convergence rate and optimality than the algorithms compared, such as NSGA II and MOEA/D.

The main contributions of this chapter can be summarized as follows.

(1) GGA: Enlighten by the existing research and natural phenomenon, we use the concept of stages to divide the classical genetic algorithm into four stages called initialization stage, infancy stage, mature stage and genetic stage. Based on this, we add a growth stage to GA and propose the Growable Genetic Algorithm (GGA) which allows the individual in GA to grow through various growth routes.

(2) HLSA-RW: We propose the Heuristic-based Local Search Algorithm (HLSA) as the

growth route of GGA and apply Random multi-Weights (RW) to decompose MDRSP to MSOPs. Using the heuristic algorithm as the search route of LSA (Local Search Algorithm) and using LSA as the growth route of the individual in GA are both novel perspectives. Combining GGA, HLSA and RW, we obtain a well-performed GGA-HLSA-RW (GHW) family of algorithms to solve MDRSP. GHW also has a flexible structure to adapt to the combination of various strategies.

(3) GHW-NSGA II and GHW-MOEA/D: We further apply the sorting strategy and population regeneration mechanism of NSGA II and MOEA/D to propose two instantiations of the GHW family i.e., GHW-NSGA II and GHW-MOEA/D, which have better convergence rate and optimality than NSGA II and MOEA/D.

(4) Extensive experiments on the simulation dataset and AzureTraceforPacking2020 [131] with various comparison sights demonstrate the superiority of the GHW family in solving MSRDPs.

The rest of this chapter is organized as follows. We review the related work in Section 3.2. The system model and problem formulation of MDRSP in cloud computing are presented in Section 3.3. The proposed methodology GHW is presented in Section 3.4. The experiment design and evaluation results are presented in Section 3.5. Finally, we conclude this chapter in Section 3.6.

3.2 Related Work

In this section, we briefly review the related work from three aspects: scheduling algorithms in cloud computing, MDRSP and the existing approaches to Multi-objective Optimization Problem (MOP).

3.2.1 Scheduling Algorithms in Cloud Computing

Approaches to optimize the resource utilization in cloud computing include VMs migration [63], queueing model [132], scheduling algorithm, etc. Among them, the scheduling algorithm is the core. In cloud computing, the existing common categories of scheduling algorithms include heuristic, machine learning and meta-heuristic algorithms.

Heuristic algorithms, generally of low computational complexity, are often used to obtain solutions with acceptable performance. Some classical heuristic algorithms include RR (Round-Robin), LPT, greedy, random, FCFS [33,126] etc. In other search algorithms, they can also be used to generate initial solutions to accelerate convergence, for example in

JBA (Jacobi Best-response Algorithm) [59], FISTA (Fast Iterative Shrinkage-Thresholding Algorithm) [133], and LARAC (Lagrange Relaxation based Aggregated Cost) [119].

Machine learning algorithms used for resource scheduling mostly belong to Reinforcement Learning (RL) or Deep Reinforcement Learning (DRL) categories. Some examples are QEEC (Q-learning based framework for Energy-Efficient Cloud) [49] and ADEC (Autonomic Decentralized Elasticity Controller) [53] from the RL category, as well as DQN (Deep Q Network) [40], ADRL (hybrid Anomaly-aware Deep Reinforcement Learning) [125], and DQTS (Deep Q-learning Task Scheduling) [110] from DRL. Combinations of machine learning and other algorithms also adapt to resource scheduling, examples of which are RL+Belief-Learning-Based Algorithm [134], DeepRM-Plus [2] and NN-DNSGA II (Neural Network with Dynamic NSGA II) [122].

The solution space of the NP-hard problem increases exponentially with the increase in data volume. Meta-heuristic is a common method to solve complex optimization problems, especially in big data systems such as the cloud systems [130]. Meta-heuristic algorithms (also evolutionary algorithms always inspired by natural phenomena) include ant colony algorithm such as MALO (Multi-objective AntLion Optimizer) [64] and S-MOAL (Spacing Multi-Objective AntLion algorithm) [42], genetic algorithms such as NSGA II [99], NSGA III [68], MOGA (Multi-Objective Genetic Algorithm) [70] and MOEAs (Multi-Objective Evolutionary Algorithms) [9], Particle Swarm Optimization (PSO) such as MOPSO (Multi-Objective PSO) [71] and HAPSO (Hybrid Adaptive PSO) [73], Artificial Bee Colony (ABC) [135], as well as Firefly Algorithm (FA) [46]. Searchability of solution enables meta-heuristics to utilize local search algorithm or other meta-heuristics as its input to accelerate the convergence, for example: OEMACS [58] leveraged OEM (Order Exchange and Migration) local search techniques; ACO-GA [136], HGA-ACO [120] and DAAGA (Dynamic Ant-colony Algorithm and Genetic Algorithm) [121] leveraged Ant Colony Optimization (ACO) and GA to optimize the search process.

For intuitive observation, we summarize the scheduling algorithms in Table. 3-1.

3.2.2 MDRSP in Cloud Computing

For realistic cloud computing systems, many types of resources need to be arranged simultaneously. The working status of each resource may affect that status of others on the same physical machine, which increases the difficulty of research on MDRSP. Aiming at optimizing resource utilization, energy consumption and cost, researchers have carried

Table 3-1 Summary of Scheduling Algorithms in Literature from Three Categories i.e., Heuristic, Machine Learning and Meta Heuristic.

Categories	Family	Algorithms	Scenarios
Heuristic		LPT [33], FCFS [126], et al.	SOP
Machine learning	RL	QEEC [49], ADEC [53] et al.	SOP, MOP
	DRL	DQN [40], ADRL [125] et al.	
Meta Heuristic	GA	NSGA II [99], NSGA III [68] et al.	SOP, MOP
	ACO	MALO [64], S-MOAL [42] et al.	
	PSO	MOPSO [71], HAPSO [73] et al.	

out numerous studies on MDRSP in the cloud. In addition to the meta-heuristics reviewed in Section 3.2.1, we continue to review some other research on MDRSP.

Goudarzi and Pedram [129] proposed a force-directed search to solve the multi-dimensional SLA-based resource allocation problem in the cloud considering power, memory and bandwidth. Xie et al [137] designed MPTMG (Multi-dimensional Pricing mechanism based on Two-sided Market Game) for distributed MDR allocation in mobile cloud computing considering storage, bandwidth and CPU in cloudset. Bao et al [138] proposed MECC (Multi-dimensional resource allocation-Enabled Cloud Cache), a SLA-aware cloud cache framework, to achieve both the SLA ensurance and cost optimization for NVM-based cloud cache. Pan et al [139] proposed a MDR sharing framework for heterogeneous nodes to reduce the total cost and task failure probability. Combining Lyapunov optimization and Lagrange dual decomposition, Yu et al [140] proposed MER-ITS (Multi-timescale multi-dimEnsion Resource allocatIon and Task Splitting algorithm) to reduce energy consumption, queuing delay, queue backlog and increase connection success ratio. Gopu and Venkataraman [141] applied MOEA/D to solve optimal VM placement in the cloud considering wastage, power consumption and propagation delay simultaneously.

In addition, research on MDRSP is also a hot topic in other scenarios. For multi-dimensional knapsack problem, negative learning in ant colony optimization [142], sum-of-ratios-based decomposition [143], Modified-BPIO (Modified Binary Pigeon-Inspired Optimization) [127] and IBSMA [128] were proposed and achieved considerable performances. For multi-dimensional transport problems, Aktar et al applied three ways, i.e., weighted sum technique, max-min Zimmermann technique and neutrosophic programming technique, to reduce multi-objective to single-objective and then used generalized

reduced gradient method for solutions [144]. For diverse safety message transmissions in vehicular networks, Chen et al [145] developed a MDR allocation scheme to jointly optimize the sensing resource allocation.

3.2.3 Existing Approaches to MOP

MDRSP is actually one of the Multi-objective Optimization Problems (MOP, also known as multicriteria optimization) [146,147]. Solving a MOP generally requires two aspects: evaluation indicator of solutions and simplification of problems. These two aspects also extended various optimization algorithms to MOP. In this subsection, we will review them from these two aspects.

Evaluation indicator-based methods include two popular types: non-dominated sorting-based method [147,148] and hypervolume-based method [149,150].

For the Non-dominated Sorting (NS) based method, Srinivas and Deb [151] proposed Non-dominated Sorting Genetic Algorithms (NSGA). Based on the concept of Pareto optimization, NSGA stratifies the individuals according to their dominant and non-dominant relationships, which improves the convergence rate to solve MOPs [151]. Using elite strategy and congestion comparison operator, Deb et al [152] proposed NSGA II, which guarantees the uniform distribution of the non-inferior optimal solution. Subsequently, NSGA III [68] applied reference points to replace congestion sorting of NSGA II, which is more suitable to high dimensional MOP. Other variants of NSGA include B-NSGA III, U-NSGA III [147], NSGA II-C [148], NN-DNSGA II algorithm [122], et al.

HyperVolume (HV), proposed by Zitzler et al, is an important indicator to evaluate the optimality of the Pareto solution set [153,154]. The indicator HV also extended a novel type of approach (HV-based method) to solve MOP. Some examples are R2HCA-EMOA (R2-based Hypervolume Contribution Approximation Evolutionary Multi-objective Optimization Algorithm)) [149] and UHV-GOMEA (Uncrowded HyperVolume and Gene-pool Optimal Mixing Evolutionary Algorithm) [150].

Simplification of problems-based methods mainly includes some dimension reduction-based methods.

Dimension reduction in MOP means using some methods to obtain problems with fewer objectives by decomposing MOP into Multi Lower-Dimensional objective Problems (MLDPs) [155–157]. Brockhoff and Zitzler [157] proposed an exact algorithm and fast heuristics to reduce the dimensions of objectives to assist evolutionary MOP with large

dimensions. Ruochen Liu et al [155] proposed a clustering and dimensionality reduction-based evolutionary algorithm for large-scale MOP with dimensions up to 5000. Zheng Tan [156] et al proposed multi-stage dimension reduction to make surrogate-assisted evolutionary algorithms capable to handle sparse MOPs.

A specific case of dimension reduction is scalarization, which means decomposing the MOPs into Multiple Single-Objective Problems (MSOPs) [146]. Aggregating the objectives into a weighted sum is a frequent approach to scalar the MOP [146]. The weighted sum approach enables computation of the properly Pareto optimal in convex cases, while may work poorly in non-convex cases [146]. Other approaches including ε-constraint method, Benson's method, and compromise programming are applicable in non-convex solution space to transform MOP to SOPs [146]. MOEA/D, proposed by Qingfu Zhang and Hui Li [158], is a typical scalarization-based method, which combines genetic algorithms and a weighted sum method [146]. On the basis of MOEA/D, Hang Xu et al [159] proposed a novel MOEA based on Hierarchical Decomposition (MOEA/HD) which decomposed the MOP into subproblems layered in different hierarchies. Jie Cao et al [160] proposed MOEA/D-TS (a Two-Stage evolutionary strategy based MOEA/D) to improve MOEA/D. In MOEA/D-TS [160], the first stage focused on pushing the solutions into the area of the Pareto front to speed up its convergence ability, as well as the second stage conducted in the operating solution's diversity to make the solutions distributed uniformly.

For the sake of observation, we summarize these approaches in Table. 3-2.

Table 3-2 Summary of Approaches to MOPs in Literature from Two Aspects i.e., Indicator- and Simplification-based Approaches.

Aspect	Approaches	Family	Algorithms
Indicator-based	NS-based	NSGA	NSGA II [152], NSGA III [68], B-NSGA III [147] et al.
	HV-based		R2HCA-EMOA [149], UHVI [150] et al.
Simplification-based	Dimension reduction-based	MOEA/D	MOEA/D [158], MOEA/HD [159], MOEA/D-TS [160] et al.
		Others	ε-constraint method, Benson's method [146] et al.

3.2.4 Analysis of Related Work

MDRSP, as a type of challenging MOPs, has complex problem features and discontinuous solution spaces. One common type of its method is the meta-heuristic algo-

rithm. Among them, NSGA family and MOEA/D family are the most popular to solve MOPs. Some comparative studies between NSGA family and MOEA/D family [161, 162] showed both these two families of algorithms have good convergence in continuous multi-objective optimization space. When in large-scale discrete solution space, the searching ability of these algorithms is insufficient. Therefore, they require abundant population size and generations to obtain an acceptable solution, which will cost a lot of computing time. Moreover, their local optimal solutions may be far from the theoretical optimal solutions due to their essential characteristics.

Conventionally, a genetic algorithm contains several processes: initializing the individuals, selecting the excellent individuals to participate in the pairing, and executing crossover and mutation to generate the children individuals, regenerate the population with a specific mechanism to generate the next generation. From the above review, the existing genetic algorithms mainly focus on the improvement of the population selection strategies and population regeneration mechanisms, such as non-dominated sorting, elite strategy, and competition mechanism [163–165], which have improved the searchability and local optimum of GA to some extent. However, as they don't pay special attention to the growth process of the individuals outside the crossover and mutation, the convergence rate and optimality need to be further improved.

Referring to the previous research, this chapter reorganizes the process of genetic algorithms by the concept of stages and adds a growth stage for each individual to obtain a novel architecture of genetic algorithm called Growable Genetic Algorithm (GGA). The GGA allows the combination of various algorithms and the individuals have more flexible evolutionary routes. To solve the MDRSP in cloud computing, we propose the Heuristic-based Local Search Algorithm (HLSA) to instantiate the growth route of GGA and use Random multi-Weights (RW) as the growth direction of individuals. Combining the above components, we propose GGA-HLSA-RW (GHW), which can effectively solve MDRSP in the cloud.

3.3 Cloud Systems and Optimization Problems Formulations considering Multi-dimensional Resources

To assist with the system model and problem formulations, Table. 3-3 lists the descriptions of some notations in this chapter.

Table 3-3 Notations and Descriptions.

Notation	Description
n	Number of tasks or VMs
m	Number of nodes
d	Number of dimensions of resources
i	Index of task or VMs
j	Index of nodes
k	Index of dimensions of resources
V_i	The task or VMs with index i
P_j	The node with index j
C_i	The property matrix of V_i
C_{ij}^{CPU}	The CPU capacity requested by T_i allocated on P_j in unit of MIPS (Million Instructions Per Second)
C_{ij}^{RAM}	The RAM capacity requested by T_i allocated on P_j in unit of Gigabytes
C_{ij}^{DS}	The Disk storage requested by T_i allocated on P_j in unit of Gigabytes
C_{ij}^{GPU}	The GPU capacity requested by T_i allocated on P_j in unit of Gigabytes
C_{ij}^{BW}	The bandwidth of network requested by T_i allocated on P_j in unit of Mbps (Million bits per second)
ψ_j	Set of tasks and VMs in node P_j
κ	The set of ψ_j where $\kappa = \langle \psi_1, \psi_2, \ldots, \psi_m \rangle$
x_{ij}	If $V_i \in P_j$ then $x_{ij} = 1$, otherwise $x_{ij} = 0$
L_{jk}	The limited capacity of resource in the k-th dimension of the node P_j
S_{jk}	The load of resource in k-th dimension of the node P_j
U_{jk}	The utilization rate in k-th dimension of the node P_j
u_{ijk}	The resource occupancy rate of V_i for the k-th dimension in P_j
$G_{jk}(S_{jk})$	The function between the load of resource in k-th dimension of server node P_j and energy consumption
E_j	The total energy consumption the node P_j
E	The total energy consumption of the cloud system
N_p	The number of individuals in each generation of genetic algorithm
N_g	The number of generations in genetic algorithm
G_{step}	The number of search steps of each individual in each generation through HLSA in GGA

3.3.1 Cloud System Model with Multi-Dimensional Resources

A cloud computing system usually consists of a large number of server nodes and integrates the resource layers of these nodes through the high-speed network as Fig. 3-1. We demonstrate cloud servers as heterogeneous nodes because of the default supportability of the cloud systems for heterogeneity. Then, we model it as a multi-dimensional

system model and model the problem of VMs allocation in it as a MDRSP.

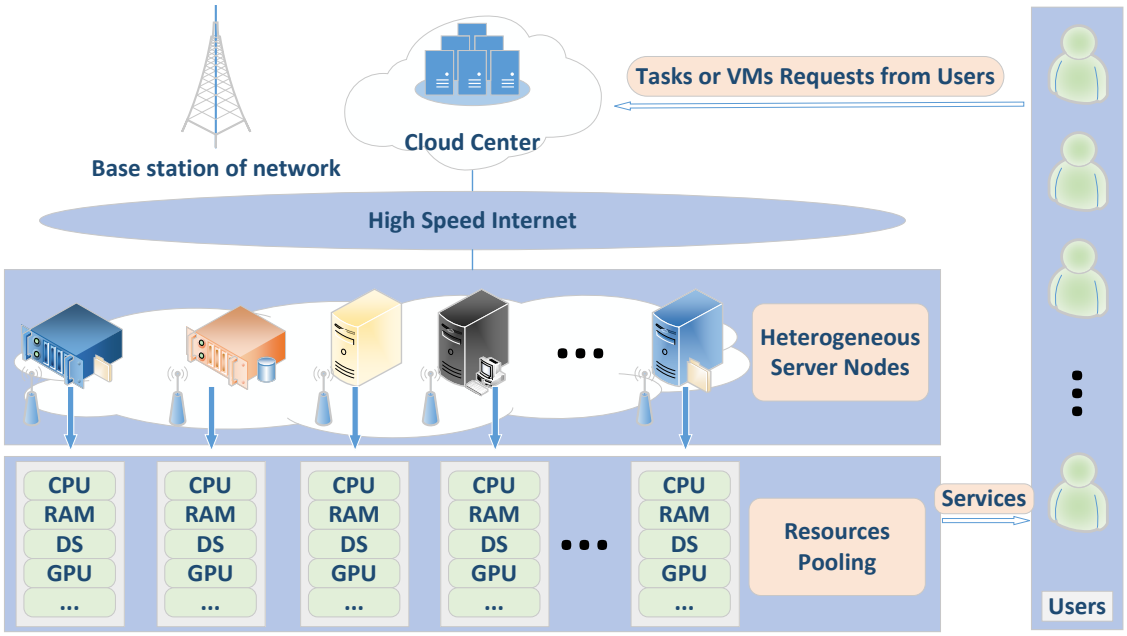


Figure 3-1 Structure of cloud computing with various resources.

We consider a cloud system with m heterogeneous nodes (denoted as $P = \langle P_1, P_2, \ldots, P_m \rangle$) and each node with d-dimensions of resources such as CPU, RAM, disk storage, GPU, bandwidth etc. The set of tasks and VMs in a time slot $[t, t+\delta t)$ is denoted as $V = \langle V_1, V_2, \ldots, V_n \rangle$. The executions of the same request are different in different nodes and always need multi resources synergistically. Thus, we assume that a task or VM request from users equals a request for resources in multiple dimensions. Based on the above, we can set the property of a task or VM V_i as a matrix $C_i = \{C_{ijk}\}_{1 \le j \le m, 1 \le k \le d}$. The j-th row $C_{ij} = \left\langle C_{ij}^{CPU}, C_{ij}^{RAM}, C_{ij}^{DS}, C_{ij}^{GPU}, C_{ij}^{BW}, \ldots \right\rangle$ denotes the capacity request for resources in each dimension when V_i is allocated to the j-th node. Each node has limited capacity in each dimension (i.e., the maximum load for healthy operation of components) that can be set as $L = \{L_{jk}\}_{1 \le j \le m, 1 \le k \le d}$ where $L_j = \left\langle L_j^{CPU}, L_j^{RAM}, L_j^{DS}, L_j^{GPU}, L_j^{BW}, \ldots \right\rangle$. For example, L_j^{CPU} is the j-th node's capacity of CPU and L_j^{DS} is its disk size. This model corresponds to various requests (computing requests, storage requests, cache requests, transmission requests, VMs requests, etc.) involved in cloud computing systems.

A diagram of the allocation of tasks or VMs to heterogeneous nodes with MDRs is shown in Fig. 3-2. Although VMs migration and task segmentation can also be leveraged to solve the resource scheduling in the cloud, they still can not avoid the allocation of tasks or VMs. In view of this, we do not consider the VMs migration and task segmentation. Then, we mainly focus on the direct allocation of tasks or VMs where any task or VM can

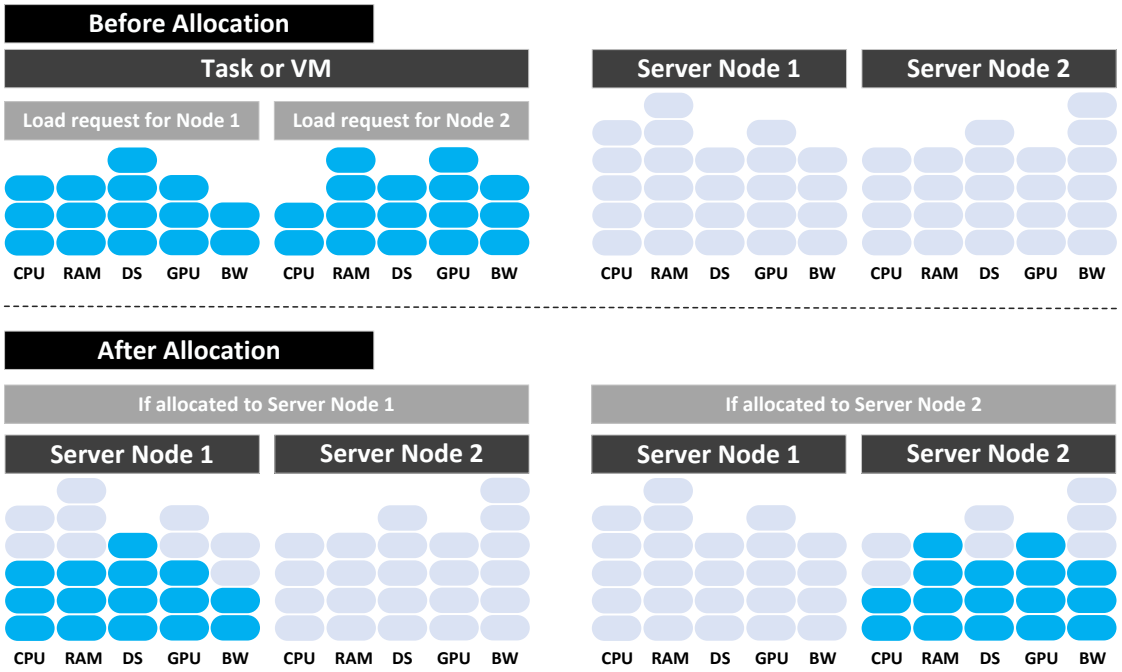


Figure 3-2 Allocation of a task or VM to heterogeneous nodes with multi-dimensional resources.

not be further split into smaller ones. This also means any task or VM will be allocated to only one server node supporting affinity while one server node can process multiple tasks or VMs simultaneously. We denote the set of tasks and VMs in node P_j as ψ_j. If a task or VM V_i is allocated to the node P_j, we use $V_i \in \psi_j$. The ψ_j of each node constitutes a vector $\kappa = \langle \psi_1, \psi_2, \ldots, \psi_m \rangle$. Therefore, we can gain the relationships of ψ_j that $\bigcup_{j=1}^{m} \psi_j = V$, $\psi_j \subset \kappa$, and $\psi_j \bigcap \psi_l = \emptyset$ for $\forall 1 \leq j \neq l \leq m$. κ determines the unique allocation result corresponding to the solution of MDRSP.

We use S_{jk} to denote the load of resource in the k-th dimension of the j-th node. Then, the load vector of the j-th node is expressed as $S_j = \langle S_{j1}, S_{j2}, \ldots, S_{jd} \rangle$. The occupancy of most components approximately satisfies linear superposition. Thus, resource occupation of each dimension on a node is equal to the sum of the requests of all VMs on it shown as Eq. (3-1). A diagram of Eq. (3-1) is presented in Fig. 3-3.

$$S_{jk} = \sum_{V_i \in \psi_j} C_{ijk} \tag{3-1}$$

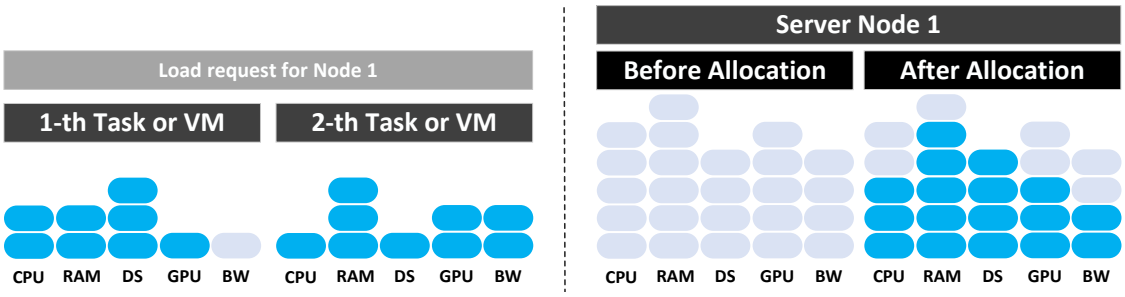


Figure 3-3 Relationship of linearly superposition for multi-dimensional resources allocating two tasks or VMs to one server node.

3.3.2 Problem Formulations for Resources Utilization and Energy Consumption

Cloud computing is based on the pay-as-you-go pattern [2] and regards the resources as ubiquitous "cloud". Generally, cloud has several targets: providing as many services as possible, ensuring flexibility in providing services, reducing the overall energy consumption, optimizing the resource utilization and prolonging the service life of components. In this chapter, we transform its aims and focus on two problems:

(1) Minimizing the maximum utilization rate of resources for each dimension under all nodes;

(2) Minimizing the energy consumption for the whole system.

3.3.2.1 Minimizing the Maximum Utilization Rate of Resources for Each Dimension under All Nodes

There are various indicators to evaluate balancing degree in the cloud such as variance or standard deviation of load [46, 110], average success rate [51], coefficient of variance [45], degree of imbalance [122], etc. In this chapter, we leverage the objectives of minimizing the maximum utilization of resources in each dimension. It is also a method to perform load balancing, improve resource utilization and ensure that the cloud system can process more VMs. The problem with multi-objectives can be formulated as Eq. (3-2), where we denote $\min \omega_k^{(1)} = \min \max (S_{1k}, S_{2k}, \ldots, S_{mk})$ and $S_{jk} \leq L_{jk}$ for $\forall j \in \{1, 2, \ldots, m\}$ and

$\forall k \in \{1, 2, \ldots, d\}$.

$$\min \omega^{(1)} = \left(\min \max_{j=1,2,\ldots,m} S_{jk} \right) \Big|_{k=1,2,\ldots,d} = \min \begin{cases} \max (S_{11}, \ldots, S_{m1}) \\ \max (S_{12}, \ldots, S_{m2}) \\ \ldots \\ \max (S_{1d}, \ldots, S_{md}) \end{cases} \tag{3-2}$$

Converting Eq. (3-2) to zero-one integer programming problem can obtain Eq. (3-3).

$$\min \omega_k^{(1)} = \min \left(\max_{j=1,2,\ldots,m} \left(\sum_{i=1}^{n} x_{ij} C_{ijk} \right) \right) \tag{3-3}$$

We assume the resource utilization rate of each dimension as the ratio of the occupied load to the limited capacity that is:

$$U_{jk} = \frac{S_{jk}}{L_{jk}} = \frac{\sum_{i=1}^{n} x_{ij} C_{ijk}}{L_{jk}} \tag{3-4}$$

In the system model of this chapter, L_{jk} and C_{ijk} are given and invariant. Thus, we can denote a parameter $u_{ijk} = C_{ijk}/L_{jk}$ to express the occupancy rate of a single VMs V_i for the k-th dimension of node P_j. The utilization rate u_{ijk} also satisfies the superposition relationship:

$$U_{jk} = \sum_{V_i \in \psi_j} u_{ijk} = \sum_{i=1}^{n} x_{ij} u_{ijk}. \tag{3-5}$$

Thus, if an algorithm can adapt to C_{ijk}, it can also apply to u_{ijk}, and vice versa. Then, a problem to reduce the utilization rate of resources for each dimension is as Eq. (3-6).

$$\min \omega_k^{(2)} = \min \left(\max_{j=1,2,\ldots,m} \left(\sum_{i=1}^{n} x_{ij} u_{ijk} \right) \right) \tag{3-6}$$

where the constraints are:

$$\textbf{s.t.} \begin{cases} \sum_{j=1}^{m} x_{ij} = 1, \\ \sum_{i=1}^{n} x_{ij} C_{ijk} \le L_{jk} \Leftrightarrow \sum_{i=1}^{n} x_{ij} u_{ijk} \le 1, \\ 0 \le C_{ijk} \le L_{jk} \Leftrightarrow 0 \le u_{ijk} \le 1, \\ x_{ij} \in \{0, 1\}, i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, m\}, k \in \{1, 2, \ldots, d\} \end{cases} \tag{3-7}$$

3.3.2.2 Minimizing the Total Energy Consumption for System

Minimizing the number of working nodes is a frequent way to optimize the energy consumption of the cloud, whose optimization objective can be written as:

$$\min \omega^{(3)} = \min \sum_{j=1}^{m} \max_{i=1,2,\ldots,n} x_{ij} \tag{3-8}$$

The use of Eq. (3-8) requires the assumption that the operating energy consumptions of all nodes are similar. However, the ratio between load and energy consumption may be varying with the different nodes. Therefore, we consider the relationship between the energy consumption and load to lean closer to the actual scene. We assume $G_{jk}(S_{jk})$ as the function between the load of resource in k-th dimension of server node P_j and its required energy consumption where we denote E_j as the total energy consumption the node P_j. In reality, $G_{jk}(S_{jk})$ is often non-linear related to the status of sever nodes such as temperature. In this chapter, we do not address the issue of the relationship between load, energy consumption and status. Thus, we assume each $G_{jk}(S_{jk})$ is a given function. Without losing generality, we also assume the energy consumption of all dimensions of resources is subject to superposition. Then, formulas for energy consumption can be obtained as:

$$E = \sum_{j=1}^{m} E_j = \sum_{j=1}^{m} \sum_{k=1}^{d} G_{jk}(S_{jk}) \tag{3-9}$$

The components in the computer mainly process the task by switching high-low voltage signals. According to Ohm's law, the electrical power is equal to the square of the voltage divided by the resistance. Thus, we set up the function $G_{jk}(S_{jk})$ as a quadratic polynomial function in this chapter:

$$G_{jk}\left(\sum_{k=1}^{q} S_{jk}\right) = a_{jk} S_{jk}^2 + b_{jk} S_{jk} + c_{jk} + sgn(S_{jk})\, d_{jk} \tag{3-10}$$

where $sgn()$ is the signum function. When $\sum_{k=1}^{q} S_{jk} = 0$, $c_{jk} = E_{jk}^{idle}$ corresponds to the energy consumption of k-th dimensional resource when node P_j is idle. When $\sum_{k=1}^{q} S_{jk} > 0$, $c_{jk} + d_{jk} = E_{jk}^{on}$ corresponds to the energy consumption of k-th dimensional resource when node P_j is on working. Then, the total energy consumption of the system can be

rewritten as follows by substituting x_{ij} and Eq. (3-10) into Eq. (3-9).

$$E = \sum_{j=1}^{m}\sum_{k=1}^{d}\left(a_{jk}\left(\sum_{i=1}^{n}x_{ij}C_{ijk}\right)^{2} + b_{jk}\left(\sum_{i=1}^{n}x_{ij}C_{ijk}\right)\right) + \sum_{j=1}^{m}\sum_{k=1}^{d}(c_{jk}) + \sum_{j=1}^{m}\left(\max_{i=1}^{n}(x_{ij})\sum_{k=1}^{d}d_{jk}\right) \tag{3-11}$$

Obviously, the objective of minimizing the total energy consumption is as:

$$\min \omega^{(4)} = \min E \tag{3-12}$$

where the constraints are also as Eq. (3-7).

3.4 Algorithm Design: Growable Genetic Algorithm

For MDRSP, the commonly used three basic steps to obtain a solution set are as:

(1) Reducing dimensionality of the MDRSP to Multi Single-Objective Problems (MSOPs) or Multi Lower-Dimensional objective Problems (MLDPs);

(2) Solving the solutions of MSOPs or MLDPs;

(3) Integrating the solutions of MSOPs or MLDPs to obtain the solution set of the original MDRSP.

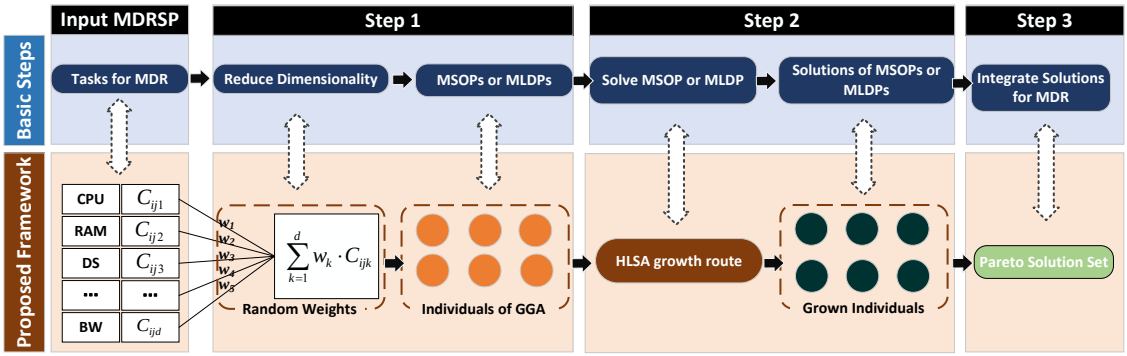


Figure 3-4 Basic steps of our proposed framework to solve MDRSPs.

Therefore, we need to achieve these three steps by building corresponding methods to solve MDRSP. We map them into a growable genetic algorithm framework as shown in Fig. 3-4. In this framework, we leverage the Random multi-Weights method (RW) to achieve dimensionality reduction and use the individuals of GGA to represent the dimensionality-reduced MSOPs; propose the HLSA-based growth route to represent the solving process of each MSOP and obtain the mature individuals after HLSA-based growth as the solution of MSOP; finally, update the Pareto solution set of MDRSP by

integrating the grown mature individuals.

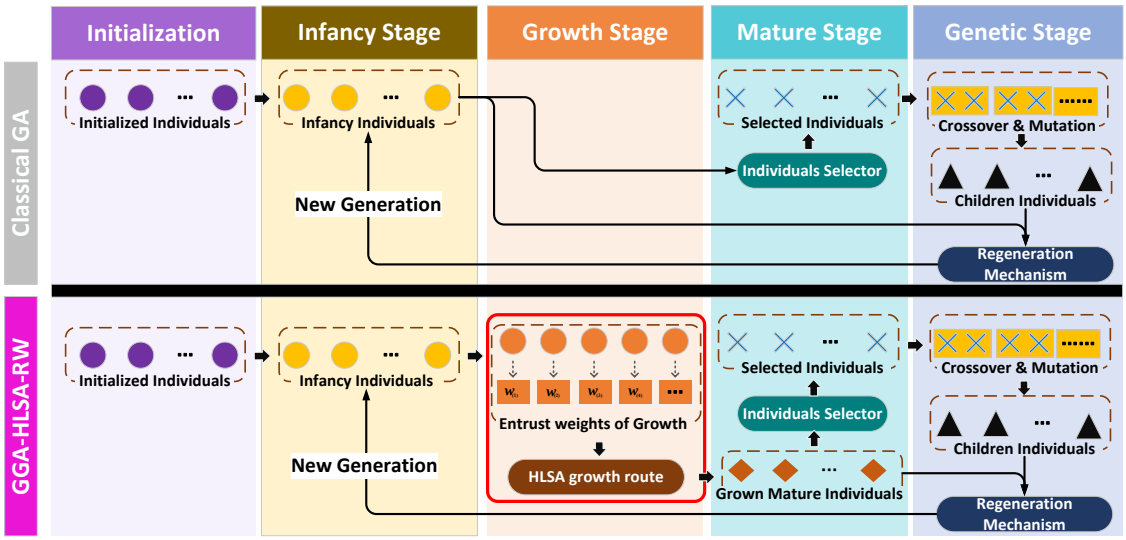


Figure 3-5 The flowchart comparison between the classical GA and our proposed GGA-HLSA-RW (GHW).

The key to our proposed framework is the Growable Genetic Algorithm (GGA) using HLSA as the growth route. To illustrate its structure and highlight the difference between it and the classical GA, we present their flowchart in Fig. 3-5, where the flowchart of the classical GA is obtained by integrating the genetic-related literature [68, 70, 99, 151, 152].

As shown in Fig. 3-5, we apply a concept of stages to divide the classical GA into four stages namely initialization stage, infancy stage, mature stage and genetic stage. The classical GA firstly takes the initialized individuals as the early individuals of the infancy stage; secondly selects some excellent individuals in the mature stage according to some strategies such as fast non-dominated sorting algorithm [68, 152]; then pairs the selected individuals; executes crossover and mutation to generate the children individuals; finally, choose a part of individuals as the individuals of the next infancy stage.

On the basis of the four stages of the classical GA, we add a growth stage in GGA shown as the part in the red box of Fig. 3-5. In the growth stage, we randomly entrust weights to the individuals as their directions of ability cultivations, use the Heuristic-based Local Search Algorithm (HLSA) to cultivate the individuals of the infancy stage, and then select the grown individuals for the subsequent genetic process. The application of a novel growth stage is conducive to quickly obtaining better optimization solutions for MDRSP. Finally, we obtain our proposed Growable Genetic Algorithm (GGA) using the HLSA-based growth route with Random multi-Weights (RW) named GGA-HLSA-RW

(abbreviated as GHW).

Next, we will present the components of GGA-HLSA-RW (GHW) respectively including RW, HLSA, and GGA based on various growth strategies.

3.4.1 Random Multi-weights-based Dimensionality Reduction

In this chapter, we mainly consider transforming MDRSP into MSOPs. However, for strict consideration, the framework of this chapter is also suitable for transforming MDRSP into MLDPs. Therefore, we still use dimensionality reduction (or decomposition) instead of scalarization in this chapter.

Before discussing the method of dimensionality reduction, we first presuppose we have a valid and efficacious algorithm (marked as A_h) to solve the single-dimensional resource scheduling problem. Taking $\omega_k^{(2)}$ in Eq. (3-6) as an example, it can be regarded as a single-dimensional resource scheduling problem (also a single-objective optimization problem) regardless of other dimensions. A_h can gain a sufficiently optimized solution of Eq. (3-6) for the k-th dimension based on our presuppose. Assuming minimizing the weighted sum of utilization in each dimension marked as $\bar{\omega}_{(w)}^{(2)}$, modification of Eq. (3-6) can gain a problem as:

$$\min \bar{\omega}_{(w)}^{(2)} = \min \left(\max_{j=1,2,\ldots,m} \left(\sum_{i=1}^{n} x_{ij} \sum_{k=1}^{d} (w_k \cdot u_{ijk}) \right) \right) \tag{3-13}$$

where $w = \langle w_1, w_2, \ldots, w_d \rangle$ is a vector of weights. It can be set $W_{ij} = \sum_{k=1}^{d} w_k \cdot u_{ijk}$. Because w and u_{ijk} are given and invariant, W_{ij} is a constant, which means Eq. (3-13) is a single-dimensional problem and has the same form as Eq. (3-6). More crucially, the algorithm A_h can apply to solve it and obtain a sufficiently optimized solution. Other problems are analogous to this property. For sake of the following discussion, we set the solution in each dimension of Eq. (3-13) obtained by A_h as $A_h\left(\bar{\omega}_{(w)}^{(2)}\right)$ where:

$$A_h\left(\bar{\omega}_{(w)}^{(2)}\right) = \left\langle \omega_1^{(2)}, \omega_2^{(2)}, \ldots, \omega_d^{(2)} \right\rangle \tag{3-14}$$

Therefore, $\omega_k^{(2)} = \max_{j=1,2,\ldots,m} \left(\sum_{i=1}^{n} x_{ij} \sum_{k=1}^{d} (w_k \cdot u_{ijk}) \right)$ where $\{x_{ij}\}$ is the obtained optimization solution of Eq.(3-13) solved by A_h.

The weight vector w is actually an angle to slice the multi-dimensions of resources and W_{ij} is the projection of this slice. This enlightens us to analyze whether the optimal solutions for the weights of all angles can cover all Pareto solutions. A general MDRSP

can be assumed as:

$$\min \omega = \min \begin{cases} \omega_1 \\ \omega_2 \\ \cdots \\ \omega_L \end{cases} \tag{3-15}$$

where:

$$\omega_k = f\left(\{x_{ij} Y_{ijk}\}_{1 \le i \le n, 1 \le j \le m}\right) \tag{3-16}$$

which expresses a function related to all the elements of the $n \times m$ matrix $\{x_{ij} Y_{ijk}\}_{1 \le i \le n, 1 \le j \le m}$ where Y_{ijk} is given parameter, and the constraints are as Eq. (3-7). Assuming T_w as a weight set traversing all angles in Euclidean space $\mathbb{R}^L$, then we can denote the optimization solution set under each weight of T_w as B where:

$$B = \left\{A_h(\omega_{(w)}) | \forall w \in T_w\right\}, \tag{3-17}$$

theoretical Pareto solution set of Eq. (3-15) as D, and the set of solutions included in B as G subject to properties that: for $\forall g \in G$ and $\forall b \in B$, b does not dominate g; for $\forall b \in B - G$, $\exists g \in G$ **s.t.** g dominates b. Therefore, the key is to find a T_w **s.t.** $card(D - G)$ as small as possible, where $card(D - G)$ means the cardinality of set $D - G$.

This reveals a method to obtain the Pareto solutions of MDRSP which is to traverse weights of all angles. The way to approximately perform traversing weights is enumerated by cyclic isometrics based on a definite interval. However, the weight set T_w is continuous containing infinite elements, and the solutions $A_h(\omega_{(w_1)})$ and $A_h(\omega_{(w_2)})$ of two weights w_1 and w_2 expressing different angles may be the same. Therefore, differentiating from cyclic isometric, a random multi-weights is applied to obtain the slices of reduced dimension. It means to select multi-weights randomly to generate corresponding single-dimensional problems and respectively obtain the solutions of these problems, as Algorithm 3-1.

As the growth stage of GGA allows the combination of various algorithms, a multi-objective optimization algorithm (including GHW itself) can be nested in the growth stage. However, considering this chapter focuses on the proposal of the GHW framework (a novel genetic-based framework) and the validation of its applicability in MDRSP, this chapter mainly uses a group of random weights to transform a MOP into MSOPs at the

Algorithm 3-1 Random multi-weights-based dimensionality reduction

Input : MDRSP with form as Eq. (3-15), constraints as Eq. (3-7) and the parameters of tasks as Y_{ij} where $Y_{ij} = \langle Y_{ij1}, Y_{ij2}, \ldots, Y_{ijd} \rangle$

Output: Solution set B of decomposed problems

1 **Generate** random weight set $T_w = \left\langle w_{(1)}, w_{(2)}, \ldots \right\rangle$ with multiple weights where $w_{(l)}$ is vector with length d

2 **Obtain** dot product $w_{(l)} \cdot Y_{ij}$ as new parameters

3 **Obtain** the problem after dimensionality reduction as:

$$\min \omega_{(w_l)} = \min f\left(\left\{ x_{ij} \left(w_{(l)} \cdot Y_{ij} \right) \right\}_{1 \le i \le n, 1 \le j \le m} \right) \tag{3-18}$$

where Eq. (3-18) is single-dimensional problem in $w_{(l)}$

4 **Solve** the problem Eq. (3-18) by the given algorithm A_h

5 **Obtain** the solution set B as Eq. (3-17)

growth stage of each generation, which also means the weights is variable after a generation.

3.4.2 Heuristic-based Local Search Algorithm

The next key is an effective algorithm A_h to gain the optimal solution for dimensionality-reduced problems, which directly determines the optimality of the solution for MDRSP. To improve the solution of A_h, we proposed the Heuristic-based Local Search Algorithm (HLSA) combining the superiorities of heuristic and local search.

Entrusting the heuristic algorithm the role of search route, we define a neighborhood in the heuristic-based local search algorithm as: It can be assumed that two solutions $\kappa = \langle \psi_1, \psi_2, \ldots, \psi_m \rangle$ and $\kappa' = \langle \psi'_1, \psi'_2, \ldots, \psi'_m \rangle$. If $\exists j_1 \neq j_2 \in \{1, 2, \ldots, m\}$ subject to that $\psi_l = \psi'_l$ for $\forall l \in \{1, 2, \ldots, m\} - \{j_1, j_2\}$. And if the result after reallocating the VMs set $\psi_{j_1} \cup \psi_{j_2}$ on two server nodes P_{j_1} and P_{j_2} through the specific heuristic algorithm is κ'. Then κ' is defined as the corresponding heuristic-based neighbor of κ and all heuristic-based neighbors of κ consist of a solution set as $Ner(\kappa)$.

A HLSA-based neighbor solution of a given solution can be also described as the solution obtained by the given solution by calling the HLSA algorithm once. If $\forall \kappa' \in Ner(\kappa)$, the solution of κ' is not optimal than that of κ, stop the local search and regard κ as a local optimum. In the proposed GHW, we set additional criteria to stop the HLSA search of one individual in one generation, which can be called the number of growth steps (denoted as G_{step}). Then, the heuristic-based local search algorithm with G_{step} criteria can be seen in Algorithm 3-2.

Algorithm 3-2 Heuristic-based local search algorithm

Input : Dimensionality reduced problem $\omega_{(w_l)}$ as Eq. (3-18) and the number of growth step (G_{step})

Output: Solution $A_h\left(\omega_{(w_l)}\right)$ of the problem

Initially Allocate tasks to resources with confirmed or random initialization policy and gain the general initial status of κ
Set $i = 0$, $Exists\_Ner = True$
while Exists_Ner and $i < G_{step}$ **do**
    $Exists\_Ner = False$, $i++$
    Search neighborhood $Ner(\kappa)$ of κ in the specific heuristic-based local search algorithm such as Algorithm 3-3 (modified LSPT-based local search algorithm)
    if $\kappa' \in Ner(\kappa)$ ***s.t.*** *the solution of* κ' *is optimal than* κ **then**
        $Exists\_Ner = True$
        Choose the optimal neighbor to update $\kappa = \kappa'$
Set κ as the solution $A_h\left(\omega_{(w_l)}\right)$ of the problem

Then, an appropriate heuristic algorithm as the search route is the critical factor. Because this framework has great adaptability to various algorithms, almost any algorithm that can solve the resource scheduling of two nodes can be applied as its search route. Without losing generality, we propose the modified LSPT (modified Longest-Shortest Process Time) to search the heuristic-based neighbor considering the performance to address the single-dimensional resource scheduling of heterogeneous nodes. A modified LSPT-based neighborhoods can be defined as: It can be assume κ' is a HLSA-based neighbor of κ, $\psi'_{j_1} \cup \psi'_{j_2} = \psi_{j_1} \cup \psi_{j_2} = \{V_{\tau_1}, V_{\tau_2}, \dots\}$ and $\xi_{\tau_i} \geq \xi_{\tau_{i+1}}$ where $\xi_{\tau_i} = w_{(l)} \cdot (Y_{\tau j_1} - Y_{\tau j_2})$. If $V_{\tau_i} \in \arg\min_{\psi'} \left(\sum_{V_{\tau_k} \in \psi'_1} w_{(l)} \cdot Y_{\tau j_1}, \sum_{V_{\tau_k} \in \psi'_2} w_{(l)} \cdot Y_{\tau j_1}\right)$ where $k < i$ for $\forall V_{\tau_i} \in \psi'_{j_1} \cup \psi'_{j_2}$, then κ' can be called the modified LSPT-based neighbor of κ. While by contraries κ may not be that of κ'. Then, the modified LSPT-based local search algorithm is presented in Algorithm 3-3. In order to minimize the maximum utilization of all heterogeneous nodes, modified LSPT requires sorting the VMs in two nodes according to ξ_{τ_i} (i.e., the difference of the weighted utilization of a VM in two nodes) and $\psi = \{V_{\tau_1}, V_{\tau_2}, \dots\}$ is an ascending set. In the process of putting VMs into nodes one by one: if the sum of weighted utilization of P_{j_1} is smaller than that of P_{j_2}, put the leftmost of the remaining ψ into ψ'_{j_1}; otherwise, put the rightmost of the remaining ψ into ψ'_{j_2}. Then, update ψ by removing the currently selected VM from ψ. The process of modified LSPT combines the characteristics of LPT and SPT so as to well solve the VMs allocation in two heterogeneous nodes.

Algorithm 3-3 Modified LSPT-based local search algorithm for heterogeneous nodes

Input : Tasks in $\psi = \psi_{j_1} \cup \psi_{j_2}$ of two server nodes P_{j_1} and P_{j_2}
Output : ψ'_{j_1} and ψ'_{j_2}

Initialize $Mark_{j_1} = 0$, $Mark_{j_2} = 0$, $\psi'_{j_1} = \emptyset = \psi'_{j_2}$
while $\psi \neq \emptyset$ **do**
  if $Mark_{j_1} \leq Mark_{j_2}$ **then**
    $\alpha = j_1, \beta = j_2$
  else
    $\alpha = j_2, \beta = j_1$
  Collect tasks $V_\tau \in \psi$ **s.t.** $w_{(l)} \cdot (Y_{\tau\alpha} - Y_{\tau\beta}) = \min\limits_{V_i \in \psi} w_{(l)} \cdot (Y_{i\alpha} - Y_{i\beta})$ to obtain a set of $\langle V_{\tau_1}, V_{\tau_2}, \ldots, V_{\tau_s} \rangle$
  if $s \geq 2$ **then**
    Choose V_τ **s.t.** $w_{(l)} \cdot Y_{\tau\alpha} = \max\limits_{1 \leq p \leq s} w_{(l)} \cdot Y_{\tau_p\alpha}$
  $Mark_\alpha += w_{(l)} \cdot Y_{\tau\alpha}$, $\psi'_\alpha += \{V_\tau\}$ and $\psi -= \{V_\tau\}$

3.4.3 Growable Genetic Algorithm based on Growth Strategies

As is known, the local search algorithm may converge to a non-global optimal solution. Thus, we consider using a genetic algorithm to increase the search scope and to improve the probability of jumping out of the local optimum. A feasible combination of local search and genetic algorithm is to use the local search algorithm as the growth route of the individuals of the genetic algorithm (i.e., GGA-HLSA).

Inspired by the existing research, we consider different individuals in the genetic algorithm to grow according to various growth strategies so as to increase the diversity of the population. This strategy can improve the global searchability of the genetic algorithm. The different growth processes of individuals can be regarded as the optimization process of individuals' ability to different weights. Additionally, if the individuals always inherit the ability weights of their parents, they may also fall into the local optimum. Therefore, we apply random multi-weights as the difference in the growth process of individuals. Due to the flexibility of the GGA framework, the solutions of HLSA can be used as mature individuals to participate in subsequent mature stage and genetic stage. Finally, we obtain our proposed GGA-HLSA-RW (GHW). The process of GHW has been shown in Fig. 3-5 and its algorithm is presented in Algorithm 3-4. The loops in Algorithm 3-3 and Algorithm 3-4 can be manipulated with matrices operations so as to utilize GPU for acceleration.

Algorithm 3-4 GGA-HLSA-RW (GHW)

Input : Tasks and VMs V_i and their parameters C_{ijk}, server nodes P_j, the limited capacities L_{jk} and the optimization problems ω

Output: Pareto solution set K

1 Set number of individuals in each generation as N_p, the number of generations as N_g

2 Generate initial individuals $\left\langle \kappa_1^{(0)}, \kappa_2^{(0)}, \ldots, \kappa_{N_p}^{(0)} \right\rangle$ with random genes and obtain initial Pareto solution set K

3 **for** *l in range(N_g)* **do**

4 Generate a random weight vector $\boldsymbol{w}_q^{(l)}$ for each individual in the l-th generation

5 Obtain the problem after dimensionality reduction of each individual in the l-th generation as Algorithm 3-1

6 Using HLSA as Algorithm 3-2 (instantiated by Algorithm 3-3 in chapter) to solve the problem after dimensionality reduction, obtain mature individuals as $\left\langle \bar{\kappa}_1^{(l)}, \bar{\kappa}_2^{(l)}, \ldots, \bar{\kappa}_{N_p}^{(l)} \right\rangle$, and update the solution set K

7 Select and Pair the better mature individuals with specific sort strategies such as non-dominated sorting [151,152], congestion degree sorting [152] and ordering of subproblems corresponding to reference vectors [158]

8 Generate the $l+1$-th infancy individuals $\left\langle \kappa_1^{(l+1)}, \kappa_2^{(l+1)}, \ldots, \kappa_{N_p}^{(l+1)} \right\rangle$ with specific strategies such as elitist strategy

9 Update the Pareto solution set K

3.4.4 Instantiation of GHW: GHW-NSGA II and GHW-MOEA/D

GGA-HLSA-RW (GHW) can be considered as a family of algorithms, which can incorporate various existing genetic optimization strategies to produce specific algorithm variants. This means that the strategies such as NSGA [151], NSGA II [99,152], NSGA III [68], MOEA/D [158], et al. can be applied in GHW to obtain new, well-performing algorithms. Specifically, this chapter applies the NSGA II strategy [99] in GHW to obtain the GGA-HLSA-RW-NSGA II (GHW-NSGA II) algorithm and also applies the MOEA/D strategy [158] to obtain GGA-HLSA-RW-MOEA/D (GHW-MOEA/D) algorithm . Referring to the existing terms of genetic algorithms [152,166], the base components of GHW are defined as:

(1) **Gene and Individual (Chromosome)**: we regard the allocated node index of each VM (or task) as a gene, where the i-th gene can be written as a vector $\lambda_i = \langle x_{i1}, x_{i2}, \ldots, x_{im} \rangle$ where $1 \leq i \leq n$ and $x_{ij} \in \{0,1\}$. If $V_i \in \psi_j$, then $x_{ij} = 1$, otherwise $x_{ij} = 0$. A vector $I = \langle \lambda_1, \lambda_2, \ldots, \lambda_n \rangle$ with n genes constructs an individual (also chromosome) corresponding to a solution of the optimization problem. Obvi-

ously, the vector I and the set of κ are interchangeable.

(2) **Individual selector**: GHW-NSGA II sorts the individuals in the mature stage by non-dominated ordering and congestion ordering, then selects better part of individuals to participate in pairing and crossover. GHW-MOEA/D initialize a group of vectors as reference vectors, selects part of individuals with the best solution under each reference vector and randomly selects two individuals of each reference vector to participate in pairing and crossover.

(3) **Crossover**: It can be assumed two individuals as $I_\alpha = \langle \lambda_{\alpha 1}, \lambda_{\alpha 2}, \ldots, \lambda_{\alpha n} \rangle$ and $I_\beta = \langle \lambda_{\beta 1}, \lambda_{\beta 2}, \ldots, \lambda_{\beta n} \rangle$. Their crossover is defined as separately extracting a part of genes from them to gain a new vector as the children individual, such as $\langle \lambda_{\alpha 1}, \lambda_{\beta 2}, \lambda_{\beta 3}, \ldots, \lambda_{\alpha n} \rangle$. In this chapter, we randomly select the number of genes of I_β participating in crossover according to the uniform distribution, and then randomly select the corresponding number of genes from I_β with the same probability to replace the genes at the corresponding position of I_α.

(4) **Mutation**: Mutation is defined as that replacing some elements of an individual $I_\alpha = \langle \lambda_{\alpha 1}, \lambda_{\alpha 2}, \ldots, \lambda_{\alpha n} \rangle$ by randomly generated genes.

(5) **Population regeneration mechanism**: Both GHW-NSGA II and GHW-MOEA/D apply elitist strategy [152] to combine the parent individuals with their children individuals to jointly compete to produce the next generation.

Unlike MOEA/D whose individual respectively corresponds to a fixed reference line (or a vector), GHW-MOEA/D is only to select the optimal part of mature individuals corresponding to each preset reference line in the individual selector, and randomly select two of them for crossover to generate a new child. This means the individuals in GHW-MOEA/D will no longer be constrained by given fixed reference lines.

In order to intuitively demonstrate the process of GHW family, we take a visualized example of GHW-NSGA II for problem $\min \omega^{(2)}$ with two-dimensional resources (CPU and RAM for the sake of observation). Then, we draw its solution process in Fig. 3-6. In the example of Fig. 3-6, $n = 10$, $m = 4$, $N_p = 10$, $G_{step} = 10$ and the parameters u_{ijk} of VMs and the genes of individuals are randomly initialized.

As shown in Fig. 3-6 which is corresponding to each stage in Fig. 3-5:

(1) **Infancy Stage**: At the 1st generation, the initialized individuals are input as the infancy individuals whose solutions are as Fig. 3-6(a);

(2) **Growth Stage**: The infancy individuals are input in the growth stage and are improved

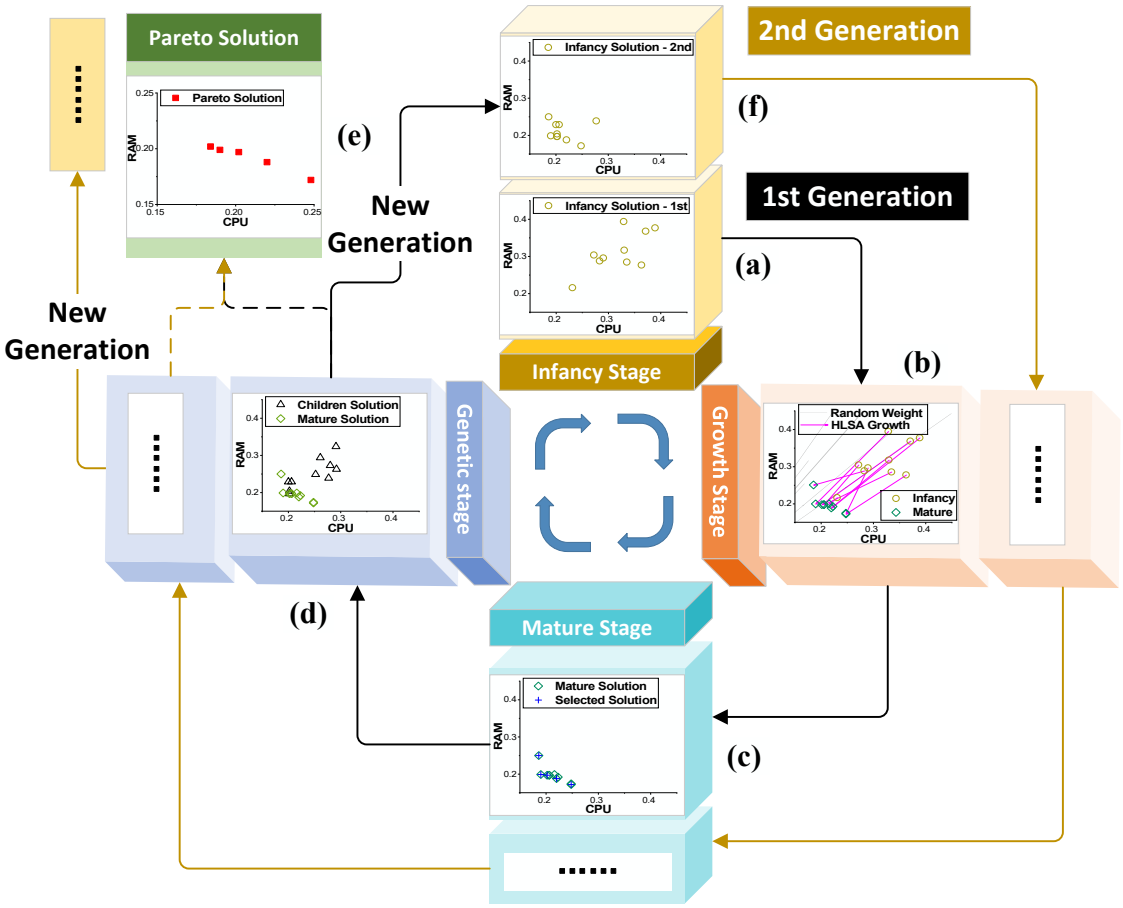


Figure 3-6 The visualized example of GHW-NSGA II with actual results in each stage.

by HLSA growth to generate mature individuals as Fig. 3-6(b). The vectors directed from circles to diamonds represent the growth process with HLSA;

(3) **Mature Stage**: After growth stage, mature individuals are screened by non-dominated sorting and congestion degree sorting to obtain some selected individuals as Fig. 3-6(c) for the subsequent crossover and mutation;

(4) **Genetic Stage**: The selected individuals participate in the crossover and mutation to generate children individuals as Fig. 3-6(d). Then, children individuals and mature individuals participate in screening together to generate the next infancy individuals as 3-6(f). At the same time, the Pareto solutions set is updated as Fig. 3-6(e);

(5) Repeat Infancy Stage → Genetic Stage.

3.5 Theoretical Analysis and Proof

3.5.1 Analysis of Computational Complexity of GHW

The computational complexity (denoted as ζ) of GHW mainly consists of two parts: the computational complexity of HLSA ($\zeta^{(H)}$) and that of genetic algorithm ($\zeta^{(G)}$).

The computational complexity of HLSA for one step search of one individual in one generation can be deduced as $\zeta^{(h)} = \mathrm{O}(mn\log n)$. Thus, for N_p individuals and N_g generations, the computational complexity of HLSA is $\zeta^{(H)} = \mathrm{O}(G_{step} \cdot N_p \cdot N_g \cdot mn\log n)$ where G_{step} is the max number of steps for HLSA search of each individual in each generation. Following, we demonstrate the proof of $\zeta^{(H)}$.

Proof: The computational complexity $\zeta^{(h)}$ corresponds to finding two nodes to execute Algorithm 3-3. The computational complexity of Algorithm 3-3 is $card\left(\psi_{j_1} \cup \psi_{j_2}\right) \log card\left(\psi_{j_1} \cup \psi_{j_2}\right)$ which mainly derives from the complexity of sorting. Thus the total computational complexity of enumerating the combinations of any two nodes is $\sum_{j=1}^{m}\sum_{l=j+1}^{m} card\left(\psi_j \cup \psi_l\right) \log card\left(\psi_j \cup \psi_l\right)$. The second derivative of $xlogx$ is $(xlogx)'' = \frac{1}{x} > 0$ and $\sum_{j=1}^{m} card\left(\psi_j\right) = n$, thus substitution into Jensen inequality can obtain:

$$\zeta^{(h)} \geq \mathrm{O}\left(\frac{m(m-1)}{2}\left(\frac{n}{m}\log\frac{n}{m}\right)\right) = \mathrm{O}(nm\log n) \tag{3-19}$$

In addition, $x_1\log x_1 + x_2\log x_2 < (x_1+x_2)\log(x_1+x_2)$ assuming $x_1, x_2 > 0$. Therefore:

$$\zeta^{(h)} \leq \mathrm{O}((n(m-1)\log n(m-1))) = \mathrm{O}(nm\log n) \tag{3-20}$$

Combining the above equations, $\zeta^{(h)} = \mathrm{O}(nm\log n)$. Thus, $\zeta^{(H)} = \mathrm{O}(G_{step} \cdot N_p \cdot N_g \cdot mn\log n)$. ■

For the computational complexity of the genetic algorithm, if using NSGA II strategies to select mature individuals and to generate the new generations, the computational complexity is $\zeta^{(G)} = \mathrm{O}\left(N_p^3 \cdot N_g \cdot d\right)$ [152]. Then, the computational complexity of GHW-NSGA II can be obtained as:

$$\zeta = \zeta^{(H)} + \zeta^{(G)} = \mathrm{O}\left(N_p \cdot N_g \cdot \left(G_{step} \cdot mn\log n + N_p^2 \cdot d\right)\right). \tag{3-21}$$

GGA-HLSA is essentially a multi-route search algorithm by adding a search route (HLSA) to the genetic algorithm. The local convergence points of various routes are usually different, so multiple routes can help each other to jump out of the local conver-

gence points of other routes so as to improve the optimality of solutions. Generally, the relationship between multiple performances (such as optimality and computational complexity) of the algorithms are varying with the change of parameters, which may not have an explicit expression. However, from the perspective of qualitative analysis based on information theory, GHW receives more effective information from GA-based route and HLSA-based route. This helps GHW spend less time finding better solutions than the existing algorithms such as NSGA II and MOEA/D. Thus, the algorithms of the GHW family theoretically have certain advantages in terms of convergence performance.

Table 3-4 Different Strategies of GGA.

Category	Strategy	Description
Growth Strategies	HLSA growth	Using HLSA as growth route of individuals in GGA
	Random growth	Using randomization as growth route of individuals
	Non growth	Direct genetic without growth
Dimension Reduction	RW	Random multi-weights for each individual
	EW	Enumeration weights for each individual
	RD	Random dimensions for each individual
	RRD	Round-robin dimensions for each individual
	TRD	Taboo round-robin dimensions for each individual
	RWS	Random dimensionality reduction strategies for each individual

3.6 Experimental Results and Analysis

3.6.1 Experiments Setting

For the sake of the comprehensive evaluations to the proposed algorithms, this section carries out four groups of experiments from various aspects including:

(1) EX_1: comparison of growth strategies for GGA;

(2) EX_2: comparison of dimensionality reduction strategies for GGA-HLSA;

(3) EX_3: evaluation of practicability on a real public trace-driven dataset (AzureTracefor-Packing2020 [131]);

(4) EX_4: verification on the advantages of proposed algorithms in convergence and optimality by comparing with state-of-the-art.

All these experiments are based on the control variable method. Because this chapter mainly focuses on discussing the influence of adding a directed growth in the GGA in-

stead of that of the type of algorithms for the growth stage, we select modified LSPT-based HLSA as the growth route of GGA. Fixing the algorithm process as GGA and regarding the growth strategy as the variable, EX_1 presents the performance of adding HLSA growth, random growth (non-directed growth) and no growth respectively as shown in Table. 3-4. Except for random multi-weights, other strategies for dimensionality reduction include Enumeration Weights (EW), Random Dimensions (RD), Round-Robin Dimensions (RRD), Taboo Round-robin Dimensions (TRD) and Random dimensionality and Weights Strategies (RWS) as shown in Table. 3-4. For example, Random Dimensions (RD) means randomly choosing one dimension to generate the single-dimensional optimization problem of each individual in each generation and using HLSA to solve the problem of the chosen dimension. Other compared strategies for dimensionality reduction are similar to RD and RW. EX_2 fixes the algorithm process as GGA-HLSA and regards the dimensionality reduction strategy as the variable. To validate the applicability of our proposed algorithms in realistic, EX_3 is a group of trace-driven experiments based on AzureTraceforPacking2020 [131]. EX_4 presents some indicators over time to validate the superiority of our proposed algorithms of GHW family compared with the state-of-the-art.

With the exception of EX_3, which is performed on the trace-based dataset, other groups of experiments are executed on the random simulation dataset, which is conducive to generating adequate data for comprehensive verification. The descriptions and operations of the datasets used in experiments are as follows.

(1) Random Simulation Dataset: is generated randomly by a Python-based simulation environment. In the simulation environment, we set up the server nodes to be heterogeneous and the parameters of VMs obey a uniform distribution.

(2) AzureTraceforPacking2020 [131]: represents part of the workload on Microsoft's Azure Compute. AzureTraceforPacking2020 provides the required CUP, RAM, SSD (Solid-State Drive) and NIC (Network Bandwidth) allocation for 487 types of VMs on 35 types of machines. Since AzureTraceforPacking2020 provides the utilization of MDRs by VMs on heterogeneous machines, it is very suitable to verify the performance of the proposed methods. If a certain dimension of a VM occupies too much utilization on all types of machines, it will directly cause this VM to occupy one machine alone, which has no impact on the optimization of other VMs and machines. Thus we choose to ignore some types of VMs with large utilization. In the experiments of this chapter, we screen out some types of VMs with a minimum resource

utilization of CPU, RAM and SSD greater than 0.3 on all types of machines, and finally retain 338 types of VMs, which means $\min_{j=1}^{m} u_{ijk} \leq 30\%$ for $\forall i, k$. Then, we randomly select the given numbers of VMs and machines from the 338 types of VMs and 35 types of machines.

Table 3-5 Setup of Experiments.

Groups	Comparison	Dataset	Objectives	Scenarios (servers, VMs)
EX_1	Growth Strategies	Simulation Dataset	$\min \omega^{(2)}$	(8, 80)
			$\min \omega^{(4)}$	(16, 20-200), (20, 20-200)
EX_2	Dimensionality Reduction	Simulation Dataset	$\min \omega^{(2)}$	(20, 200)
			$\min \omega^{(4)}$	(16, 20-200), (20, 20-200)
EX_3	Generations	AzureTracefor Packing2020	$\min \omega^{(2)}$	(400, 1000), (700, 2000)
			$\min \omega^{(4)}$	(400, 1000), (700, 2000)
EX_4	SOTA: NSGA II MOEA/D	Simulation Dataset	$\min \omega^{(2)}$	Large Scale
				Small Scale

EX_1 to EX_3 are executed for two problems, i.e. $\min \omega^{(2)}$ (minimizing the maximum utilization rate of resources for each dimension under all nodes) and $\min \omega^{(4)}$ (minimizing energy consumption for total cloud system), while EX_4 is performed only for the $\min \omega^{(2)}$ problem. For the sake of presenting the Pareto solution in the same figure simultaneously, we choose three dimensions of resources, that is CPU, RAM and DS, to execute the experiments. Since the results from various setups of clouds allowed us to draw the same conclusions, we only present results for parts of them corresponding to specific scenarios. The experimental setup is shown in Table. 3-5.

We use the Mxnet framework to enable the program to run in GPU. Then, the experiments are launched on a GPU desktop computer with configurations as:

- CPU: Intel(R) Core(TM) i5-8400 CPU @ 2.8GHZ;
- SSD: KINGSTON SA400S37 240GB;
- GPU: NVIDIA GeForce GTX 1060 6GB;
- Program version: Python 3.6 + mxnet-cu90 1.5.0.

3.6.2 EX_1: Comparison of the Growth Strategies for GGA

EX_1 is carried out to evaluate the performance of different growth strategies in the genetic algorithm including non-growth, random growth and HLSA growth shown as Table. 3-4. In EX_1, we set $N_g = 100$, $N_p = 50$, mutation rate is 0.2, $G_{step} = 10$ as well as the individual selector and regeneration mechanism of Fig. 3-5 are conducted with random

screening with equal probability rather than using survival of the fittest.

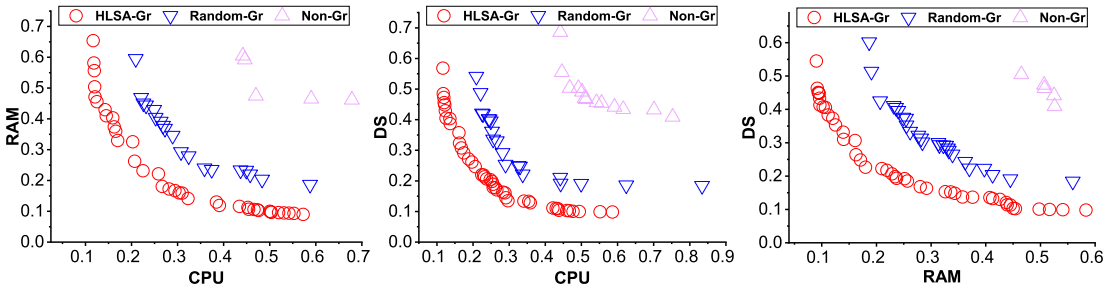
3.6.2.1 Minimizing the Maximum Utilization of Resources

For the problem of $\min \omega^{(2)}$ (i.e., minimizing the maximum utilization rate of resources for each dimension under all nodes), we present the experiment results under the scenarios with 8 nodes and 80 VMs. And the utilization of VMs obeys the uniform distribution:

$$u_{ijk} \sim \mathrm{U}(0, 12)\,\% \tag{3-22}$$

where $\mathrm{U}(0, 12)$ means the uniform distribution in $[0, 12]$ generated by calling the function of mxnet as mxnet.nd.random.randint(low=0, high=121, shape=(n, d, m))/1000.

For sake of observation, Fig. 3-7 plots the Pareto solutions of two-dimensional resources (the boundary of projection on each coordinate plane of 3D Pareto solution) for 8 server nodes with 80 VMs.



(a) CPU and RAM (b) CPU and DS (c) RAM and DS

Figure 3-7 2D Pareto solution of minimizing the maximum utilization of each dimensional resources under non-growth, random growth and HLSA growth strategies of GGA with random crossover and regeneration for 8 server nodes and 80 VMs.

In Fig. 3-7, the Pareto set of HLSA growth strategy outperforms random growth and non-growth. That means for each solution of non-growth and random growth strategies, there is still at least one solution of HLSA growth strategy dominating it. The comparison between random-growth and non-growth illustrates that the GGA can optimize the result of the non-growth genetic algorithm although only using randomization as the growth route. The comparison between HLSA-growth and random growth illustrates that directional growth (i.e., using the HLSA as the growth route of the GA) can obtain better Pareto solutions than that of the random-growth route. Further, the HLSA-growth can eliminate

the instability of the random growth strategy.

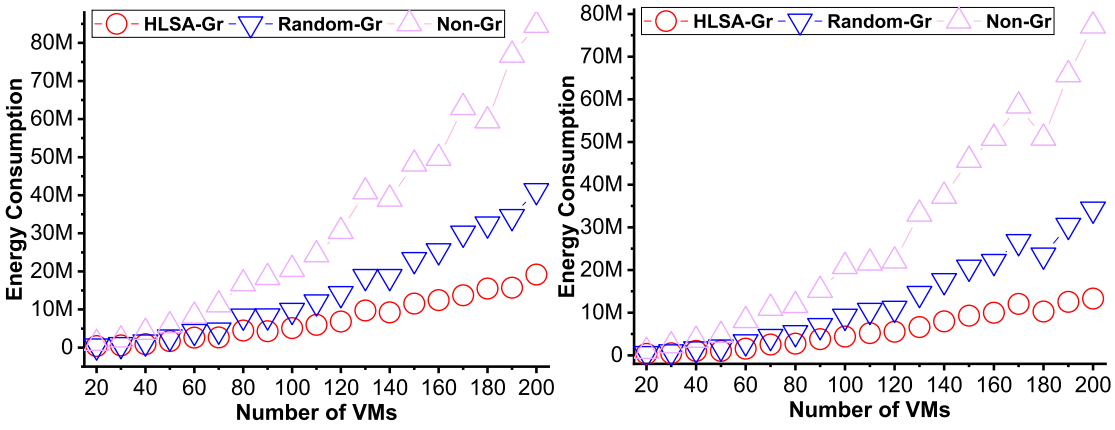
3.6.2.2 Minimizing Energy Consumption

Next, we carry out the experiments in the scenarios of $\min \omega^{(4)}$ (i.e., minimizing energy consumption for the total cloud system) with 16 and 20 server nodes respectively and set the numbers of VMs from 20 to 200 where the capacities requested of each VMs are $C_{ijk} \sim \mathrm{U}(75, 150)$. In the simulated experiments, we randomly generate the coefficients of Eq. (3-11) as integers according to uniform distribution that:

$$\begin{cases} a_{jk} \sim \mathrm{U}(1, 10), b_{jk} \sim \mathrm{U}(0, 100), \\ c_{jk} \sim \mathrm{U}(100, 200), d_{jk} \sim \mathrm{U}(500, 1000). \end{cases} \tag{3-23}$$

Then, we plot the results of each growth strategy in Fig. 3-8.

Fig. 3-8(a) plots the minimum energy consumption from three strategies for 16 server nodes and Fig. 3-8(b) plots that for 20 server nodes of the 100-th generations. From Fig. 3-8, the HLSA-Growth strategy achieves the lowest energy consumptions compared with random-growth and non-growth for all the sets of (n, m). In addition, random-growth achieves lower energy consumption than non-growth. The sorting of the performance of these three strategies in Fig. 3-8(b) demonstrates GGA outperforms non-growable one and the directional growth outperforms the random growth. For all experimental combinations, HLSA reduces energy consumption on average by 95.21% and 333.7% than random-growth and non-growth respectively.



(a) 16 server nodes (b) 20 server nodes

Figure 3-8 Energy consumption under non-growth, random growth and HLSA growth strategies of GGA with random crossover and regeneration.

Overall, EX_1 verifies that adding a growth stage in the process of the genetic algo-

rithm solving MDRSPs can significantly improve solutions. This may be because GGA actually applies a multi-route, which usually has better local optimal solutions than the single-route search.

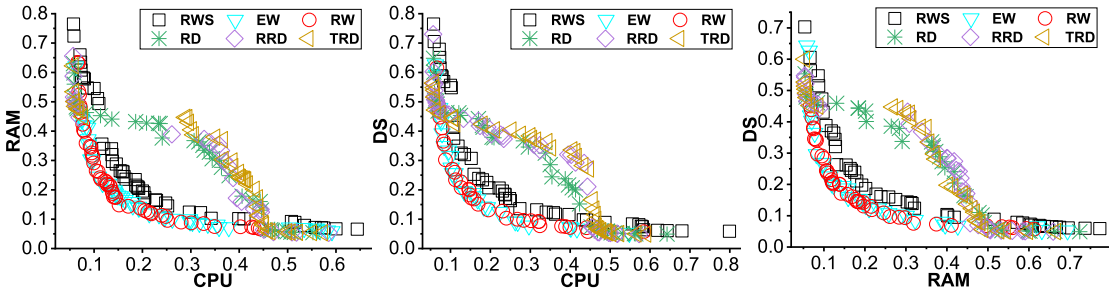
3.6.3 EX_2: Comparison of Dimensionality Reduction Strategies for GGA-HLSA

To evaluate the performance of different dimensionality reduction strategies, we carry out experiments respectively using: RW, EW, RD, RRD, TRD and RWS as shown in Table3-4. Same as EX_1, we set $N_g = 100$, $N_p = 50$, mutation rate is 0.2, $G_{step} = 10$, as well as the individual selector and regeneration mechanism, are also conducted with random screening with equal probability.

3.6.3.1 Minimizing the Maximum Utilization of Resources

For the problem of $\min \omega^{(2)}$, we only plot the experiment results under the scenarios with 20 server nodes and 200 VMs, as similar conclusions can also be obtained under other combinations of (n, m). The parameters of VMs also obey Eq. (3-22). After 100 generations, the Pareto solution sets of each dimensionality reduction strategy are plotted in Fig. 3-9. In each subfigure of Fig. 3-9, the Pareto solutions of RWS, EW and RW are obviously better than that of RD, RRD and TRD, which shows that the strategy only by switching a single-dimension is far from finding a better solution hence weighted solutions are necessary. For RWS, EW and RW with relatively close Pareto solutions, the solutions of EW and RW are better than that of RWS, which demonstrates RWS has more uncertainty resulting worse solution set. The Pareto boundaries of EW and RW almost coincide, which demonstrates that RW is sufficient to search the Pareto solutions set. The computational complexities of EW are too large because each generation in the genetic algorithm needs to enumerate as many weights as possible. However, the Pareto solutions corresponding to different weights may be the same, so EW costs extra computation, but may miss some solutions as it can't really enumerate all weights.

To more quantitatively evaluate the performance of EW and RW, we combine the three-dimensional Pareto solution sets of EW and RW for 20 server nodes and 200 VMs into a new Pareto solution set to calculate the proportion of EW and RW in the new solution set. The new Pareto solutions set has 621 solutions in total where 474 solutions originate from RW and 149 solutions from EW. That means RW solutions account for 76.08% and EW for 23.92% in the combined Pareto solutions set. These results illustrate that the RW



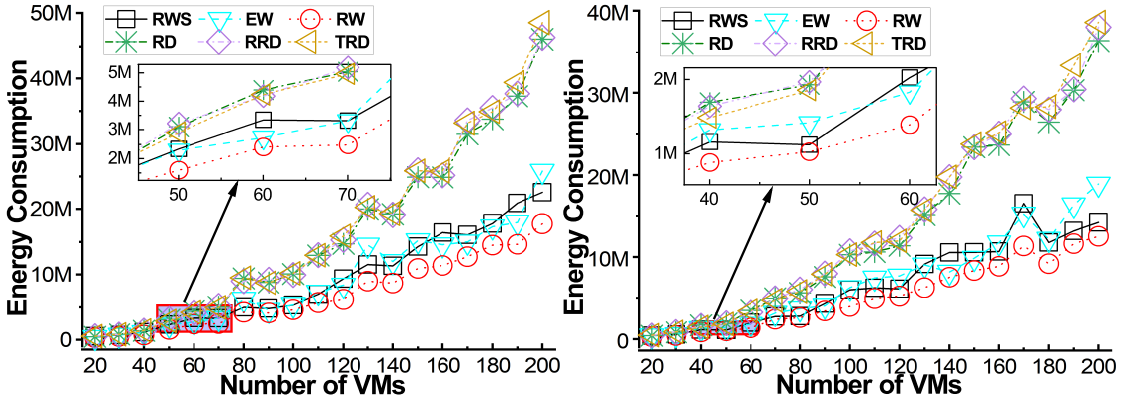
(a) CPU and RAM (b) CPU and DS (c) RAM and DS

Figure 3-9 2D Pareto solution using different dimensionality reduction strategies of GGA-HLSA with random crossover and regeneration for 20 server nodes and 200 VMs.

strategy has a higher probability of obtaining a better Pareto solution set than EW. It means the usage of RW can not only reduce the computational complexity but also improve the Pareto solution set.

3.6.3.2 Minimizing Energy Consumption

To further verify the performance of different dimensionality reduction strategies, we carry out the experiments in the scenarios of minimizing energy consumption ($\min \omega^{(4)}$) with 16 and 20 server nodes respectively. We set the numbers of VMs from 20 to 200 where the capacities requested of each VMs are $C_{ijk} \sim \mathrm{U}(75, 150)$. Then, the minimum energy consumptions of each dimensionality reduction strategy are plotted in Fig. 3-10.



(a) 16 server nodes (b) 20 server nodes

Figure 3-10 Energy consumption using different dimensionality reduction strategies of GGA-HLSA with random crossover and regeneration.

In Fig. 3-10, EW, RW and RWS obtain far lower energy consumptions than RD,

RRD and TRD, which shows the strategies on weights are still better than that on a single resource dimension in the MDRSP of minimizing energy consumption. RW achieves the best performance among all the strategies, followed by RWS and EW. Although EW has better Pareto solutions than RWS in $\min \omega^{(2)}$, EW has larger energy consumption than RWS in $\min \omega^{(4)}$. This is because the objective of $\min \omega^{(4)}$ has nonlinear terms as Eq. (3-11), which makes balancing the utilization of dimensional resources not equal to the minimum energy consumption. This phenomenon shows again EW can not really enumerate all weight combinations. Overall, RW reduces the energy consumption on average by 29.63%, 35.30%, 122.8%, 129.3%, and 129.2% respectively than RWS, EW, RD, RRD and TRD.

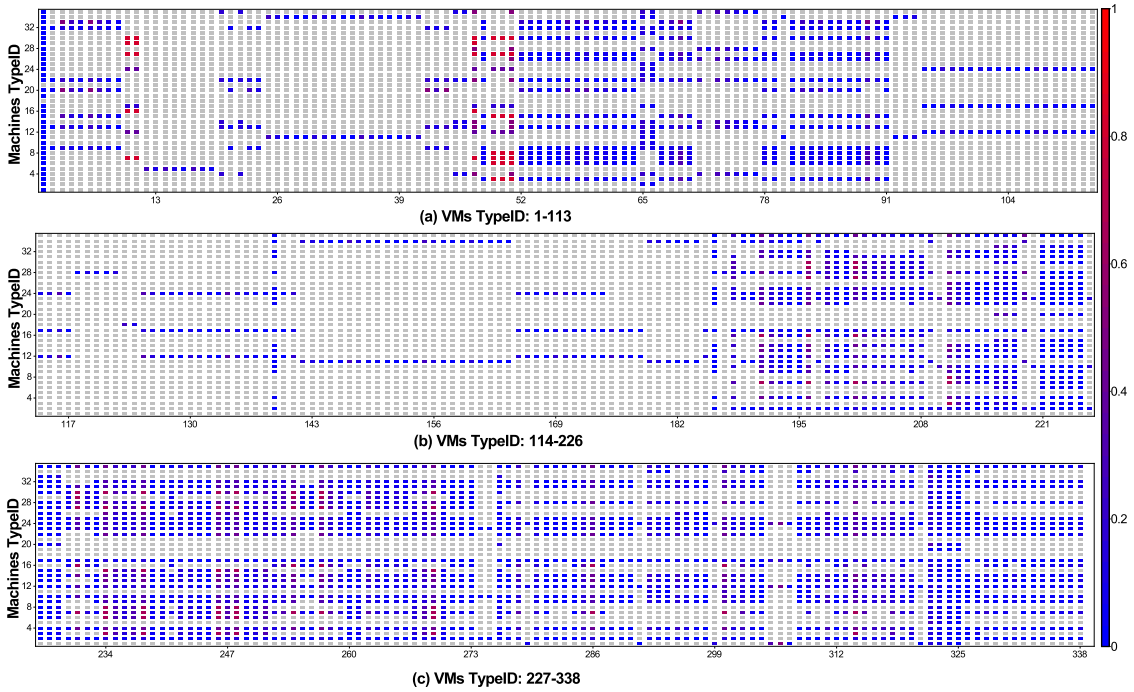


Figure 3-11 Heat-map of the CPU utilization required by our selected 338 types of VMs on 35 types of machines, where the gray represents the VM of the specified type can not run on the corresponding machine.

The results of EX_2 can conclude that using random weight in GGA-HLSA can gain better solutions than other dimensionality reduction strategies through the same generations for MDRSPs.

Firstly, when using dimension reduction strategies based on resource dimensions, including random dimensions, round-robin dimensions, and taboo round-robin dimensions, GGA-HLSA evolves each individual in only one dimension at a time. In fact, RD, RRD

and TRD are unable to balance the utilization rates of resources in multiple dimensions. Thus, they can not obtain good solutions with the same optimization level as RW when multiple dimensions all have a certain weight proportion. Consequently, in Fig. 3-9, the Pareto solutions of RD, RRD, and TRD in the middle are far worse than that of RW. The same reason works in the experiments of Fig. 3-10. That is, the optimization of resources in a single-dimension often cannot reduce the total energy consumption of the cloud system.

Then, RW, EW and RWS in GGA-HLSA can obtain a close solution set, but that of EW and RWS are worse than RW. This may be because: in RWS, poor strategies have shared the running time; and in EW, some close weights actually get repeated solutions. Both RWS and EW make GGA-HLSA not fully utilized to obtain more information in the process of searching for solution space. Similarly, they failed to obtain the solution with lower energy consumption than RW.

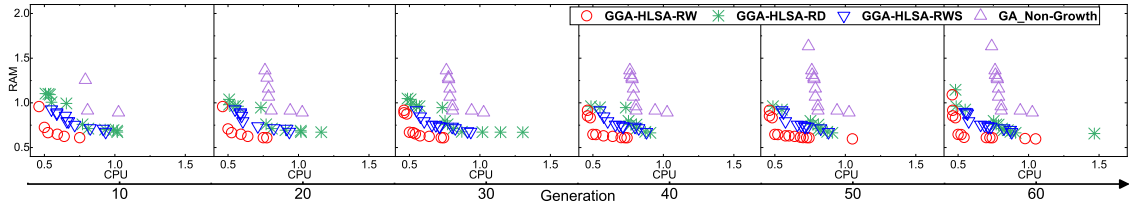
3.6.4 EX_3: Evaluation of Practicability on Azure Trace

To further evaluate the practicality and the convergence of our proposed GHW in solving MDRSP, we carry out a group of trace-driven experiments EX_3 based on Azure-TraceforPacking2020 [131]. As mentioned above, we select 338 types of VMs with $\min_{j=1}^{m} u_{ijk} \leq 30\%$ for $\forall i, k$, and renumber these types of VMs. Then we can draw a heat-map of the CPU utilization required by our selected 338 types of VMs on 35 types of machines as shown in Fig. 3-11. The heat maps of their RAM and SSD are similar to that of the CPU.

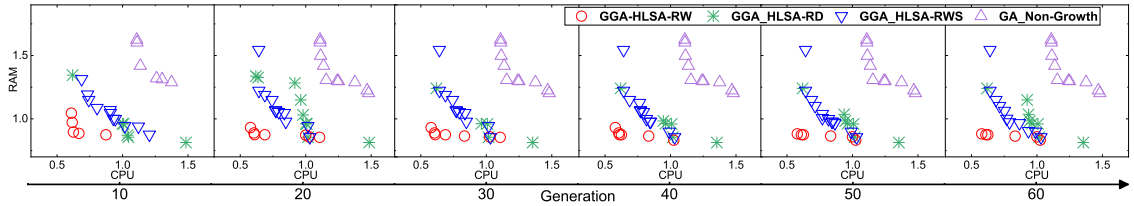
In EX_3, we set each generation to have 20 individuals to observe the pipeline of Pareto solutions in $\min \omega^{(2)}$ and that of energy consumptions in $\min \omega^{(4)}$ within 60 generations. We also set the mutation rate is 0.2, $G_{step} = 10$, as well as the individual selector strategies and regeneration mechanism of Fig. 3-5 are based on NSGA II. We conducted extensive experiments under different numbers of VMs and machines. Since each experiment can lead to the same qualitative conclusion, we only present the results under two sets of parameters (400 machines, 1000 VMs) and (700 machines, 2000 VMs).

Using GGA-HLSA-RD-NSGA II (GHD-NSGA II), GGA-HLSA-RWS-NSGA II (GHWS-NSGA II) and NSGA II as baselines, the pipelines of Pareto solution sets of $\min \omega^{(2)}$ on the plane of (CPU, RAM) are plotted in Fig. 3-12. Fig. 3-12(a) is for 1000 VMs and 400 machines, as well as Fig. 3-12(b) for 2000 VMs and 700 machines. Because

most types of VMs in AzureTraceforPacking2020 [131] cannot be allocated on many types of machines, the Pareto solution sets of the trace-driven experiments are not as regular as that on the simulation dataset (EX_1 and EX_2) which approximately form the continuous curves. However, both in Fig. 3-12(a) and Fig. 3-12(b), GHW-NSGA II not only still obtains better Pareto solution set at the 60-th generation, but also achieves convergence faster in min $\omega^{(2)}$ than baselines.



(a) 1000 VMs and 400 machines



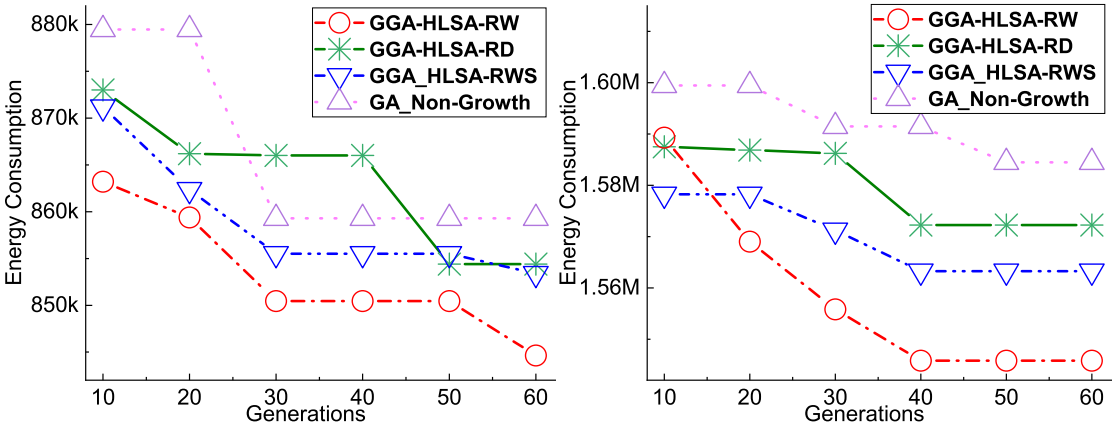
(b) 2000 VMs and 700 machines

Figure 3-12 Pipeline of Pareto solution sets within 60 Generations for min $\omega^{(2)}$ of Azure Trace.

To continually evaluate the performance of GHW-NSGA II in other objectives, we plot the pipeline of energy consumption within 60 generations of the problem min $\omega^{(4)}$ in Fig. 3-13. For this problem, we also assume energy consumption satisfies Eq. (3-24) by replacing C (capacity) with u (utilization) and the coefficients also satisfy Eq. (3-23) considering AzureTraceforPacking2020 [131] didn't provide energy consumption data of resources.

$$E = \sum_{j=1}^{m}\sum_{k=1}^{d}\left(a_{jk}\left(\sum_{i=1}^{n}x_{ij}u_{ijk}\right)^2 + b_{jk}\left(\sum_{i=1}^{n}x_{ij}u_{ijk}\right)\right) + \sum_{j=1}^{m}\sum_{k=1}^{d}(c_{jk}) + \sum_{j=1}^{m}\left(\max_{i=1}^{n}(x_{ij})\sum_{k=1}^{d}d_{jk}\right) \tag{3-24}$$

As shown in Fig. 3-13, GHW-NSGA II obtains lower solutions of energy consumption than the baselines since the first generation. In the iteration of 60 generations, the curve of GHW-NSGA II is still the lowest compared with the baselines. At the end of the



(a) 1000 VMs and 400 Machines (b) 2000 VMs and 700 Machines

Figure 3-13 Pipeline of Energy Consumption within 10 Generations for min $\omega^{(4)}$ of Azure Trace.

60-th generation for the two scenarios, the energy consumptions of GHW-NSGA II are $(844633, 1545821)$, which are respectively reduced by $(1.14, 1.68)\%$, $(1.03, 1.11)\%$ and $(1.71, 2.43)\%$ respectively compared with the baselines GHD-NSGA II, GHWS-NSGA II and NSGA II.

EX_3 on AzureTraceforPacking2020 verifies our proposed GGA-HLSA-RW not only has fast convergence but also applies to the MDRSP in realistic cloud computing.

3.6.5 EX_4: Comparison with the State-of-the-art

To further evaluate the advantages of our proposed algorithms, we execute a group of experiments EX_4 to compare the proposed GHW family with the state-of-the-art in the terms of convergence and optimality. The state-of-the-art participating in comparison are NSGA II and MOEA/D which are two well-performed and frequent baselines in MOPs. For the algorithms of the GHW family, we verify three algorithms including GHW-RCE (GGA-HLSA-RW with Random Crossover and rEgeneration mechanism), GHW-NSGA II and GHW-MOEA/D.

The previous subsections have fully verified the practicability of our proposed algorithms in both the simulation dataset and public trace-driven dataset for both the problems of min $\omega^{(2)}$ and min $\omega^{(4)}$. Therefore, we only evaluate their performance in the simulation dataset for the problem of min $\omega^{(2)}$ (minimizing the maximum utilization rate of resources for each dimension under all nodes) in EX_4, which does not lose generality.

In addition, we select different parameters from those in the EX_1, EX_2 and EX_3 to

increase the diversity of experimental results. The distribution of the utilization is:

$$u_{ijk} \sim \mathrm{U}(5, 12)\,\%, \tag{3-25}$$

generated by mxnet.nd.random.randint(low=50, high=121, shape=(n, d, m))/1000.

For the sake of quantitative analysis of convergence, we apply HVs-over-time [153,154] as the evaluation indicator. Considering to observe the HyperVolume (HV) of each algorithm into the range $[0, 1]$, the experiments call the function **pymoo.indicators.hv.Hypervolume** [167] with the following settings:

(1) ref_points = $(1, 1, 1)$,

(2) norm_ref_point = False,

(3) zero_to_one = True,

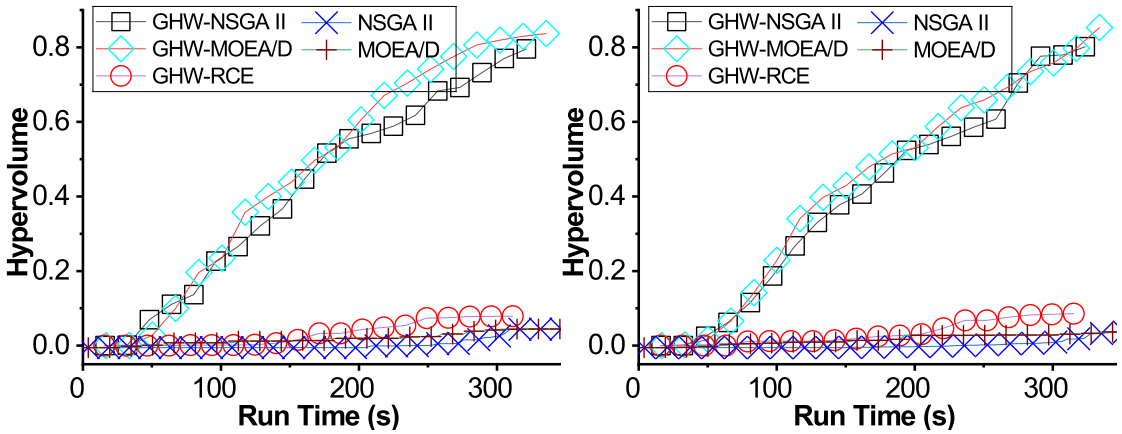
(4) ideal = $\left(\min_j U_{j1}, \min_j U_{j2}, \min_j U_{j3}\right)$,

(5) nadir = $\left(\max_j U_{j1}, \max_j U_{j2}, \max_j U_{j3}\right)$.

According to the calculation time of each algorithm, we convert the generation number into the corresponding running time to obtain the HVs-over-time of each algorithm.

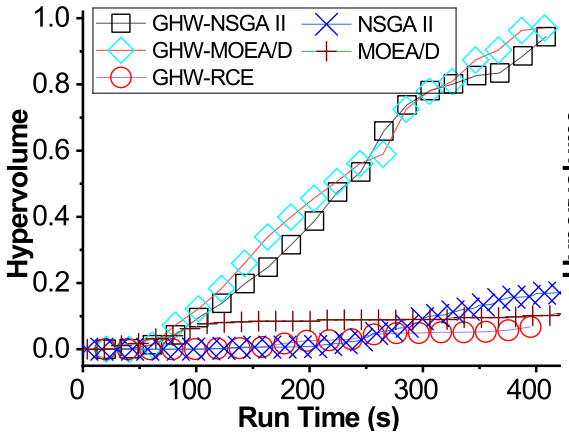
We present four groups of experiments respectively as $(n = 100, m = 40)$, $(n = 200, m = 40)$, $(n = 500, m = 100)$ and $(n = 1000, m = 100)$, which are sufficient to illustrate the convergence of the algorithms. Since GHW adds a growth stage in each generation, the time consumption of each generation will be longer than that of NSGA II and MOEA/D. In order to balance the running time of algorithms so as to facilitate comparison at the approximate time point, we set the $N_g = 20, N_p = 100$ for the algorithms of GHW family, as well as $N_g = 400, N_p = 500$ for NSGA II and MOEA/D. The mutation rate is set as 0.2 and $G_{step} = 10$. Then, we plot the HVs-over-time of each algorithm in Fig. 3-14.

Comparing GHW-RCE with NSGA II and MOEA/D in Fig. 3-14(a) and Fig. 3-14(b), GHW-RCE reaches higher HVs than NSGA II and MOEA/D in each same time point. Concretely in Fig. 3-14(a), the HV of GHW-RCE at the 20-th generation (corresponding to 311.41s) is 0.079. Correspondingly, NSGA II and MOEA/D at the 90-th generation obtain lower HVs both as 0.056. In Fig. 3-14(c) and Fig. 3-14(d) with larger scale of VMs and server nodes, NSGA II and MOEA/D obtain higher HVs than GHW-RCE. Changes in comparison results from Fig. 3-14(a) to Fig. 3-14(d) show that the performance of GHW-RCE degrades faster than that of NSGA II and MOEA/D with the increasing number of VMs and nodes. This is because random crossover and population regeneration counter-

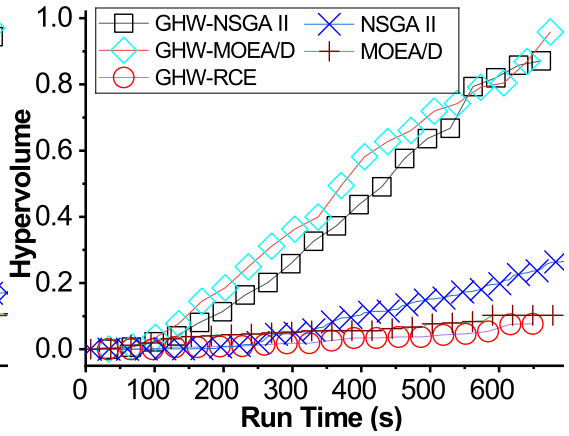


(a) 100 VMs and 40 Machines.

(b) 200 VMs and 40 Machines.



(c) 500 VMs and 100 Machines.



(d) 1000 VMs and 200 Machines.

Figure 3-14 HVs-over-time of proposed GHW family, NSGA II and MOEA/D for the problem $\min \omega^{(2)}$ in simulation dataset.

productively make GHW-RCE fail to inherit the excellent solutions found before. In fact, GHW plays the role of optimizing solutions, as well as random crossover and regeneration play the role of degrading solutions. Even so, GHW-RCE still maintains advantages to a certain extent in Fig. 3-14(a) and Fig. 3-14(b), which reversely demonstrates the powerful advantages of GHW.

Comparing GHW-NSGA II and GHW-MOEA/D with NSGA II and MOEA/D in Fig. 3-14, the results obviously and solidly demonstrate the superiorities of GHW family. In Fig. 3-14, the HVs of GHW-NSGA II and GHW-MOEA/D are far higher than that of compared algorithms. Concretely in Fig. 3-14(a), GHW-NSGA II and GHW-MOEA/D only take 48.20s and 67.16s to obtain the HVs as 0.070 and 0.099 respectively, which are higher than that of NSGA II and MOEA/D got in 320s. This illustrates GHW-NSGA II and GHW-MOEA/D have much faster convergence rates. At 11-th generation, GHW-NSGA II

(at 176.73s) and GHW-MOEA/D (at 184.67s) achieve 0.516 and 0.532 HV respectively more than 0.5. And at 20-th generation, the HVs of GHW-NSGA II (at 321.33s) and GHW-MOEA/D (at 335.78s) achieve 0.796 and 0.837 respectively, which are both ten times more than that of NSGA II and MOEA/D. The other figures in Fig. 3-14 also have similar phenomena. With the increase in the number of VMs and nodes, GHW-NSGA II and GHW-MOEA/D still maintain obvious advantages steadily.

Table 3-6 The HVs and corresponding time compared the algorithms of GHW family with state-of-the-art.

Algorithms	Final HV					When one achieves HV ≥ 0.5		
	Gen.	Time	HV	ε_1	ε_2	Gen.	Time	HV
Fig. 3-14(a): $(n,m) = (100,40)$								
NSGA II	90	335.3	0.056	48.2	67.1	50	186.3	0.000
MOEA/D	90	349.4	0.056	48.2	67.1	45	174.7	0.021
GHW-RCE	20	311.4	0.079	64.2	67.1	11	171.2	0.033
GHW-NSGA II	20	321.3	**0.796**	-	-	11	176.7	**0.516**
GHW-MOEA/D	20	335.7	**0.837**	-	-	11	184.6	**0.532**
Fig. 3-14(b): $(n,m) = (200,40)$								
NSGA II	80	331.2	0.043	64.7	66.7	43	178.0	0.002
MOEA/D	80	308.6	0.041	64.7	66.7	45	173.5	0.022
GHW-RCE	20	315.2	0.086	80.8	83.4	11	173.3	0.025
GHW-NSGA II	20	323.5	**0.802**	-	-	11	177.9	**0.463**
GHW-MOEA/D	20	333.8	**0.853**	-	-	11	183.6	**0.514**
Fig. 3-14(c): $(n,m) = (500,100)$								
NSGA II	120	403.8	0.169	143.0	122.2	66	222.1	0.017
MOEA/D	120	349.4	0.119	122.5	101.8	60	226.8	0.089
GHW-RCE	20	395.0	0.068	102.1	81.4	11	217.2	0.027
GHW-NSGA II	20	408.6	**0.945**	-	-	11	224.7	**0.475**
GHW-MOEA/D	20	407.3	**0.970**	-	-	11	224.0	**0.506**
Fig. 3-14(d): $(n,m) = (1000,200)$								
NSGA II	200	674.3	0.357	363.6	303.4	120	404.6	0.296
MOEA/D	200	727.9	0.151	231.4	202.2	110	400.3	0.114
GHW-RCE	20	648.6	0.077	165.3	134.8	12	389.2	0.034
GHW-NSGA II	20	661.2	**0.871**	-	-	12	396.7	**0.437**
GHW-MOEA/D	20	674.2	**0.959**	-	-	12	404.5	**0.581**

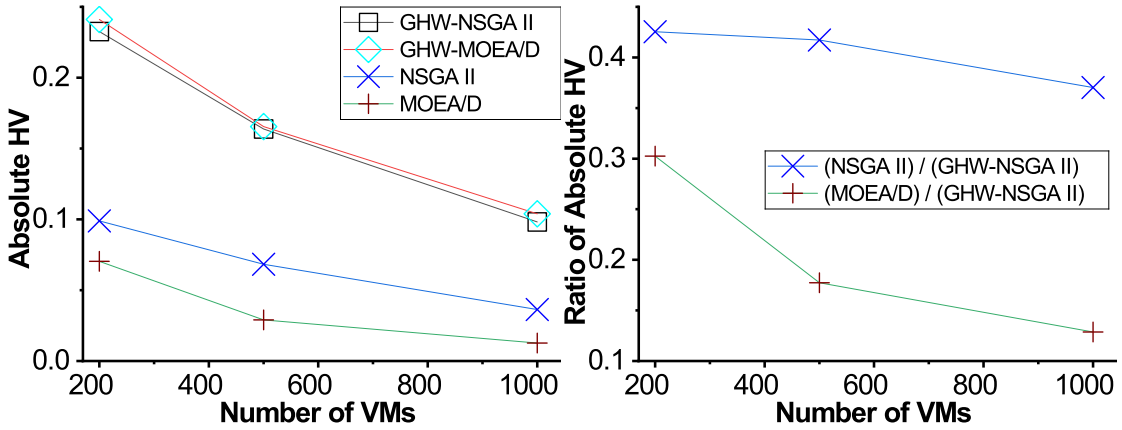
Note: ε_1 (s) and ε_2 (s) respectively express the time when GHW-NSGA II and GHW-MOEA/D achieve corresponding HV.

For quantitative description, we list the values of HVs and the corresponding time of each group of experiments in Table. 3-6. The quantitative comparison of Table. 3-6 shows that the HVs of our proposed algorithms keep higher than compared algorithms.

Moreover, our proposed GHW-NSGA II and GHW-MOEA/D reduce more than 200 seconds to achieve the approximate HV that the comparison algorithms achieve at the end of each figure in Fig. 3-14. And, our proposed algorithms only spend about 11 generations to achieve the HVs close to 0.5 in each scenario of Fig. 3-14. With the expansion of the scale, the HVs of all algorithms show an increasing trend continuously. This is because we normalize the solutions to $[0, 1]$ by setting zero_to_one = True when calculating the HV, thus raising the value of the comparison algorithm. This also leads to the narrowing of the gap between our algorithm and the comparison algorithm. Different settings when computing HV will cause different numerical trends, but their qualitative results will not change. Thus, the results of Table. 3-6 can at least intuitively prove our proposed algorithms are significantly better than compared algorithms.

In fact, the advantages of our algorithm are expanding with the increase of the scale of (n, m). To verify it, we compute the absolute HV of the three scales of VMs and nodes $(200, 40)$, $(500, 100)$, $(1000, 200)$ by setting zero_to_one=False. Then, we plot the absolute HVs in Fig. 3-15(a) and the ratio of absolute HV between compared algorithms and GHW-NSGA II in Fig. 3-15(b). In Fig. 3-15, the numbers of VMs are all 5 times of nodes.Under the same proportion between VMs and nodes, large scale has more feasible solutions and smaller scheduling granularity. Thus, the theoretical optimal absolute HVs will increase with scale. However, in Fig. 3-15(a), the absolute HVs of all algorithms decrease with the increase of scale. This is because the search space of the solution set of the NP-hard problem increases exponentially with the increase of scale, so that the optimality of the algorithms' solution set will decline in the exponential increase of search time. To measure the decline amplitude of different algorithms, Fig. 3-15(b) computes the ratio of absolute HVs between compared algorithms and GHW-NSGA II under each combination of (n, m). In Fig. 3-15(b), the ratios are decreasing with the increase of VMs, which means the optimality of compared algorithms decreases more than that of GHW-NSGA II. This may be because the searchability of NSGA II and MOEA/D can not keep up with the exponential increase of solution space with the increase of VMs and nodes.

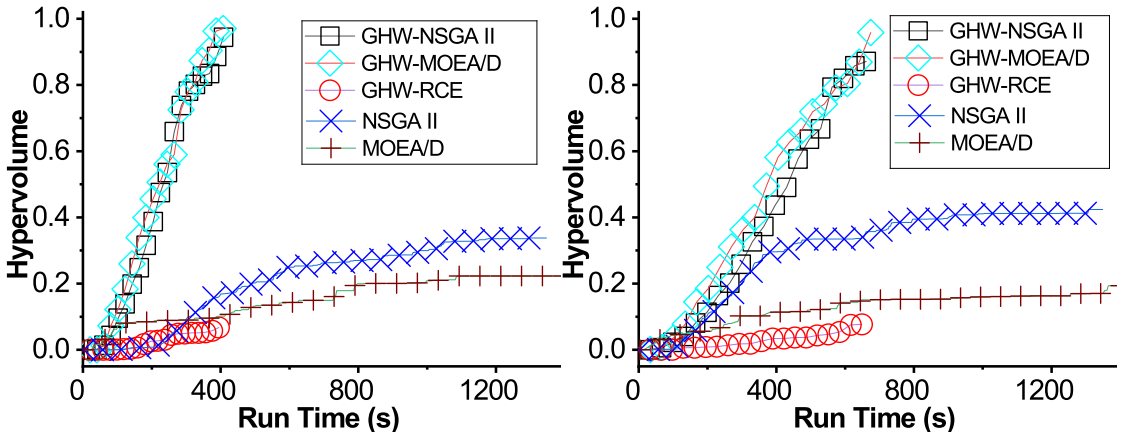
Additionally Fig. 3-16 plots the results with a larger time range of Fig. 3-14(c) and Fig. 3-14(d). From Fig. 3-16, increasing the time range cannot make the HVs of NSGA II and MOEA/D reach the level of GHW-NSGA II and GHW-MOEA/D. This demonstrates our proposed GHW-NSGA II and GHW-MOEA/D have advantages not only in convergence rate but also in optimality.



(a) Absolute HVs.

(b) Ratio of absolute HVs between compared algorithms and GHW-NSGA II.

Figure 3-15 Absolute HVs and ratio over number of VMs where $(200, 40)$, $(500, 100)$, $(1000, 200)$ and zero_to_one=False.



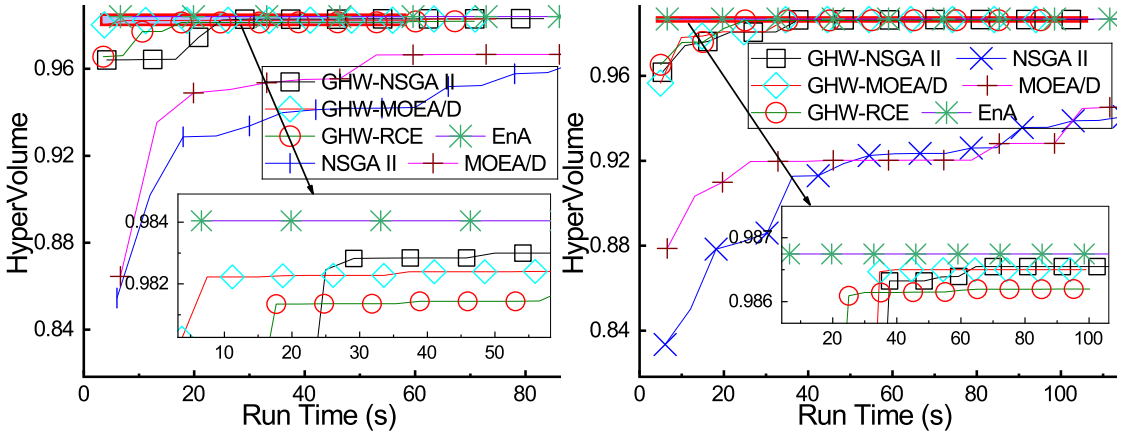
(a) 500 VMs and 100 Machines.

(b) 1000 VMs and 200 Machines.

Figure 3-16 Extended results of Fig. 3-14 with larger time range.

In order to further illustrate the performance of our proposed algorithms in the theoretical optimal solution, we carry out several groups of experiments in small scales. We use an enumerative algorithm (marked as EnA) to obtain the theoretical optimal Pareto solutions further to obtain theoretical optimal HVs as reference. Considering multi groups of experiments have similar conclusion, we only present two groups of experiments respectively under $(n, m) = (10, 3)$ and $(n, m) = (10, 4)$. Then, the HVs-over-time are shown in Fig. 3-17. In the experiments of Fig. 3-17, we set $N_p = 100$ for GHW-RCE, GHW-NSGA II and GHW-MOEA/D, as well as set $N_p = 500$ for NSGA II and MOEA/D. The N_g of all the algorithms is 20 and the mutation rate is 0.2. In Fig. 3-17, GHW-NSGA II and GHW-MOEA/D are the two fastest algorithms to approach or even reach the theoretical optimal

HV followed by GHW-RCE, while the state-of-the-art NSGA II and MOEA/D don't reach the theoretical optimum in Fig. 3-17. This shows that our proposed GHW family of algorithms is more likely to find the theoretical optimal solution in the small-scale dataset than compared algorithms.

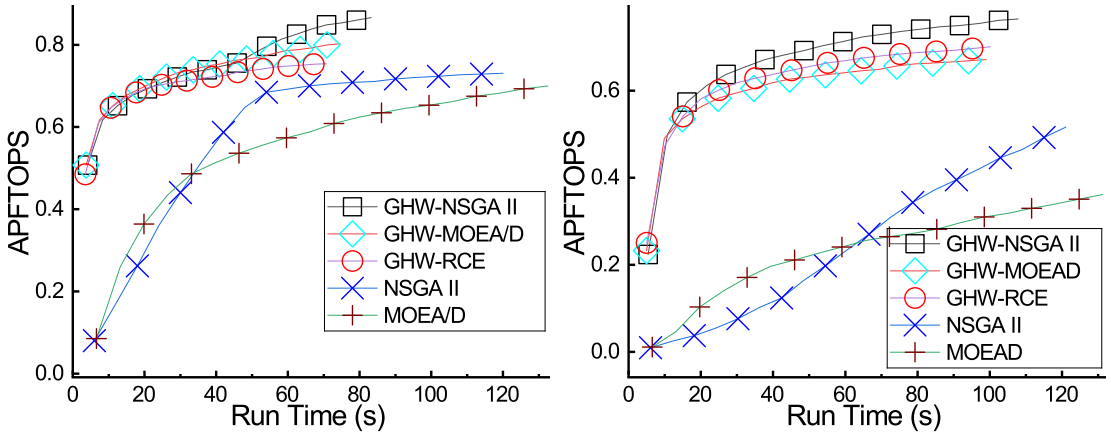


(a) 10 VMs and 3 Machines. (b) 10 VMs and 4 Machines.

Figure 3-17 HVs-over-time of proposed GHW family, NSGA II and MOEA/D for the problem $\min \omega^{(2)}$ with enumerative algorithm as reference in small scale simulation dataset.

To further comprehensively validate this conclusion, we conduct 100 instances under each combination of parameters $(n, m) = (10, 3)$ and $(n, m) = (10, 4)$ respectively, record the proportion of each algorithm finding the theoretical optimal solution at the corresponding time, and compute the average proportion corresponding to each time as the Average Probability-over-time of each algorithm Finding the Theoretical Optimal Pareto Solutions (denoted as APFTOPS). Then, we plot the results in Fig. 3-18.

In Fig. 3-18, the curves of GHW-NSGA II and GHW-MOEA/D are higher than that of the compared algorithms. Concretely, in the Fig. 3-18(a), the APFTOPSs of GHW-RCE (0.754), GHW-NSGA II (0.866) and GHW-MOEA/D (0.802) reach more than 0.75 at about 80s (20-th generation), while that of NSGA II (0.711) and MOEA/D (0.623) are less than 0.72; in the Fig. 3-18(b), that of GHW-RCE (0.701), GHW-NSGA II (0.765) and GHW-MOEA/D (0.671) reach more than 0.65 at about 100s (20-th generation), while that of NSGA II (0.446) and MOEA/D (0.310) are less than 0.45. The statistical significance of Fig. 3-18 shows that our proposed algorithms have higher probabilities to find the theoretical Pareto solutions, which means our proposed algorithms have better optimality than compared algorithms in solving the problem $\min \omega^{(2)}$.



(a) 10 VMs and 3 Machines. (b) 10 VMs and 4 Machines.

Figure 3-18 APFTOPS of the proposed GHW family, NSGA II and MOEA/D for the problem min $\omega^{(2)}$ with enumerative algorithm as reference in small scale simulation dataset where each combination of (n, m) has 100 instances.

These results in EX_4 prove that the advantages of GHW are comprehensive in the aspects of faster convergence rate and better optimality. This is consistent with the analysis in Section 3.5.1 that the combination of two search routes (i.e., HLSA route and genetic route) has more diverse information and can make full use of effective information to find better optimization solutions.

3.6.6 Summary of Experiments

Through the multiple groups of experiments from various sights in this section, we can observe that GHW, as a novel family of genetic algorithms, has significant advantages in addressing the MDRSP that is usually NP-hard problem. Among these experiments:

- EX_1 evaluates the effect of different growth routes on GGA. It demonstrates adding a growth stage can significantly improve the performance of the genetic algorithm in solving MDRSPs. It also demonstrates a directional growth route using HLSA has better solutions than compared random growth and non-growth. The order of algorithms by performance is: GGA-HLSA > GGA-RandomGrowth > GA (GGA-Non Growth), where GGA-HLSA > GGA-RandomGrowth means GGA-HLSA outperforms GGA-RandomGrowth;
- EX_2 evaluates the effect of different dimensionality reduction strategies on GGA-HLSA. It demonstrates using RW (Random Weight) in GGA-HLSA can obtain more comprehensively better solutions than compared dimensionality reduction strategies

including EW (Enumeration Weights), RD (Random Dimensions), RRD (Round-Robin Dimensions), TRD (Taboo Round-robin Dimensions) and RWS (Random dimensionality and Weights Strategies). The order by performance is: GGA-HLSA-RW > (GGA-HLSA-EW, GGA-HLSA-RWS) > (GGA-HLSA-RD, GGA-HLSA-RRD, GGA-HLSA-TRD). (GGA-HLSA-EW, GGA-HLSA-RWS) means GGA-HLSA-EW and GGA-HLSA-RWS have performance with approximate level, or GGA-HLSA-EW is not obviously superior to GGA-HLSA-RWS;

- Based on the previous experimental conclusions, EX_3 validates the feasibility of the algorithm we proposed to solve the MDRSP in realistic cloud computing. In addition, it demonstrates that combining the strategies of NSGA II with GHW can obtain better solutions than compared algorithms. The order by performance is: GHW-NSGA II > (GHD-NSGA II, GHWS-NSGA II) > NSGA II;
- Finally, EX_4 compares the GHW family (GHW-NSGA II and GHW-MOEA/D) with the state-of-the-art NSGA II and MOEA/D by observing the HVs-over-time in some large-scale dataset and observing the average probability to find theoretical optimal Pareto solutions over time in some small scale dataset. EX_4 demonstrates that the GHW family algorithms we propose have a faster convergence rate and better optimality than NSGA II and MOEA/D, that is (GHW-NSGA II, GHW-MOEA/D) > (NSGA II, MOEA/D). The results of EX_4 show that the algorithms we propose provide a comprehensive and significant improvement compared to the reference algorithms.

3.7 Summary of this Chapter

The Multi-Dimensional Resources Scheduling Problem (MDRSP) in cloud computing, a multi-objective optimization problem, is challenging because the resources of each dimension are usually heterogeneous and coupled. The solution of MDRSP requires simultaneous consideration of multi types of resources, which makes MDRSP far more complex than the single-dimensional resource scheduling problem.

In this chapter, we focus on the allocation of VMs in heterogeneous multi-dimensional resources of cloud computing and formulate several MDRSPs including minimizing the maximum utilization rate of each dimension of resources and minimizing the energy consumption of the total system. To solve these MDRSPs, we firstly use the concept of stages in genetic algorithm to divide its processes into four stages namely initialization stage, infancy stage, mature stage and genetic stage; secondly add a growth

stage to the genetic algorithm and propose GGA-HLSA-RW (GHW, i.e., Growable genetic algorithm using the Heuristic-based local search algorithm with random Weights as growth route). To concretize HLSA, we proposed a modified LSPT algorithm, which can improve the solutions of MSOPs in heterogeneous nodes. GHW, a hybrid algorithm combining meta-heuristic, heuristic and local search, has strong adaptability and optimality for various MDRSPs. The proposal of the GHW family allows the flexible combinations of various algorithms. To further improve the performance of GHW family of algorithms, we finally propose GHW-NSGA II and GHW-MOEA/D applying the sorting strategies and regeneration mechanism of NSGA II and MOEA/D into GHW.

To validate the performance of the GHW family, we carried out extensive experiments on the simulation dataset and AzureTraceforPacking2020-driven dataset. The experiments not only validate the growth strategy and dimensionality reduction strategy of GHW outperforming baselines, but also validate the feasibility and superiority of GHW in realistic cloud computing. Compared with state-of-the-art NSGA II and MOEA/D in experiments, GHW-NSGA II and GHW-MOEA/D have better convergence rate and optimality, which shows the comprehensive advantages of the GHW family of algorithms.

The other significance of the GHW family is that it shows the great potential of adding a growth stage to GA and demonstrates that combining with multi-search routes may be able to improve convergence and optimality.

On this basis, it is a worthwhile direction to explore the architecture and theory of more stable genetic algorithms or other multi-search route algorithms. The processing speed and energy consumption of electronic components are always affected by many factors, such as network congestion, temperature, continuous working time et al. We in the future plan to explore the variants of GHW and apply them to address MDRSPs in more complex scenarios considering the capacities of resources and conversion formula of energy consumption are time-varying in dynamic systems. In addition, accelerating GHW through distributed computing is also an important topic affecting the development of GHW family in the big data era.

Chapter 4 Joint Optimization of Multi-subproblems in Parallel Training of Deep Learning Models Based on Cross Search Algorithms

Parallel training of large-scale networks has attracted the attention of both artificial intelligence and high-performance distributed systems. One of efficiency parallelism is the micro-batch-based pipeline, e.g., GPipe. Based on the GPpie, we establish a time-cost model with the basic time function of layers, which considers computing time and communication time simultaneously as well as regards them as nonlinear to batch size. Focusing on the optimal solutions of network division and data partition, we propose a Cross-Search algorithm with Improved Multi-dimensional Dichotomy (CSIMD). Through theoretical derivation, we prove IMD has appreciable theoretical optimality and computational complexity. Extensive experiments on both CNN-based networks and transformer-based networks demonstrate our proposed CSIMD can obtain optimal network division and data partition schemes under GPipe parallelism. On average, CSIMD achieves $(2.0, 2.5)\times$ and $(1.5, 1.6)\times$ speedup respectively in CNN- and transformer-related networks over GPipe-R and GPipe-E.

4.1 Introduction

As one of the representative technologies of artificial intelligence (AI), deep learning (DL) has a significant trend that the scale of model parameters continues to increase [168,169]. The recent large deep neural networks (DNN) have hundreds of billions parameters such as FLAN with 137B parameters [170], GPT-3 [171] with more than 175B parameters, Gopher with 280B parameters [172] and ERNIE 3.0 Titan [173] with 260B parameters. From the existing research, large DNNs can generally maintain higher accuracy for some complex scenarios, especially for natural language processing (NLP) and computer vision (CV).

In realistic, training these extremely large DNNs in realistic requires cooperative and parallel work of a large number of distributed AI processor devices (e.g., GPU cluster), which usually consume too much time [174,175]. How to improve the efficiency of GPU cluster become a key and hotspot in large-scale DNN research [168,174,176]. However, the structure optimization of distributed parallel training is a complex NP-Hard problem where

the solution space (scheme space) is often an uncountable set [174,177]. To obtain some approximate schemes, existing research regards data parallelism, model parallelism and pipeline parallelism as three foundational policies [178–180], which directs current parallel training research to make gradual improvements.

Although the tensor model parallelism (such as Megatron-LM [181]) can reduce the computing times, it requires additional communication times and costs, which may greatly increase the total training time and training cost [180]. To reduce the communication cost and time, the pipeline parallel structure only has communication between two adjacent model nodes. Currently, pipeline-related parallel structures are the most frequent in the parallel training of large DNNs. Some existing well-performed pipeline parallelism-related architectures are Gpipe [182], PipeDream [183], Dapple [184] and Hpipe [185] et al.

Data partition (data sharding) is one of the most critical factors of parallel training especially for micro-batch-based pipeline parallelism [186,187], which directly affects the schedulable granularity of computing time and communication time in distributed parallelism [180]. Improving schedulable granularity allows more overlap between the time of various processes and can increase feasible solution space with better theoretical global optimization solutions [184,188,189]. In realistic, when using the devices (including GPU and network) to participate in training, the computing time or communication time is probably not proportional to their data volume [177,190], although some studies [191–194] regard it as a proportional relationship in order to simplify the problem. This indicates that continuously improving the granularity of the data partition may introduce extra time consumption. To find the optimal partition of data parallelism or model parallelism, some recent research mainly utilized dynamic programming [169,184]. Due to the large numbers of model parameters and devices, the existing partition methods have massive computational complexity as the solution space generally increases exponentially [191,195,196]. In addition, the lack of accurate analysis models and formulas makes it quite difficult to pre-estimate the corresponding training performance of each partition scheme.

In this chapter, we choose total training time as the indicator of performance and mainly focus on the optimization of Micro-Batch-based Pipeline Parallelism (MBPP). We not only consider computing time and communication time simultaneously but also consider them probably nonlinear with data size. Then, we construct a time-cost model with respect of network divisions and mini-batch partitions. If the functional expressions of computing time and communication time for each micro-batch corresponding to each node

under different partitions are known, the global optimal partition number can be obtained by direct derivation or difference of theoretical equations and inequalities. If unknown which is the common case in realistic, the performance data or curve can still be obtained through the performance test program as the reference [191], and the extreme point can still be obtained by substituting the performance data into the theoretical formula.

With the theoretical model, the joint problem can be formulated as a multi-dimensional array segmentation which is usually an NP-Hard problem. Then, we propose a cross-search algorithm with improved multi-dimensional dichotomy (CSIMD) to jointly solve the network division and mini-batch partition. Through theoretical derivation, we prove IMD has an appreciable approximation to solve multi-dimensional array segmentation. To validate the comprehensive performances of our proposed algorithm, we conduct multiple sets of experiments from various aspects in the homogeneous GPU Cluster. The experiments demonstrate our proposed CSIMD can further reduce the training time of MBPP.

The main contributions of this chapter can be summarized as follows.

(1) In theory, we derive a formula of the total training time with respect of partition number and network division of MBPP considering computing time and communication time simultaneously as well as considering the relationship between data size and time may be nonlinear, which is more realistic. And then, we formulate the problem of parallel training under MBPP to multi-dimensional array segmentation.

(2) Based on the time-cost model, we propose a cross-search algorithm with improved multi-dimensional dichotomy (CSIMD) to obtain the optimal network division and mini-batch partitions.

(3) We theoretically prove the relationship between the approximation ratio and computational complexity of the IMD algorithm to solve multi-dimensional array segmentation. We also present the theoretical basis formula for parameter selection of the IMD algorithm.

(4) Extensive experiments from various sights not only demonstrate the optimality and fastness of our proposed IMD but also demonstrate that our proposed CSIMD can obtain an optimal scheme of MBPP.

The rest of this chapter is organized as follows. We review the related work in Section 4.2; The formulation of distributed parallel training under MBPP and the relevant theoretical formulas are derived in Section 4.3. We propose the methodology and present

its theoretical analysis in Section 4.4. The experiment design and evaluation results are presented in Section 4.5. Finally, we conclude this chapter in Section 4.6.

4.2 Related Work

In this section, we mainly review three aspects that are related to our research in this chapter: parallelism, mathematical cost model, and partition algorithms.

Data parallelism and model parallelism are two basic modes of parallel training, which derive various new parallel parallelism [169,185]. Data parallelism partitions training data into multiple pieces [197,198] and model parallelism divides the AI model into multiple parts [178,191,199]. Combining data parallelism and model parallelism, one important series is pipeline parallelism. Gpipe [182] divided mini-batch data into multi micro-batches, which allowed the computation and communication of different stages corresponding to different model nodes (on different devices) can overlap. The more overlapping parts of various processes on different devices correspond to the less idle time (bubbles) of devices in the whole system [169]. On the premise that the pipeline does not introduce redundant computing and communication costs, the total training time of GPipe is smaller than the original pipeline. Based on the micro-batch data parallelism, PipeDream [183] added a strategy that is shifting the gradient backward-propagation (BP) earlier to the moment immediately after its last part of forward-propagation (FP). Other well-performed methods or parallelism including Dapple [184], Hpipe [185], TeraPipe [180], NasPipe [200], et al are all the variants of Gpipe or PipeDream based on micro-batch data parallelism. Terapipe [180] followed the pipeline of GPipe and improved the pipelining granularity to reduce the pipeline bubbles of the transformer-based NLP model by proposing a new dimension, i.e. token dimension. Dapple [184] followed the method of PipeDream and shift the BP of other sub-model nodes to earlier besides that of the last sub-model node for the same mini-batch [184].

The mathematical cost model is also an important factor of parallel training, which is the basis for the optimization solution of the subsequent parallel training scheme. Next, we mainly review the time-cost model of parallel training in existing research. The complexity of parallel training, makes it difficult to obtain an accurate expression of the cost model. Gpipe did not consider the corresponding time-cost model in [182]. In the subsequent studies [180,186,191], the time-cost model was considered as:

$$T_{PP} = (m-1) \cdot (\max(F_i) + \max(B_i)) + \sum (F_i + B_i) \tag{4-1}$$

where T_{PP} means the total training time for one iteration when using pipeline parallelism, F_i and B_i are the time respectively for forward propagation and backward propagation of the i-th stage, and m is the number of micro-batches.

The cost model of PipeDream was considered as a recursive formula [183]. Deepak Narayanan et al [201] proposed PipeDream-2BW and considered a time-cost model for pipelining as

$$T_{PP} = \max_i \max \left(T_i^{cp} + \sum_j T_{j \to i}^{cm}, \frac{1}{m} \cdot T_i^{cm} \right)$$

where T_i^{cp} means the computing time of the i-th stage, $T_{j \to i}^{cm}$ means the communication time between stages i and j, and T_i^{cm} means the communication time of exchanging gradients. PipePar [202] considered a cost model as

$$T_{PP} = T_{cm} + T_{cp},$$

which directly adds communication and computing time. Deepak Narayanan et al [181] proposed Megatron-LM and considered a cost model as

$$T_{PP} = C \cdot (F + B))$$

where C is a coefficient related to micro-batch size and data parallel size. Venmugil Elango [194] proposed PaSE and considered the FLOP-to-bytes ratio in the cost model.

Because the cost model was generally non-analytical or recursive, dynamic programming is a suitable and widely used method to obtain the optimal partition of data parallelism or model parallelism [169]. Some examples include PipeDream [184], Dapple [184], Terapipe [180], EffTra [186], Alpa [191], PaSE [194], PipePar [202] et al. Linear programming is also a frequent method in parallel training [176,198,203]. Some examples include NetPlacer [203], HGP4CNN [176], DPDA [198]. Other partition methods include off-the-shelf graph partitioning algorithms [204], recurrence [196], grouping genetic algorithm [205], minimum vertex-cut graph partitioning algorithm [206], near-optimal layer partition of local search method [195].

From the literature review, the formulation of the cost model affected the choice of partition algorithm. Some of the cost models only considered one of computation and communication; some only provided recursive formulas; and some with direct expressions relied on a lot of ideal assumptions which was far away from the real scenario. Referring to but distinguished from the existing research, this chapter derives a time-cost model of

MBPP considering computation and communication simultaneously. The model doesn't limit the features of devices, so it adapts to both homogenous and heterogeneous systems. The cost model also considers the nonlinear relationship between cost and data size, which is closer to reality. The optimal algorithm in this chapter is cross-search with improved multi-dimensional dichotomy which has far less complexity.

4.3 Cloud System and Optimization Problem Formulations Considering Parallel Training Workflow of Deep Leaning Model

Before the derivation of the cost model, we list some notations and their descriptions in Table 4-1.

Table 4-1 Notations and Descriptions

Notation	Description
N	Number of stages
p	Number of partitions
K	Number of layers of DNN
$F_i^P(p)$	The forward computing time of one micro-batch in the i-th stage when partitioning the mini-batch into p micro-batches
$F_i^M(p)$	The forward communication time of one micro-batch
$B_i^P(p)$	The backward computing time of one micro-batch
$B_i^M(p)$	The backward communication time of one micro-batch
$S_{ij}^P(p)$	The start computing time of the j-th partition in the i-th stage of forward
$S_{ij}^M(p)$	The start communication time of the j-th partition in forward
$R_{ij}^M(p)$	The start computing time of the j-th partition in backward
$R_{ij}^M(p)$	The start communication time of the j-th partition in backward
T_{GP}	The time-cost for one iteration of one mini-batch under Gpipe parallelism
$H_i(p)$	The computing time of the i-th layer when partitioning the mini-batch into p micro-batches
$J_i(p)$	Time required to communicate the output data of the i-th layer to other device
L_i	The i-th layer
C_i	The collection of layer on the i-th device
α_i	The maximum number of layers in C_i
λ	The collection of α_i as $\lambda = \langle \alpha_1, \ldots, \alpha_N \rangle$
Dz_i	The data size of the output for the i-th layer

In this chapter, we mainly focus on one structure of MBPP (i.e., GPipe) shown in Fig 4-1. The characteristic of GPipe is that the BPs of micro-batches in each mini-batch need to start after the FP of the last micro-batch in the last layer ends. To simplify the structure of MBPP so that the analytical formulas of time-cost are obtainable, we consider that the

receiving data and sending data of the same device do not affect each other, otherwise, the recursive formula will be more complex. This is consistent with the full-duplex communication mode which is widely used in current distributed computing systems [188]. Thus in Fig 4-1, the occupation of data received by each device for the communication component of this device is omitted.

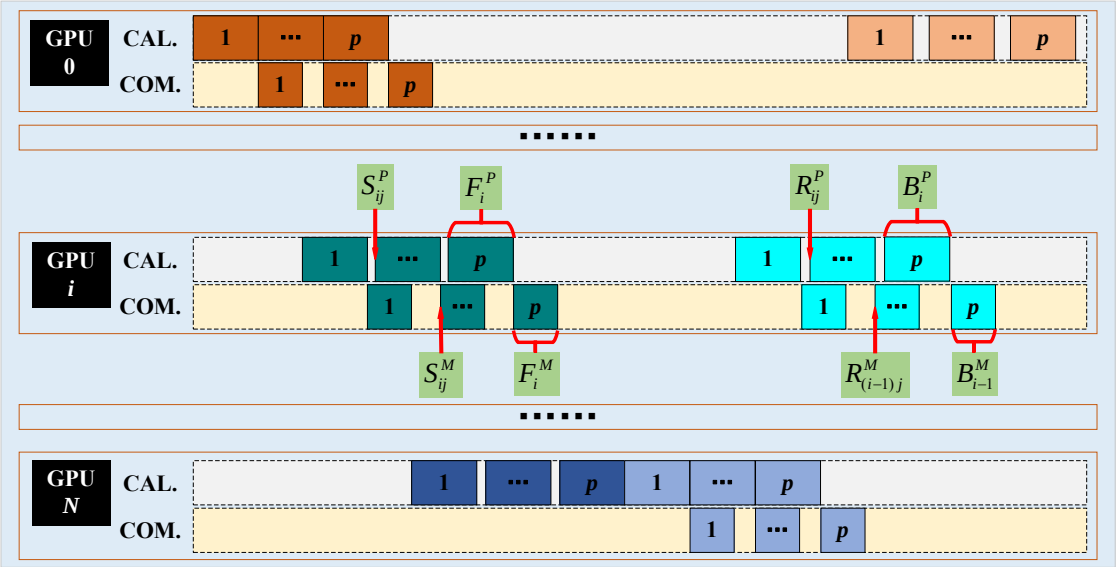


Figure 4-1 The structures of the GPipe parallelism.

4.3.1 Cost Model for GPipe considering Computing and Communication Time

For GPipe, the expression of time is as Eq 4-1 without consideration of communication.

To construct an expression related to partitions, we set that the computing time and communication time of one micro-batch are the functions in terms of partitions: $F_i^P(p)$ and $F_i^M(p)$ are respectively the computing time and communication for FP of one micro-batch in the i-th stage, where p is the number of data partitions; $B_i^P(p)$ and $B_i^M(p)$ are that for BP. The process time consumption of different micro-batches in the same stage is relatively stable. It can be set that the $S_{ij}^P(p)$ is the start computing time of the j-th partition in the i-th stage of FP and $S_{ij}^M(p)$ is the start communication time; $R_{ij}^P(p)$ and $R_{ij}^M(p)$ are for BP.

When the functions $F_i^P(p), F_i^M(p), B_i^P(p)$ and $B_i^M(p)$ are fixed, $S_{ij}^P(p), S_{ij}^M(p), R_{ij}^P(p)$ and $R_{ij}^M(p)$ can completely represent the whole process of parallel training of a model. Since these variables are functions of partition number p, we omit the (p) in the subsequent expression. Different from PipeDream, the characteristic of the Gpipe structure is that the

backward propagation in the same mini-batch needs to wait for all the forward propagation to complete before starting. Then, the recursive formula can be written as Eq 4-2 and Eq 4-3 according to its characteristic when considering computing and communication time simultaneously.

$$\begin{cases} S_{ij}^{P} = \max\left(S_{(i-1)j}^{M} + F_{i-1}^{M}, S_{i(j-1)}^{P} + F_{i}^{P}\right) \\ S_{ij}^{M} = \max\left(S_{ij}^{P} + F_{i}^{P}, S_{i(j-1)}^{M} + F_{i}^{M}\right) \end{cases} \tag{4-2}$$

$$\begin{cases} R_{ij}^{P} = \max\left(R_{ij}^{M} + B_{i}^{M}, R_{i(j-1)}^{P} + B_{i}^{P}\right) \\ R_{ij}^{M} = \max\left(R_{(i+1)j}^{P} + B_{i+1}^{P}, R_{i(j-1)}^{M} + B_{i}^{M}\right) \end{cases} \tag{4-3}$$

where $1 \leq i \leq N$ is the index of stage, N is the number of stages,$1 \leq j \leq p$ is the index of partition. If $i \notin [1, N]$ or $j \notin [1, p]$, then $S_{ij}^{P}(p) = R_{ij}^{P} = -\infty$. And if $i \notin [1, N-1]$ or $j \notin [1, p]$, then $S_{ij}^{M} = R_{ij}^{M} = -\infty$. The initial conditions of Eq 4-2 and Eq 4-3 are:

$$\begin{cases} S_{11}^{P} = 0 \\ R_{N1}^{P} = S_{Np}^{P} + F_{N}^{P} \end{cases} \tag{4-4}$$

Substitution of Eq 4-4 into Eq 4-2 can obtain the expression of FP:

$$\begin{cases} S_{ij}^{P} = \sum_{k=1}^{i-1}\left(F_{k}^{P} + F_{k}^{M}\right) + (j-1)\max\left(\max_{1\leq k\leq i-1}\left(F_{k}^{P}, F_{k}^{M}\right), F_{i}^{P}\right) \\ S_{ij}^{M} = \sum_{k=1}^{i}\left(F_{k}^{P} + F_{k}^{M}\right) - F_{k}^{M} + (j-1)\max_{1\leq k\leq i}\left(F_{k}^{P}, F_{k}^{M}\right) \end{cases} \tag{4-5}$$

where $1 \leq i \leq N$ and $1 \leq j \leq p$.

Substitution of Eq 4-4 into Eq 4-3 can obtain the expression of BP:

$$\begin{cases} R_{(N-i)j}^{P} = S_{Np}^{P} + F_{N}^{P} + \sum_{k=1}^{i-1}\left(B_{N-k}^{P} + B_{N-k}^{M}\right) + B_{N}^{P} \\ \qquad + (j-1)\max\left(\max_{1\leq k\leq i-1}\left(B_{N-k}^{P}, B_{N-k}^{M}\right), B_{N}^{P}\right) \\ R_{(N-i)j}^{M} = S_{Np}^{P} + F_{N}^{P} + \sum_{k=1}^{i}\left(B_{N-k+1}^{P} + B_{N-k}^{M}\right) \\ \qquad + (j-1)\max_{1\leq k\leq i}\left(B_{N-k+1}^{P}, B_{N-k}^{M}\right) \end{cases} \tag{4-6}$$

Then, the time-cost for one iteration of one mini-batch under GPipe parallelism can

be derived as:

$$\begin{aligned} T_{GP} = R_{1p}^{P} + B_1^{P} = & \sum_{k=1}^{N} \left(F_k^{P} + F_k^{M} + B_k^{P} + B_k^{M} \right) \\ & + (p-1) \max \left(\max_{1 \le k \le N-1} \left(B_k^{P}, B_k^{M} \right), B_N^{P} \right) \\ & + (p-1) \max \left(\max_{1 \le k \le N-1} \left(F_k^{P}, F_k^{M} \right), F_N^{P} \right) \end{aligned} \tag{4-7}$$

where we set $F_N^M = B_N^M = 0$. Compared to Eq 4-1, Eq 4-7 puts the communication time in consideration.

4.3.2 Theoretical Analysis of Cost Model

In parallel training of MBPP, two aspects need to be optimized: dividing DNN into N stages (denoted as network division) and finding the optimal number of data partitions (denoted as data partition).

If the functions $F_i^P(p)$, $F_i^M(p)$, $B_i^P(p)$ and $B_i^M(p)$ are given, the optimal partition number can be obtained by the extreme values of Eq 4-7. Thus, we first discuss the division of network layers which needs to be determined before solving the optimal data partition.

It can be set that the DNN has K layers denoted as $L = \langle L_1, L_2, \ldots, L_K \rangle$ where L_i corresponds to the i-th layer and $K \geq N$. For the sake of analysis, we also set that the computing time and communication time (time required to communicate its output data to the next layer) for the FP of the i-th layer are respectively $H_i^P(p)$ and $H_i^M(p)$ where $1 \leq i \leq K$. That for BP are $J_i^P(p)$ and $J_i^M(p)$. Thus, we can plot a diagram of the whole calculation process and the corresponding symbols for one micro-batch of DNN as Fig 4-2.

The division of the network layer is actually to determine on which device each layer executes. It can be set that the collection of layer on the i-th device is C_i, i.e., $L_i \in C_j$ means the i-th layer is executed on the j-th device. The MBPP structure divides the network layer in order, thus: if $L_{i_1} \in C_j$ and $L_{i_2} \in C_j$ where $i_1 \leq i_2$, then $L_i \in C_j$ for $i_1 \leq \forall i \leq i_2$. Therefore, we can set the maximum number of layers in C_i as α_i where $\alpha_{i-1} < \alpha_i$. This means if $\alpha_{i-1} < j \leq \alpha_i$ then $L_j \in C_i$, else $L_j \notin C_i$, which also means $C_j = \left\langle L_{\alpha_{i-1}+1}, L_{\alpha_{i-1}+2}, \ldots, L_{\alpha_i} \right\rangle$. Then, we can use H^P, H^M, J^P and J^M to express the F^P, F^M, B^P

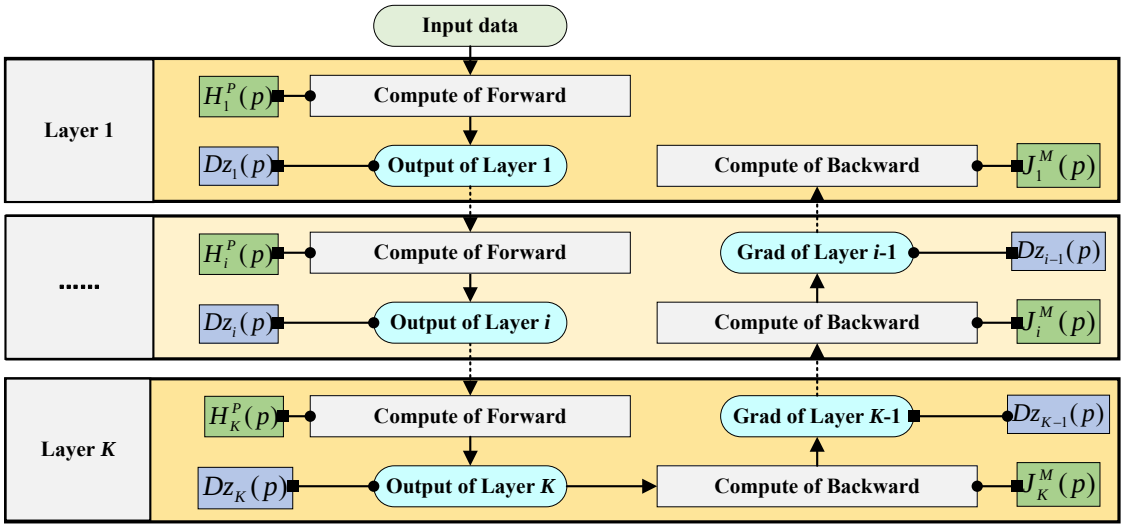


Figure 4-2 The diagram of the whole calculation process and the corresponding symbols for one micro-batch of DNN.

and B^M as Eq 4-8.

$$\begin{cases} F_i^P = \sum_{k=\alpha_{i-1}+1}^{\alpha_i} H_k^P \\ F_i^M = H_{\alpha_i}^M \\ B_i^P = \sum_{k=\alpha_{i-1}+1}^{\alpha_i} J_k^P \\ B_i^M = J_{\alpha_i+1}^M \end{cases} \tag{4-8}$$

For the sake of presentation of the relationship between stages and layers of DNN, we plot the diagram of network division with the FP in Fig 4-3 on the basis of the process of Fig 4-2. The process of BP is analogous to that of FP.

Substituting Eq 4-8 into Eq 4-7 can obtain the expression of training time with respect of α_i and p as Eq 4-9.

$$\begin{aligned} T_{GP} = & \sum_{i=1}^{K}\left(H_i^P + J_i^P\right) + \sum_{i=1}^{N-1}\left(H_{\alpha_i}^M + J_{\alpha_i+1}^M\right) \\ & + (p-1)\max\left(\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}\left(J_k^P\right)\right), \max_{i=1}^{N-1}\left(J_{\alpha_i+1}^M\right)\right) \\ & + (p-1)\max\left(\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}\left(H_k^P\right)\right), \max_{i=1}^{N-1}\left(H_{\alpha_i}^M\right)\right) \end{aligned} \tag{4-9}$$

Therefore, the key to solving the problem of minimizing total training time is to find

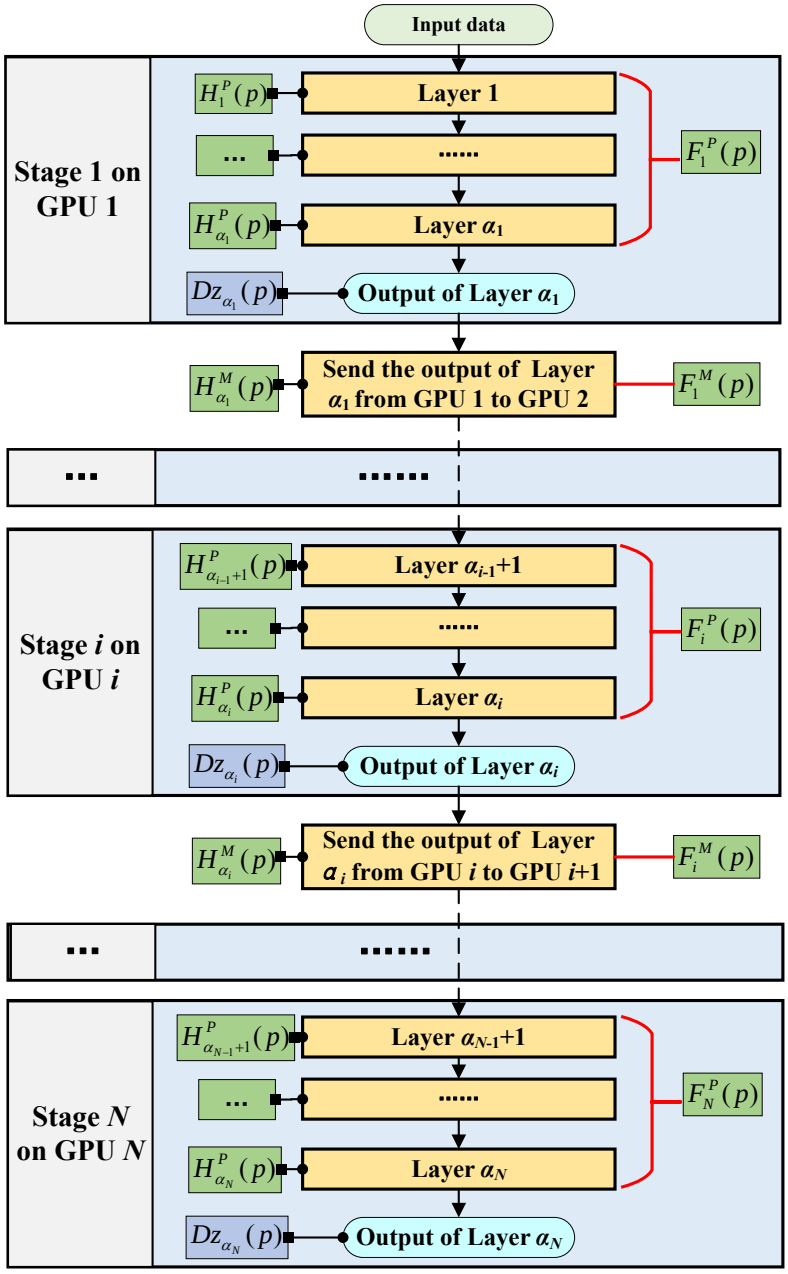


Figure 4-3 The diagram of network division (K layers to N stages) with the forward propagations.

the suitable collection of $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$ and p. In the scenario of this chapter, λ and the number of data partitions p necessarily and sufficiently correspond to a unique parallel training scheme under MBPP. Thus, we can use the vector $\langle \lambda, p \rangle$ to represent the solution of the optimal solution of a parallel training problem and use $T_{GP}(\lambda, p)$ to represent its corresponding training time under parallel training structure for one mini-batch.

When p is given, $\sum_{i=1}^{K} (H_i^P + J_i^P)$ is a constant. Thus, the problem can be transformed into the balancing division of layers (a multi-dimensional array seg-

mentation problem). If ignoring communication cost, the problem is converted to $\min\left(\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}(H_k^P)\right)+\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}(J_k^P)\right)\right)$, which is segmenting K numbers to N groups without changing orders to minimizing the maximum sum of these groups. However, considering communication and computing simultaneously increases the complexity of the problem significantly, because different network division schemes will change the layers of DNN involved in communication.

4.3.3 Theoretical Analysis of Basic Function

When $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$ is given, the key to solving the problem of minimizing the total training time under MBPP is to find the optimal partition number p of mini-batch. In this case, it only requires to calculate the extreme value of T_{GP}. Thus, the functions of $H_i^P(p)$, $H_i^M(p)$, $J_i^P(p)$ and $J_i^M(p)$ are crucially required. In realistic, these functions $H_i^P(p)$, $H_i^M(p)$, $J_i^P(p)$ and $J_i^M(p)$ are nonlinear to the micro-batch size. This means they aren't inversely proportional to partition number p.

In the realistic network, the communication time is approximately proportional to data size only when the data size is far larger than bandwidth Bw. However, when the data size is less than a specific value (specific bytes), the communication cost is approximately constant. This is because each communication requires the least cost. Thus, a piecewise function as Eq 4-10 can be used to fit the relationship between data size Dz and communication cost Mc.

$$Mc = \begin{cases} C_2 \cdot Dz, & Dz \geq C_3 \\ C_1, & D_z < C_3 \end{cases} \tag{4-10}$$

where C_1, C_2 and C_3 are bandwidth Bw-related constants.

For the i-th layer of DNN, assuming the output data size of one mini-batch as Dz_i, the function H_i^M of one micro-batch for FP can be written as Eq 4-11.

$$H_i^M(p) = \begin{cases} C_2 \cdot \dfrac{Dz_i}{p}, & p \leq \dfrac{Dz_i}{C_3} \\ C_1, & p > \dfrac{Dz_i}{C_3} \end{cases} \tag{4-11}$$

With the same constants C_1, C_2 and C_3, the function of $J_i^M(p)$ is similar to $H_i^M(p)$.

For computing time, there is a similar phenomenon to communication time, i.e., when data is less than a specific size, the computing time will be relatively unchanged with the data size. However, since the operation of the model is tensor calculation (matrix calcu-

lation) involving multiple dimensions, it is not proportional to the size of one dimension. For data parallelism with micro-batch, the inflection point is related to its corresponding layer and the performance of devices. It can set the inflection point of the i-th layer as γ_i^F for forward computing and γ_i^B for backward computing. When devices are given, the least time required for computation of a small amount is also given, which can be set as P_1. The functions of computing time can be represented as Eq 4-12 with respect of partition number p.

$$\begin{cases} H_i^P(p) = \begin{cases} \dfrac{\beta_i^F}{p}, & p \le \gamma_i^F \\ P_1, & p > \gamma_i^F \end{cases} \\ J_i^P(p) = \begin{cases} \dfrac{\beta_i^B}{p}, & p \le \gamma_i^B \\ P_1, & p > \gamma_i^B \end{cases} \end{cases} \tag{4-12}$$

where β_i^F and β_i^B are the time-cost for FP and BP respectively when $p = 1$.

For a homogeneous GPU cluster configured for a given hardware environment, the constants C_1, C_2, C_3 and P_1 are fixed which can be obtained by statistics of multiple communication experiments. The data sizes of output data are relatively explicit and can be obtained by bytes of data. However, the β_i^F, β_i^B γ_i^F and γ_i^B are implicitly related to the intrinsic matrix operation of the layers. However, these inflection points are relatively stable and can still be obtained through certain statistical methods.

4.4 Algorithm Design: Cross Search Algorithm

Based on the above analysis, the problem can be transformed into three aspects:

(1) ω_1: solving optimal network division $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$ with given partition number p of mini-batch;

(2) ω_2: solving optimal partition number p with given network division $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$;

(3) Jointly solving problem ω_1 and problem ω_2 to obtain the optimal parallel training scheme $\langle \lambda, p \rangle$.

In this section, we will propose the corresponding method to solve these three aspects.

4.4.1 Cross-Search for Joint Solution of ω_1 and ω_2

Firstly, we discuss the systematic method, i.e., cross-search for a joint solution of ω_1 and ω_2. Supposing two algorithms, denoted as A_1 and A_2, can respectively obtain the theoretical optimal solution of problems ω_1 and ω_2. Two avenues to get the joint solution of ω_1 and ω_2 are:

(1) Traverse all mini-batch partitions p to find the optimal network division corresponding to each partition, and then compare their solutions to obtain the optimal scheme;
(2) Traverse all network divisions to find their corresponding optimal mini-batch partitions and then compare to obtain the scheme.

However, these two traversal avenues both need to consume a lot of computational complexity. To improve the calculation speed for jointly solving ω_1 and ω_2 with maintaining the optimization, we propose a cross-search algorithm whose steps are as follows.

(1) Firstly, divide the network layer under the original structure;
(2) Find the optimal partition number of mini-batch under this layer division;
(3) Re-adjust network layer division under the optimal partition number;
(4) Solving the optimal partition number again;
(5) Repeat step (3) to step (4) until there is no better network layer partition and partition number.

The pseudo-code of cross-search can be seen in Algorithm 4-1. Through the way of cross-switching optimization, Algorithm 4-1 achieves the search for optimal network division and optimal partition number.

With the framework of the systematic method (cross-search), two algorithms A_1 and A_2 which can respectively solve problems ω_1 and ω_2 are significantly required. Next, we will discuss them successively.

4.4.2 Improved Multiple Dichotomy Algorithm to Divide Network Layers

4.4.2.1 Algorithm and Framework

The problem ω_1, that solving the optimal network division with a given partition number, is a variant of the array balanced segmentation problem, i.e. multi-dimensional array segmentation problem. The typical array balanced segmentation problem only considers a one-dimensional array. For example, when only considering computing of BP, the

Algorithm 4-1 Cross-Search Algorithm for Joint Solution of ω_1 and ω_2

Input : K, N, The functions of H_i^P, H_i^M, J_i^P, and J_i^M for $\forall 1 \leq i \leq K$
Output: Solution $\langle \lambda, p \rangle$ of parallel training and its corresponding training time $T(\lambda, p)$

Set $p = 1$
Use algorithm A_1 to obtain the optimal network division $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$ under p and record its corresponding training time $T_{GP}(\lambda, p)$ (denoted as $T(\lambda, p)$ uniformly)
while *True* **do**
    Use algorithm A_2 to obtain the optimal data partition number p' under λ and obtain its corresponding training time as $T(\lambda, p')$
    if $T(\lambda, p') = T(\lambda, p)$ **then**
        Break
    Use algorithm A_1 to obtain the optimal network division λ' under p' and obtain $T(\lambda', p')$
    if $T(\lambda', p') = T(\lambda, p')$ **then**
        Break
    Update $p = p'$ and $\lambda = \lambda'$

objective of problem ω_1 can be simplified to $\min \max_{i=1}^{N} \left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i} J_k^P \right)$ setting p is given. To solve the problem ω_1 considering J^P, J^M, H^P, and H^M simultaneously, we improve the dichotomy and propose multiple dichotomy method.

Following the strategy of typical dichotomy, the improved multi-dimensional dichotomy method regards the problem ω_1 to the bin-packing problem and transforms the solution target to find the minimum volume of bins. The dichotomy method introduces preset volume and mainly updates the preset volume by taking the median of the preset volume and the maximum or minimum volume currently obtained. The volume is measured by $\max\left(\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}(J_k^P)\right), \max_{i=1}^{N-1}\left(J_{\alpha_i+1}^M\right)\right) + \max\left(\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}(H_k^P)\right), \max_{i=1}^{N-1}\left(H_{\alpha_i}^M\right)\right)$. If the minimum number of bins is larger than N, it implies that the preset volume is small and requires to be enlarged, otherwise, the preset volume is large and can be decreased. Then, we can use the dichotomy method to update the preset volume.

Improved multiple dichotomy algorithm (IMD) can be seen in Algorithm 4-2. As the IMD transforms the problem ω_1 to an array segmentation packing problem, it needs to call an improved two-dimensional array segmentation packing algorithm as Algorithm 4-3.

Algorithm 4-2 Improved dichotomy to solve multi-dimensional array segmentation (IMD)

Input : K, N, p, The functions of H_i^P, H_i^M, J_i^P, and J_i^M for $\forall 1 \leq i \leq K$
Output: Solution $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$ and $T(\lambda, p)$

Set the small values ε
Set $D_1 = \max\left(\sum_{i=1}^{K}(H_i^P), \max_{i=1}^{N-1}\left(H_{\alpha_i+1}^M\right)\right)$, $D_2 = \max\left(\sum_{i=1}^{K}(J_i^P), \max_{i=1}^{N-1}\left(J_{\alpha_i+1}^M\right)\right)$, $D^{max} = D_1 + D_2$ and $D^{min} = \max\left(\max\left(H^P, J^P\right), \min\left(H^M, J^M\right)\right)$
Set the weight groups with $\eta + 1$ (or other given number) groups $W = \langle w_0, w_1, w_2, \ldots, w_\eta \rangle$ where $w_i = \langle w_i(1), w_i(2) \rangle = \left\langle \frac{i}{\eta}, 1 - \frac{i}{\eta} \right\rangle$
while $D^{max} - D^{min} \geq \varepsilon$ **do**
    Set $D^{mean} = \frac{D^{max} + D^{min}}{2}$, $check = False$
    for w *in* W **do**
        Set $\eta_1 = w(1) \cdot D^{mean}$, $\eta_2 = w(2) \cdot D^{mean}$
        Substitute η_1 and η_2 into Algorithm 4-3 to obtain τ and λ^{mean}
        if $\tau \leq N$ **then**
            $check = True$, $\lambda = \lambda^{mean}$
            Break the "for" loop
    if *check* **then**
        $D^{max} = D^{mean}$
    else
        $D^{min} = D^{mean}$
    Calculate the training time $T(\lambda, p)$

Algorithm 4-3 Two-dimensional array segment packing algorithm

Input : K, N, p, upper limit η_1 and η_2, the functions of H_i^P, H_i^M, J_i^P, and J_i^M for $\forall 1 \leq i \leq K$
Output: The number of bins τ, the solution of array segmentation $\lambda = \langle \alpha_1, \alpha_2, \ldots, \alpha_\tau \rangle$ and its corresponding training time $T(\lambda, p)$

Set $\psi_1 = \psi_2 = 0$, $\lambda = \emptyset$, $\tau = 0$, $i = 0$
while $i \leq K$ **do**
    $i++$
    $\psi_1 += H_i^P$, $\psi_2 += J_i^P$
    if $\max(\psi_1, H_i^M) \leq \eta_1$ *and* $\max(\psi_2, J_i^M) \leq \eta_2$ **then**
        $\alpha = i$
    if $\max(\psi_1, H_i^M) > \eta_1$ *and* $\max(\psi_2, J_i^M) > \eta_2$ **then**
        $\tau++$, $\lambda += \{\alpha\}$, $i = \alpha$, $\psi_1 = \psi_2 = 0$
$\tau++$, $\lambda += \{K\}$

4.4.3 Method to Obtain Optimal Partition Number

To obtain the optimal solution of parallel training, we also need to find the optimal partition number p of data parallelism under a given network division scheme λ, i.e., solving the problem ω_2.

When, H^P, H^M, J^P and J^M are known, it is easy to calculate the running time $T(\lambda, p)$ for $\forall p$. Then, it only needs to choose the partition number corresponding to the minimum running time. As the network division scheme λ is given, matrix operation can be used to accelerate the calculation of running time for all possible partition numbers. The algorithm to obtain the number of the optimal partitions under given λ is as Algorithm 4-4.

4.5 Theoretical Analysis and Proof

4.5.0.1 Properties and Analysis to the Converge Solution of IMD

The improved dichotomy to solve multi-dimensional array segmentation (IMD, Algorithm 4-2) aims at solving the problem that

$$\min \omega^{(1)}(\lambda) \tag{4-13}$$

where $\omega^{(1)}(\lambda) = \max\left(\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}\left(J_k^P\right)\right), \max_{i=1}^{N-1}\left(J_{\alpha_i+1}^M\right)\right) + \max\left(\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}\left(H_k^P\right)\right), \max_{i=1}^{N-1}\left(H_{\alpha_i}^M\right)\right)$. Denoting the solution of Algorithm 4-2 as λ^{ID}, the feasible solution set as Λ, and the theoretical optimal solution is λ^O, then the solution has the following property according to the process of Algorithm 4-2.

Property 4 If $\varepsilon \to 0$ and the weight groups W has adequate weights $\eta \to +\infty$ which can ergodic all possible proportions between $\max\left(\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}\left(J_k^P\right)\right), \max_{i=1}^{N-1}\left(J_{\alpha_i+1}^M\right)\right)$ and $\max\left(\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}\left(H_k^P\right)\right), \max_{i=1}^{N-1}\left(H_{\alpha_i}^M\right)\right)$, then for $\forall \omega^{(1)} > 0$ the following formula must be true.

$$\begin{cases} \tau(\omega) \le N, & if \quad \omega \ge \omega^{(1)}\left(\lambda^{(ID)}\right) \\ \tau(\omega) > N, & if \quad \omega < \omega^{(1)}\left(\lambda^{(ID)}\right) \end{cases} \tag{4-14}$$

where $\tau(\omega)$ is the minimum number of bins when setting the volume of bins as ω.

The proof of Property 4 can be seen as follows considering $\tau(\omega)$ is a non-increasing function.

Algorithm 4-4 Optimal Data Partition Algorithm via Basic Time Function

Input : K, N, λ, the functions of H_i^P, H_i^M, J_i^P, and J_i^M for $\forall 1 \leq i \leq K$

Output: The optimal partition number p of data parallelism

1 Get four matrices $Q^{(1)}$, $Q^{(2)}$, $Q^{(3)}$ and $Q^{(4)}$ to respectively represent the basic time functions of H_i^P, H_i^M, J_i^P, and J_i^M. For example, the element $Q_{ji}^{(1)}$ of the j-th row and the i-th column in $Q^{(1)}$ equals to $H_i^P(j)$

2 Get two $K \times N$ matrices ($P^{(1)}$ and $P^{(2)}$) and a one-dimensional matrix $P^{(3)}$ where

$$\begin{cases} P_{ij}^{(1)} = \begin{cases} 1, & if \quad \alpha_{i-1} + 1 \leq j \leq \alpha_i \\ 0, & others \end{cases} \\ P_{ij}^{(2)} = \begin{cases} 1, & if \quad j = \alpha_i \\ 0, & others \end{cases} \\ P_i^{(3)} = i - 1 \end{cases}$$

3 Calculate the matrices multiplication and obtain the maximum value of each row

$$\begin{cases} \rho_1(\lambda) = \max \begin{pmatrix} \max\left(Q^{(1)} \times P^{(1)}, dim = 1\right) \\ \max\left(Q^{(2)} \times P^{(2)}, dim = 1\right) \end{pmatrix} \cdot P^{(3)} \\ \rho_2(\lambda) = \max \begin{pmatrix} \max\left(Q^{(3)} \times P^{(1)}, dim = 1\right) \\ \max\left(Q^{(4)} \times P^{(2)}, dim = 1\right) \end{pmatrix} \cdot P^{(3)} \end{cases}$$

where $X \times Y$ means matrix multiplication, $\max(X, dim = 1)$ means maximizing each row of the matrix X, $\max(X, Y)$ means taking the maximum value of the homologous elements of the two matrices, $X \cdot Y$ means multiplying the homologous elements of two matrices

4 Calculate the matrix of running time as

$$Z = \text{sum}\left(Q^{(1)} + Q^{(2)} + Q^{(3)} + Q^{(4)}, dim = 1\right) + \rho_1(\lambda) + \rho_2(\lambda)$$

where $X + Y$ means taking the sum of the homologous elements of the two matrices

5 Obtain the index corresponding to the minimum value of matrix Z as the optimal partition number that

$$p = \arg\min(Z)$$

Proof: Because $\tau\left(\omega^{(1)}\left(\lambda^{(ID)}\right)\right) = N$, therefore $\tau(\omega) \leq N$ when $\omega \geq \omega^{(1)}\left(\lambda^{(ID)}\right)$. If $\exists \omega < \omega^{(1)}\left(\lambda^{(ID)}\right)$ **s.t.** $\tau(\omega) \leq N$, then the value $D^{min} < D^{mean}$. Then, the Algorithm 4-2 needs to be continued, which is in contradiction with $\lambda^{(ID)}$ is a convergence solution. Therefore, for $\forall \omega < \omega^{(1)}\left(\lambda^{(ID)}\right)$, $\tau(\omega) > N$. Thus, Property 4 is proved. ■

On the basis of Property 4, we can obtain a property as Property 5.

Property 5 For $\forall\lambda \in \Lambda$, $\omega^{(1)}(\lambda) \geq \omega^{(1)}\left(\lambda^{(ID)}\right) = \omega^{(1)}\left(\lambda^{(O)}\right)$ under the conditions of Property 4, i.e., $\lambda^{(ID)}$ is one theoretical optimal solution of problem $\min \omega^{(1)}$.

The proof of Property 5 is as follows using reduction to absurdity.

Proof: If $\exists\lambda \in \Lambda$ **s.t.** $\omega^{(1)}(\lambda) < \omega^{(1)}\left(\lambda^{(ID)}\right)$, then $\tau\left(\omega^{(1)}(\lambda)\right) > N$ according to the second formula of Eq 4-14. Because $\lambda \in \Lambda$ is a feasible solution of problem $\min \omega^{(1)}$, therefore $\tau\left(\omega^{(1)}(\lambda)\right) \leq N$. These two inequalities are contradictory. Thus, for $\forall\lambda \in \Lambda$, $\omega^{(1)}(\lambda) \geq \omega^{(1)}\left(\lambda^{(ID)}\right)$, i.e., $\lambda^{(ID)}$ is a theoretical optimal solution. ■

In fact, Property 4 and Property 5 are equivalent to reveal that the convergent solution of Algorithm 4-2 is the theoretically optimal solution. However, there are two indispensable conditions as follows for Property 4 and Property 5 to be tenable.

1. $\varepsilon \to 0$,
2. W has adequate weights, i.e., $\eta \to +\infty$.

In real computer programs, these two conditions are generally unable to be achieved, since the computer cannot generate infinitely small numbers. In realistic application of Algorithm 4-2 to solve the problem $\min \omega^{(1)}$, the error between $\omega^{(1)}\left(\lambda^{(ID)}\right)$ and $\omega^{(1)}\left(\lambda^{(O)}\right)$ is related to both ε and η, which is revealed by Property 6 through deduction.

Property 6 When $\varepsilon > 0$ and $\eta < +\infty$, the error ξ between $\omega^{(1)}\left(\lambda^{(ID)}\right)$ and $\omega^{(1)}\left(\lambda^{(O)}\right)$ is

$$0 \leq \xi = \omega^{(1)}\left(\lambda^{(ID)}\right) - \omega^{(1)}\left(\lambda^{(O)}\right) \leq \varepsilon + \frac{1}{\eta - 1}\omega^{(1)}\left(\lambda^{(O)}\right) \tag{4-15}$$

According to the process of Algorithm 4-2, we can present the proof of Property 6 as follows.

Proof: It can be set that the value $\max\left(\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}\left(J_k^P\right)\right), \max_{i=1}^{N-1}\left(J_{\alpha_i+1}^M\right)\right)$ and $\max\left(\max_{i=1}^{N}\left(\sum_{k=\alpha_{i-1}+1}^{\alpha_i}\left(H_k^P\right)\right), \max_{i=1}^{N-1}\left(H_{\alpha_i}^M\right)\right)$ corresponding to the optimal solution λ are respectively $\varrho_1(\lambda)$ and $\varrho_2(\lambda)$. When the Algorithm 4-2 reaches converge, the following relationships are tenable:

(1) $0 < \exists i \leq \eta$ **s.t.** $D^{max}\frac{i}{\eta} \geq \varrho_1\left(\lambda^O\right)$ and $D^{max}\frac{\eta - i}{\eta} \geq \varrho_2\left(\lambda^O\right)$;

(2) for $0 \leq \forall i \leq \eta$, $D^{min}\frac{i}{\eta} \leq \varrho_1\left(\lambda^O\right)$ or $D^{min}\frac{\eta - i}{\eta} \leq \varrho_2\left(\lambda^O\right)$;

(3) $0 \leq D^{max} - D^{min} \leq \varepsilon$.

If $0 \leq \exists i \leq \eta$ **s.t.** $D^{min}\frac{i}{\eta} \leq \varrho_1\left(\lambda^O\right)$ and $D^{min}\frac{\eta - i}{\eta} \leq \varrho_2\left(\lambda^O\right)$, then $D^{min} \leq \varrho_1\left(\lambda^O\right) + \varrho_2\left(\lambda^O\right) = \omega^{(1)}\left(\lambda^{(O)}\right)$. Because, $\omega^{(1)}\left(\lambda^{(ID)}\right) \leq D^{max} \leq D^{min} + \varepsilon$, thus $\omega^{(1)}\left(\lambda^{(ID)}\right) \leq \omega^{(1)}\left(\lambda^{(O)}\right) + \varepsilon$.

If for $0 \leq \forall i \leq \eta$, $D^{min}\frac{i}{\eta} \leq \varrho_1\left(\lambda^O\right) \wedge D^{min}\frac{\eta - i}{\eta} \leq \varrho_2\left(\lambda^O\right) = False$, then there

must $0 \leq \exists i < \eta$ **s.t.** $D^{min\frac{i}{\eta}} \leq \varrho_1\left(\lambda^O\right)$, $D^{min\frac{\eta-i}{\eta}} \geq \varrho_2\left(\lambda^O\right)$, $D^{min\frac{i+1}{\eta}} \geq \varrho_1\left(\lambda^O\right)$ and $D^{min\frac{\eta-i-1}{\eta}} \leq \varrho_2\left(\lambda^O\right)$. Therefore, $D^{min\frac{\eta-1}{\eta}} \leq \varrho_1\left(\lambda^O\right) + \varrho_2\left(\lambda^O\right)$. Thus, $\omega^{(1)}\left(\lambda^{(ID)}\right) \leq \frac{\eta}{\eta-1}\omega^{(1)}\left(\lambda^{(O)}\right) + \varepsilon$.

Thus, Property 6 is proved. ■

Property 6 demonstrates the optimality of IMD. From Eq 4-15 of Property 6, we can derive that

$$\lim_{\varepsilon\to 0,\eta\to+\infty}\left(\omega^{(1)}\left(\lambda^{(ID)}\right) - \omega^{(1)}\left(\lambda^{(O)}\right)\right) = 0 \tag{4-16}$$

which is consistent with the Property 5.

4.5.0.2 Analysis of Computational Complexity and Selection of Parameters ε and η

To further discuss the performance of Algorithm 4-2, we analyze its computational complexity $Cc^{(2)}$.

Since the dichotomy will reduce the search space by half each time, it needs $\log_2\left(\frac{D}{\varepsilon}\right)$ times to reach convergence where $D = D_1 + D_2 - \max\left(\max\left(H^P, J^P\right), \min\left(H^M, J^M\right)\right)$. The complexity of each time is determined by η and Algorithm 4-3, which can be obtained as $\mathbf{O}\left(\eta\left(2K\right)\right)$, where the complexity of Algorithm 4-3 is $\mathbf{O}\left(2K\right)$. Thus, the complexity of Algorithm 4-2 can be derived as

$$Cc^{(2)} = \mathbf{O}\left(2K\eta\log_2\left(\frac{D}{\varepsilon}\right)\right) \tag{4-17}$$

If the maximum allowable error is given as ξ where $\xi = \varepsilon + \frac{1}{\eta-1}\omega^{(1)}\left(\lambda^{(O)}\right)$ according to Eq 4-15 of Property 6, we can obtain the complexity of Algorithm 4-2 with respect of ε as

$$\begin{aligned} Cc^{(2)} &= \mathbf{O}\left(2K\left(\frac{\omega^{(1)}\left(\lambda^{(O)}\right)}{\xi-\varepsilon}+1\right)\log_2\left(\frac{D}{\varepsilon}\right)\right) \\ &\approx \mathbf{O}\left(2K\frac{\omega^{(1)}\left(\lambda^{(O)}\right)}{\xi-\varepsilon}\log_2\left(\frac{D}{\varepsilon}\right)\right) \end{aligned} \tag{4-18}$$

Minimizing $Cc^{(2)}$ is approximately equivalent to minimizing $\frac{1}{\xi-\varepsilon}\ln\left(\frac{D}{\varepsilon}\right)$ where ξ and D are given. It can be derived that when

$$\varepsilon\left(\ln D - \ln\varepsilon + 1\right) = \xi, \tag{4-19}$$

$Cc^{(2)}$ achieves minimum. Eq 4-19 presents a way to select the appropriate parameters ε and η to reduce the computational complexity under the given maximum error. Eq 4-

19 can be solved through various numeric methods such as Newton iteration method and secant method.

4.6 Experimental Results and Analysis

For the sake of the comprehensive evaluations of our proposed cross-search algorithm with improved multi-dimensional dichotomy (CSIMD), we carry out three groups of experiments from various aspects including:

(1) EX_1: evaluation of IMD to solve two-dimensional array segmentation problem;

(2) EX_2: evaluation of CSIMD in the CV-related networks to obtain optimal network division and data partition;

(3) EX_3: evaluation of CSIMD in the NLP-related networks to obtain optimal network division and data partition.

The networks and their corresponding dataset in experiment evaluations are listed in Table 4-2.

Table 4-2 The networks and their corresponding dataset in experiment evaluations

Category	DataSet	Networks
CV	Mnist	Self-designed CNNs
	ImageNet	VGG11
		VGG16
NLP	WikiText2	Self-designed Transformers
		GPT-1 with 12 transformer layers

Then, the experiments are launched on a cluster environment with multi servers. The center server acts as the management center. The configurations of the cluster environment are as follows.

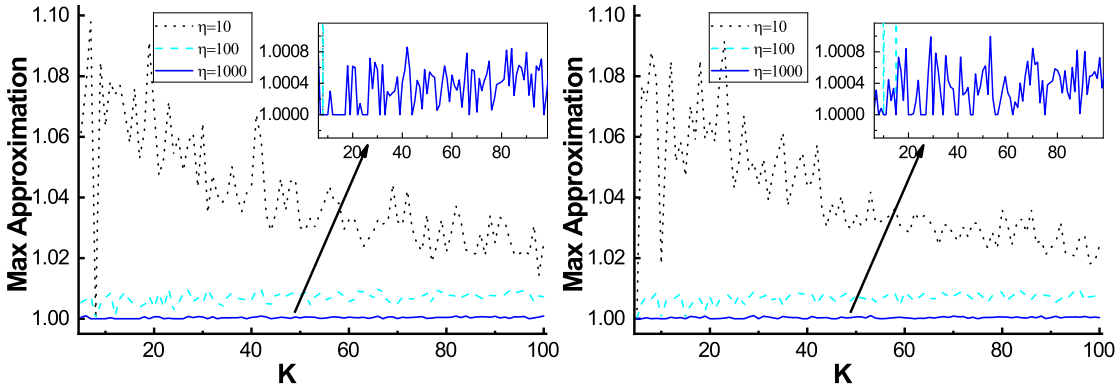
- Communication Network: 1 Gigabit or 10 Gigabit, full duplex;
- Program version: Python 3.7 + Pytorch 1.13.1;
- Servers × x:
  - CPU: Intel i9 10850K @ 3.6GHz, 10 cores;
  - SSD: Samsung 980 NVMe M.2 @ 1TB;
  - RAM: LPX 64GB DDR4 3200;
  - GPU: NVIDIA TESLA V100 @ 32GB × 2;

In order to observe the algorithm performance in the scenarios with more GPUs, we use process concatenation to simulate the parallel operation of neural networks, which can

reflect the qualitative comparison of various strategies in more GPUs by the results of that in a small number of GPUs.

4.6.1 Evaluation of Improved Multi-Dimensional Dichotomy

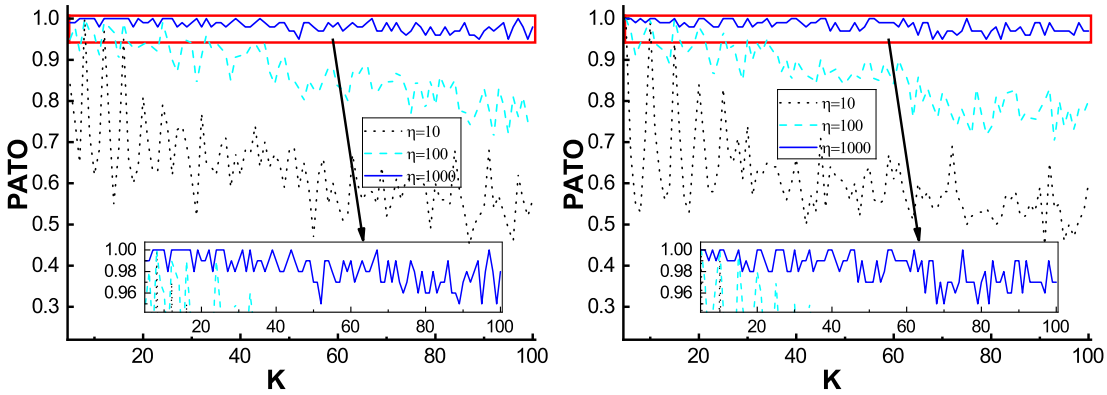
To evaluate the optimality of the improved dichotomy algorithm (IMD) in solving multi-dimensional array segmentation, we carry out experiments in two groups of scenarios with the small scale that $(K \in [5, 100], N = 4)$ and $(K \in [5, 100], N = 5)$ to observe the approximation and probabilities to achieve the theoretical optimization (PATO). In each combination of (K, N), we execute 100 instances by randomly generating the values of the array in a uniform distribution, i.e., $H_i^P, H_i^M, J_i^P, J_i^M \sim \mathrm{U}[50, 100]$; use an enumerative algorithm to obtain theoretical optimization solution; and then record the maximum approximation and PATO of each combination of (K, N) using IMD respectively with the number of weight group as 11, 101, and 1001. Then, we plot the approximation in Fig 4-4 and the corresponding PATO in Fig 4-5.



(a) $(K \in [5, 100], N = 4)$ (b) $(K \in [5, 100], N = 5)$

Figure 4-4 The maximum approximations for different numbers of weight groups when using improved multi-dimensional dichotomy algorithm to solve multi-dimensional array segmentation where each combination of (K, N) has 100 instances.

From the results of Fig 4-4, the maximum approximations of $\eta = 1000$ (within the range of $[1, 1.001]$) are lowest, followed by that of $\eta = 100$ (within $[1, 1.01]$) and $\eta = 10$ (within $[1, 1.1]$). As η increases, the maximum approximation decreases, indicating that the solution of the algorithm is closer to the theoretically optimal solution. The fluctuation range of the maximum approximations is approximately inversely proportional to η. Additionally, as K increases, the maximum approximation ratio does not show a signif-



(a) $(K \in [5, 100], N = 4)$ corresponding to Fig 4-4(a)

(b) $(K \in [5, 100], N = 5)$ corresponding to Fig 4-4(b)

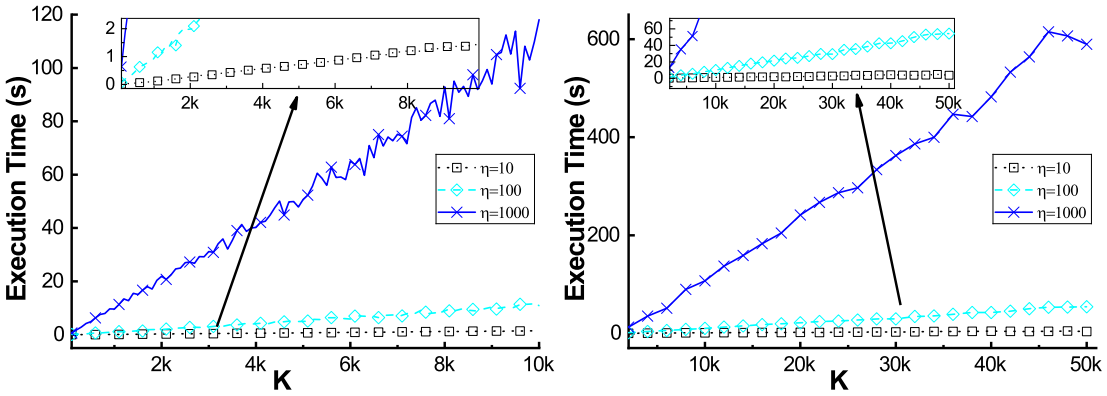
Figure 4-5 The probabilities achieving the theoretical optimization (PATO) for different numbers of weight groups when using improved multi-dimensional dichotomy algorithm to solve multi-dimensional array segmentation where each combination of (K, N) has 100 instances corresponding to Fig 4-4.

icant upward trend, which indicates that the approximation (or relative error) is mainly influenced by η and has no explicit positive or negative correlation with (K, N). These observations are consistent with the theoretical conclusion of Property 6, which can serve as supplementary proof of theory verifying the theoretical error of the IMD algorithm proposed in this chapter.

From Fig 4-5, the probability of the algorithm reaching the theoretical optimal solution increases with the η. When $\eta = 10$, the PATO is within the range of $(0.45, 1]$; that of $\eta = 100$ is $(0.70, 1]$; and $\eta = 1000$ is $(0.95, 1]$. This is also consistent with the conclusion of Property 6.

As concluded by Property 6, Fig 4-4 and Fig 4-5, when η approaches infinity, the approximation and PATO will both tend to 1. However, the actual selection of η needs to be based on the requirements of computational complexity and optimality. To further observe the complexity of IMD, we execute experiments in two groups of scenarios with the huge scale that $(K \in [100, 10000], N = 50)$ and $(K \in [2000, 50000], N = 1000)$. Each combination of (K, N) has 20 instances. Then, we plot the average execution time of IMD in Fig 4-6.

As shown in Fig 4-6, the average execution time of each η increases approximately linearly with K. As η increases, the gradient of the curve significantly increases. To further observe the relationship between slope and η, we linearly fit the curves in Fig 4-6, and then



(a) $(K \in [100, 10000], N = 50)$ (b) $(K \in [2000, 50000], N = 1000)$

Figure 4-6 The average execution time (computational complexity) for different sizes of weight groups when using improved multi-dimensional dichotomy algorithm to solve multi-dimensional array segmentation where each combination of (K, N) has 20 instances.

obtained their fitted slope and goodness-of-fit (R-square) as shown in Table 4-3.

Table 4-3 The fitted slope (FS) and goodness-of-fit to linearly fit the execution time of IMD corresponding to Fig 4-6.

Scenario	η	FS	R^2
$(K \in [10^2, 10^4], N = 50)$	10	0.00015	0.9977
	10^2	0.00112	0.9857
	10^3	0.01116	0.9887
$(K \in [2 \times 10^3, 5 \times 10^4], N = 10^3)$	10	0.00009	0.9459
	10^2	0.00112	0.9951
	10^3	0.01285	0.9938

The R-squares of all rows in Table 4-3 are larger than 0.9, indicating that the execution time can be statistically acceptable as proportional to K. The fitted slope in Table 4-3 is approximately proportional to η, which equals to that the execution time is proportional to η. These conclusions on computational complexity are consistent with Eq 4-18 in Section 4.5.0.2, which means that in the statistical sense, IMD is a linear time algorithm with controllable errors in solving the multi-dimensional array segmentation.

The experiments on approximation, PATO, and computational complexity in this subsection not only verify the theoretical derivation of IMD algorithm performance in Section 4.4.2, but also again demonstrate its optimality and rapidity. We set $\eta = 1000$ considering

the number of DNN's layers in the subsequent experiment is much smaller than the order of magnitude of K in the computational complexity experiment of this subsection.

4.6.2 Evaluation of CSIMD in the CV-related networks

To evaluate the CSIMD in obtaining the optimal network division and mini-batch partition, we first carry out the experiments in CV-related networks which mainly consist of convolutional layers and fully connected layers. The compared strategies are selected as:

(1) GPipe-R: GPipe with random divisions and partitions;

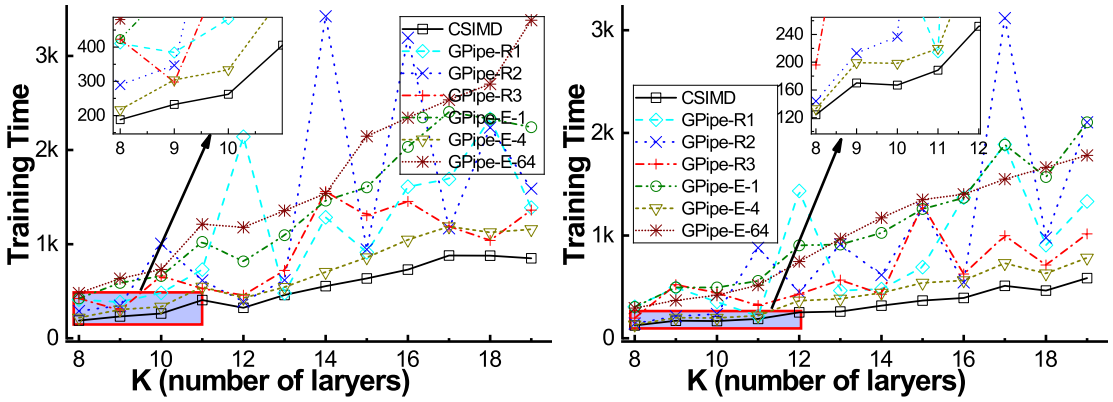
(2) GPipe-E: GPipe with equal divisions and fixed partitions.

Table 4-4 Detail of Self-designed CNNs

Layer	Types	Input Channel	Output Channel	Kernel
L_1	CNN	$In_1 = 1$	$Ou_1 = \mathrm{U}(20, 50)$	
L_x	CNN	$In_x = Ou_{x-1}$	$Ou_x = \mathrm{U}(20, 50)$	3×3
L_K	FC	$In_K = Ou_{K-1}$	$Ou_K = 10$	

In self-designed CNNs, as shown in Table 4-4, we continuously increase the number of CNN layers, where the number of output channels of each CNN layer is a random value generated by uniform distribution $\mathrm{U}(20, 50)$. Three fixed partitions of GPipe-E are set as 1, 4 and mini-batch size (BS). The mini-batch size is set as 64, the input figures are resized to 100×100. Then, we record the time for training 6400 images in two scenarios that $(K \in [8, 19], N = 4)$ and $(K \in [8, 19], N = 8)$ and plot results in Fig 4-7.

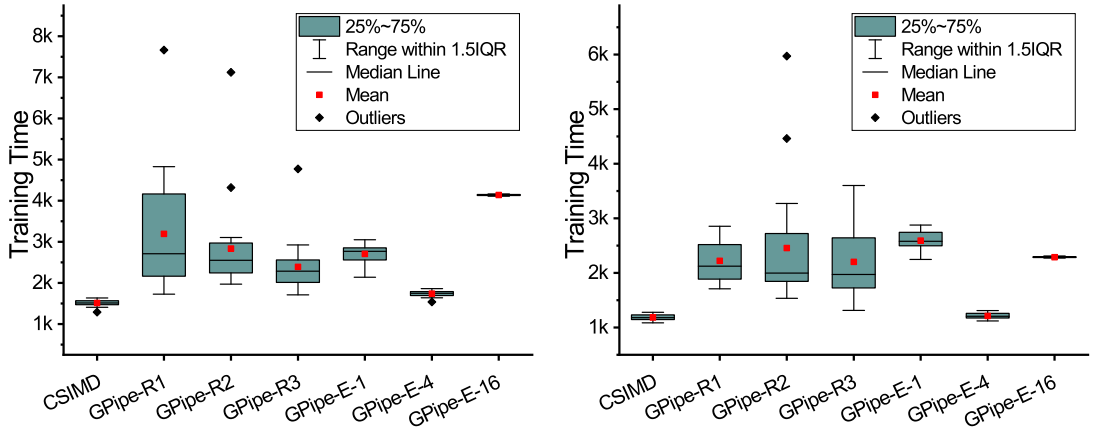
As shown in Fig 4-7, the curve of CSIMD remains the lowest, followed by GPipe-E-4. The results of Fig 4-7 demonstrate using CSIMD to search the network division and mini-batch partitions can further improve the performance of micro-batch-based pipeline parallelism. This also indicates that setting fixed network division and mini-batch partitions cannot adapt to all scenarios. The observation that GPipe-E-4 is better than GPipe-E-1 indicates performing certain mini-batch partitioning can improve training speed, which is consistent with the goal proposed by GPipe. However, as the number of partitions increases to 64, the training time actually becomes larger on the contrary. This indicates that the existence of one or more network layers has a nonlinear relationship between time (computing time or communication) and data volume (batch size), i.e. $H_i^P, H_i^M, J_i^P, J_i^M$ may not be inversely proportional to p.



(a) $(K \in [8, 19], N = 4)$ (b) $(K \in [8, 19], N = 8)$

Figure 4-7 The training time with respect of K (number of layers) for self-designed CNN in Table 4-4 to train 6400 images of Mnist under different network division and batch partition strategies where mini-batch size is 64, the resize of image is 100×100.

To further observe the statistical performance of CSIMD, we carry out experiments for self-designed CNNs in two scenarios that $(K = 20, N = 4)$ and $(K = 20, N = 8)$ where the mini-batch is set as 16 and each scenario has 20 instances. Then, we plot the boxchart of time to train 6400 images of Mnist in Fig 4-8. From Fig 4-8, CSIMD consistently outperforms comparison strategies, which shows CSIMD can stably obtain better network division and mini-batch partition schemes.

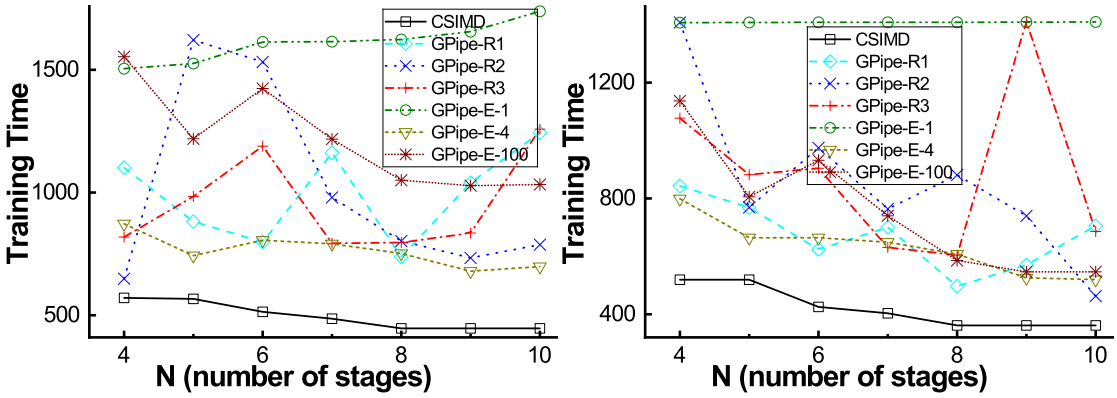


(a) $(K = 20, N = 4)$ (b) $(K = 20, N = 8)$

Figure 4-8 The boxchart of training time for self-designed CNN in Table 4-4 to train 6400 images of Mnist under different network division and batch partition strategies where mini-batch size is 16.

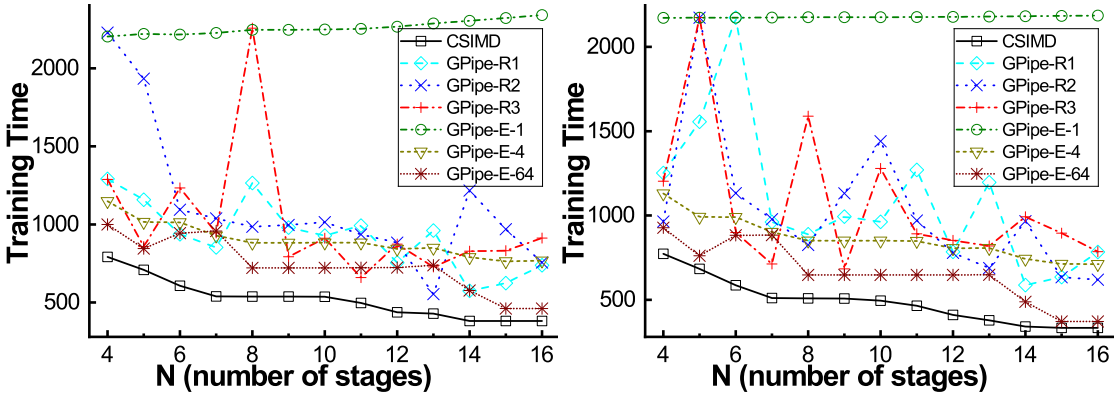
Additionally, to verify the performance of CSIMD in parallel training of existing

common CV-related neural networks, we execute experiments in VGG11 and VGG16 on the ImageNet dataset. We collect training performance data in two network bandwidth environments that are Gigabit bandwidth and 10 Gigabit bandwidth. As the layers of VGG are given, we select the number of stages (N, also the number of GPUs) as abscissa. Then, we plot their training time for training 6400 images in Fig 4-9 and Fig 4-10.



(a) Gigabit bandwidth (b) 10 Gigabit bandwidth

Figure 4-9 The training time with respect of N (number of stages) for VGG11 to train 6400 images of ImageNet under different network division and batch partition strategies where mini-batch size is 100, the resize of image is 224 × 224 and $N \in [4, 10]$).



(a) Gigabit bandwidth (b) 10 Gigabit bandwidth

Figure 4-10 The training time with respect of N (number of stages) for VGG16 to train 6400 images of ImageNet under different network division and batch partition strategies where mini-batch size is 64, the resize of image is 224 × 224 and $N \in [4, 16]$).

In Fig 4-9 and Fig 4-10, the curve of CSIMD also remains the lowest showing a

decreasing trend with the increase of N. While the curve of GPipe-E-1 increase with N, which is because adding the stages actually increases additional communication time. Comparing Fig 4-9(a) to Fig 4-9(b) and Fig 4-10(a) to Fig 4-10(b), as bandwidth increases, the training time of CSIMD and GPipe-E decreases, which indicates that improving communication speed can accelerate parallel training. However, the acceleration effect by improving communication for GPipe-E-1 with the longest training time is the most significant, and that for CSIMD is the smallest. This is because CSIMD minimizes the impact of communication time on the overall training speed by optimizing network division and mini-batch partition, which once again validates the advantage of cross-search when considering both nonlinear computation time and communication time. To our knowledge, there is currently no algorithm that can achieve this.

To quantitatively evaluate the improvement effect of CSIMD, we compile various experimental results on CV-related networks in this subsection and record the (minimum, average, maximum) ratios of training time between the comparison strategies and CSIMD under each network. Then, we list the ratios in Table 4-5.

Table 4-5 The (minimum, average, maximum) ratios of training time between the comparison strategies and CSIMD under each CV-related network

Strategies	Self CNNs	VGG11	VGG16	Average
GPipe-R1	(1.04, 2.29, 6.60)	(1.38, 1.81, 2.78)	(1.51, 2.01, 3.71)	2.037
GPipe-R2	(1.16, 2.24, 6.18)	(1.14, 2.02, 2.98)	(1.25, 2.12, 3.19)	2.128
GPipe-R3	(1.14, 1.76, 3.53)	(1.43, 2.04, 3.91)	(1.21, 2.10, 4.20)	1.967
GPipe-E-1	(1.66, 2.40, 3.69)	(2.64, 3.35, 3.90)	(2.78, 4.68, 6.55)	3.479
GPipe-E-4	(1.07, 1.24, 1.48)	(1.28, 1.53, 1.68)	(1.43, 1.80, 2.18)	1.522
GPipe-E-BS	(2.16, 2.93, 3.97)	(1.51, 2.11, 2.77)	(1.11, 1.40, 1.78)	2.146

As shown in Table 4-5, CSIMD can achieve 2.0×, 3.5×, 1.5× and 2.1× speedup on average respectively over GPipe-R, GPipe-E-1, GPipe-E-4 and GPipe-E-BS in the CV-related networks of the experiments in this chapter.

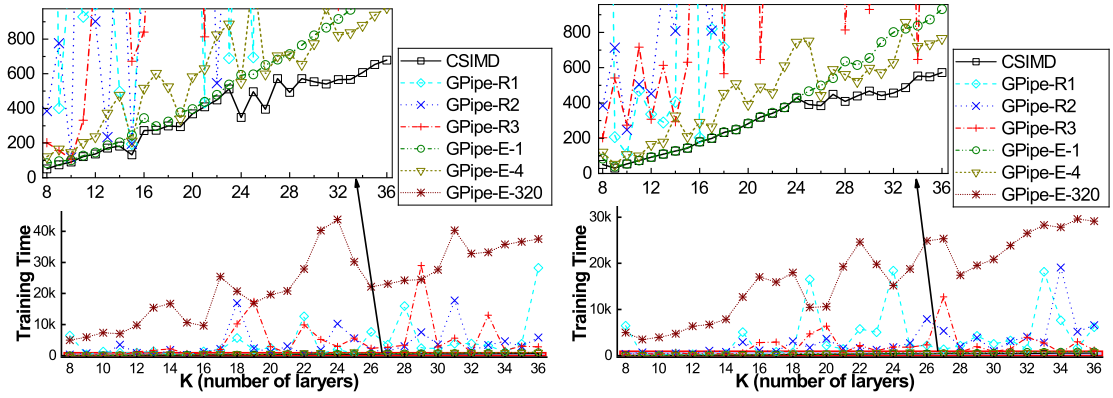
4.6.3 Evaluation of CSIMD in the NLP-related networks

Different from CNNs, transformer-based NLP models can not only be partitioned in batch size but also in tokens, i.e., Terapipe [180]. Since these two aspects are actually both data parallelism obeying the same cost model of Eq 4-7, we only present the verification for the partition of batch size.

We first use self-designed transformers as shown in Table 4-6 to execute incremental

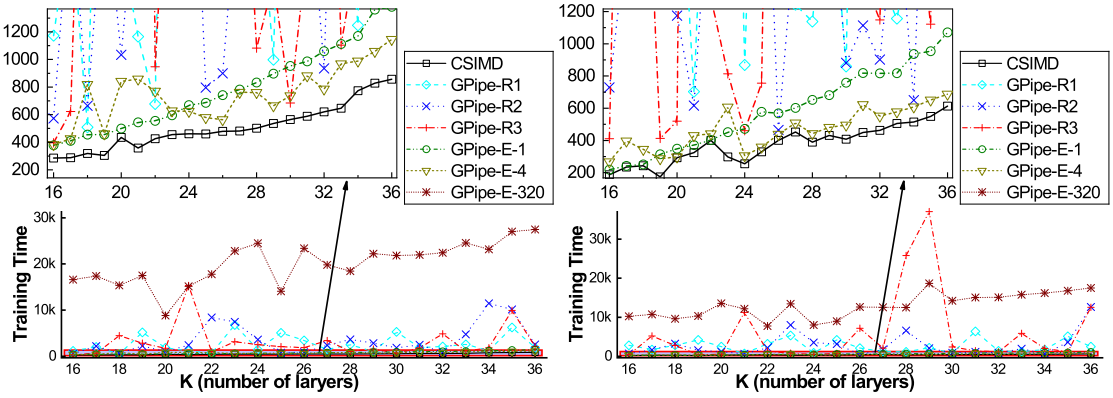
Table 4-6 Detail of Self transformers-based networks

Layer	Types
L_1	Embedding+Positional Encoding
L_x	Transformer Encoder Layer
L_K	FC



(a) $(K \in [8,36], N = 8)$, 1GB

(b) $(K \in [8,36], N = 8)$, 10GB



(c) $(K \in [16,36], N = 16)$, 1GB

(d) $(K \in [16,36], N = 16)$, 10GB

Figure 4-11 The training time with respect of K (number of layers) for self-designed transformer-based NLP networks in Table 4-6 to train 320 × 100 seqs of of WikiText-2 under different network division and batch partition strategies where the seq_length is 16, the mini-batch size is 320 and the embedding size is 10.

experiments. Similar to self-designed CNNs, we set $(K \in [8,36], N = 8)$ and $(K \in [16,36], N = 16)$. In self-designed transformers, the dataset is WikiText2, the seq_length is set as 16, the mini-batch size is 320 and the embedding size is 10. We use the random generation to randomly select the factor of embedding size (i.e., 10) as the number of heads

in each transformer layer and randomly generate dim_feedforward by U$[1, 5]$. Then, we plot the training time of each strategy for training 100 mini-batches in Fig 4-11.

In Fig 4-11, the curves of CSIMD are also lower than compared strategies, which verifies the feasibility and superiority of CSIMD in optimizing parallel training of transformer-based NLP DNN. GPipe-E-4 is better than GPipe-E-1 and GPipe-E-320, which indicates there are layers with nonlinear time (computing time or communication time) with respect of data volume in transformer-based NLP networks, otherwise, GPipe-E-320 should be better than GPipe-E-4. Because if following the linear assumptions of time functions, more partitions of mini-batch will cause a smaller training time. In addition, as the number of layers continues to increase, the advantages of CSIMD become increasingly apparent, which is because an increase in the number of network layers will bring more possibilities for network division, and CSIMD can find optimization solutions among these possibilities due to the optimality of IMD, which has been verified in the above.

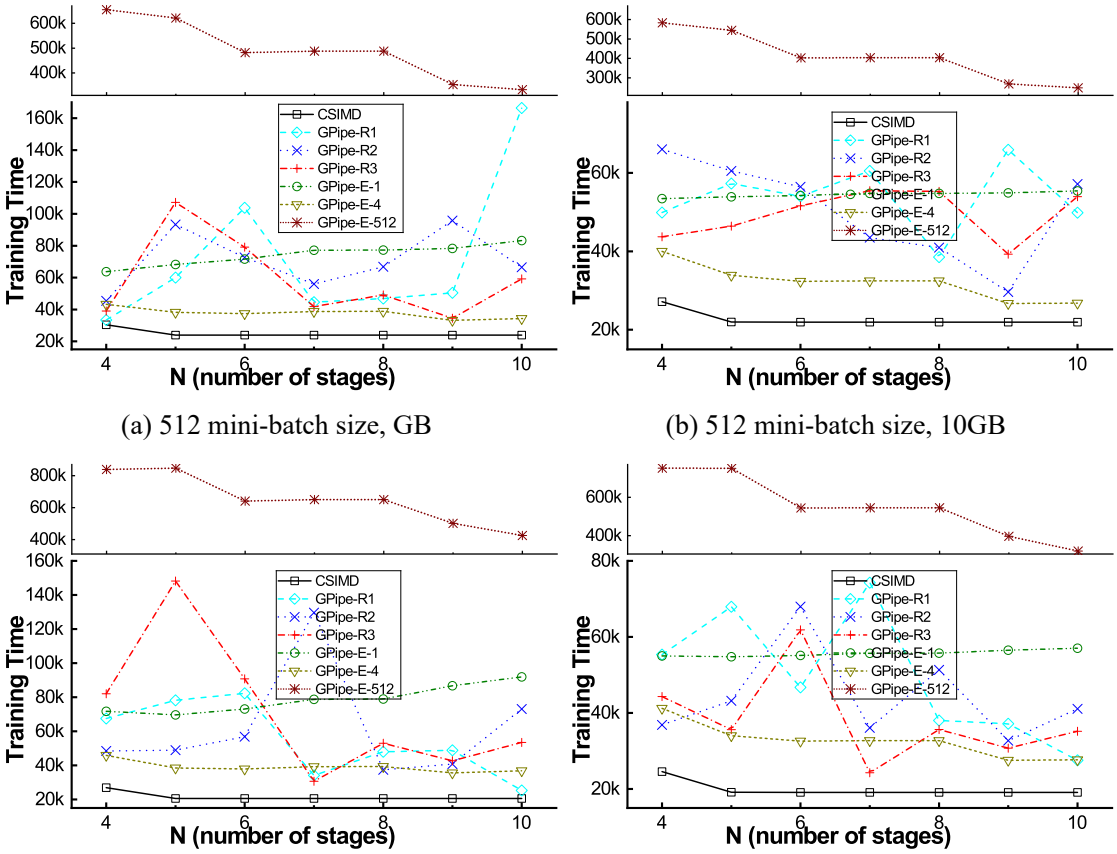
To further test the performance of CSIMD in frequently-used networks, we carry out experiments for GPT-1 in WikiText2, where the number of layers is 14 (12 transformer layers), embedding_dim is 768, number of heads is $768/4$, dim_feedforward is 768×4 and seq_length is 16. Similar to VGG, we use the number of stages as abscissa and plot the time to train 102400 sequences in Fig 4-12. Overall, CSIMD remains lowest in Fig 4-12, which demonstrates the superiority of CSIMD in frequently-used NLP networks. Unlike the results of VGG, after N is greater than 5, the training time of CSIMD remains almost unchanged and does not decrease as N increases. This is because using 5 GPUs in CSIMD search results is optimal, while more GPUs actually introduce additional communication time.

Similarly, in order to quantitatively evaluate the improvement effect of CSIMD in NLP networks, we compile results in this subsection and list the average ratios of training time between the comparison strategies and CSIMD under each network in Table 4-7.

As shown in Table 4-7, CSIMD can achieve 2.6×, 3.0×, 1.6× and 23.81× speedup on average respectively over GPipe-R, GPipe-E-1, GPipe-E-4 and GPipe-E-BS in GPT-1 of the experiments in this chapter.

4.7 Summary of this Chapter

Parallel training is now a hotspot in the development of artificial intelligence models. micro-batch-based pipeline parallelism, i.e., GPipe parallelism, is one of the popular



(a) 512 mini-batch size, GB

(b) 512 mini-batch size, 10GB

(c) 1024 mini-batch size, GB

(d) 1024 mini-batch size, 10GB

Figure 4-12 The training time with respect of N (number of stages) for GPT-1 to train 1024×100 seqs of WikiText-2 under different network division and batch partition strategies where number of layers is 14 (12 transformer layers), embedding_dim is 768, number of heads is $768/4$, dim_feedforward is 768×4 and seq_length is 16.

strategies to improve the performance of parallel training. In GPipe, two key factors are the division of the network layers and the partition of mini-batches. Additionally, due to the complexity of the parallel training process, some cost models make a lot of assumptions, which makes them deviate from the real scenario.

Considering these, we consider computing time and communication time simultaneously and consider them nonlinear to batch size. Then focusing on the scenario where the number of network layers is greater than the number of devices, we derive a time-cost model with respect of network division and data partitions, which can be regarded as the joint of multiple high-dimensional array segmentation problems. To obtain the joint solutions of network division and data partitions, we propose a cross-search algo-

Table 4-7 The average ratios of training time between the comparison strategies and CSIMD under each NLP-related network

Strategies	Self Transformers	GPT-1
GPipe-R1	11.32	2.60
GPipe-R1	8.74	2.58
GPipe-R1	9.96	2.47
GPipe-E-1	1.39	2.99
GPipe-E-4	1.54	1.60
GPipe-E-BS	53.87	23.81

rithm, in which we propose an improved multi-dimensional dichotomy to solve multi-dimensional array segmentation (CSIMD). Through theoretical derivation, we present and prove the theoretical performance of IMD. Finally, we carry out extensive experiments in CNN-based networks and transformer-based networks. Extensive experiments demonstrate our proposed CSIMD can obtain optimal network division and data partition schemes under GPipe parallelism. On average, our proposed CSIMD achieves $(2.0, 2.5)\times$ and $(1.5, 1.6)\times$ speedup respectively in CV- and NLP-related networks over GPipe-R and GPipe-E.

This chapter not only provides an algorithm to obtain optimal parallel schemes but also provides a complete idea that solving high-performance parallel training schemes from the perspective of solving optimization problems. By making certain improvements to the algorithm, it can also be applied to scenarios where the number of network layers is smaller than the number of devices. In the future, We consider continuing to explore more realistic time-cost models; as well as consider combining network layer division and mini-batch partition (also including token partition) to explore optimization algorithms for parallelism that adapt to more complex scenarios.

Chapter 5 Design of a novel architecture for parallel training of deep learning based on Unequal Date Partitioning and Dual-chromosome Genetic Algorithms

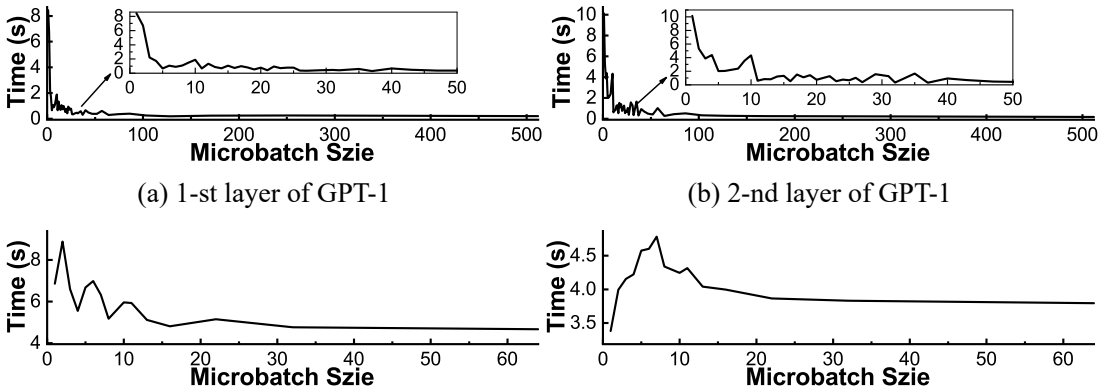
The increasing need for large-scale deep neural networks (DNN) has made parallel training an area of intensive focus. One effective method, microbatch-based pipeline parallelism (notably GPipe), accelerates parallel training in various architectures. However, existing parallel training architectures normally use equal data partitioning (EDP), where each layer's process maintains identical microbatch-sizes. EDP may hinder training speed because different processes often require varying optimal microbatch-sizes. To address this, we introduce UMPIPE, a novel framework for unequal microbatches-based pipeline parallelism. UMPIPE enables unequal data partitions (UEDP) across processes to optimize resource utilization. We develop a recurrence formula to calculate the time cost in UMPIPE by considering both computation and communication processes. To further enhance UMPIPE's efficiency, we propose the Dual-Chromosome Genetic Algorithm for UMPIPE (DGAP) that accounts for the independent time costs of forward and backward propagation. Furthermore, we present TiDGAP, a two-level improvement on DGAP. TiDGAP accelerates the process by simultaneously calculating the end time for multiple individuals and microbatches using matrix operations. Our extensive experiments validate the dual-chromosome strategy's optimization benefits and TiDGAP's acceleration capabilities. TiDGAP can achieve better training schemes than baselines, such as the local greedy algorithm and the global greedy-based dynamic programming. Compared to (GPipe, PipeDream), UMPIPE achieves increases in training speed: $(13.89, 11.09)\%$ for GPT1-14, $(17.11, 7.96)\%$ for VGG16 and $\geq (170, 100)\%$ for simulation networks.

5.1 Introduction

With the development of intelligent technology, the deep neural network (DNN) has been widely used in image processing [207,208], natural language processing (NLP) [209] and other fields. The existing DNNs include various structures such as convolutional neural network (CNN) [176], transformer layer-based network [180,181], graph neural network [210,211] et al. The emerging application trend of increasingly complex scenarios requires an increasing scale of parameters especially for NLP. T5 with 11B parameters [212],

FLAN with 137B parameters [170], and GPT-3 [171] with more than 175B parametersare some existing large-scale DNN model. For DNN which is generally trained on the Graph Processing Unit (GPU), the increasing parameters pose two significant bottlenecks, i.e., 1) Training large-scale DNN consumes too much time [174,175]; 2) The amount of data in the training process makes a single GPU or a few GPUs out of memory [195,213]. Therefore, parallel training on the distributed system (i.e., GPU cluster) becomes a key paradigm and a research hotspot to address the bottlenecks of large-scale DNN [168,174,176].

Data Parallelism (DP), Tensor Model Parallelism (TMP), and Pipeline Model Parallelism (PMP) are three foundational parallel modes for training DNNs on distributed systems [178,180,214,215]. PMP and TMP, in particular, often leverage a combination with DP to enhance their architectures. Renowned PMP architectures include GPipe [182], PipeDream [183], Dapple [184], and Hpipe [178], while TMP is represented by Megatron-LM [181] and Tofu [216]. Further, hybrid 3D parallelism models integrating DP, TMP, and PMP, like FOLD3D [217] and Merak [215], have significantly boosted parallel training performance. These existing architectures all rely on microbatch-based data parallelism. In the parallel training, forward propagation (FP) and backward propagation (BP) comprise computation and communication processes. A notable challenge is the "bubbles" or idle times each device experiences while waiting for preceding processes on other devices to complete [184,185]. Reducing bubbles and increasing the overlap between computation and communication is a key focus in optimizing parallel training [217]. Techniques such as dynamic programming [169], linear programming [203], and recurrence methods [196] are developed to mitigate bubbles.



(a) 1-st layer of GPT-1 (b) 2-nd layer of GPT-1

(c) 1-st layer of VGG16 (d) 2-nd layer of VGG16

Figure 5-1 Computation time to process 512 pieces data in realistic GPU devices.

Despite the above progress, Equal Data Partitioning (EDP) remains a significant lim-

itation across most parallelism. EDP means that all processes of computations and communications, must handle the same number of microbatches within a minibatch [218]. The limitation caused by EDP is prevalent in almost all existing parallel architectures that rely on microbatch-based data parallelism (e.g., GPipe [182], PipeDream [183], Dapple [184]. In the dynamic landscape of DNNs, layer heterogeneity significantly impacts computation and communication times, which are not necessarily proportional to the microbatch-size [177,190]. For example in Fig. 5-1, the layers of GPT-1 with transformer and VGG16 with CNN have variations in processing the same 512 pieces of data under different microbatch-sizes, especially when the microbatch-size is small (if computation time is proportional to data size, the curves should be straight lines parallel to X-axis). Additionally, different layers have not only different time consumption but also different inflection points, which reflects heterogeneity between layers. The heterogeneity and nonlinearity suggests that optimal partition numbers for different processes might vary. Hence, EDP may result in inefficiencies.

Motivated by this observation, we propose UMPIPE, a novel pipeline parallel structure utilizing unequal microbatches. This approach allows for unequal data partitioning (UEDP) across both computation and communication processes in DNNs. UMPIPE represents the first exploration into unequal partitioning within parallel training and effectively harnesses the heterogeneous time consumption characteristics of DNN layers. We develop a general recurrence formula and conduct an in-depth analysis of UMPIPE's optimality. Our findings indicate that UMPIPE's optimal scheme is at least as efficient, if not more so, than existing methods like GPipe.

To obtain the optimization scheme of UMPIPE, we propose a dual-chromosome genetic algorithm (DGAP), leveraging the independence of data partitions of FP and BP. Since the microbatch-sizes for processes in UMPIPE can be different, the formula of estimating training time in GPipe [191,192] does not apply to UMPIPE. To obtain explicit expressions of training time in UMPIPE is a challenge. It significantly complicates the computational process in determining UMPIPE's optimal scheme using recurrence formulas. To address this challenge, we propose a two-level improvement method based on matrix operations, which expedites calculations by simultaneously processing multiple individuals and microbatches. We integrate this method with DGAP to form Two-level Improved Dual-Chromosome Genetic Algorithm (TiDGAP).

Our contributions are summarized as follows:

(1) **UMPIPE**: Our approach considers not just the computation and communication simultaneously but also their nonlinear time consumption relative to microbatch-size. We propose UMPIPE, a UEDP-based pipeline parallelism framework. This innovation allows different microbatch-sizes for processes within DNNs, thereby accelerating parallel training. We derive a recurrence formula and conduct a comprehensive theoretical analysis for UMPIPE.

(2) **DGAP**: Recognizing that time costs for FP and BP are independent in UMPIPE, we develop Dual-Chromosome Genetic Algorithm (DGAP) to identify the optimal scheme. Our theoretical analysis of the dual-chromosome strategy highlights its statistical advantages in efficiently resolving UMPIPE's unique scheme requirements.

(3) **TiDGAP**: Addressing the challenges in calculating UMPIPE's training time due to UEDP, we conduct theoretical derivation and propose TiDGAP, a matrix operation-based two-level improved method to accelerate DGAP. TiDGAP can simultaneously calculate the end time corresponding to multiple individuals and multiple microbatches, substantially boosting DGAP's search capability.

(4) Extensive experiments demonstrate the fast speed and optimality of TiDGAP compared to baseline methods. Additionally, results confirm that UMPIPE achieves a faster training speed than baseline parallelism without compromising the training convergence of DNNs. Compared to (GPipe, PipeDream), UMPIPE achieves $(13.89, 11.09)\%$ improvement in GPT1-14, $(17.11, 7.96)\%$ in VGG16, and $\geq (170, 100)\%$ in simulation networks.

The rest of this chapter is organized as follows. We review the related work in Section 5.2. The BABYPIPE parallelism is presented in Section 5.3. The methodology is proposed in Section 5.4. The experiment evaluations are presented in Section 5.5. Finally, we conclude this chapter in Section 5.6.

5.2 Related Work

Data parallelism (DP), tensor model parallelism (TMP) and pipeline model parallelism (PMP) are three basic architectures to train large-scale DNNs [215,217].

Data level parallelism generally divides the data into multiple parts for time-sharing or equipment-sharing training [197,198]. Joshua Romero et al. [214] implemented a lightweight decentralized coordination strategy by utilizing a response cache to accelerate collective communication in data parallel training. Lei Guan et al. [219] proposed pd-

lADMM splitting the optimization problem into sub-problems to train the fully connected DNN in a data-parallel manner.

Tensor model parallelism divides layers of DNN into multiple parts from the dimension of tensor operations [178, 191, 199]. Based on model parallelism, Deepak Narayanan et al. [181] proposed Megatron-LM to divide the self-attention layer into a multi-tensor operation model which allowed the self-attention layer to be put on different devices.

Pipeline-related parallelism is an important method in parallel training, which is usually combined with DP and TMP. Gpipe [182] split the training minibatch into multiple microbatches and utilized the pipeline to train each part of the model on its corresponding distributed node. Terapipe [180] followed the pipeline of Gpipe and improved the pipelining granularity to reduce the pipeline bubbles of the transformer-based NLP model by proposing a new dimension, i.e. token dimension. Based on Gpipe, PipeDream [183] shifted the gradient backward-propagation (BP) of each microbatch earlier to the moment immediately after its last part of forward. Dapple [184] followed PipeDream and shifted the BP of other sub-model nodes to earlier besides that of the last sub-model node for the same minibatch [184].

Hybrid 3D parallelism, which integrates DP, PMP, and TMP, is currently an effective framework to achieve high training efficiency [215,217]. Fanxin Li et al. [217] proposed FOLD3D to slice a DNN into multiple segments, which allowed the computations in the same segment to be scheduled together. Zhiquan Lai et al. [215] proposed Merak that can automatically deploy 3D parallelism with an automatic model partitioner. Some other hybrid parallellism includes EffTra [192], ParaDL [220] and PipePar [221].

Other strategies to improve the performance of parallel training include momentum-driven adaptive synchronization model [222], NeoFlow (flexible framework for enabling efficient compilation) [223], pase parallelization [196], etc.

The given parallelism (i.e., parallel architecture) needs optimization methods to solve for the optimal parallel training schemes. Because the cost model was generally non-analytical or recursive, dynamic programming is a suitable and widely used method to obtain the optimal partition of data parallelism or model parallelism [169]. Some examples using dynamic programming include PipeDream [184], Dapple [184], Terapipe [180], EffTra [192], PaSE [196], PipePar [221] et al. Linear programming is also a frequent method in parallel training [176, 198, 203]. Some examples include NetPlacer [203], HGP4CNN [176], DPDA [198]. Other partition methods include off-the-shelf graph partitioning algorithms

[210], recurrence [196], grouping genetic algorithm [205], and near-optimal layer partition of local search method [195].

From the reviewed literature, how to construct a well-performed parallel training architecture to accelerate training speed by reducing pipeline bubbles and redundant communication is an urgent topic. The existing DP and TMP rely on equal partitioning. However, the optimal partitioning numbers of layers in DNN may be different, and equal partitioning will restrict the optimization boundary of parallelism. Referring to but different from the existing research, the notable feature of our proposed BABYPIPE is the unequal data partitioning (UEDP). In BABYPIPE, communication and computation time are non-negligible and are not necessarily proportional to the microbatch-size. These properties indicate that the targeted scenarios of our proposed BABYPIPE are far more complex and realistic. TiDGAP is also significantly different from existing algorithms due to the novelty and complexity of BABYPIPE. Although TiDGAP still relies on recurrence formulas, its calculation process has been resolved and improved through theoretical derivation based on matrix operations. GPU-based matrix operations enable TiDGAP to obtain schemes of BABYPIPE in the order of seconds.

5.3 Design and Formulations of a New Parallel Training Architecture (UMPIPE) for Deep Learning Models

Table 5-1 Notations and Descriptions.

Notation	Description
N	Number of stages
p_k	Number of microbatch in the k-th forward process
q_k	Number of microbatch in the k-th backward process
P	Minibatch-size
$F_i^P(p)$	The forward compute time of one microbatch in the i-th stage when partitioning the minibatch into p microbatches
$F_i^M(p)$	The forward communication time of one microbatch
$B_i^P(p)$	The backward compute time of one microbatch
$B_i^M(p)$	The backward communication time of one microbatch
$F_k(p_k)$	The time cost of one microbatch in the k-th forward process
E_{kj}	The end time of the j-th data of the k-th forward process
T_{ik}	The begin time of the i-th microbatch in the k-th process
$B_k(q_k)$	The time cost of one microbatch in the k-th backward process
R_{kj}	The end time of the j-th data of the k-th backward process

For the sake of the presentation of BABYPIPE's architecture and cost model, we list the notations in Table 5-1.

In one epoch of training DNNs, a dataset is divided into multiple minibatches and each minibatch needs to start after the previous minibatch ends. Due to the same calculation process for each minibatch, the time consumption for each minibatch is approximately equal. Therefore, we can focus on the training time for one iteration of one minibatch. It can be set that the minibatch-size of training DNN is P, which means training P pieces of data simultaneously in one minibatch. These P data can be set as $\{d_1, d_2, \ldots, d_P\}$. To reduce parallel training time, the existing microbatch-based pipeline parallelism (e.g., GPipe) partitions a minibatch into multiple microbatches. A main characteristic is that it partitions the input data of all layers into the same number of microbatches. In parallel training, a DNN is usually divided into multiple stages corresponding to the number of devices. In reality, each stage may contain one or more layers of DNN. Since this chapter mainly focuses on the improvement of parallel architecture in data partitioning and the algorithm for the optimization schemes, we set that each stage has only one layer of DNN, which does not affect the promotion of our proposals in this chapter. Our proposal is also applicable to scenarios where a stage contains multiple layers. Assuming a parallel DNN has 4 stages and the minibatch-size is $P = 6$, the data partitioning of a GPipe with 3 microbatches is shown in Eq 5-1 and that with 2 microbatches is shown in Eq 5-2. In Eq 5-1, the form $\begin{bmatrix} d_1 & d_2 \end{bmatrix}$ means calculating data d_1 and d_2 in one microbatch.

$$\begin{bmatrix} \begin{bmatrix} d_1 & d_2 \end{bmatrix} & \begin{bmatrix} d_3 & d_4 \end{bmatrix} & \begin{bmatrix} d_5 & d_6 \end{bmatrix} \\ \begin{bmatrix} d_1 & d_2 \end{bmatrix} & \begin{bmatrix} d_3 & d_4 \end{bmatrix} & \begin{bmatrix} d_5 & d_6 \end{bmatrix} \\ \begin{bmatrix} d_1 & d_2 \end{bmatrix} & \begin{bmatrix} d_3 & d_4 \end{bmatrix} & \begin{bmatrix} d_5 & d_6 \end{bmatrix} \\ \begin{bmatrix} d_1 & d_2 \end{bmatrix} & \begin{bmatrix} d_3 & d_4 \end{bmatrix} & \begin{bmatrix} d_5 & d_6 \end{bmatrix} \end{bmatrix} \tag{5-1}$$

$$\begin{bmatrix} \begin{bmatrix} d_1 & d_2 & d_3 \end{bmatrix} & \begin{bmatrix} d_4 & d_5 & d_6 \end{bmatrix} \\ \begin{bmatrix} d_1 & d_2 & d_3 \end{bmatrix} & \begin{bmatrix} d_4 & d_5 & d_6 \end{bmatrix} \\ \begin{bmatrix} d_1 & d_2 & d_3 \end{bmatrix} & \begin{bmatrix} d_4 & d_5 & d_6 \end{bmatrix} \\ \begin{bmatrix} d_1 & d_2 & d_3 \end{bmatrix} & \begin{bmatrix} d_4 & d_5 & d_6 \end{bmatrix} \end{bmatrix} \tag{5-2}$$

$$
\begin{aligned}
T_{GP} = & \sum_{k=1}^{N} \left(F_k^P + F_k^M + B_k^P + B_k^M \right) \\
& + (p-1) \max \left(\max_{1 \le k \le N-1} \left(B_k^P, B_k^M \right), B_N^P \right) \\
& + (p-1) \max \left(\max_{1 \le k \le N-1} \left(F_k^P, F_k^M \right), F_N^P \right)
\end{aligned} \tag{5-3}
$$

It can be set that the number of stages of DNN is N, the forward computation time of one microbatch in the i-th stage is $F_i^P(p)$ (abbreviated as F_i^P) when partitioning the mini-batch into p microbatches, the forward communication time of that is F_i^M, the backward computation time of that is B_i^P, and the backward communication time of that is B_i^M. Then, the total training time of DNN under GPipe parallelism for one iteration of one minibatch can be derived as Eq 5-3 considering computation and communication simultaneously.

5.3.1 Architecture of BABYPIPE

As GPipe makes a constraint that each layer has the same number of microbatches, some time-consuming processes (computation or communication) may limit the optimization of the total training time. To reduce the training time, BABYPIPE introduces unequal data partitioning (UEDP) into microbatch-based pipeline parallelism. In BABYPIPE, different processes in DNN can have different microbatch-sizes. UEDP is beneficial for layers flexibly selecting appropriate numbers of data partitions. Also, for the DNN with 4 stages and $P = 6$ minibatch-size, two examples of BABYPIPE can be seen in Eq 5-4 and Eq 5-5.

$$
\begin{bmatrix}
\begin{bmatrix} d_1 & d_2 & d_3 \end{bmatrix} & \begin{bmatrix} d_4 & d_5 & d_6 \end{bmatrix} \\
\begin{bmatrix} d_1 & d_2 \end{bmatrix} & \begin{bmatrix} d_3 & d_4 \end{bmatrix} & \begin{bmatrix} d_5 & d_6 \end{bmatrix} \\
\begin{bmatrix} d_1 & d_2 & d_3 \end{bmatrix} & \begin{bmatrix} d_4 & d_5 & d_6 \end{bmatrix} \\
\begin{bmatrix} d_1 & d_2 \end{bmatrix} & \begin{bmatrix} d_3 & d_4 \end{bmatrix} & \begin{bmatrix} d_5 & d_6 \end{bmatrix}
\end{bmatrix} \tag{5-4}
$$

$$\begin{bmatrix} \begin{bmatrix} d_1 & d_2 & d_3 \end{bmatrix} & \begin{bmatrix} d_4 & d_5 & d_6 \end{bmatrix} \\ \begin{bmatrix} d_1 & d_2 & d_3 & d_4 & d_5 & d_6 \end{bmatrix} & \\ [d_1] & [d_2] & [d_3] & [d_4] & [d_5] & [d_6] \\ \begin{bmatrix} d_1 & d_2 \end{bmatrix} & \begin{bmatrix} d_3 & d_4 \end{bmatrix} & \begin{bmatrix} d_5 & d_6 \end{bmatrix} & \end{bmatrix} \tag{5-5}$$

In Eq 5-4, the number of microbatches of each stage is respectively $(2,3,2,3)$. That in Eq 5-5 is $(2,1,6,3)$. In addition to varying the number of partitions between different stages, the partitions for computation and communication within the same stage can also be unequal. The partitions for forward and backward propagations in the same layer can also be unequal. For the convenience of discussion, we regard all computations and communications in training DNN to belong to a set of processes. Based on the characteristics of training DNN, there are $4N-2$ processes for N stages, marked as $\kappa = \langle \kappa_1, \kappa_2, \ldots, \kappa_{4N-2} \rangle$ according to the order of execution. Therefore, we can obtain that for $k \leq 2N-1$, κ_k corresponds to the forward computation of the $(k+1)/2$-th stage if k is an odd number, else it is the forward communication between the $k/2$-th and the $\left(\frac{k}{2}+1\right)$-th stages. For $k > 2N-1$, κ_k is the backward computation of the $(k-2N+2)/2$-th stage when $k-2N+1$ is an odd number, otherwise it is the backward communication between the $(k-2N+1)/2$-th and the $(k-2N-1)/2$-th stages. It can be set that the time cost of one microbatch in the k-th forward process is $F_k(p_k)$ and that in the k-th backward process is $B_k(p_k)$. Assuming the number of microbatches in the k-th forward process is p_k and that in the k-th backward process is q_k, the relationships can be obtained as Eq 5-6.

$$\begin{cases} F_k(p_k) = \begin{cases} F^P_{\frac{k+1}{2}}(p_k), & k \mod 2 = 1 \\ F^M_{\frac{k}{2}}(p_k), & otherwise \end{cases} \\ B_k(q_k) = \begin{cases} B^P_{N-\frac{k-1}{2}}(q_k), & k \mod 2 = 0 \\ B^M_{N-\frac{k}{2}}(q_k), & otherwise \end{cases} \end{cases} \tag{5-6}$$

With p_k and q_k, we can give the mathematical definitions of GPipe and BABYPIPE to highlight their differences.

- In GPipe, p_k and q_k must satisfy that for $0 \leq \forall i \leq \forall j \leq 2N-1$, $p_i = p_j = q_i = q_j = p$.

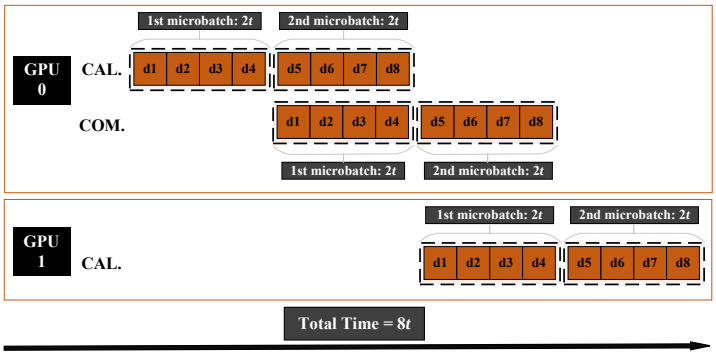
- In BABYPIPE, it is allowed that $\exists i,j\ s.t.\ p_i \neq p_j$, or $q_i \neq q_j$, or $p_i \neq q_j$. It is also allowed that for $0 \leq \forall i \leq \forall j \leq 2N-1$, $p_i = p_j = q_i = q_j$.

This indicates that GPipe is a set of BABYPIPE's special cases when $p_i = p_j = q_i = q_j$ for $0 \leq \forall i \leq \forall j \leq 2N-1$.

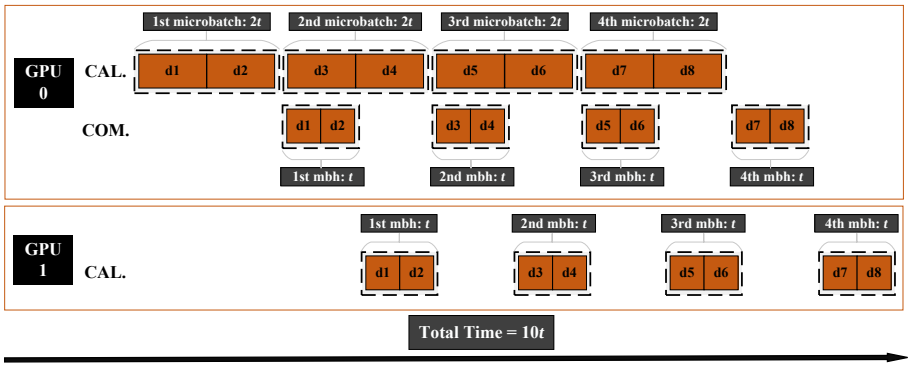
As mentioned above, some time-consuming processes in GPipe slow down the entire training. It is mainly because F_k (or B_k) and p_k (or q_k) are not necessarily inversely proportional in actual training. In some cases, there may exist a number $p\ s.t.$ that $F_k(x) = F_k(y)$ for $\forall x > y \geq p$. A similar phenomenon appears in B_k. If different processes choose different partitions in this case (i.e., applying BABYPIPE), it may further reduce the total training time, which will be better than all partition schemes of GPipe. For a more concrete explanation, we will list two sets of examples with two stages, shown as Fig 5-2 and Fig 5-3.

The first example is for the forward propagation of a network whose minibatch-size is $P = 8$ and time functions satisfy that $F_1^P(2) = F_1^P(4) = 2t$, $F_1^M(2) = 2F_1^M(4) = 2t$, $F_2^P(2) = 2F_2^P(4) = 2t$. We draw the timelines of two schemes for GPipe and one for BABYPIPE in Fig 5-2. Fig 5-2(a) is for the GPipe with 2 microbatches in one minibatch where $p_1 = p_2 = p_3 = 2$ and Fig 5-2(b) is for that with 4 microbatches where $p_1 = p_2 = p_3 = 4$. The best scheme of GPipe is Fig 5-2(a) with $8t$ training time for one minibatch. Based on Fig 5-2(a), the communication in the first stage and computation in the second stage can be further improved. If further dividing the microbatches in the communication of the first stage, the computation process in the second stage can start earlier. The scheme of BABYPIPE in Fig 5-2(c) combines the schemes of Fig 5-2(a) and Fig 5-2(b) and takes the scheme as $p_1 = 2$ and $p_2 = p_3 = 4$. Then, the training time under BABYPIPE parallelism in Fig 5-2(c) is $7t$ better than the best scheme of GPipe, accelerating the training speed by 14.29%.

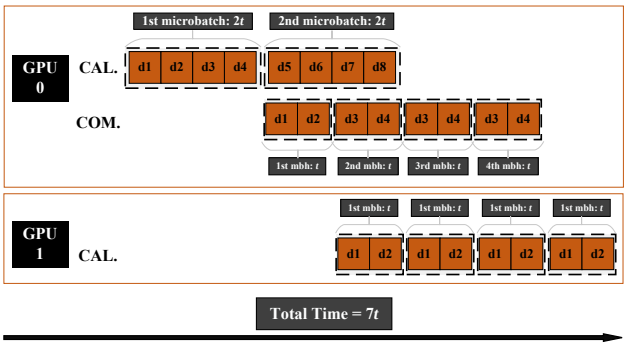
The example of Fig 5-2 discusses the advantages of unequal data partitioning between different processes in forward propagation. The next example discusses UEDP between forward propagation and backward propagation. It can be set the minibatch-size is $P = 8$ and time functions satisfy that $F_1^P(2) = F_1^P(4) = 2t$, $F_1^M(2) = F_1^M(4) = 2t$, $F_2^P(2) = F_2^P(4) = 2t$ and $B_1^P(2) = 2B_1^P(4) = 2t$, $B_1^M(2) = 2B_1^M(4) = 2t$, $B_2^P(2) = 2B_2^P(4) = 2t$. Then, we can draw the timelines for two schemes of GPipe and one for BABYPIPE in Fig 5-3. The scheme in Fig 5-3(a) takes 2 microbatches in one minibatch with $16t$ training time, and Fig 5-3(b) takes 4 microbatches with $18t$ training time. It can be noted that the



(a) GPipe (2 microbatches): $8t$



(b) GPipe (4 microbatches): $10t$

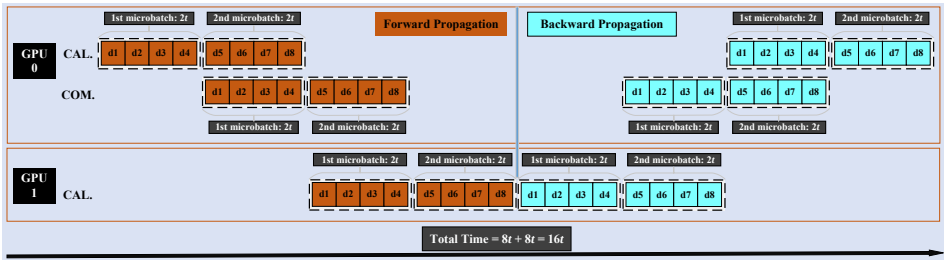


(c) BABYPIPE (2-4-4 microbatches): $7t$

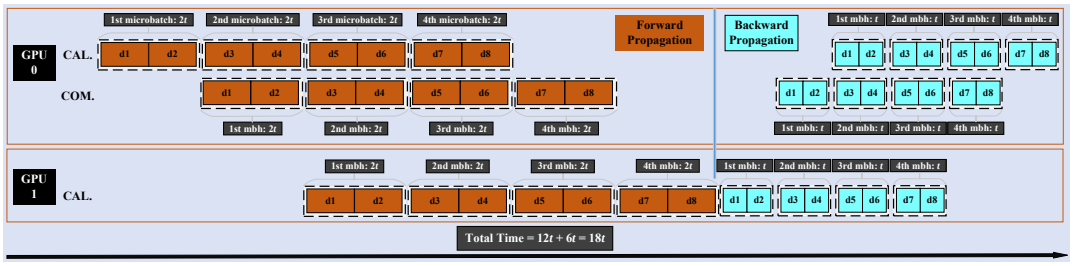
Figure 5-2 Forward propagation timeline for the GPipe and BABYPIPE in two stages of GPUs where: $F_1^P(2) = F_1^P(4) = 2t$, $F_1^M(2) = 2F_1^M(4) = 2t$, $F_2^P(2) = 2F_2^P(4) = 2t$.

training time for the backward propagation in Fig 5-3(b) is $6t$ less than that in Fig 5-3(a). Thus, if combining the scheme of forward propagation in Fig 5-3(a) and the scheme of backward propagation in Fig 5-3(b), a better scheme of BABYPIPE can be obtained as Fig 5-3(c). According to Fig 5-3(c), the training time under BABYPIPE parallelism is $14t$ better than the best scheme of GPipe.

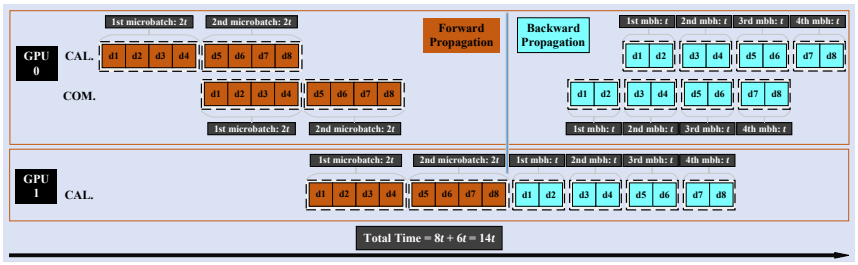
These two examples demonstrate that unequal data partitioning of BABYPIPE can



(a) GPipe (2 microbatches): $8t + 8t = 16t$



(b) GPipe (4 microbatches): $12t + 6t = 18t$



(c) BABYPIPE (2 microbatches for forward propagation and 4 partitions for backward propagation): $8t + 6t = 14t$

Figure 5-3 Timeline with forward propagation and backward propagation for the GPipe and BABYPIPE in two stages of GPUs where: $F_1^P(2) = F_1^P(4) = 2t$, $F_1^M(2) = F_1^M(4) = 2t$, $F_2^P(2) = F_2^P(4) = 2t$ and $B_1^P(2) = 2B_1^P(4) = 2t$, $B_1^M(2) = 2B_1^M(4) = 2t$, $B_2^P(2) = 2B_2^P(4) = 2t$.

introduce better solutions than EDP, which can further improve the speed of parallel training. Data parallelism is one of the foundations of most existing parallel architectures. Therefore, UEDP, which improves data parallelism, is significant for parallel training.

5.3.2 Formulas for BABYPIPE

Due to the introduction of UEDP, the training time of DNN using BABYPIPE is not as easy to derive as using GPipe. However, solving optimization solutions requires evaluation of the optimized solutions inevitably. Therefore, in this section, we present a recurrence formula and provide a time-consuming recurrence calculation algorithm.

Setting the begin time of the i-th microbatch in the k-th process is T_{ik}, a recurrence formula for BABYPIPE is

$$T_{ijk} = \max\left(T_{x_ij(k-1)} + F_{k-1}, T_{(i-1)jk} + F_k\right) \tag{5-7}$$

where x_i satisfies

$$\frac{x_i - 1}{p_{k-1}} < \frac{i}{p_k} \leq \frac{x_i}{p_{k-1}} \tag{5-8}$$

i.e., $x_i = \lceil (ip_{k-1})/p_k \rceil$ where $\lceil\rceil$ is upward rounding function.

It is complex to derive its analytical expression of the training time. In practice, we can calculate the end time of each data based on the dependencies between each data in each layer of DNN. We can set the end time of the j-th data in the k-th process as E_{kj}. Then, the dependencies of E_{kj} can be derived as:

$$E_{kj} = \max\left(E_{(k-1)z_{kj}}, E_{ky_{kj}}\right) + F_k \tag{5-9}$$

where

$$\begin{cases} z_{kj} = \min\left(p_k \cdot \left\lceil \dfrac{j}{p_k} \right\rceil, P\right) \\ y_{kj} = p_k \cdot \left(\left\lceil \dfrac{j}{p_k} \right\rceil - 1\right) \end{cases} \tag{5-10}$$

According to the Eq 5-9, E_{kj} is only determined by $E_{(k-1)j}$ and $E_{ky_{kj}}$. Thus, we can use the recurrence formula with two layers of loops to calculate the training time under BABYPIPE parallelism as Algorithm 5-1 assuming F_k is known.

Algorithm 5-1 Recurrence algorithm for the forward training time of one mini-batch under BABYPIPE

Input : $F_k(p_k)$ and p_k for $\forall k$, minibatch-size P
Output: E_{kj} for $1 \leq k \leq 2N-1$ and $1 \leq j \leq P$

```
Initial E_kj = 0 for 0 ≤ k ≤ 2N − 1 and 0 ≤ j ≤ P
for k ∈ [1, 2N − 1] do
    for j ∈ [1, P] do
        Calculate z_kj and y_kj according to Eq 5-10
        Calculate E_kj according to Eq 5-9
```

As the backward propagation under BABYPIPE parallelism has a similar process with forward propagation, Algorithm 5-1 also applies to backward propagation, which only needs to replace E_{kj} by R_{kj}, F_k by B_k and p_k by q_k. Although Algorithm 5-1 can be utilized to obtain the training time under BABYPIPE parallelism, it will consume a plethora of computational time due to its two layers of loops, especially in the genetic algorithm. In the next section, when introducing the optimization algorithm for solving BABYPIPE's scheme, we will detail the improvement methods for Algorithm 5-1 to accelerate the calculation of training time.

When F_k and B_k are given for $\forall k$ and $\forall p_k$, $\langle p_1, p_2, \ldots, p_{2N-1}, q_1, q_2, \ldots, q_{2N-1} \rangle$ will determine the final training time under BABYPIPE parallelism. Therefore, $\langle p_1, p_2, \ldots, p_{2N-1}, q_1, q_2, \ldots, q_{2N-1} \rangle$ can be regarded as the solution of BABYPIPE's scheme. Then, we can obtain the optimization problem of BABYPIPE as

$$\min \omega = E_{(2N-1)P} + R_{(2N-1)P} \tag{5-11}$$

where $E_{(2N-1)P}$ is determined by $\langle p_1, p_2, \ldots, p_{2N-1} \rangle$ and $R_{(2N-1)P}$ by $\langle q_1, q_2, \ldots, q_{2N-1} \rangle$.

5.3.3 Theoretical Analysis of Basic Functions

The key to solving the problem of minimizing the total training time is to find the optimal partition numbers $\langle p_1, p_2, \ldots, p_{2N-1}, q_1, q_2, \ldots, q_{2N-1} \rangle$. Thus, the functions of $F_i^P(p)$, $F_i^M(p)$, $B_i^P(p)$ and $B_i^M(p)$ are crucial. In realistic, these functions $F_i^P(p)$, $F_i^M(p)$, $B_i^P(p)$ and $B_i^M(p)$ are nonlinear to the microbatch-size. It means they aren't inversely proportional to partition number p.

In realistic DNNs, the communication time is approximately proportional to data size only when the data size is far more than bandwidth Bw. However, the communication cost is approximately constant when the data size is less than a specific value (specific bytes). This is because each communication requires a fixed minimum cost. Thus, a piecewise function is feasible to fit the relationship between data size Dz and communication cost Mc, shown as Eq 5-12.

$$Mc = \begin{cases} C_2 \cdot Dz, & Dz \geq C_3 \\ C_1, & D_z < C_3 \end{cases} \tag{5-12}$$

where C_1, C_2 and C_3 are bandwidth Bw-related constants.

For the i-th layer of DNN, assuming the output data size of one minibatch as Dz_i, the

function F_i^M of one microbatch for FP can be written as Eq 5-13.

$$F_i^M(p) = \begin{cases} C_2 \cdot \dfrac{Dz_i}{p}, & p \leq \dfrac{Dz_i}{C_3} \\ C_1, & p > \dfrac{Dz_i}{C_3} \end{cases} \tag{5-13}$$

With the same constants C_1, C_2 and C_3, the function of $B_i^M(p)$ is similar to $F_i^M(p)$.

For computation time, there is a similar phenomenon to communication time, i.e., when data is less than a specific size, the computation time will be relatively unchanged with the data size. However, since the operation of the model is tensor calculation (matrix calculation) involving multiple dimensions, it is not proportional to the size of one dimension. For data parallelism based on microbatch, the inflection point is related to its corresponding layer and the performance of devices. It can set the inflection point of the i-th layer as γ_i^F for forward computation and γ_i^B for backward computation. When devices are given, the least time required for computation of a small amount is also given, which can be set as P_1. The functions of computation time with respect to partition number p can be represented as Eq 5-14.

$$\begin{cases} F_i^P(p) = \begin{cases} \dfrac{\beta_i^F}{p}, & p \leq \gamma_i^F \\ P_1, & p > \gamma_i^F \end{cases} \\ B_i^P(p) = \begin{cases} \dfrac{\beta_i^B}{p}, & p \leq \gamma_i^B \\ P_1, & p > \gamma_i^B \end{cases} \end{cases} \tag{5-14}$$

where β_i^F and β_i^B are the time-cost for FP and BP when $p = 1$.

For a homogeneous GPU cluster configured for a given hardware environment, the constants C_1, C_2, C_3 and P_1 are stationary and can be obtained by statistics of multiple communication experiments. The output data sizes are relatively explicit and can be obtained by bytes of data. The β_i^F, β_i^B γ_i^F and γ_i^B are implicitly related to the intrinsic matrix operation of the layers. However, these inflection points are relatively stable and available based on statistical methods.

5.3.4 Analysis for Optimality of BABYPIPE

According to properties of basic functions and GPipe, we can obtain a theorem about the optimal partition number.

Theorem 5.1 Assuming the set of optional partitions is $\left\langle \beta_1, \beta_2, \ldots, \beta_\eta \right\rangle$ where $\beta_1 = 1 < \beta_2 < \cdots < \beta_\eta = P$ and the optimal partition number of GPipe is β_α where $\alpha < \eta$, that means $p_k = \beta_\alpha$ for $\forall k$, there must exist γ *s.t.* $F_\gamma(\beta_\alpha) = F_\gamma(\beta_{\alpha+1})$.

Theorem 5.1 can be proved by reduction to absurdity.

Proof: According to the property of basic function, the following relationship is tenable for $\forall\gamma$.

$$F_\gamma(\beta_\alpha) \geq F_\gamma(\beta_{\alpha+1})$$

If $F_\gamma(\beta_\alpha) > F_\gamma(\beta_{\alpha+1})$, it can be derived that the partition $\beta_{\alpha+1}$ is better than β_α. It contradicts with that β_α is the optimal partition number. Thus, Theorem 5.1 is proved.

■

With Theorem 5.1, we can obtain a relationship between the optimal solutions of GPipe and BABYPIPE.

Theorem 5.2 (1) If $F_\gamma(\beta_{\alpha-1}) > F_\gamma(\beta_\alpha) = F_\gamma(\beta_{\alpha+1})$ for $\forall\gamma$, then $p_k = \beta_\alpha$ is also the optimal solution of BABYPIPE, and the best scheme of GPipe equals to that of BABYPIPE. (2) If p_k is the optimal solution of BABYPIPE and better than GPipe, then there must exist $i \neq j$ *s.t.* $p_i \neq p_j$.

Theorem 5.2 reveals that the theoretical optimal training scheme of BABYPIPE must be no worse than that of GPipe.

Proof: For the first property of Theorem 5.2, we use the inductive method to prove it. Assuming the number of processes in DNN is M, for $M = 1$, the first property of Theorem 5.2 is obviously tenable. For $M = 2$, it can be assumed that the optimal solution of BABYPIPE is $\langle p_1 = a_1, p_2 = a_2 \rangle$. As the GPipe belongs to BABYPIPE, the solution of BABYPIPE must be no worse than GPipe. When $a_1 = a_2$, the optimal solution of BABYPIPE is also that of GPipe. In this case, Theorem 5.2 is tenable. For $a_1 \neq a_2$, when only one of a_1, a_2 equals to β_α, it can be set $a_1 = \beta_\alpha$ and $a_2 \neq \beta_\alpha$. As $F_\gamma(\beta_{\alpha-1}) > F_\gamma(\beta_\alpha) = F_\gamma(\beta_{\alpha+1})$, thus it can be derived that $\langle p_1 = a_1, p_2 = a_2 \rangle$ is worse than $p_1 = \beta_\alpha = p_2$. When $a_1 \neq \beta_\alpha$ and $a_2 \neq \beta_\alpha$, it can be proved that $p_1 = a_1, p_2 = a_2$ must be worse than $(p_1 = \beta_\alpha, p_2 = a_2)$ or $(p_1 = a_1, p_2 = \beta_\alpha)$. $(p_1 = \beta_\alpha, p_2 = a_2)$ or $(p_1 = a_1, p_2 = \beta_\alpha)$ are both worse than $(p_1 = \beta_\alpha, p2 = \beta_\alpha)$. Thus, $(p_1 = a_1, p_2 = a_2)$ is worse than $(p_1 = \beta_\alpha, p_2 = \beta_\alpha)$. Therefore, the first property is tenable for $M = 2$.

Assuming the first property of Theorem 5.2 is tenable for $\forall M \leq K - 1$. For $M = K$, we can assume there exists a solution of BABYPIPE better than $p_k = \beta_\alpha$. Thus, there

must exist one piece of data at one layer with an earlier ending time in the scheme of BABYPIPE than that of GPipe and its previous microbatches all have the same ending time in BABYPIPE as that of GPipe. It can be set that the index of this layer is $l \leq K$. Thus, for $\forall k \leq l-1$, $p_k = \beta_\alpha$ and $p_l \neq \beta_\alpha$ according to the assumption that Theorem 5.2 is tenable for $\forall M \leq K-1$. As $F_\gamma(\beta_{\alpha-1}) > F_\gamma(\beta_\alpha) = F_\gamma(\beta_{\alpha+1})$, thus $\forall p_l \neq \beta_\alpha$ the ending time of any piece data in the l-th layer must be not earlier than that of $p_l = \beta_\alpha$. Thus, Theorem 5.2 is tenable for $M = K$. Thus, the first property is proved.

As the first property is tenable, the second property clearly holds, otherwise, it contradicts the condition that the solution of BABYPIE is better than that of GPipe.

■

In fact, all the training schemes of GPipe are special cases of BABYPIPE, which means the schemes of BABYPIPE include that of GPipe. Therefore, our subsequent algorithm considers using the optimal solution of GPipe as the initial solution. This setting ensures that the obtained scheme of BABYPIPE must be not inferior to GPipe.

5.4 Algorithm Design: Double-chromosome Genetic Algorithms for UMPIPE

It can be assumed that the minibatch-size P has Q possible cases for partitioning. Therefore, GPipe also has Q feasible solutions as it requires EDP. As BABYPIPE allows UEDP for different processes, it has Q^{4N-2} feasible solutions, which are far more than that of GPipe and increase exponentially with the number of stages. In the previous section, we analyze that as the solution set expands, BABYPIPE has a better theoretical optimal solution. Leveraging dynamic programming, recurrence algorithm or enumerate algorithm requires large computational complexity. Thus, one of the keys is to find an algorithm with acceptable complexity that can find a better solution than the GPipe optimal solution.

As $\langle p_1, p_2, \ldots, p_{2N-1}, q_1, q_2, \ldots, q_{2N-1} \rangle$ can be regarded as the solution of BABYPIPE, we consider using the genetic algorithm to search the optimal solution of Eq 5-11. In genetic algorithms, the fitness of each individual is related to the training time corresponding to its solution. According to Algorithm 5-1, calculating the fitness of all individuals in each generation will require three layers of loops, which consume a large amount of time. Therefore, we also propose a method to eliminate loops through GPU-based matrix operations, significantly improving the search speed of genetic algorithms.

5.4.1 DGAP: Dual Chromosomes-based Genetic Algorithm

Referring to the existing terms of genetic algorithms [152,166], the base components of DGAP are set as:

(1) **Gene, Chromosome and Individual**: we regard the partition number p_k or q_k of each process as a gene. Generally, an individual corresponds to a chromosome, where C and D can merge into one chromosome. However, in BABYPIPE, the data partitions for forward and backward propagation are mutually independent, and their corresponding training time is a linear sum of forward and backward as Eq 5-11. Thus, we set each individual has two chromosomes: first is a vector $C = \langle p_1, p_2, \ldots, p_{2N-1} \rangle$ and the second is a vector $D = \langle q_1, q_2, \ldots, q_{2N-1} \rangle$ both with $2N-1$ genes corresponding to an optimal data partition scheme of the BABYPIPE.

(2) **Fitness and Chromosomes selector**: An individual's fitness equals the training time of DNN corresponding to individual's genes-determined partition scheme under BABYPIPE parallelism (called time corresponding to the individual). The fitness of the chromosome C is equal to its corresponding forward propagation time $E_{(2N-1)P}$ and that of D is to backward time $R_{(2N-1)P}$.DGAP sorts the two sets of chromosomes of all individuals separately and then selects better parts of each set of chromosomes in pairing and crossover respectively.

(3) **Crossover**: In this chapter, we set the crossover to occur between four chromosomes $C_\alpha = \langle p_{\alpha_1}, p_{\alpha_2}, \ldots, p_{\alpha_{2N-1}} \rangle$, $C_\beta = \langle p_{\beta_1}, p_{\beta_2}, \ldots, p_{\beta_{2N-1}} \rangle$, $D_\gamma = \langle q_{\gamma_1}, q_{\gamma_2}, \ldots, q_{\gamma_{2N-1}} \rangle$, $D_\eta = \langle q_{\eta_1}, q_{\eta_2}, \ldots, q_{\eta_{2N-1}} \rangle$. Their crossover is defined as separately extracting a part of genes from them to gain two new chromosomes $C^{(new)}$ and $D^{(new)}$ to construct the children individual, such as $C^{(new)} = \langle p_{\alpha_1}, p_{\beta_2}, p_{\beta_3}, \ldots, p_{\alpha_{2N-1}} \rangle$ and $D^{(new)} = \langle p_{\eta_1}, p_{\gamma_2}, p_{\eta_3}, \ldots, p_{\eta_{2N-1}} \rangle$.

(4) **Mutation**: Mutation is replacing some elements of a chromosome by randomly generated genes.

(5) **Population regeneration mechanism**: The genetic algorithm for BABYPIPE applies elitist strategy [152] to combine the parent individuals with their children individuals to jointly compete to produce the next generation.

Then, we can present the dual-chromosome genetic algorithm for BABYPIPE in Algorithm 5-2, which uses Algorithm 5-1 to calculate the fitness.

Algorithm 5-2 is a conventional version. As Algorithm 5-2 calls Algorithm 5-1

Algorithm 5-2 Dual-chromosome genetic algorithm for BABYPIPE (DGAP): using the conventional calculation procedure

Input : $F_k(p)$ and $B_k(q)$ for $\forall k, p, q$. P, N_p, N_g and N_c. $W = [w_0, w_1, \ldots, w_{Q-1}]$ of optional partitions, i.e., $p \in W$ and $q \in W$.
Output: $\langle p_1, p_2, \ldots, p_{2N-1}, q_1, q_2, \ldots, q_{2N-1} \rangle$

Set Q individuals both with equal partitions per layer, where the chromosomes of the i-th individuals are as $C_i = D_i = \langle w_{i-1}, w_{i-1}, \ldots, w_{i-1} \rangle$
Randomly initial the rest $N_p - Q$ individuals
for *generation* $\in [1, N_g]$ **do**
    for $i \in [1, N_p]$ **do**
        Initial two arrays $E_{kj} = 0$ and $R_{kj} = 0$ for $0 \leq k \leq 2N-1$ and $0 \leq j \leq P$
        for $k \in [1, 2N-1]$ **do**
            for $j \in [1, P]$ **do**
                Calculate E_{kj} and R_{kj} based on Eq 5-9
        Obtain the fitness of $h(C_i) = E_{(2N-1)P}$ and $h(D_i) = R_{(2N-1)P}$
    for $i \in [1, N_c]$ **do**
        Select and Pair the chromosomes C_α, C_β, D_γ and D_η with better fitness
        Execute crossover and mutation to generate children chromosomes $C^{(new)}$ and $D^{(new)}$
        Initial two arrays $E_{kj} = 0$ and $R_{kj} = 0$ for $0 \leq k \leq 2N-1$ and $0 \leq j \leq P$
        for $k \in [1, 2N-1]$ **do**
            for $j \in [1, P]$ **do**
                Calculate E_{kj} and R_{kj} based on Eq 5-9
        Obtain the fitness of children individuals
    Sort the children and parents according to the fitness of individuals, and retain the best N_p individuals as the next generation
Set the genes of the individuals with the best fitness as solution $\langle p_1, p_2, \ldots, p_{2N-1}, q_1, q_2, \ldots, q_{2N-1} \rangle$

to calculate the fitness (i.e., training time) of each individual, it requires four layers of loops, including loop in generation index, loop in individual index i, loop in network layer index k and loop in input data index j. Due to too many loop layers, Algorithm 5-2 will take a long time to search for an optimal solution with time complexity $\mathrm{O}\left(N_g \cdot P \cdot (N_p + N_c) \cdot (4N-2)\right)$, which is mainly consumed in the calculation of fitness. To enable genetic algorithms feasible to solve the optimal data partitioning $\langle p_1, p_2, \ldots, p_{2N-1}, q_1, q_2, \ldots, q_{2N-1} \rangle$ in practical applications, it is necessary to reduce the time spent on each iteration of the genetic algorithm. We will introduce the improvement way to accelerate DGAP subsequently. Before that, we analyze the convergence of the dual chromosome strategy theoretically.

5.4.2 Analysis of Convergence for Dual-Chromosomes Strategy

In one minibatch, all the backward propagations need to start after the ending of all forward propagation, so the optimization of ω in Eq 5-11 can be divided into two independent objectives $\min E_{(2n-1)P}$ and $\min R_{(2n-1)P}$. Correspondingly, we set one individual to have two chromosomes C and D in DGAP.

Next, we can analyze the convergence probability in each generation to demonstrate the superiority of two chromosomes compared to using one chromosome.

It can be obtained that the number of feasible solutions for BABYPIPE is Q^{4N-2}. We can set all genes of individuals in each generation to be randomly generated again without considering the evolutionary ability of the crossover and mutation in genetic algorithms. Then, the probability of obtaining the theoretically optimal individual in one generation is $\varphi_1 = \frac{N_p}{Q^{4N-2}}$. Thus, for one chromosome, the probability of obtaining the global optimal solution at the g-th generation is $(1-\varphi_1)^{g-1}\cdot\varphi_1$. Therefore, the expectation of generations required to achieve theoretically optimal individuals is

$$\sum_{g=1}^{+\infty} g\cdot(1-\varphi_1)^{g-1}\cdot\varphi_1 = \frac{1}{\varphi_1} = \frac{Q^{4N-2}}{N_p} \tag{5-15}$$

For two chromosomes, the probabilities of obtaining the theoretically optimal C and D in one generation are both $\varphi_2 = \frac{N_p}{Q^{2N-1}}$. The probability of obtaining the global optimal C at the g-th generation is $(1-\varphi_2)^{g-1}\cdot\varphi_2$, which is same to D. Therefore, the expectation of generations required to achieve theoretical optimal C chromosome is $\frac{1}{\varphi_2}$. As C and D are independent, the expectation of generations required to both achieve theoretical optimal individual for dual-chromosomes-based genetic algorithm (DGAP) is $2\frac{1}{\varphi_2} = 2\frac{Q^{2N-1}}{N_p}$ which is far less than that of one-chromosome-based genetic algorithm.

This huge difference between a dual-chromosomes-based genetic algorithm and a one-chromosome-based genetic algorithm comes from that:

(1) For one chromosome with $4N-2$ genes, even if the first $2N-1$ genes reach the theoretical optimal, they still need to be replaced when the last $2N-1$ genes don't achieve optimal.

(2) For dual chromosomes respectively with $2N-1$ genes, when the $2N-1$ genes of C chromosome reach theoretical optimal, the evolution of C chromosome can be stopped regardless of the evolutionary state of the D chromosome. The same goes for the D chromosome.

This conclusion is also valid when considering the evolutionary ability of crossover and mutation in genetic algorithms. For the general situation considering crossover and mutation, it can be proved that the dual-chromosome-based genetic algorithm has a higher probability of reaching the optimal solution for each generation than the one-chromosome-based genetic algorithm. Therefore, the solution of DGAP in each generation will be statistically superior to GAP.

In subsequent experiments, we will compare the convergence performance of GAP and DGAP in solving the optimal solution of BABYPIPE.

5.4.3 OiDGAP: One-level improved DGAP

To accelerate DGA for BABYPIPE, we improve Algorithm 5-2 from two aspects, including eliminating the loop in the individuals' index (the "for loop" of line 4 and line 10 in Algorithm 5-2) and eliminating the loop in input data's index (the "for loop" of line 7 and line 15 in Algorithm 5-2).

Eliminating the loop in the individual index indicates simultaneously calculating the training time corresponding to multiple individuals. It can be set that the available partition numbers construct a one-dimensional array $W = [w_0, w_1, \ldots, w_{Q-1}]$, the corresponding time functions of each layer construct two two-dimensional arrays $F = \{f_{kj}\}_{(2N-1)\times Q}$ and $B = \{b_{kj}\}_{(2N-1)\times Q}$ where $f_{kj} = F_{k+1}(w_j)$ and $b_{kj} = B_{k+1}(w_j)$ for $0 \leq k < 2N-1$ and $0 \leq j < Q$, the indexes for all individuals' genes in array W construct a two-dimensional array $G = \{g_{ik}\}_{N_p\times(4N-2)}$ which means the partition number of the $(k+1)$-th layer determined by the $(i+1)$-th individual is

$$W[g_{ik}] = w_{g_{ik}} = \begin{cases} p_{(i+1)(k+1)}, & if \quad k < 2N-1 \\ q_{(i+1)(k+2-2N)}, & others \end{cases} \tag{5-16}$$

Therefore, when W is given, the array G is equivalent to genes in the genetic algorithm. As the calculations of training time for forward propagation and backward propagation have similarities, we only discuss the calculation for the training time of forward propagation. The array, composed of the k-th gene of all individuals, can be obtained as $G[:,k]$ where $0 \leq k < 2N-1$, thus the array of its corresponding time functions of all individuals are $F[k, G[:,k]]$ and that of the partition numbers are $W[G[:,k]]$. The microbatch-sizes of the k-th process for all individuals are $\left\lceil \frac{P}{W[G[:,k]]} \right\rceil$. It can be assumed that the end time of the j-th data in the k-th forward process for the $(i+1)$-th individual is e_{ijk} which can construct a three-dimensional array $E = \{e_{ijk}\}_{N_p\times(P+1)\times 2N}$. Thus, the recur-

rence relationship for calculating the fitness of multiple individuals simultaneously can be derived as

$$\begin{cases} Z^{(e)} = \min\left(Z^{(b)} + \left\lceil \dfrac{P}{W[G[:,k]]} \right\rceil, P+1\right) \\ E\left[:, Z^{(b)}[:] : Z^{(e)}[:], k\right] = F[k, G[:,k]] \\ \quad + \max\left(E\left[:, Z^{(e)}[:]-1, k-1\right], E\left[:, Z^{(b)}[:]-1, k\right]\right) \\ Z^{(b)} = Z^{(e)} \end{cases} \circlearrowleft \tag{5-17}$$

where the array $Z^{(b)}$ indicates the starting index of data whose initial value is $[1,1,\ldots,1]_{N_p}$ and $Z^{(e)}$ is end index. Due to the possible differences in the number of partitions corresponding to the same layer for multiple individuals, the number of columns required to be broadcast in each row of the second formula of Eq 5-17 will be different. Generally, the programming with array operation does not support imbalanced broadcasting, hence using matrix multiplication to achieve this function to achieve it. Then, the genetic algorithm for BABYPIPE after one level of improvement can be shown in Algorithm 5-3.

Assuming executed on one or multiple GPUs with sufficient parallel capability, Algorithm 5-3 has the time complexity as $\mathrm{O}\left(N_g \cdot P \cdot (4N-2)\right)$.

5.4.4 TiDGAP: Two-level improved DGAP

Eliminating the loop in input data indicates simultaneously calculating the end time of multiple microbatches, i.e., obtaining $E[i,:,k]$ after one operation.

As shown in Eq 5-9, the start time of the j-th data in the k-th process mainly depends on the maximum end time between the z_{kj}-th data in the $(k-1)$-th process (called preprocess baseline) and the y_{kj}-th data in the k-th process (called preorder baseline). Therefore, we can divide the start time of data in the k-th process into p_k parts (i.e., p_k microbatches), where all data in the same part has the same start time. Thus, we only need to quickly calculate the start time of one data in each part to know the start time of other data in the same part. Assuming the end time of each data in the $(k-1)$-th process is known, we can obtain the preprocess baseline for each microbatch in the k-th process is $\left[E\left[i, \left\lceil \frac{P}{p_k} \right\rceil, k-1\right], E\left[i, 2\left\lceil \frac{P}{p_k} \right\rceil, k-1\right], \ldots, E\left[i, P, k-1\right]\right]$ which can be set as $\left[\eta_1, \eta_2, \ldots, \eta_{p_k}\right]$ for the sake of presentation. Therefore, the start time of the 1st microbatch in the k-th process is η_1, and that of the 2nd microbatch is $\max\left(\eta_1 + f, \eta_2\right)$ where f means the time function. By mathematical induction, we derive a provable property that

Algorithm 5-3 One-level improved DGA for BABYPIPE (OiDGAP): simultaneously calculating the training time corresponding to multiple individuals

Input : $F=\{f_{kj}\}_{(2N-1)\times Q}$, $B=\{b_{kj}\}_{(2N-1)\times Q}$, P, $W=[w_0,w_1,\ldots,w_{Q-1}]$. N_p, N_g and N_c.

Output: $G\left[\arg\min\left(H^{(C)}+H^{(D)}\right)\right]$

1 Set Q individuals both with equal partitions per layer as $G[i,:]=[W[i],W[i],\ldots,W[i]]$

2 Randomly initial the rest N_p-Q individuals

3 Set $T=\{t_{ij}=j\}_{N_p\times(P+1)}$, $E=\{e_{ijk}=0\}_{N_p\times(P+1)\times 2N}$ and $R=\{r_{ijk}=0\}_{N_p\times(P+1)\times 2N}$

4 **for** $k\in[0,2N-2]$ **do**

5 Initial $Z^{(b)}=[1,1,\ldots,1]_{N_p}$

6 **while** $\min\left(Z^{(b)}\right)<P+1$ **do**

7 $Z^{(e)}=\min\left(Z^{(b)}+\left\lceil\frac{P}{W[G[:,k]]}\right\rceil,P+1\right)$

8 $Z=\text{where}\left(Z^{(b)}<P+1\right)$

9 $M=\left(T[Z]\geq Z^{(b)}[Z]\right)*\left(T[Z]<Z^{(e)}[Z]\right)$

10 $A_1=E[k+1]\left[Z,Z^{(b)}[Z]-1\right]$ and $A_2=E[k]\left[Z,Z^{(e)}[Z]-1\right]$

11 $E[k+1,Z]+=F[k,G[Z,k]]+M*\max(A_1,A_2)$

12 $Z^{(b)}=Z^{(e)}$

13 Use the similar way as line 4-12 to calculate R

14 Obtain the arrays of fitness of $H^{(C)}=E[:,P+1,2N]$ and $H^{(D)}=R[:,P+1,2N]$

15 **for** *generation* $\in[1,N_g]$ **do**

16 **for** $i\in[1,N_c]$ **do**

17 Execute crossover and mutation to generate children $\bar{G}$

18 Obtain $\bar{H}^{(C)}$ and $\bar{H}^{(D)}$ of children with similar way as line 4-14 ; `// Two-layer loops`

19 Concatenate children and parent individuals as $G=\text{concat}\left(G,\bar{G}\right)$, $H^{(C)}=\text{concat}\left(H^{(C)},\bar{H}^{(C)}\right)$, $H^{(D)}=\text{concat}\left(H^{(D)},\bar{H}^{(D)}\right)$

20 Update individuals as: $U=\text{argsort}\left(H^{(C)}+H^{(D)}\right)$, $G=G[Sort[:N_p]]$ with its fitness $H^{(C)}=H^{(C)}[U[:N_p]]$ and $H^{(D)}=H^{(D)}[U[:N_p]]$

the start time of the j-th microbatch in the k-th process is

$$\max_{l=1}^{j}(\eta_l+(j-l)f)=\max_{l=1}^{j}(\eta_l-lf)+jf \tag{5-18}$$

Thus, we only need to calculate $\max_{l=1}^{j}(\eta_l-lf)$ for each j. We can construct a array as $PB=\left[\eta_1-f,\eta_2-2f,\ldots,\eta_j-jf,\ldots,\eta_{p_k}-p_kf\right]$. Significantly, $\max_{l=1}^{j}(\eta_l-lf)$ is exactly the maximum value of the first j items for the array PB, which can be quickly obtained by calling the 'cummax' function of array operation or using upper triangular matrix (if

without 'cummax' function). Then, $\text{cummax}(PB) + [1, 2, \ldots, p_k]f$ is the array composed of the start time of each data in the k-th process.

This approach still applies to calculating the starting time of multiple data of multiple individuals simultaneously by combining with Algorithm 5-3, which can simultaneously eliminate two layers of loops in Algorithm 5-2 including the loop in the individuals' index (the "for loop" of line 4 and line 10 in Algorithm 5-2) and the loop in input data's index (the "for loop" of line 7 and line 15 in Algorithm 5-2). Then, the two-level improved DGA for BABYPIPE simultaneously calculating the starting time of multiple data and multiple individuals can be seen in Algorithm 5-4.

On one or multiple GPUs with sufficient parallel capability, the time complexity of Algorithm 5-4 is $\mathrm{O}\left(N_g \cdot (4N-2)\right)$ which is approximately $\frac{1}{P}$ of Algorithm 3.

Then, we can list the time complexity of DGAP (Algorithm 5-2), OiDGAP (Algorithm 5-3) and TiDGAP (Algorithm 5-4) in Table 5-2. From the comparison of Table 5-2, our proposed TiDGAP has a much lower time complexity that provides a method to quickly calculate the training time corresponding to different data partitioning schemes and allows BABYPIPE (unequal data partitions for microbatch-based pipeline parallelism) to apply in the practical optimization and acceleration of parallel training. With the increase of N_p and P, the execution time of TiDGAP in real GPU devices will also increase slowly, while it is still much less than DGAP and OiDGAP, suitable for optimization of parallel training for large-scale DNN. In subsequent experiments, we will also evaluate the execution time of TiDGAP compared to DGAP and OiDGAP.

5.5 Experimental Results and Analysis

5.5.1 Experiment Settings

For the sake of the comprehensive evaluations of the unequal data partitions-based parallelism (i.e., UMPIPE) and two-level improved dual-chromosome genetic algorithm (TiDGAP), we carry out four groups of experiments from various aspects including:

(1) EX_1: Comparing TiDGAP with TiGAP to demonstrate the optimization effect of dual-chromosome strategy;

(2) EX_2: Comparing TiDGAP with OiDGAP and DGAP to demonstrate the acceleration effect on the evolution of two-level improvement with array operation;

(3) EX_3: Comparing TiDGAP with baselines local greedy algorithm and global greedy-based dynamic programming to demonstrate the optimality of DGAP for UMPIPE.

Algorithm 5-4 Two-level improved DGA for BABYPIPE (TiDGAP): simultaneously calculating the starting time of multiple data and multiple individuals

Input : $F=\{f_{kj}\}_{(2N-1)\times Q}$, $B=\{b_{kj}\}_{(2N-1)\times Q}$, P, $W=[w_0,w_1,\ldots,w_{Q-1}]$. N_p, N_g and N_c.

Output: $G\left[\arg\min\left(H^{(C)}+H^{(D)}\right)\right]$

1 Set Q individuals both with equal partitions per layer as $G[i,:]=[W[i],W[i],\ldots,W[i]]$

2 Randomly initialize the rest N_p-Q individuals

3 Set $T=\{t_{ij}=j\}_{N_p\times(P+1)}$, $E=\{e_{ijk}=0\}_{N_p\times(P+1)\times 2N}$ and $R=\{r_{ijk}=0\}_{N_p\times(P+1)\times 2N}$, $Z'=\{t_{ij}=i\}_{N_p\times(P+1)}$ .reshape(shape $=(-1,)$)

4 **for** $k\in[0,2N-2]$ **do**

5 Set $A=\{0\}_{N_p\times(P+1)}$ and $M=\left\lceil\frac{T}{W[G[:,k]]}\right\rceil$

6 Obtain the index of the reference end time in the previous process as $Z=M*(W[G[:,k]])$

7 Calculate the basic time for each individual of current p_k as $M_1=M*(F[k,G[:,k]])$

8 Obtain the array of PB as $PB=E[k][Z',Z]$.reshape(shape $=(N_p,P+1))-M_1$

9 $A[:,1:]=\text{cummax}\,(PB[:,1:],dim=1)$

10 $E[k+1]+=A+(M+1)*F[k,G[:,k]]$

11 Use the similar way as line 4-10 to calculate R

12 Obtain the arrays of fitness of $H^{(C)}=E[:,P+1,2N]$ and $H^{(D)}=R[:,P+1,2N]$

13 **for** *generation* $\in[1,N_g]$ **do**

14 Use array operation to execute the crossover and mutation to generate children $\bar{G}$

15 Obtain fitness $\bar{H}^{(C)}$ and $\bar{H}^{(D)}$ of children with similar way as line 4-10 ; `// One loop`

16 Concatenate the children and parent individuals as $G=\text{concat}\left(G,\bar{G}\right)$, $H^{(C)}=\text{concat}\left(H^{(C)},\bar{H}^{(C)}\right)$, $H^{(D)}=\text{concat}\left(H^{(D)},\bar{H}^{(D)}\right)$

17 Update individuals as: $U=\text{argsort}\left(H^{(C)}+H^{(D)}\right)$, $G=G[Sort[:N_p]]$ with its fitness $H^{(C)}=H^{(C)}[U[:N_p]]$ and $H^{(D)}=H^{(D)}[U[:N_p]]$

(4) EX_4: Comparing UMPIPE with GPipe, UMPipeDream and PipeDream to demonstrate the superiority of UEDP.

EX_1 and EX_2 are conducted on a randomly generated simulation dataset, which is beneficial for executing sufficient experiments. EX_3 and EX_4 are conducted on real parallel training in multiple GPUs, which is conducive to demonstrating the advantages and feasibility simultaneously of our proposed UMPIPE architecture and TiDGAP algorithm. The baseline algorithms in EX_3 represent that:

(1) Local Greedy Algorithm for UMPIPE (LG): For each process, select the number of

Table 5-2 The Description and Time Complexity of Genetic Algorithm and its Improved Algorithms for BABYPIPE.

Algorithm	Description	Time Complexity	Ratio
DGAP	Using recurrence algorithm with three layers of loops to calculate the training time corresponding to individuals	$\mathrm{O}\left(N_g \cdot P \cdot (N_p + N_c) \cdot (4N-2)\right)$	1
OiDGAP	Simultaneously calculating the training time corresponding to multiple individuals	$\mathrm{O}\left(N_g \cdot P \cdot (4N-2)\right)$	$\frac{1}{N_c+N_p}$
TiDGAP	Simultaneously calculating the starting time of multiple data corresponding to multiple individuals	$\mathrm{O}\left(N_g \cdot (4N-2)\right)$	$\frac{1}{(N_c+N_p)\cdot P}$

partitions that make the current process have the earliest end time. The algorithm can be seen in Algorithm 5-5.

(2) Global Greedy-based Dynamic Programming for UMPIPE (GG): For each process, selecting the number of partitions that make the last process have the earliest end time, whose algorithm can be seen in Algorithm 5-6. In Algorithm 5-6, the parameter 'rounds' can be flexibly set and can also be replaced by that $\nexists C'$ better than C which is a convergence condition of Algorithm 5-6.

Algorithm 5-5 Local greedy for UMPIPE (LG)

Input : $F_k(p), B_k(q); P; W = [w_0, w_1, \ldots, w_{Q-1}]$
Output: $\langle p_1, p_2, \ldots, p_{2N-1}, q_1, q_2, \ldots, q_{2N-1} \rangle$
for $k \in [0, 4N-3]$ **do**
  if $k < 2N-1$ **then**
    Choose the p_k in W which makes E_{kP} to take the minimum value
  else
    Choose the q_k in W which makes R_{kP} to take the minimum value

Each group of experiments adopts the control variables to ensure the reliability of comparisons. In EX_1, the indicators are the optimization results over generations and the probability of finding the global optimal solution over generations to demonstrate the convergence of dual-chromosome strategy, where the optimization results present the parallel training time corresponding to the optimization solutions. EX_2 is mainly used to observe the execution time of different algorithms controlling the number of individuals (N_p) and

Algorithm 5-6 Global greedy-based dynamic programming for UMPIPE (GG)

Input : $F_k(p), B_k(q); P; W = [w_0, w_1, \ldots, w_{Q-1}]$
Output: $\langle p_1, p_2, \ldots, p_{2N-1}, q_1, q_2, \ldots, q_{2N-1} \rangle$

(Randomly or specifically) initial a partition scheme as $C = \langle p_1, \ldots, p_{2N-1} \rangle$ and $D = \langle q_1, \ldots, q_{2N-1} \rangle$
for $i <$ *rounds* **do**
  for $k \in [0, 4N-3]$ **do**
    if $k < 2N-1$ **then**
      Choose the p'_k in W *s.t.* $E_{(2N-1)P}(C') = \min_{l=0}^{Q-1} \left(E_{(2N-1)P}(C|_{p_k=w_l}) \right)$ where $C|_{p_k=w_l}$ means only changing the partition number of the k-th process to w_l
      Update $p_k = p'_k$, $C = C'$
    else
      Choose the q'_k in W to make $R_{(2N-1)P}(D')$ take the minimum value
      Update $q_k = q'_k$, $D = D'$

generations (N_g) in different scales. EX_3 is to observe the stable optimization results of the different algorithms. EX_4 is to observe the optimal training time and convergence under UMPIPE parallelism, compared with other parallelism. We test a large number of instances in each group of experiments, and instances from the same group of experiments point to similar conclusions. Therefore, we only provide a subset of them in this chapter. Then, the optimization algorithm is launched on a desktop and the realistic parallel training is launched on the servers. The configurations of them are as follows.

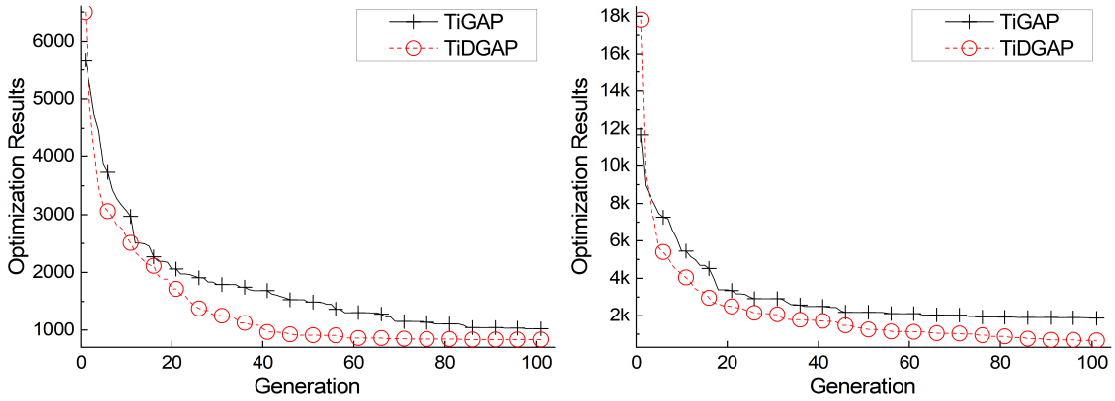
- Program version: Python 3.7 + Pytorch 1.13.1;
- Desktop: NVIDIA GeForce RTX 3060 Ti @ 8GB;
- Servers: NVIDIA TESLA V100 @ 32GB × 2.

5.5.2 EX_1: Evaluation of Dual-Chromosome Strategy of TiDGAP Compared with TiGAP

To observe the optimization effect of the dual-chromosome strategy, we compare TiDGAP with TiGAP in the simulation scenarios. In the simulation scenarios, we randomly generate $F = \{f_{kj}\}_{(2N-1)\times Q}$ and $B = \{b_{kj}\}_{(2N-1)\times Q}$. As the outcomes from different random distributions echo the consistent trends and results, we only present the results obtained from the uniform distribution $\mathrm{U} = [1, 100] \cap \mathbb{N}^*$.

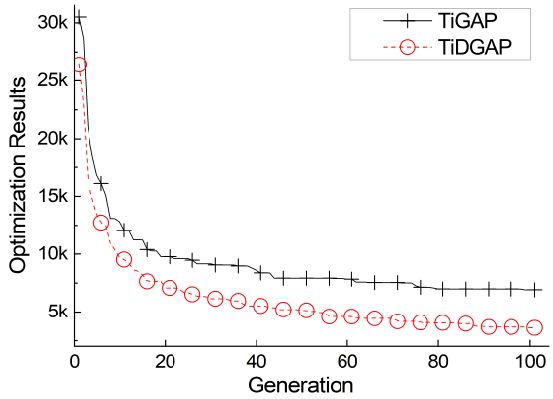
Firstly, we observe the trends of optimization results over generations in several scenarios, including $(N = 10, P = 64)$, $(N = 10, P = 512)$, $(N = 20, P = 512)$ and

$(N = 100, P = 1024)$. In experiments, we set the number of individuals as $N_p = 100$, and the number of generations as $N_g = 100$, and the algorithms don't set equal partitions into initial states (i.e., random initialization). Then, we plot the results in Fig. 5-4.

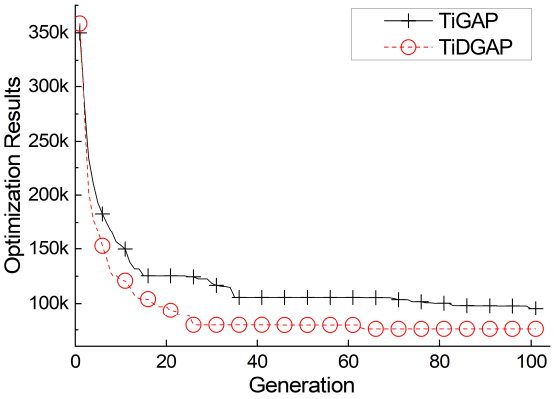


(a) $(N = 10, P = 64)$

(b) $(N = 10, P = 512)$



(c) $(N = 20, P = 512)$



(d) $(N = 100, P = 1024)$

Figure 5-4 The optimization results (corresponding to time for training one minibatch) over generations in randomly generated basic time arrays comparing TiDGAP with TiGAP, where: $N_p = 100$, $N_g = 100$, $F, B \sim \mathrm{U} = [1, 100]$, randomly initializing N_p individuals.

From Fig. 5-4, the curves of TiDGAP with dual-chromosome strategy remain lower than that of TiGAP. The unique difference between TiDGAP and TiGAP is the number of chromosomes per individual in genetic algorithms. TiDGAP and TiGAP both have $4N-2$ genes in one individual. The two chromosomes of the individual in TiDGAP are composed of $2N-1$ genes respectively, while the individual in TiGAP only has one chromosome. The comparative results in Fig. 5-4 show that the usage of the dual-chromosome strategy is beneficial for improving convergence of GAP, which is consistent with the analysis in Section 5.4.2. The time costs of forward and backward propagation in UMPIPE are

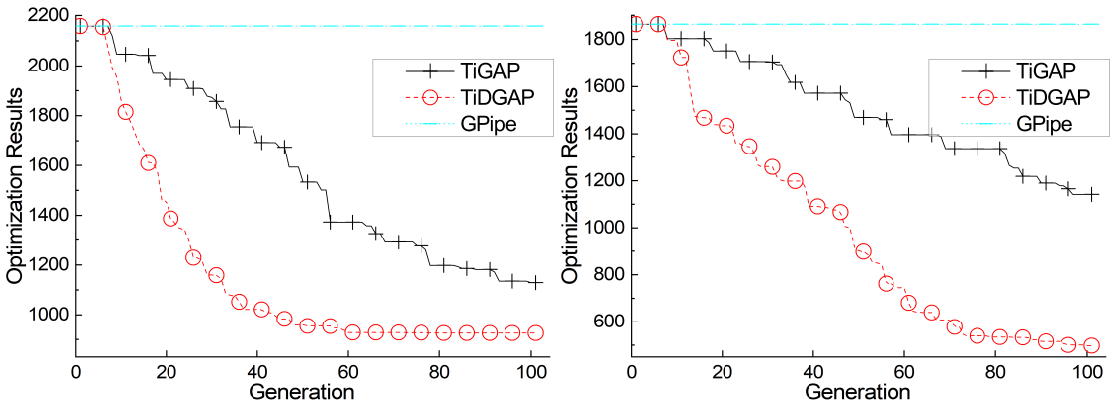
independent mutually. Therefore, setting two chromosomes to represent forward propagation and backward propagation respectively has lower expected generations than the single-chromosome to achieve the optimal solution of UMPIPE. The statistical indicator (lower expected generations) points to better optimization results (corresponding to lower training time under UMPIPE parallelism) over the generation as Fig. 5-4. For the sake of observation of the quantitative comparison between TiDGAP and TiGAP in Fig. 5-4, we have listed the optimization results of TiDGAP and TiGAP at the 100-th generation in Table 5-3 where $\epsilon_{\text{TiDGAP}}^{\text{TiGAP}} = \frac{\text{TiGAP}-\text{TiDGAP}}{\text{TiGAP}}$ means the reduction magnitude of TiDGAP in training time compared to TiGAP. From Table 5-3, TiDGAP within 100 generations reduces the training time under UMPIPE by 18.33%, 64.02%, 47.88% and 20.60% compared to TiGAP.

Table 5-3 The quantitative optimization results of TiDGAP and TiGAP at the 100-th generation in the experiments of Fig. 5-4.

Scenarios	TiDGAP	TiGAP	$\epsilon_{\text{TiDGAP}}^{\text{TiGAP}}$
$(N = 10, P = 64)$	833	1020	18.33%
$(N = 10, P = 512)$	671	1865	64.02%
$(N = 20, P = 512)$	3601	6909	47.88%
$(N = 100, P = 1024)$	75765	95419	20.60%

The experiments in Fig. 5-4 do not set the solution of GPipe as one of the initial individuals. To verify the advantages of dual-chromosome strategy for UMPIPE over single chromosome have universality, we carry out experiments in two combinations of $(N = 10, P = 64)$ and $(N = 10, P = 512)$ by setting equal partitions into initial states as the line 1 in Algorithm 5-4. Then, we plot the optimization results over generations of TiDGAP and TiGAP in Fig. 5-5. In Fig. 5-5, we also draw a straight line paralleling to the horizontal axis to represent the optimal training time of DNN under GPipe parallelism. From Fig. 5-5, the convergence of TiDGAP to solve the scheme of UMPIPE is still better than that of TiGAP. This also confirms once again that the dual-chromosome strategy is more suitable for UMPIPE than the single-chromosome. Moreover, the solutions of TiDGAP and TiGAP are both better than those of GPipe, which proves that the UMPIPE architecture is superior to GPipe in the randomly generated basic time functions. This phenomenon can reflect the significance of UMPIPE. In order to evaluate the performance of TiDGAP and UMPIPE in this scenario, we list the optimization results at the 100-th

generation in Table 5-4 where $\varepsilon_{\text{TiDGAP}}^{\text{GPipe}} = \frac{\text{GPipe}-\text{TiDGAP}}{\text{TiDGAP}}$ means the improvement ratio of TiDGAP (also representing UMPIPE) in training speed compared to GPipe.



(a) $(N = 10, P = 64)$ (b) $(N = 10, P = 512)$

Figure 5-5 The optimization results (corresponding to training time for one minibatch) over generations in randomly generated basic time arrays comparing TiDGAP with TiGAP, where: $N_p = 100$, $N_g = 100$, $F, B \sim \text{U} = [1, 100]$, using the optimal GPipe solution as the initial individuals.

Table 5-4 The quantitative optimization results of TiDGAP, TiGAP at the 100-th generation and that of GPipe in the experiments of Fig. 5-5 for $N = 10$, setting equal partitions into initial states.

Scenarios	GPipe	TiDGAP	TiGAP	$\epsilon_{\text{TiDGAP}}^{\text{TiGAP}}$	$\varepsilon_{\text{TiDGAP}}^{\text{GPipe}}$
$P = 64$	2158	926	1129	17.98%	1.33×
$P = 512$	1863	498	1142	56.39%	2.74×

From Table 5-4, the parallel training scheme of UMPIPE solved by the TiDGAP algorithm has a speed increase of 1.33× and 2.74× respectively in $(N = 10, P = 64)$ and $(N = 10, P = 512)$ compared to GPipe, which is consistent with the analysis of UMPIPE's optimality in Section 5.3.4.

To further evaluate the convergence of TiDGAP, we carry out experiments in small-scale scenarios including $(N = 2, P = 512)$, $(N = 3, P = 64)$, $(N = 5, P = 8)$, $(N = 12, P = 2)$. We use an enumerated algorithm to obtain the theoretical optimal solution of UMPIPE. In each scenario, we execute 100 instances, record the generations when the algorithm reaches the theoretical optimal for each instance, and calculate the probability of achieving global optimization over generations. The genetic algorithm in GAP

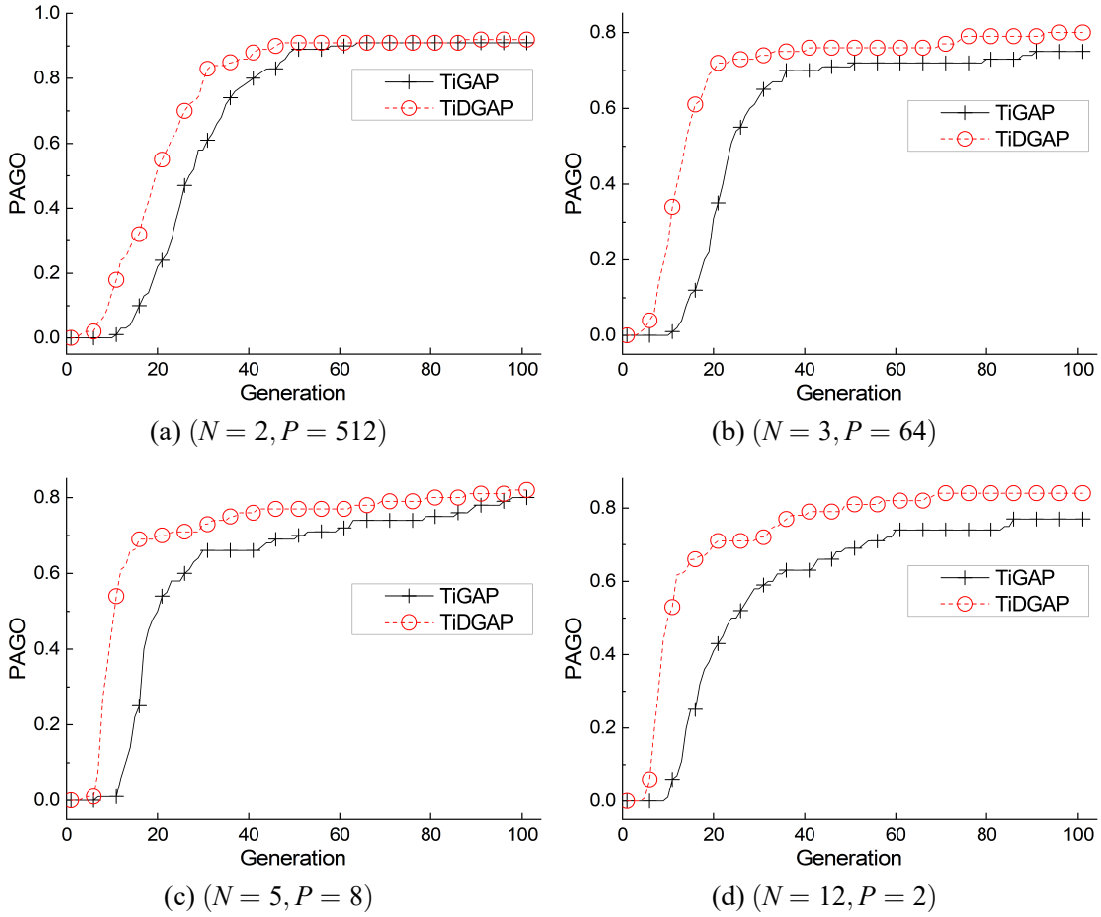


Figure 5-6 The probabilities of achieving global optimization (PAGO) over generations in randomly generated basic time arrays comparing TiDGAP with TiGAP, where: $N_p = 100$, $N_g = 100$, $F, B \sim \mathrm{U} = [1, 100]$, randomly initializing N_p individuals.

preserves the current best individual to the next generation, so if the algorithm reaches theoretical optimal in a certain generation, it will remain theoretically optimal in all subsequent generations, and hence the count of achieving theoretical optimal in subsequent generations will be increased by 1. Then, we plot the results in Fig. 5-6. Obviously, TiDGAP has higher probabilities than TiGAP to obtain the theoretical optimal solution within the same generations in Fig. 5-6, which also proves the superiority of the dual-chromosome strategy.

5.5.3 EX_2: Evaluation of Two-level improvement of TiDGAP Compared with OiDGAP and DGAP

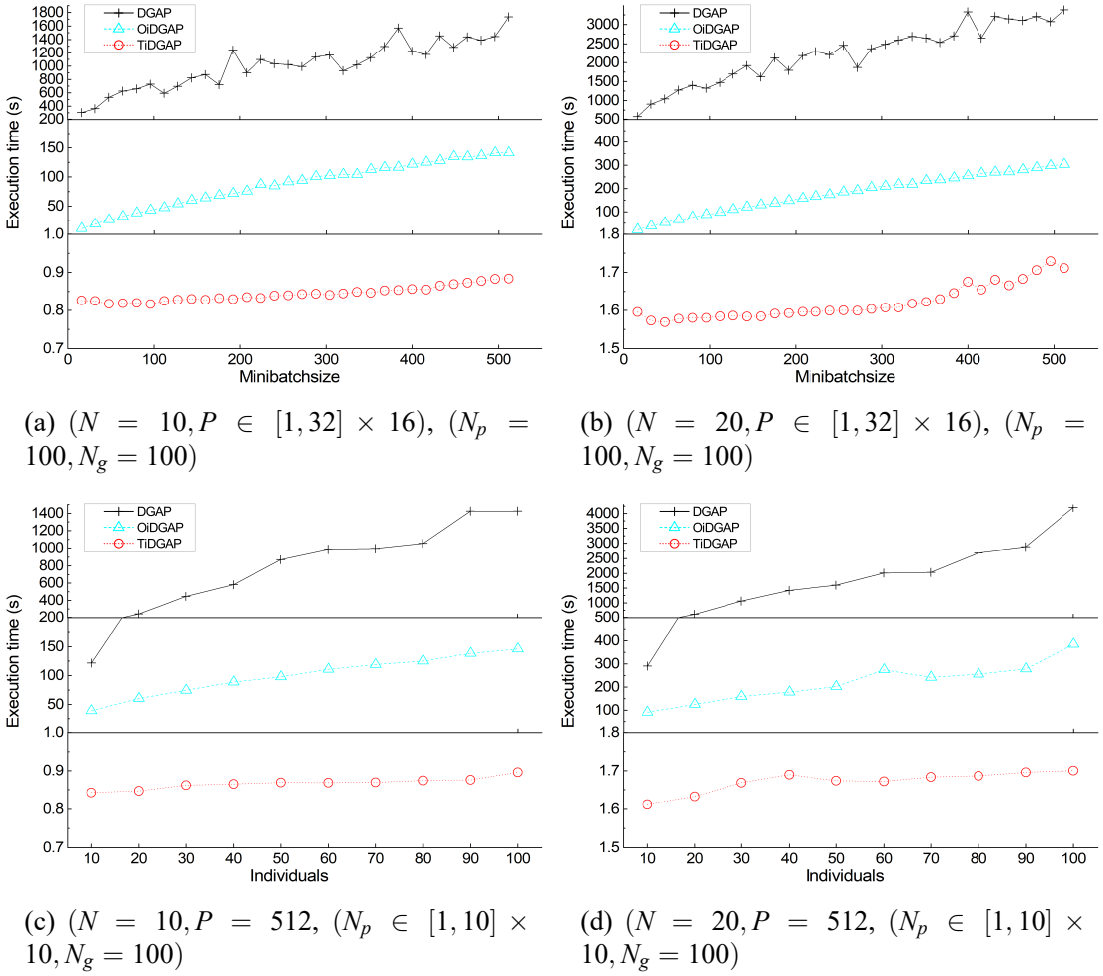
To observe the acceleration effect of the two-level improvement, we compare TiDGAP with OiDGAP and DGAP in the simulation scenarios with random basic time functions. As the two-level improvement of TiDGAP mainly eliminates two layers of loops, including the loop in the individuals' index and the loop in the input data's index, we execute experiments in four configurations with varying minibatch-size or number of individuals as follows:

- $(N = 10, P \in [1, 32] \times 16)$, $(N_p = 100, N_g = 100)$;
- $(N = 20, P \in [1, 32] \times 16)$, $(N_p = 100, N_g = 100)$;
- $(N = 10, P = 512$, $(N_p \in [1, 10] \times 10, N_g = 100)$;
- $(N = 20, P = 512$, $(N_p \in [1, 10] \times 10, N_g = 100)$.

These algorithms are executed on the Desktop.

Using the changed parameters as the abscissa and the algorithm execution time as the ordinate, we plot the execution time of TiDGAP, OiDGAP and DGAP in Fig. 5-7.

Firstly, the execution time of TiDGAP in Fig. 5-7 is significantly smaller in magnitude than OiDGAP and DGAP. Concretely, for $N = 10$ and $N_g = 100$ of Fig. 5-7(a) and 5-7(c), The execution time of TiDGAP is about $[0.8\text{s}, 0.9\text{s}]$ less than 1s; and that of OiDGAP and DGAP are respectively $[1\text{s}, 150\text{s}]$ and $[100\text{s}, 1800\text{s}]$. For $N = 20$ and $N_g = 100$ of Fig. 5-7(b) and 5-7(d), that of TiDGAP is about $[1.5\text{s}, 1.8\text{s}]$ less than 2s; and that of OiDGAP and DGAP are respectively $[2\text{s}, 400\text{s}]$ and $[300\text{s}, 4000\text{s}]$. Secondly, comparing OiDGAP to DGAP can demonstrate the effectiveness of improvement to simultaneously calculate the end time corresponding to multiple individuals, as well as comparing TiDGAP to OiDGAP can demonstrate the effectiveness of improvement to simultaneously calculate the end time of multiple microbatches. Lastly, the execution time of OiDGAP is approximately proportional to both minibatch-size P and the number of individuals N_p, i.e., linearly increasing with the increase of P and N_p. DGAP also has the same phenomenon. Unlike OiDGAP and DGAP, in the scenarios of Fig. 5-7, the execution time of TiDGAP remains relatively stable without increasing with P and N_p. These are generally consistent with the theoretical time complexities. This indicates that within a certain range of parameters, the computing speed of TiDGAP can be maintained unaffected by P and N_p beneficial from the parallel computing ability of GPU. In detail, according to the results of Fig. 5-7(c) and 5-7(d),, the execution speeds of TiDGAP are $(162.86\times, 226.36\times)$ of



(a) $(N = 10, P \in [1, 32] \times 16)$, $(N_p = 100, N_g = 100)$

(b) $(N = 20, P \in [1, 32] \times 16)$, $(N_p = 100, N_g = 100)$

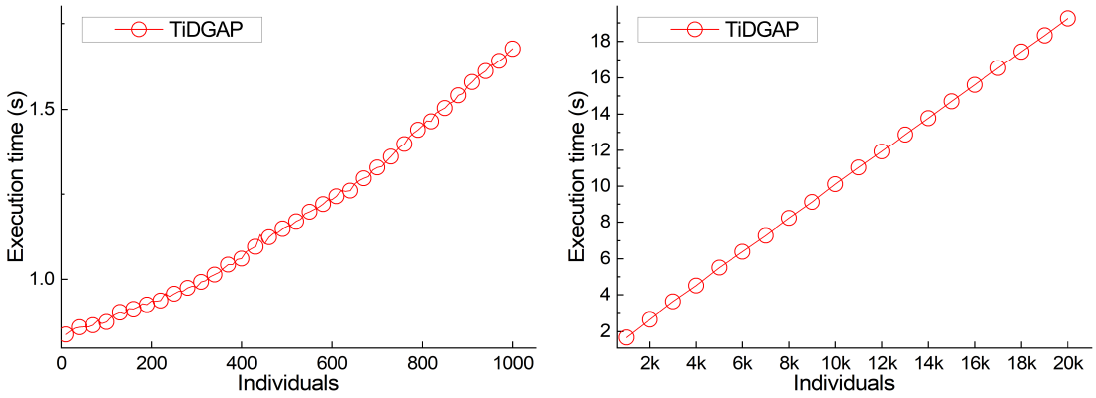
(c) $(N = 10, P = 512, (N_p \in [1, 10] \times 10, N_g = 100)$

(d) $(N = 20, P = 512, (N_p \in [1, 10] \times 10, N_g = 100)$

Figure 5-7 The execution time of TiDGAP, OiDGAP and DGAP for solving UMPIPE in simulated scenarios launched on GeForce RTX 3060 Ti.

OiDGAP, and $(1590.23\times, 2473.02\times)$ of DGAP in $(N = 10, N = 20)$ for $P = 512$ and $N_p = 100$.

However, the execution time of OiDGAP in Fig. 5-7(c) and 5-7(d) is directly proportional to the number of individuals N_p which seems to contradict the theoretical complexity of OiDGAP but actually not. Theoretical complexity assumes that the ideal GPU has sufficient parallel capability, while the parallel capability of GPUs is not infinite in reality. When the computational complexity reaches a certain level that exceeds the maximum value that the GPU's parallel cores can carry, its computational complexity will also increase with the N_p. This property will also apply to TiDGAP. To further evaluate the execution time of TiDGAP with respect to the number of individuals, we conduct experiments in simulated scenarios of $N_p \in [10, 1000]$ and $N_p \in [1000, 20000]$ both with $(N = 10, P = 512, N_g = 100)$. Then, we plot the execution time of TiDGAP in Fig. 5-8.



(a) $N_p \in [10, 1000]$ (b) $N_p \in [1000, 20000]$

Figure 5-8 The execution time of TiDGAP for solving UMPIPE in simulated scenarios where $(N = 10, P = 512, N_g = 100)$.

In Fig. 5-8, we can observe that when the number of individuals is large enough, the execution time of TiDGAP will also linearly increase with the number of individuals N_p. However, the slope of TiDGAP is still much smaller than that of OiDGAP and DGAP. It is worth emphasizing that, the execution time of TiDGAP in $N_p = 1000$ is only 1.68s for $N = 10$ and that in $N_p = 20000$ is only 19.24s. In the experiments of Fig. 5-4 and Fig. 5-5, 100 individuals are enough to obtain competitive optimal solutions. This strongly validates the rapidity of TiDGAP with the two-level improvement based on matrix operations on GPU. Similarly, when minibatch-size P is large enough, the execution time of TiDGAP will also linearly increase with P, while with a far smaller slope than that of OiDGAP and DGAP. The speed of TiDGAP is adequate to meet the requirements of current realistic large-scale DNNs.

5.5.4 EX_3: Evaluation of TiDGAP for UMPIPE Compared with Local Greedy Algorithm and Dynamic Programming

To additionally demonstrate the superiority and feasibility of our proposed TiDGAP for UMPIPE, we choose two typical neural networks (VGG16 and GPT-1) and some self-designed networks, and then execute the experiments in realistic environments. VGG16 (trained in MNist dataset) and GPT-1 (in WikiText-2 dataset) are respectively typical CNN-based DNN in CV and transformer-based DNN in NLP. Then self-designed DNNs are based on CNN, whose configuration can be seen in Table 5-5. In order to compare algorithms at the same computational speed level, we apply our proposed two-level improvements to the baseline algorithms to greatly improve their computational speed. As

the local greedy algorithm does not have continuous search capability, we use its convergence solution to supplement the curve of subsequent time.

Table 5-5 Detail of Self-designed CNNs.

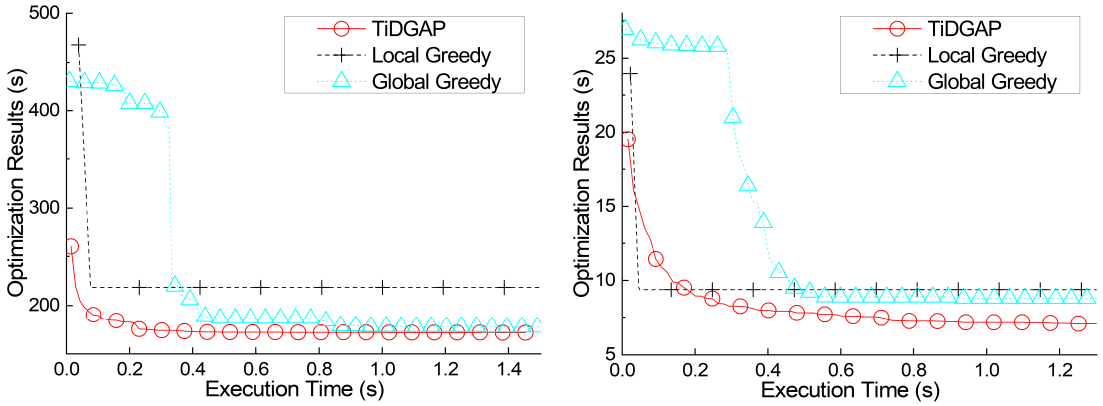
Layer	Types	Input Channel	Output Channel	Kernel
L_1	CNN	$In_1 = 1$	$Ou_1 = \mathrm{U}(20, 50)$	
L_x	CNN	$In_x = Ou_{x-1}$	$Ou_x = \mathrm{U}(20, 50)$	3×3
L_K	FC	$In_K = Ou_{K-1}$	$Ou_K = 10$	

For VGG16 and GPT-1, we carry out the training on the servers with multiple Tesla V100 GPUs and obtain the basic time functions. We do not use the solution of GPipe as the initial solution of these algorithms, i.e., the initial states of the algorithms participating in the comparison are all randomly generated. Then, the optimal results over the execution time of TiDGAP and baselines (local greedy and global greedy) are plotted in Fig. 5-9. From the overall trends in Fig. 5-9, it can be seen that solutions of TiDGAP are better than baselines over time. When the time is long enough, TiDGAP achieves the best convergence solutions in both VGG16 and GPT-1 followed by global greedy and local greedy. Although the global greedy algorithm also has search-ability, each search step in it requires calculating the overall training time corresponding to the current optimization solution, which consequently consumes redundant computational complexities. Moreover, the global greedy algorithm can only search for one solution at a time. TiDGAP, based on the individual evolution strategy of the genetic algorithm, can solve multiple solutions simultaneously, which makes it less likely to fall into local optima. The optimal solutions of TiDGAP at the 100-th generation and the solutions of baselines at the corresponding time point are listed in Table 5-6. Concretely, the optimization results of TiDGAP are 171.71s and 7.06s respectively for GPT-1 and VGG16, reducing the training time by $(3.25, 17.78)\%$ compared to global greedy and by $(21.44, 24.68)\%$ compared to local greedy algorithm.

To show the role of UEDP in the solution process of UMPIPE's training scheme, Fig. 5-10 provides waterfall charts of the current generation's optimal partition number for TiDGAP in solving the optimization scheme. It can be seen that the optimal number of partitions for each process of DNN throughout the entire solving process has always been unequal, revealing the advantage of UEDP. In fact, GG and TiDGAP can be combined to establish a growable genetic algorithm using GG as the growth route of TiDGAP, which

Table 5-6 The quantitative optimization results of TiDGAP, local greedy (LG) and global greedy (GG) in the experiments of Fig. 5-9 with randomly generated initial solutions.

Scenarios	TiDGAP	LG	GG	$\epsilon^{\mathrm{LG}}_{\mathrm{TiDGAP}}$	$\epsilon^{\mathrm{GG}}_{\mathrm{TiDGAP}}$
GPT-1	171.71	218.56	177.47	21.44%	3.25%
VGG-16	7.06	9.37	8.80	24.68%	17.78%

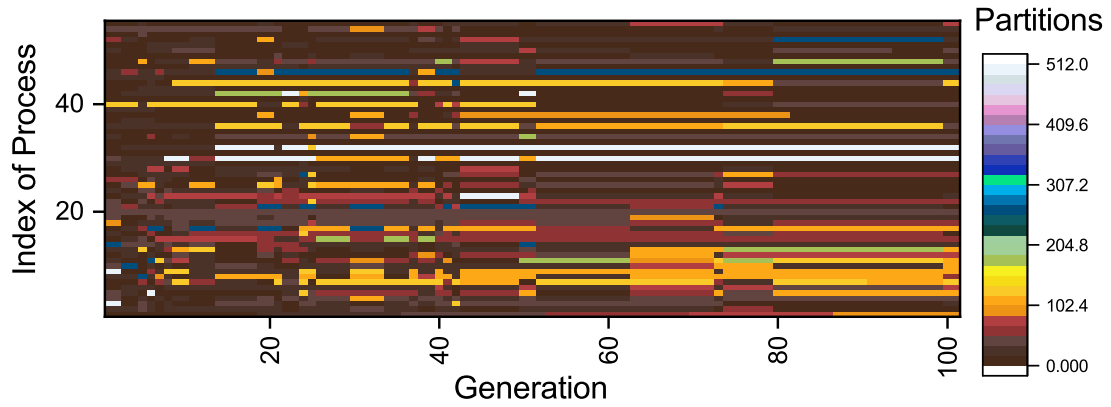


(a) GPT-1, minibatch-size is 512 (b) VGG16, minibatch-size is 64

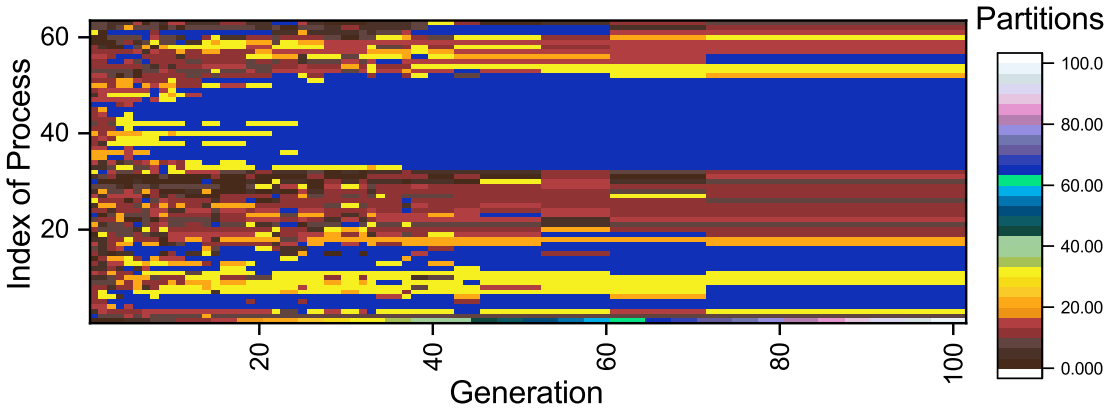
Figure 5-9 The optimization results (i.e., time for training one minibatch) over times in realistic environments for GPT-1 and VGG16 comparing TiDGAP with local greedy and global greedy algorithms with randomly generated initial solutions, where: $N_p = 100$, $N_g = 100$, under distributed systems with Tesla V100 GPUs, randomly initializing N_p individuals.

will have improved performances in terms of convergence and optimality. The growable genetic algorithm is an innovative framework, allowing different algorithms to serve as growth routes to solve optimization problems [87]. As this chapter focuses on proposing the novel UMPIPE parallel architecture and its corresponding optimization algorithm with two-level improvement to accelerate DGAP, we do not delve into the discussion of the growable genetic algorithm for UMPIPE.

In order to increase the comprehensiveness of the validation scenario, we perform training of self-designed DNNs on RTX 3060Ti GPUs to obtain the basic time functions. In self-designed DNNs, we change the number of layers to observe the trend of algorithms. We use the stable convergence solution as the ordinate. Then, the optimal results and relative reduction with different numbers of layers are plotted in Fig. 5-11.

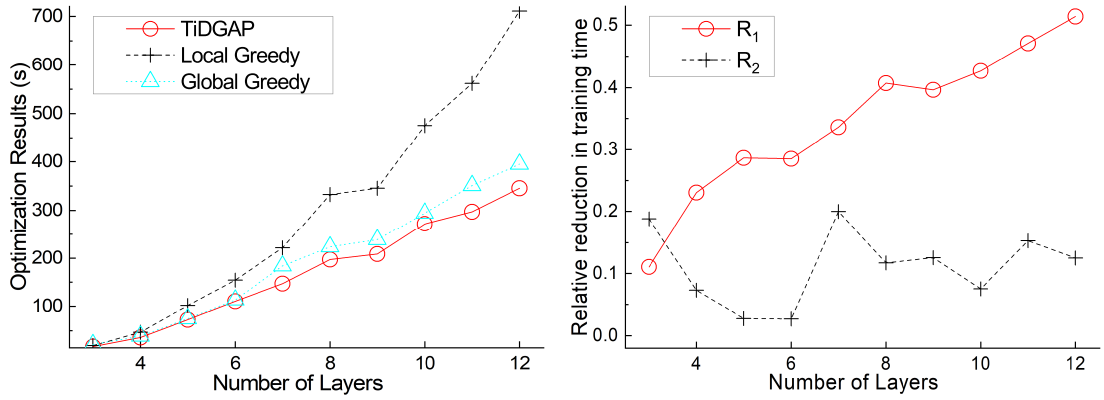


(a) GPT-1, minibatch-size is 512



(b) VGG16, minibatch-size is 64

Figure 5-10 Waterfall charts of optimal partitions of TiDGAP in Fig. 5-9.



(a) Optimization training time (b) Relative reduction in training time

Figure 5-11 The results of self-designed CNN-based networks for 100 minibatches with different numbers of layers in realistic environments comparing TiDGAP with LG and GG algorithms, where $N_p = 100$, $N_g = 100$, randomly initializing N_p individuals. $R_1 = \epsilon^{\mathrm{LG}}_{\mathrm{TiDGAP}}$, $R_2 = \epsilon^{\mathrm{GG}}_{\mathrm{TiDGAP}}$.

In Fig. 5-11(a), the curves of TiDGAP remain the lowest followed by global greedy and local greedy. The performance ranking is consistent with that of Fig. 5-9. Combined with Fig. 5-11(a), the results validate the advantages of our proposed TiDGAP are universal. From Fig. 5-11(a), it can be seen that as the number of layers increases, the absolute differences between our proposed TiDGAP and the comparison baselines become increasingly larger. In Fig. 5-11(b), $\epsilon^{\mathrm{LG}}_{\mathrm{TiDGAP}}$ shows an increasing trend from 0.1 to 0.5 with the number of layers. This is because the feasible solution space of UMPIPE increases exponentially with the number of layers increases, which leads to a decrease in the algorithm's solving ability. LG does not have search capability and decreases faster than TiDGAP. Additionally, $\epsilon^{\mathrm{GG}}_{\mathrm{TiDGAP}}$ fluctuates between 0 and 0.2. TiDGAP can maintain its advantage of around 10%.

5.5.5 EX_4: Evaluation of UEDP Compared UMPIPE with State-of-the-Art Parallelism

Table 5-7 The comparison of various parallel architectures (GPipe, UMPIPE, PipeDream, UMPipeDream (UPD)) in different scenarios.

Scenarios	Networks	Training Time (s) for One Minibatch				$\varepsilon^{\mathrm{GPipe}}_{\mathrm{UMPIPE}}$	$\varepsilon^{\mathrm{PipeDream}}_{\mathrm{UMPIPE}}$	$\varepsilon^{\mathrm{PipeDream}}_{\mathrm{UPD}}$
		UMPIPE	GPipe	PipeDream	UPD			
Typical Nets	GPT-1	167.939	191.264	186.568	164.389	**13.89%**	**11.09%**	13.49%
	VGG16	7.060	8.268	7.622	6.462	**17.11%**	**7.96%**	17.95%
Sim. Nets	$(N=2,P=512)$	97	271	203	67	178.38%	109.28%	202.98%
	$(N=2,P=1024)$	103	595	456	81	477.67%	342.72%	462.96%
	$(N=5,P=512)$	207	1141	957	170	451.21%	362.32%	462.94%
	$(N=5,P=1024)$	254	1058	894	211	316.54%	251.97%	323.70%
	$(N=8,P=512)$	353	1694	1610	303	379.89%	356.09%	431.35%
	$(N=8,P=1024)$	409	1735	1647	394	324.21%	302.69%	318.02%
	$(N=10,P=512)$	616	2007	1911	482	225.81%	210.22%	296.47%
	$(N=10,P=1024)$	478	2220	2024	398	364.44%	323.43%	408.54%
	$(N=20,P=512)$	786	3855	3759	722	390.46%	378.24%	420.64%
	$(N=20,P=1024)$	1000	4055	3961	895	305.50%	296.10%	342.57%
	$(N=50,P=512)$	2297	9963	9865	2150	333.74%	329.47%	358.84%
	$(N=50,P=1024)$	2102	9853	9645	1987	368.74%	358.85%	385.41%

Note: $\varepsilon^{\mathrm{GPipe}}_{\mathrm{UMPIPE}} = \frac{\mathrm{GPipe}_{\mathrm{training\ time}} - \mathrm{UMPIPE}_{\mathrm{training\ time}}}{\mathrm{UMPIPE}_{\mathrm{training\ time}}}$ means the ratio of improvement in training speed of UMPIPE compared to GPipe. Using recursive algorithms to obtain optimized 1F1B schemes for PipeDream and UMPipeDream; using the enumeration to obtain optimized EDP schemes for GPipe and PipeDream; using genetic algorithm (through 4000 generations for the sake of sufficient optimization) to obtain the optimized UEDP schemes for UMPIPE and UMPipeDream; using the training time of optimized scheme solved as the representative of the corresponding architecture's performance.

To demonstrate the superiority of our proposed parallelism UMPIPE and UEDP, we compare it to GPipe and PipeDream in this group of experiments.

To further verify the potential of UEDP for various parallel architectures based on the control variable method, we also include the architecture that combines UEDP with pipedream (called UMPipeDream referring to the naming convention of UMPIPE where UMPipeDream $=$ UMPIPE $+$ 1F1B $=$ PipeDream $+$ UEDP) in the comparison. Due to the lack of fast calculation formulas for PipeDream and UMPipeDream, our experiments use recursive algorithms to calculate the training time corresponding to the data partitioning scheme of PipeDream or UMPipeDream to support the optimization. In experiments, the optimal data partitioning schemes of GPipe and PipeDream are obtained through the enumeration method, as EDP allows enumeration; that of UMPIPE and UMPipeDream are solved by the genetic algorithm with 4000 generations (for the sake of obtaining sufficiently optimized results to reflect the inherent performance of the architectures themselves). Experiments include two representative networks VGG16 (with CNN layers, minibatch-size is 64) and GPT-14 (with transformer layers, minibatch-size is 512) in realistic environments executed with multiple Tesla V100 GPUs, as well as multiple simulation networks generated by realistic devices-based random simulation environments. The optimization results (corresponding to training time) for each architecture in different scenarios are shown in Table 5-7.

From Table 5-7, UMPIPE is generally superior to GPipe and PipeDream, indicating UMPIPE has certain advantages as a new parallel architecture. The comparison between UMPIPE and GPipe demonstrates that UEDP can improve the speed of AFAB (all forward all backward) architectures. Compared with PipeDream, UMPipeDream achieves faster training speed, indicating that UEDP also has a positive effect on accelerating 1F1B (one forward one backward) architectures such as PipeDream. Moreover, UMPipeDream overall achieved the best results, indicating the potential of extending UEDP to other parallel architectures. In GPT-1 and VGG16, UMPIPE accelerates the training speed by $(13.89, 11.09)\%$ and $(17.11, 7.96)\%$ respectively compared with (GPipe, PipeDrea); UMPipeDream accelerates the training speed by 13.49% and 17.95% respectively compared with PipeDream. In the simulation scenarios, the heterogeneity of the network layer is more apparent. Thus, UEDP-based parallel architectures (UMPIPE and UMPipeDream) have a greater improvement on the basis of EDP-based parallel architectures (GPipe, PipeDream), with an improvement $\geq (170\%, 100\%)$ in terms of training speed.

When evaluating the optimization scheme under the 1F1B architecture in experiments, we calculated the optimal 1F1B scheme under the given data partition as its evaluation value. Therefore, the optimization results of PipeDream in the experiment represent the performance of architectures such as PipeDream and Dapple [183, 184]. Therefore, the above results can not only demonstrate the significance of UMPIPE in reducing the training time under microbatch-based pipeline parallelism in AFAB, but also reveal the potential of UEDP in improving various architectures in 1F1B such as PipeDream and Dapple. It is notable that the UMPIPE architecture can be further improved, because adding 1F1B strategy on the basis of UMPIPE (i.e., UMPipeDream) can enhance the training speed on the basis of UMPIPE according to the experimental results in Table 5-7. This chapter addresses a challenge: for UMPIPE with the addition of UEDP on the basis of GPipe, we propose the matrix operations-based fast calculation formulas (i.e., Eq. 5-17 and Eq. 5-18) to simultaneously evaluate multiple training schemes of UMPIPE, which allows optimization algorithms to complete the search for optimization solutions of data partitioning in a short period of time. However, for UMPipeDream (UMPipeDream $=$ UMPIPE $+$ 1F1B $=$ PipeDream $+$ UEDP $=$ GPipe $+$ UEDP $+$ 1F1B), the combination of UEDP and 1F1B strategies cause a more complex calculation process for the training time corresponding to the scheme. The recursive algorithm chosen in experiments for UMPipeDream consumes a significant amount of computation time, which also inspires future work to focus on deriving fast calculation formulas for UMPipeDream (i.e., the further extension of UMPIPE). In addition, UEDP will bring additional programming work and data-switching processes in parallel architecture, which require further research on generalization.

In order to supplement the practical application significance of UMPIPE, we need to verify that the UEDP of UMPIPE does not worsen the DNNs' accuracy over epochs during the training process. We choose two self-designed CNN-based networks with 5 layers and carry out the training respectively in MNist dataset and CIFAR10 dataset. The configures of networks and data partitioning schemes of UMPIPE are listed in Table 5-8, where "U_1" means UMPIPE_1 (the scheme of UMPIPE with index of 1), the column "Or". means the original structure without data partitions, GPipe-2 means dividing the minibatch into two microbatches for all layers in GPipe, and type of layer $C(1, 20)$ means CNN layer with the input channel as 1 and output channels as 20, and $FC(160, 10)$ means full connection layer with input neurons as 160 and output neurons as 10. The minibatch-

Table 5-8 The configures of networks for MNist and CIFAR10 dataset with their data partitioning schemes of parallelism.

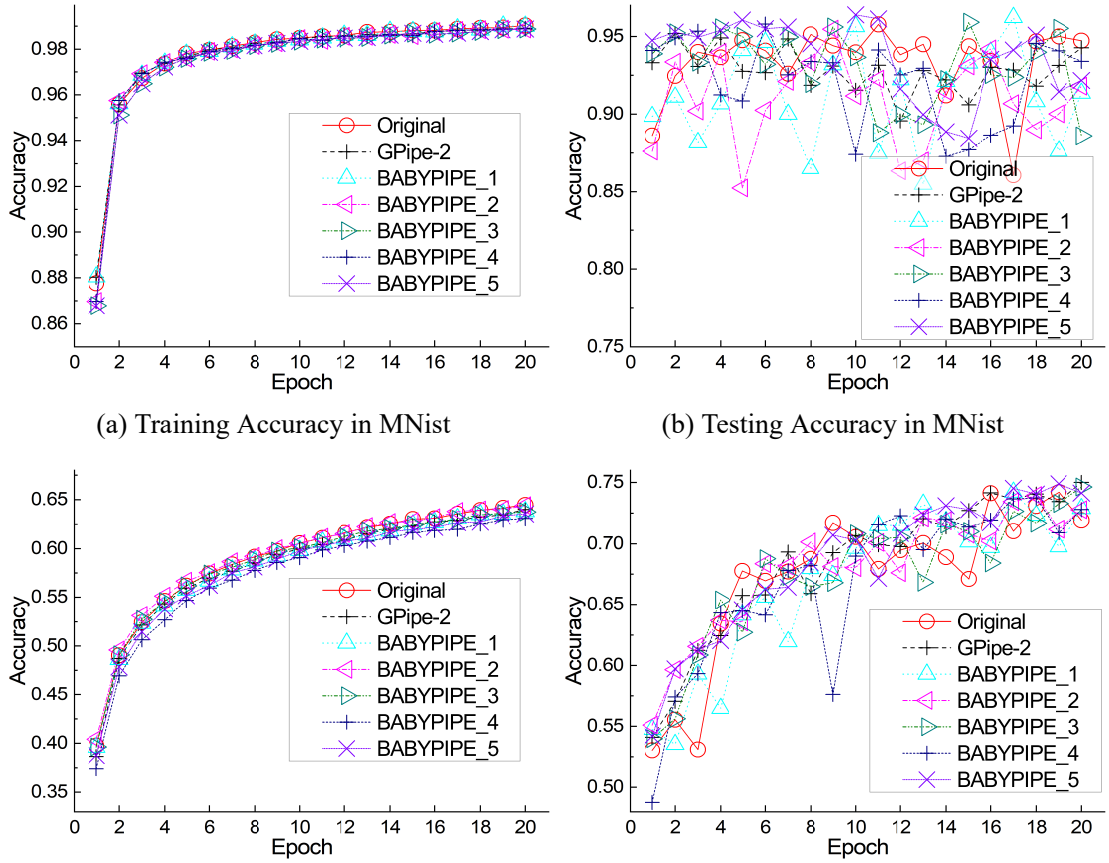
Dataset	Layers	Type	Data Partitions Schemes						
			Or.	GPipe-2	U_1	U_2	U_3	U_4	U_5
MNist	1	$C(1,20)$	1	2	2	2	2	2	4
	2	$C(20,40)$	1	2	2	2	2	8	2
	3	$C(40,80)$	1	2	2	1	2	4	4
	4	$C(80,160)$	1	2	1	2	2	1	8
	5	$FC(160,10)$	1	2	2	1	4	1	1
CIFAR10	1	$C(3,16)$	1	2	2	2	2	8	2
	2	$C(16,32)$	1	2	2	2	1	2	2
	3	$C(32,64)$	1	2	1	1	2	1	4
	4	$C(64,128)$	1	2	2	1	2	1	2
	5	$FC(128,10)$	1	2	2	2	1	1	2

sizes of Table 5-8 are both 64, and the convolution kernels are all 3×3. For the sake of presentation, we use the power of 2 as the number of microbatches for each layer. UMPIPEs with different subscripts correspond to different UEDP schemes. As communication doesn't affect the accuracy of over epochs, we don't consider the data partitions of communication processes. In this set of experiments, forward propagation and backward propagation in the same layer have the same number of data partitions, which does not affect the experimental conclusion, because the main process that determines the convergence of network training is the gradient descent in backward propagation. Then, we plot the training accuracy and testing accuracy within 20 training epochs in Fig. 5-12. The results of Fig. 5-12 show that the accuracy trends of UMPIPE with UEDP do not lag behind the original or GPipe-2. This demonstrates that UMPIPE will not worsen the DNNs' accuracy over epochs. With less time per epoch, UMPIPE will have better convergence over time than GPipe.

5.5.6 Summary of Experiments

Through the multiple groups of experiments from various perspectives in this section, we can observe that BABYPIPE with TiDGAP, as a novel architecture for parallel training, has significant advantages in improving the training speed of DNN. Among these experiments:

(1) EX_1 not only demonstrates that the convergence of dual-chromosome is better than



(a) Training Accuracy in MNist

(b) Testing Accuracy in MNist

(c) Training Accuracy in CIFAR10

(d) Testing Accuracy in CIFAR10

Figure 5-12 The accuracy over epochs in self-designed CNN-based networks in MNist and CIFAR10 dataset comparing UMPIPE with GPipe, where the data partitioning schemes of UMPIPE are listed in Table 5-8, minibatch-size is 64.

that of single-chromosome, but also demonstrates the advantages of BABYPIPE over GPipe through the scenarios with the randomly generated basic time functions of DNN.

(2) EX_2 demonstrates the acceleration effectiveness of two-level improvement to simultaneously calculate the end time corresponding to multiple individuals and multiple microbatches. TiDGAP only consumes seconds of execution time to obtain competitive optimization solutions.

(3) EX_3 demonstrates the optimality of our proposed TiDGAP for BABYPIPE in realistic environments compared with baselines local greedy and global greedy. Through the experiments in GPT-1, VGG16 and self-designed CNNs, TiDGAP achieves the best optimization solution.

(4) EX_4 demonstrates the superiority of BABYPIPE in realistic environments and simu-

lation datasets compared to GPipe and PipeDream related parallelism. In EX_4, we not only verify the parallel scheme of BABYPIPE has less training time, but also verify unequal microbatch-sizes of BABYPIPE do not worsen the training convergence of accuracy over epochs.

5.6 Summary of this Chapter

Based on the microbatch-based pipeline parallelism, this chapter proposes unequal microbatches-based (i.e., unequal data partitions-based) pipeline parallelism (UMPIPE), not only considering computation time and communication time simultaneously, but also considering them probably nonlinear with data size. This chapter derives the recurrence formula for the training time of DNN under UMPIPE parallelism and proves the optimality of UMPIPE in theory.

To obtain the optimization scheme of UMPIPE, this chapter proposes the dual-chromosome genetic algorithm (DGAP). Dual-chromosome is proved with better convergence than the single chromosome for solving training scheme of UMPIPE. Aiming at accelerating DGAP algorithm, we further delve into theoretical derivations of the recurrence formula for UMPIPE and propose the two-level improved DGAP (TiDGAP). TiDGAP can simultaneously calculate the end time of multi-schemes and multi-microbatches for UMPIPE.

The experimental results comprehensively get corroboration to our theoretical analysis, demonstrating the advantages of dual-chromosome strategy and matrix operation-based two-level improvement method of TiDGAP. Compared with baseline optimization methods (local greedy and global greedy), TiDGAP has better convergence and optimality. Compared with baseline parallelism (GPipe and PipeDream), UMPIPE achieves less training time. Compared to (GPipe, PipeDream), UMPIPE improves training speed by $(13.89, 11.09)\%$ in GPT1-14, $(17.11, 7.96)\%$ in VGG16, and $\geq (170\%, 100\%)$ in other simulation networks.

The significance of our proposed BABYPIPE with TiDGAP is that it not only provides a novel architecture for parallel training, but also provides a new solution approach. The introduction of unequal data partitioning brings more optional schemes for parallel training, which actually further optimizes the granularity of distributed parallelism. BABYPIPE with TiDGAP will still have superiority in the hybrid parallel architecture (e.g., 3D parallelism), which we plan to focus on studying. Moreover, the growable ge-

netic algorithm combining TiDGAP with global greedy or other search algorithms to find the optimal training schemes is also a potential direction.

Chapter 6 Hierarchical Cloud System and Machine Learning based Algorithm Selectors

Cloud computing environment is becoming increasingly complex due to its large-scale information growth and increasing heterogeneity of computing resources. Hierarchical Cloud computing dividing the system into multi-levels with multiple subsystems to support the adaptability to abundant requests from users has been widely applied and brings great challenges to resource scheduling. It is critical to find an effective way to address the complex scheduling problems in hierarchical Cloud computing, whose scenarios and optimization objectives often change with the types of subsystems. In this chapter, we propose a scheduling framework to select the scheduling algorithms (SFSSA) for different scheduling scenarios considering no algorithm well suitable to all scenarios. To concretize SFSSA, we propose deep learning-based algorithms selectors (DLS) trained by labeled data and deep reinforcement learning-based algorithms selectors (DRLS) trained by feedback from dynamic scenarios to complete the algorithms selection regarding the scheduling algorithms as selectable tools. Then, we apply strategies including pre-trained model, long experience reply and joint training to improve the performance of DRLS. To enable the quantitative comparison of selectors, we introduce a weighted cost model for the trade-off between solution and complexity. Through multiple sets of experiments in hierarchical Cloud computing with multi subsystems for five types of scheduling problems and varying weights of cost, we demonstrate DLS and DRLS outperform baseline strategies. Compared with random selector, greedy selector, round-robin selector, single best selector, virtual best selector and single fast selector, DLS reduces the cost by 47.4%, 46.1%, 33.9%, 47.9%, 19.3%, 18.8% under stable parameter ranges, and DRLS reduces the cost by 41.1%, 40.6%, 11.7%, 42.3%, 11.5%, 12.5% in dynamic scenarios respectively. In experiments, we also validate DRLS has stronger adaptability than DLS in dynamic scheduling scenarios and DRLS using all of strategies achieves the best performance.

6.1 Introduction

The advent of Industry 4.0 and 5G Era is producing an increasing volume of data to Internet putting forward requirements for mighty software systems of network [25]. Cloud

computing, as a large-scale distributed system that provides flexible, reliable, dynamic and high coverage services, has shown significant advantages in meeting the demand of Internet and is playing an indispensable role in various professions including scientific research and engineering production [4]. With its heterogeneous resources and policy of pay-as-you-use, Cloud computing can respond to different types of users' requests such as transmission requests, storage requests, computing requests, etc [10,124]. Currently, Cloud computing has become the foundation for many business applications [2].

The increasing task requests and data transmission make the scale of Cloud computing system gradually expand based on deployment of hardware devices. However, the deployment capability lags behind the increase of Internet data, which presses exactly the management of resources in Cloud. In addition, inappropriate utilization ratio of resources will cause excessive energy consumption, high running time of tasks, and significant burden to systems so as to reduce the quality of service, reduce service life of components and increase CO_2 emission, which presents the necessity of requirement for efficient scheduling [10,124,224]. Various factors containing time, resource state and environment, will affect the optimization objective and scenario, resulting in greatly high complexity of the solution process. Therefore, the research of resource scheduling in Cloud computing has always been a hotspot and nodus in the era of big data, which also influences the position and development of Cloud computing in society.

One novel structure of hierarchical Cloud computing (including mobile edge computing) is to divide the Cloud system into multiple subsystems (Multi-Level Cloud System) to provide specific types of services for corresponding users, which reduces the computational complexity of resource management and achieve considerable performance [48,100]. Thus, research on multi Cloud environments is one of the tendencies to improve the application performance of Cloud [225]. The potential user types are the basis of subsystem division and the types of services, which also cause the difference of equipment composition between different subsystems. This difference determines the various objectives of each subsystem and increases the difficulty of resource management. Resource management has been broadly studied and there are various scheduling algorithms such as heuristic, meta-heuristics and machine learning. Some algorithms, such as Ant colony algorithm [63], NSGA-II algorithm [99] and deep reinforcement learning [2,100], have shown excellent performance in the existing research, whereas the scheduler based on single scheduling algorithm is far from meeting the demand of the actual operation environment, especially

with multi subsystems as no algorithm can fit all scenarios of Cloud computing and perform better than other algorithms currently. Meta-heuristic, deep reinforcement learning and other algorithms, that can resolve the problem of variable optimization objectives and scenarios to a certain extent, require and consume extensive computing resources and time with the increasing complexity of the problem. And some algorithms with certain rapidity such as LPT, FCFS, RR, BFD and Greedy can only address the resource scheduling in simple scenarios with the low requirement as their solution is obviously worse than other complex algorithms [2,101].

The existing contradiction, that determines the upper limit of the profit, is the balance between optimization and complexity of the algorithm. Moreover, the increasing complexity of the algorithm will abate the realistic value to deploy and apply the algorithm as well as cause resource bottleneck and risk in the operation process of Cloud systems. Therefore, the improvement of one algorithm is inadequate to solve this contradiction. Some existing research on the resource scheduling in hierarchical Cloud computing applied DQN [48,100], Joint DC (JCORA) algorithm [226], a penalized successive convex approximation (P-SCA)-based algorithm [20], and VCEPSO based on particle swarm optimization [227] and achieved good performance in their researched problem. However, they mainly focused on the improvement of a single scheduling algorithm without considering the differences between subsystems and between their objectives. In reality, different subsystems in hierarchical Cloud computing may have different equipment compositions to provide different services for various users, which causes the change of optimization objectives concerned by resource scheduling of different subsystems. Therefore, it is considerable to dynamically select algorithms for variable scenarios. This prompts us to consider making full use of the existing algorithm to meet more complex scenes with various objectives in hierarchical Cloud computing with multi subsystems (HCCMS).

In the existing resource scheduling algorithms of Cloud computing and other distributed systems, a phenomenon from observation is that each optimization algorithm has its targeted factors and scenarios since its proposition. Considering another point of view that all the scheduling algorithms belong to human's wisdom and referring to research on algorithm selectors in other fields [228,229], this chapter proposes a scheduling framework to select the scheduling algorithms (SFSSA) by combining advantages of various scheduling algorithms. To concretize SFSSA, this chapter also proposes deep learning-based algorithm selectors (DLS) and deep reinforcement learning-based algorithm selectors (DRLS)

to deal with the resource scheduling problems of HCCMS with different optimization objectives and varying weights of cost. Although training DRL or DL to schedule resources directly will consume abundant computing capacities and time because the input and output spaces of resource scheduling are too large, DLS and DRLS only consume tiny complexities to select the scheduling algorithms. Finally, according to the types of scheduling problems, the weight of solution and complexity, as well as the number of tasks and resources, DLS or DRLS can select an algorithm with good performance from the algorithm pool to minimize the cost of HCCMS.

The main contributions of this chapter can be summarized as:

(1) Scheduling framework to select the scheduling algorithms (SFSSA): this chapter proposes a framework to select scheduling algorithm according to optimization objective and scenarios regarding the scheduling algorithms also as selectable resources, which can integrate the advantages of various scheduling algorithm, decompose the optimization objective and also effectively decompose the computational complexity of the optimization algorithm in complex resource management scenarios of HCCMS.

(2) DLS and DRLS: this chapter utilizes deep learning (DL) and deep reinforcement learning (DRL) respectively to represent the decision-making process of selecting scheduling algorithm to construct DLS and DRLS. Using DL and DRL, the DLS and DRLS can effectively model multiple optimization scenarios and effectively train the strategy of scheduling algorithm selection, which adapts various scenarios of variable optimization objectives. Our proposed DLS and DRLS also explore a novel role for DL and DRL as scheduling algorithm selector of Cloud computing.

(3) Combination of various strategies of DRLS: to further improve the performance of DRLS, this chapter applied three strategies in DRLS including pre-trained model, long experience replay and joint training, where each strategy optimizes the selection results of DRLs to a certain extent. The DRLS using all these three strategies performs best among all the DRL-based selectors.

(4) Multiple sets of experiments: this chapter constructs scheduling algorithms pool and carries on multiple sets of experiments in complex scenarios of HCCMS to verify the superiority of proposed algorithm selector from various sights. In experiments, DLS and DRLS achieve better results in their own corresponding scenarios compared with baseline strategies including random selector (RS), greedy selector (GS), round-robin selector (RRS), single best selector (SBS), virtual best selector (VBS) and single fast

selector (SFS).

The remainder of this chapter is organized as follows. We review the related works in Section 6.2. System model and formulation of scheduling problems in complex scenarios of HCCMS are presented in Section 6.3. The scheduling framework to select the scheduling algorithms (SFSSA) as well as the DL-based and DRL-based algorithms selectors with multiple strategies are proposed in Section 6.4. The design and results of multiple sets of experiments from various sights are presented in Section 6.5. Finally, we conclude this chapter in Section 6.6.

6.2 Related Work

Frequently used approaches for resource scheduling of Cloud computing contain migration such as VMs migration [63], application migration [230], task migration [19], and workload migration [13]; Queuing model such as M/M/S [49] and M/G/1 [231]; multi-phase approach [9]; as well as scheduling algorithm. The core of these approaches is still the scheduling algorithm including several categories according to the solution way: direct allocation algorithms, search algorithms, and machine learning algorithms.

Direct allocation algorithm, mostly heuristic algorithm, is one type of the commonly applied algorithms in realistic Cloud systems that benefit from its low computing complexity, light-weight, analyzability, ease of implementation and deployment. In the direct allocation algorithm, LPT [103], greedy [101], random, RR (Round-Robin) [232], FFD [15], FCFS (First Come First Serve) [233] are frequently utilized algorithms to gain an approximate solution of scheduling problems and have also been applied as baselines of comparison in recent literature. Z. Guan et al. applied Jacobi Best-response Algorithm and proposed a novel globally optimization algorithm based on a combination of the branch and bound framework to solve the resulting non-convex cooperative problem and minimize the costs of Mobile Cloud Computing [59]. Wenhong Tian et al. proposed a 2-approximation algorithm, LLIF, with theoretical analysis and proof to minimize the energy consumption of VMs scheduling [15]. Z. Hong proposed FCFI+ACTI, a QoS-Aware Distributed Algorithm to address multi-hop computation-offloading problem in IoT-Edge-Cloud Computing [61].

Search algorithm, mainly meta-heuristic algorithm, is a local search or global search to get final scheme based on an initial allocation state, which can generally achieve better solution than direct allocation algorithm but will consume more complexity of time and space. This type of algorithm has adaptability to the varied objectives hence applies

to multi-objective optimization problems. Xiao-Fang Liu et al. develop an ant colony system-based approach named OEMACS allocating VMs to reduce energy consumption of Cloud computing [58]. M. Mahil et al. proposed PSO-ACS algorithm incorporating particle swarm optimization and Ant Colony System to optimize energy efficiency and SLA Violation of cloud data centers [83]. MOEAs [9] have performed superiority in multi-objective scheduling problems in Cloud computing. Ali Abdullah Hamed Al-Mahruqi et al. proposed HH-ECO, a Hybrid Heuristic Algorithm using chaotic based particle swarm optimization (C-PSO), to improve optimal makespan and energy conservation [234]. Amir Iranmanesh et al. proposed DCHG-TS, a hybrid genetic algorithm based on load balancing routing, to optimize both cost and makespan for scientific workflow scheduling in Cloud computing [235].

Machine learning, mainly reinforcement learning and deep reinforcement learning usually occupying better optimization solutions in dynamic scheduling problems than search algorithms, adapts various scenarios apt to lifelong learning, but requires abundant data-based training with extensive computing complexity and possesses unpredictable execution results without analyzability, which may cause risk to large-scale Cloud systems in realistic. Combining neural network and NSGA-II algorithm, Goshgar Ismayilov et al. proposed the NN-DNSGA-II algorithm, a multi-objective evolutionary algorithm for dynamic workflow scheduling in Cloud computing [122]. Based on M/M/S queuing model and Q-learning, Ding Ding et al. proposed QEEC, a two phases framework, to enhance the energy efficiency of Cloud computing [49]. However, the state space is high dimensional and continuous resulting in massive computing complexity to train Q-table in scheduling problems. K. Lolos et al. proposed MDP_DT, an adaptive Reinforce Learning with three strategies that Chain Split, Reset Split and Two-phase Split to reduce the size of state space [106]. One other strategy to express the continuous state space in reinforcement learning is the deep neural network. DRL, integrating reinforcement learning and deep learning, such as modified DRL algorithm [108], DQTS [110], Deep Q Network (DQN) [40], ADRL [125] and DeepRM-Plus [2], has been applied to solve scheduling problems in Cloud computing and performed the superiorities of DRL verified by experiments in their chapters.

Additionally, for hierarchical or multi Cloud computing: Haifeng Lu et al. proposed IDRQN to improve the energy consumption, load balancing, latency and average execution time for MEC with multi-area subsystems [48]; Meng Li et al. applied DDQN to improve the system performance for blockchain-enabled M2M communications in EC with

multi groups of M2M network [100]; Hu et al. proposed a MOS (multi-objective scheduling) algorithm based on the particle swarm optimization (PSO) to minimize workflow makespan and cost simultaneously of multi-cloud environment [225]; other algorithms including Joint DC (JCORA) algorithm [226], P-SCA-based algorithm [20], and PSO-based VCEPSO [227] have been proposed to address the resource scheduling problems of hierarchical Cloud computing.

Overall, the existing research mainly targets improving the performance of the scheduling algorithm itself. Differentiating from previous research, this chapter focuses on the algorithms selection of HCCMS, regards scheduling algorithms as selectable resources and proposes a scheduling framework to select the scheduling algorithms (SF-SSA). Additionally, this chapter accords a novel role, i.e. algorithms selectors, to DL and DRL by proposing DL-based selector (DLS) and DRL-based selector (DRLS) for various scenarios.

6.3 Design of Hierarchical Cloud System with Multi-subsystem

To assist the description of the system model and problem formulations, Table 6-1 gives some notations used in this chapter.

6.3.1 System model of Multi-Level Cloud System

Cloud computing integrates the physical or virtual machines of various servers through high-speed Internet to form a resources pool. When a user submits a task request, the Cloud computing management center allocates the resources in the resource pool to the user according to the current operation status of the Cloud computing system and the attributes of the user's request. After allocation, Cloud computing management updates the status of the Cloud system and monitors the operation of the whole system in real-time. The traditional process of Cloud management without subsystem can be seen in Fig 6-1.

Currently, a single Cloud computing resource management center can no longer meet the rapidly growing number of task requests from all over the world. Moreover, a single resource management center is not robust with a low ability to anti risk. During actual operation, multi-point control is usually adopted to manage the Cloud system i.e. according to the attributes of Cloud computing server nodes and the classification of target users, the large-scale Cloud computing system is divided into multiple subsystems to form a

Table 6-1 Notations and Descriptions.

Notation	Description
q	Number of subsystems
k	Index of subsystem
S_k	The subsystem with index k
n_k	The number of server nodes in S_k
N_k	The set of server nodes in S_k
i	Index of server node
R_{ki}	The i-th server node of S_k
T_k	The set of tasks allocated to subsystem S_k
m_k	The number of tasks in subsystem S_k
j	Index of tasks
T_{kj}	The j-th task in subsystem S_k
TS_{ki}	Set of tasks in server node R_{ki}
KS_k	The set of TS_{ki} where $KS = \langle TS_{k1}, TS_{k2}, \ldots, TS_{kn_k} \rangle$
C_{ki}^{CPU}	The maximum capacity of CPU for node R_{ki}
C_{ki}^{DS}	The maximum capacity of DS for node R_{ki}
v_{kji}^{CPU}	The CPU capacity request of T_{kj} for R_{ki}, similarly v_{kji}^{DS}
L_{ki}^{CPU}	The occupied CPU capacity of R_{ki}, similarly L_{ki}^{DS}
ET_{kji}	The processing time of task T_{kj} when executed in R_{ki}
RT_{ki}	The running time of server node R_{ki}
MT_k	The makespan of the k-th subsystem S_k
$U_k(t)$	The cost of the k-th subsystem S_k at time slot $[t, t+\delta t)$
$\Omega(t)$	The cost of the whole system at time slot $[t, t+\delta t)$

multi-level Cloud computing system (hierarchical Cloud computing with multi subsystems (HCCMS)). Similar to but different from the references [48,100], Fig 6-2 presents one architecture of HCCMS.

In Fig 6-2, a user or a group of users submits diverse requests to Cloud system, the classifier in Cloud center will classify these requests and send the classified requests to the corresponding subsystems for processing. Then the management center (scheduler) of the subsystem allocates these requests to specific physical or virtual machines. This HCCMS structure in Fig 6-2 reduces the overall failure probability of system and simultaneously increases the security between subsystems. In case of local subsystem failure, the management center of other subsystems or Cloud center can temporarily take over the

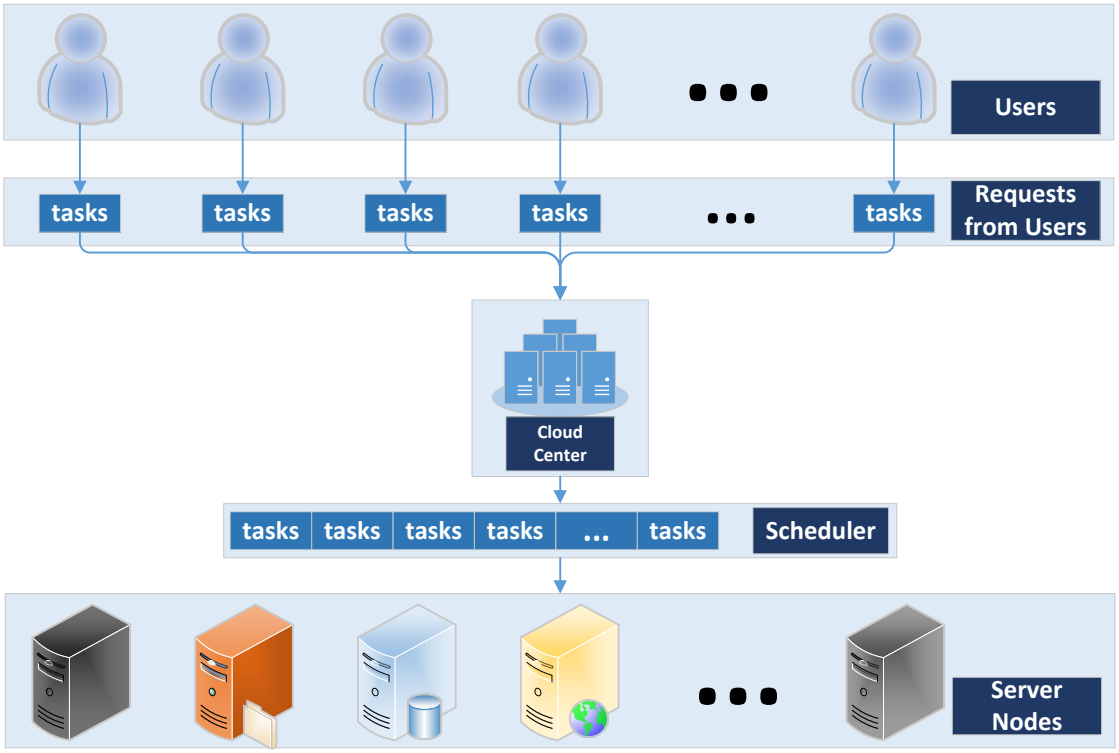


Figure 6-1 The traditional process of Cloud resource management.

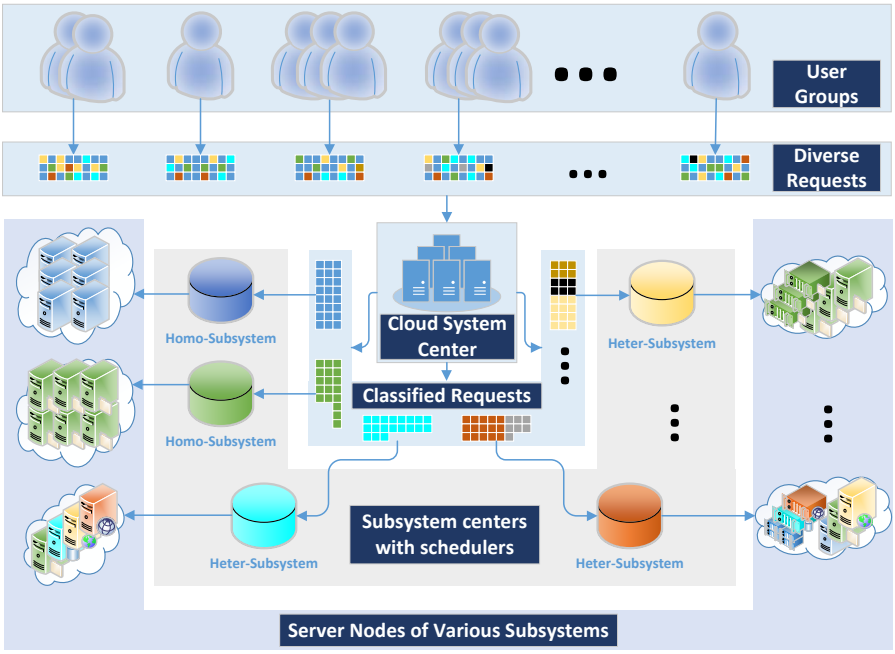


Figure 6-2 The hierarchical Cloud computing with multi subsystems (HCCMS).

resource scheduling of the failed subsystem under the authorization of the Cloud system management center. In order to meet more complex scenarios during operation of Cloud computing, the nodes in all subsystems can be re-divided and combined into new sub-

systems dynamically. HCCMS also indicates that the resource scheduling strategies of different subsystems may be different, which poses a great challenge to Cloud.

Focusing on resource scheduling problems in HCCMS, we consider a HCCMS with q subsystems donated as $S = \langle S_1, S_2, \ldots, S_q \rangle$ and the k-th subsystem with n_k server nodes donated as $N_k = \langle N_{k1}, N_{k2}, \ldots, N_{kn_k} \rangle$. As each server node has a limited capacity of CPU and disk storage (DS), we use C_{ki}^{CPU} and C_{ki}^{DS} to denote them. It can be assumed that the set of tasks allocated to the subsystem S_k at a time slot $[t, t+\Delta t)$ as $T_k = \langle T_{k1}, T_{k2}, \ldots, T_{km_k} \rangle$ with m_k tasks. Then, we use v_{kji}^{CPU}, and v_{kji}^{DS} as the capacity request of task T_{kj}, ET_{kji} as the processing time of T_{kj}, as well as L_{ki}^{CPU} and L_{ki}^{DS} as the load of R_{ki}.

In order to illustrate the advantages of HCCMS, we theoretically analyze the computational complexity of the resource scheduling algorithm in HCCMS. Supposing the computational complexity of an algorithm to allocate m tasks to n resources is $\mathrm{O}(f(m,n))$. For the existing resource scheduling algorithms, except that algorithms with linear complexity such as random algorithm, other algorithms have polynomial or exponential computational complexities and meet the $f'_m(m,n) \geq 0$ and $f'_n(m,n) \geq 0$. Therefore, Eq (6-1) can be obtained based on Jensen Inequality.

$$f(m,n) = f(m,n) + f(0,n) \geq \sum_{k=1}^{q} f(m_k,n) \geq \sum_{k=1}^{q} f(m_k,n_k) \tag{6-1}$$

Especially, when an algorithm has polynomial complexity above quadratic or exponential complexity, the computational complexity satisfies $\exists \alpha \geq 2$ **s.t.** $\mathrm{O}(f(m,n)) \geq \mathrm{O}(m^{\alpha})$. Then, Eq (6-2) can be obtained based on Power-Mean Inequality.

$$f(m,n) \geq q^{(\alpha-1)} \sum_{k=1}^{q} f\left(\frac{m}{q}, n\right) \geq q^{(\alpha-1)} \sum_{k=1}^{q} f\left(\frac{m}{q}, n_k\right) \tag{6-2}$$

According to Eq (6-2), evenly allocating the tasks to subsystems can reduce the computational complexity of resource scheduling to $q^{1-\alpha}$ times. Eq (6-1) and Eq (6-2) demonstrate the significance of HCCMS in improving resource management capability.

With the theoretical analysis of dividing Cloud system into multi subsystems, we can continue to construct its system model. Assuming a server node (PM or VM) is only in one subsystem, we consider a subsystem has two basic classifications i.e. homogeneous and heterogeneous. The homogeneous subsystem means the execution of user's task requests on each node is the same as well as the upper load limit of each node is the same, whose

relationship can be shown as Eq (6-3). Otherwise, the subsystem is heterogeneous.

$$\begin{cases} \left\langle v_{kji_1}^{CPU}, v_{kji_1}^{DS} \right\rangle = \left\langle v_{kji_2}^{CPU}, v_{kji_2}^{DS} \right\rangle, \\ ET_{kji_1} = ET_{kji_2}, \\ \left\langle C_{ki_1}^{CPU}, C_{ki_1}^{DS} \right\rangle = \left\langle C_{ki_2}^{CPU}, C_{ki_2}^{DS} \right\rangle, \\ 1 \leq i_1 < i_2 \leq n_k, 1 \leq \forall j \leq m_k. \end{cases} \tag{6-3}$$

In Cloud computing, the server nodes, commonly used to process the same type of tasks, may approximately be homogeneous. For example, the disks dedicated to the service nodes that processes storage requests may be the same batch of production with the same model and for storage requests, if disks are identical then the server nodes can be regarded as homogeneous. Additionally, the server nodes, which are frequently applied to process some computing requests or comprehensive requests, may be heterogeneous because computing requests and comprehensive requests from different users vary greatly.

In this chapter, we consider the task requests are integral and can not be split into smaller ones, which means any task will be fully allocated to only one server node, however one server node may process more than one task simultaneously. We denote the set of tasks in node R_{ki} as TS_{ki} where if a task T_{kj} is allocated to R_{ki}, then $T_{kj} \in TS_{ki}$. All the TS_{ki} in subsystem S_k constitute a vector $KS_k = \langle TS_{k1}, TS_{k2}, \ldots, TS_{kn_k} \rangle$. As KS_k determines the unique allocation result of tasks in subsystem S_k, it can be leveraged to represent the solution of tasks allocation of subsystem S_k. According to the occupation status of the task or VMs on the server nodes, it can be divided into two categories: a task occupies the server node completely within a certain time (full-occupancy), and the task occupies part of the server node (partial-occupancy). If a task occupies the server node completely, the server node can only process one task simultaneously [33,49], which implies other tasks need to enter the queue and wait for the server node to be idle before being processed. For this situation, running time of a server node RT_{ki} is the main parameter to be considered, and the total running time of the server node is equal to the sum of the processing time of all tasks assigned to this node as Eq (6-4).

$$RT_{ki} = \sum_{T_{kj} \in TS_{ki}} ET_{kji} \tag{6-4}$$

where we consider the processing time of tasks is fixed without affection from the resources' status or the order of tasks. For partial-occupancy of server node, running time is no longer the superposition of tasks' processing time. However, the occupation of various

components in the server node is a more noteworthy parameter. Then, we consider the tasks currently allocated to the server node will occupy the resources of the node simultaneously in a minimum time slot. As the occupancy of components in the server node generally satisfies the relationship of linearly superposition, Eq (6-5) can be given

$$L_{ki}^{CPU} = \sum_{T_{kj} \in TS_{kj}} v_{kji}^{CPU} \tag{6-5}$$

where the similar relationship applies to DS.

In this chapter, we do not address the issue of multi-dimensional resource scheduling, so assume each scheduler only needs to consider the one-dimensional resource. Finally, multiple factors, that are the homogeneity and heterogeneity of server nodes, full-occupancy and partial-occupancy will construct different types of subsystems in Cloud computing, hence increasing the abilities of Cloud systems to provide flexible services.

6.3.2 Subsystems and Subproblems of Resource Scheduling

Based on the types of tasks requests and services, several types of the subsystem in HCCMS can be presented as:

(1) Homogeneous full-occupied subsystem, which is usually utilized to provide specific services for user groups with exclusive needs considering that these users require independence and stability during the service period and the type of their tasks are stationary;

(2) Homogeneous partial-occupied subsystem, which is usually utilized to provide services for the broader user groups with a small capacity of the single task but the huge number of all task requests, such as dedicated subsystem for file storage supporting Cloud disk;

(3) Heterogeneous full-occupied subsystem, which is usually utilized to provide comprehensive or customized services for specific user groups that these users require to independently occupy the allocated server nodes and the types of their tasks are more complex, such as some server nodes for commercial or scientific research activities;

(4) Heterogeneous partial-occupied subsystem, which is constructed with various physical machines and usually utilized to provide comprehensive services for broader users.

For the full-occupied subsystem, the running time applied by users is the main parameter affecting the service capability of server nodes. Thus, optimizing the total running time of the server node is one of the keys to manage the full-occupied subsystem.

For the homogeneous full-occupied subsystem, the processing time of a task is the same for different server nodes, so the total running time of all the server nodes in the homogeneous full-occupied subsystem is invariant when tasks are given. Then, an optimization problem to minimizing the makespan of the homogeneous full-occupied subsystem can be given as Eq (6-6), where the makespan means the maximum running time of all server nodes in subsystem S_k, and the constraints are shown as Eq (6-7).

$$\min \omega_k^{(1)} = \min \left(\max_{i=1,2,\ldots,n_k} \left(\sum_{j=1}^{m_k} x_{ji} ET_{kji} \right) \right) \tag{6-6}$$

$$\textbf{s.t.} \begin{cases} \sum_{i=1}^{m} x_{ji} = 1, \sum_{j=1}^{n} x_{ji} v_{kji} \leq C_{ki}, \\ x_{ji} \in \{0,1\}, j \in \{1,2,\ldots,m_k\}, i \in \{1,2,\ldots,n_k\} \end{cases} \tag{6-7}$$

For heterogeneous full-occupied subsystem, the processing time of the same task may vary on heterogeneous server nodes, which indicates different schemes KS_k of tasks allocation will lead different total running time of subsystem. Therefore in this case, the total running time and makespan are both critical optimization objectives shown as Eq (6-8) also subject to Eq (6-7).

$$\min \omega_k^{(2)} = \begin{cases} \min \left(\max_{i=1,2,\ldots,n_k} \left(\sum_{j=1}^{m_k} x_{ji} ET_{kji} \right) \right) \\ \min \left(\sum_{i=1}^{n_k} \sum_{j=1}^{m_k} x_{ji} ET_{kji} \right) \end{cases} \tag{6-8}$$

For homogeneous partial-occupied subsystem, we consider two types of subsystems classified by their services: one is the subsystem applied to storage requests and the other is that applied to computing requests.

The main component for Cloud storage requests is hard disk which mainly requires load balancing to reduce network congestion. In this chapter, we use variance of server nodes' load of disk storage to measure the degree of balancing. Next, the objective of load balancing can be given as

$$\begin{aligned} \min \omega_k^{(3)} = \min & \frac{1}{n_k} \sum_{i=1}^{n_k} \left(\sum_{j=1}^{m_k} x_{ji} v_{kji}^{DS} \right)^2 \\ & - \frac{1}{n_k^2} \left(\sum_{i=1}^{n_k} \left(\sum_{j=1}^{m_k} x_{ji} v_{kji}^{DS} \right) \right)^2 \end{aligned} \tag{6-9}$$

where the constraints are as Eq (6-7) and $\sum_{j=1}^{m_k} x_{ji} v_{kji}^{DS} \leq C_{ki}^{DS}$.

The computing requests mainly rely on CPU or GPU which usually consume more electrical energy than disk, hence one of the objectives is to reduce the number of working nodes of subsystem (bin packing problem). In this chapter taking the utilization of the CPU as an instance, the single-dimensional bin packing problem can be given as

$$\min \omega_k^{(4)} = \min \sum_{i=1}^{n_k} \max_{j=1,2,\ldots,m_k} x_{ji} \tag{6-10}$$

where the constraints are as Eq (6-7) and $\sum_{j=1}^{m_k} x_{ji} v_{kji}^{CPU} \leq C_{ki}^{CPU}$.

For the heterogeneous partial-occupied subsystem providing comprehensive services for broader user groups, the number of instructions accounts for a high proportion of its tasks to support various tasks. Thus, the CPU is one of the most frequently used components. In this case, balancing the utilization of the CPU and minimizing the total occupied capacity of the CPU are two considerable objectives, hence a bi-objective optimization problem can be given as

$$\min \omega_k^{(5)} = \begin{cases} \min \dfrac{1}{n_k} \displaystyle\sum_{i=1}^{n_k} \left(\sum_{j=1}^{m_k} x_{ji} v_{kji}^{CPU} \right)^2 \\ \quad -\dfrac{1}{n_k^2} \left(\displaystyle\sum_{i=1}^{n_k} \left(\sum_{j=1}^{m_k} x_{ji} v_{kji}^{CPU} \right) \right)^2 \\ \min \displaystyle\sum_{i=1}^{n_k} \sum_{j=1}^{m_k} x_{ji} v_{kji}^{CPU} \end{cases} \tag{6-11}$$

According to the properties of different subsystems, we have given five subproblems of HCCMS summarized in Table 6-2. Although there are still more optimization problems for Cloud computing not listed in Table 6-2, the core target of this chapter is to address the joint scheduling problems of HCCMS with various subproblems. Thus, using these five optimization problems as examples has representativeness actually. For the joint scheduling of more types of scheduling problems, the proposed method in this chapter will still be applicable.

6.3.3 Joint Scheduling Problem and Cost Model for Various Subproblems

In this chapter, we define the joint scheduling problem for various subproblems as optimization problems with variable optimization objectives where the optimization objectives are known before the scheduling of resources. Similar to [236], we also construct

Table 6-2 Five Subproblems in This Chapter.

Sign	Description of Problem
$\omega^{(1)}$	Minimizing makespan for homogeneous resources
$\omega^{(2)}$	Minimizing makespan for heterogeneous resources
$\omega^{(3)}$	Load balancing of DS for homogeneous resources
$\omega^{(4)}$	Bin packing for homogeneous resources
$\omega^{(5)}$	Minimizing standard deviation and total load of CPU for heterogeneous resources

cost model $U_k(t)$ for the trade-off the solution quality $\omega_k(t)$ and computational complexity $\tau_k(t)$ of scheduling algorithm at the time slot $[t, t+\Delta t)$ to uniformly measure the solutions of various subproblems. Without losing generality, we assume the cost equals the weighted sum of optimization results and computational complexity using two weights $w_\omega^{(k)}(t)$ and $w_\tau^{(k)}(t)$ as

$$U_k(t) = w_\omega^{(k)}(t) \cdot \omega_k + w_\tau^{(k)}(t) \cdot \tau_k \tag{6-12}$$

where $w_\omega^{(k)}(t), w_\tau^{(k)}(t) \in \mathbb{R}$. Considering the complexity of realistic scenarios, we regard the weights $w_\omega^{(k)}(t)$ and $w_\tau^{(k)}(t)$ as time-varying for each subsystems. Generally, the objective of minimizing the cost $\Omega(t)$ of the whole system at the time slot $[t, t+\Delta t)$ can be given as

$$\min \Omega(t) = \min \sum_{k=1}^{q} U_k(t) \tag{6-13}$$

6.4 Algorithm Design: Algorithm Selectors based on Machine Learning Methods

6.4.1 SFSSA: Scheduling Framework to Select the Scheduling Algorithms

For the joint scheduling problem for various subproblems as Eq (6-13), a single algorithm is not enough to deal with varying and complex scenarios, which proposes a demand for flexible utilization of various algorithms regarding scheduling algorithms as selectable tools. Naturally, the joint scheduling problem for various subproblems can be converted to: *how to select an appropriate algorithm to schedule resources of a subsystem aiming*

to minimize total cost of HCCMS.

Obviously, the most appropriate algorithm corresponding to different weights may be non-fixed. For example, when $w_{\omega}^{(k)}(t) > 0 \wedge w_{\tau}^{(k)}(t) = 0$ or $w_{\omega}^{(k)}(t) \gg w_{\tau}^{(k)}(t) \geq 0$, an algorithm with high complexity but can get the optimal solution is the most appropriate; contrarily, when $w_{\omega}^{(k)}(t) = 0 \wedge w_{\tau}^{(k)}(t) > 0$ or $w_{\tau}^{(k)}(t) \gg w_{\omega}^{(k)}(t) \geq 0$, a fast algorithm such as greedy algorithm or random algorithm may be the most appropriate. It can be assumed an algorithm pool $A = \langle A_1, A_2, \ldots, A_d \rangle$ has d algorithms and the optimization result and computational complexity of l-th algorithm for the subproblem at the time slot $[t, t + \Delta t)$ of the subsystem S_k are respectively $\omega_{kl}(t)$ and $\tau_{kl}(t)$. If an algorithm A_l can not solve the subproblem of the subsystem S_k, we set $\omega_{kl}(t) = +\infty$ and $\tau_{kl}(t) = +\infty$. Then, Eq (6-13) can be transformed into Eq (6-14) based on: *selecting an appropriate algorithm*, where the constraints are as Eq (6-15).

$$\min \Omega(t) = \min \sum_{k=1}^{q} \sum_{l=1}^{d} y_{kl}(t) \left(w_{\omega}^{(k)}(t) \omega_{kl}(t) + w_{\tau}^{(k)}(t) \tau_{kl}(t) \right) \tag{6-14}$$

$$\textbf{s.t.} \begin{cases} \sum\limits_{l=1}^{d} y_{kl}(t) = 1, y_{kl}(t) \in \{0, 1\}, \\ k \in \{1, 2, \ldots, q\}, l \in \{1, 2, \ldots, d\} \end{cases} \tag{6-15}$$

On the premise of given weights and tasks of all subsystems, the center of HCCMS needs to select a suitable algorithm for each subsystem to minimize the total cost of the system. As the scheduling algorithms can be deployed by multiple subsystems simultaneously, we only need to construct one public algorithms pool. Then, the scheduling framework to select the scheduling algorithms (SFSSA) from the algorithms pool to schedule the resources of the subsystem is as Fig 6-3, and its algorithm is as Algorithm 6-1.

Algorithm 6-1 Scheduling framework to select the scheduling algorithms (SFSSA)

Input : Tasks set $T_k(t)$ and their parameters, objective of scheduling problems $\omega^{(k)}(t)$, weights of cost $w_{\omega}^{(k)}$ and $w_{\tau}^{(k)}$, and algorithms pool A

Output: Selected algorithm A_y also the solution $\langle y_{k1}(t), y_{k2}(t), \ldots, y_{kq}(t) \rangle$ of problem Eq (6-14)

1 Use algorithm selector to select an algorithm from algorithms pool
2 Execute the selected algorithm to generate the solution of scheduling problem
3 Schedule the resources based on the solution
4 Calculate the cost and update the strategies of selector

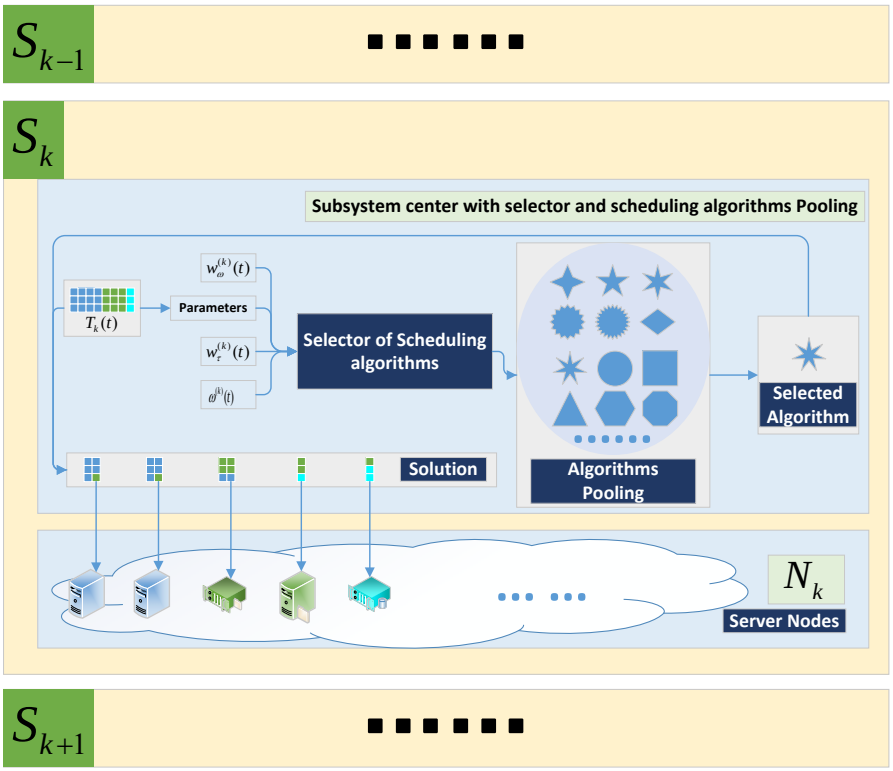


Figure 6-3 The Scheduling framework to select the scheduling algorithms (SFSSA).

In Eq (6-14), if solution $\omega_{kl}(t)$ and complexity $\tau_{kl}(t)$ are given before scheduling, it only needs to select the algorithm with the minimum cost $w_\omega^{(k)}(t)\omega_{kl}(t) + w_\tau^{(k)}(t)\tau_{kl}(t)$. However, before executing a scheduling algorithm, $\omega_{kl}(t)$ and $\tau_{kl}(t)$ are usually unknown. Thus, the directly required function of algorithm selector is to predict $\omega_{kl}(t)$ and $\tau_{kl}(t)$, which nevertheless is difficult. Considering this issue, we convert the prediction of $\omega_{kl}(t)$ and $\tau_{kl}(t)$ to the classification according to the input of Algorithm 6-1 and utilize a classifier to act as the algorithm selector of SFSSA. Now, we can see that two main elements of SFSSA are actually classifier (i.e. algorithm selector) and algorithms pool.

6.4.2 Algorithms Pool

Focusing on the five subproblems presented in Table 6-2, we construct an algorithms pool with various algorithms including heuristic algorithms, meta-heuristic algorithms, local search algorithms, hybrid algorithms and deep reinforcement learning. The specific algorithm and its corresponding subproblems are shown in Table 6-3. In Table 6-3, LPTS, MLPTS, BFDS, BestBFDS, LPT-NS, MLPT-NS, BFD-NS and LPT-BFD-NS are all heuristic-based local search algorithms utilizing heuristic algorithms as search routes.

Table 6-3 Various Types of Algorithms in Algorithms Pool.

Category	Algorithm	Description	Subproblems				
			$\omega^{(1)}$	$\omega^{(2)}$	$\omega^{(3)}$	$\omega^{(4)}$	$\omega^{(5)}$
Randomization	Random	Randomly allocating tasks to resources	✓	✓	✓	✓	✓
Heuristic	Greedy	Scheduling with greed priory	✓	✓	✓	✓	✓
	RR	Round Robin algorithm	✓	✓	✓	✗	✓
	LPT	Longest Processing Time algorithm	✓	✗	✓	✗	✗
	SPT	Shortest Processing Time algorithm	✓	✓	✓	✗	✓
	BFD	Best Fit Decreasing algorithm	✓	✗	✓	✓	✗
	FFD	First Fit Decreasing algorithm	✗	✗	✗	✓	✗
	FF	First Fit algorithm	✗	✗	✗	✓	✗
Meta-Heuristic	ACO	Ant Colony Optimization algorithm	✓	✓	✓	✓	✓
	PSO	Particle Swarm Optimization algorithm	✓	✓	✓	✓	✓
	GA-Random	Genetic algorithm with random initial state	✓	✓	✓	✓	✓
Single Route Local Search	LPTS	LPT-Search algorithm using LPT as search route	✓	✓	✓	✗	✓
	MLPTS	MLPT-Search algorithm using MLPT as search route	✓	✓	✓	✗	✓
	BFDS	BFD-Search algorithm using BFD as search route	✓	✗	✓	✓	✗
	NS	Neighborhood-Search algorithm	✓	✗	✓	✓	✗
	BestBFDS	BestBFD-Search using BestBFD as search route	✓	✓	✓	✓	✓
Multi Routes Local Search	LPT-NS	Using LPT and NS as search routes	✓	✓	✓	✗	✓
	MLPT-NS	Using modified LPT and NS as search routes	✓	✓	✓	✗	✓
	BFD-NS	Using BFD and NS as search routes	✓	✗	✓	✓	✗
	LPT-BFD-NS	Using the LPT, BFD and NS as search routes	✓	✗	✓	✗	✗
Hybrid	GA-MinMin	Genetic algorithm using MinMin initialized state	✓	✓	✓	✓	✓
	ACO-GA	Using the output of ACO as the input of GA	✓	✓	✓	✓	✓
	PSO-GA	Using the output of PSO as the input of GA	✓	✓	✓	✓	✓
	MLPT-GA	Using GA and MLPT as search routes	✓	✓	✓	✓	✓
Machine Learning	DRL	Deep reinforcement learning	✓	✓	✓	✓	✓

The algorithm of the heuristic-based local search algorithm is shown as Algorithm 6-2, which can significantly reduce the computational complexity and enhance the optimality of algorithms. Among Table 6-3, MLPTS and BestBFDS are modified algorithms origi-

nating from LPT and BFD respectively to solve subproblems, as well as their search routes are shown as Algorithm 6-3 and Algorithm 6-4.

Algorithm 6-2 Heuristic-based local search algorithm

Input : Subproblem ω_k and the set of tasks T_k
Output: Solution KS_k of the subproblem
Initially Allocate tasks to resources with confirmed or random initialization policy and gain the general initial status of KS_k
while Exists_Ner **do**
    NoExists_Ner
    Search neighborhood $Ner(KS_k)$ of KS_k in the heuristic-based local search algorithm
    if $KS'_k \in Ner(KS_k)$ ***s.t.*** *the solution of* KS'_k *is optimal than* KS_k **then**
        Exists_Ner
        Choose the optimal neighbor to update $KS'_k \rightarrow KS_k$
Set KS_k as the solution of ω_k

Algorithm 6-3 Modified LPT search route for heterogeneous resources

Input : Tasks $TS = TS_{ki} \cup TS_{kl}$ of R_{ki} and R_{kl}
Output: TS_{ki} and TS_{kl}
$Mark_i = 0$, $Mark_l = 0$, $TS_{ki} = \emptyset$ and $TS_{kl} = \emptyset$
while $TS \neq \emptyset$ **do**
    if $Mark_i \leq Mark_l$ **then**
        $c = i, b = l$
    else
        $c = l, b = i$
    Find tasks $T_{k\alpha_o} \in TS$ **s.t.** $E_{k\alpha_o} = \min\limits_{T_{kj} \in TS} (ET_{kjc} - ET_{kjb})$ to obtain a set of $\{T_{k\alpha_1}, T_{k\alpha_2}, \ldots, T_{k\alpha_s}\}$
    if $s \geq 2$ **then**
        Choose $T_{k\alpha}$ **s.t.** $ET_{k\alpha} = \max\limits_{1 \leq o \leq s} ET_{k\alpha_o c}$
    $Mark_c+ = E_{k\alpha c}$, $TS_{kc}+ = \{T_{k\alpha}\}$ and $TS- = \{T_{k\alpha}\}$

6.4.3 DLS: DL-based Selector of Scheduling Algorithms

After giving algorithms pool, one of method to select appropriate algorithms is Deep Learning-based selector (DLS). As deep learning belongs to supervised learning, a labeled dataset is required to train the DLS. To build a labeled dataset, we randomly generate these

Algorithm 6-4 BestBFD search route

Input : Tasks $TS = TS_{ki} \cup TS_{kl}$ of R_{ki} and R_{kl}
Output: TS_{ki} and TS_{kl}

Set $\gamma = \frac{\sum_{T_{kj} \in TS_{ki} \cup TS_{kl}} ET_{kji}}{2}$, $s = 0$ and $Mark_i = 0$
for *j in tasks from largest to smallest* **do**
    if $Mark_i + ET_{kji} \leq \gamma$ **then**
        Put task in R_{ki} and update $Mark_i += ET_{kji}$
    else
        $s++$
        Mark $|Mark_i + E_{kji} - \gamma| = L_s$
        Mark the scheme as KS_s
Mark the scheme as KS_0 and Choose $\arg\min_{KS} (|Mark_i - \gamma|, L_1, \ldots, L_s)$ as solution

five categories of problems through the simulation system; execute all algorithms in the algorithms pool suitable for the problem; then record the solutions and complexities of these scheduling algorithms; lastly sort the cost of each algorithm from small to large, and set the label value according to the sorting ranking.

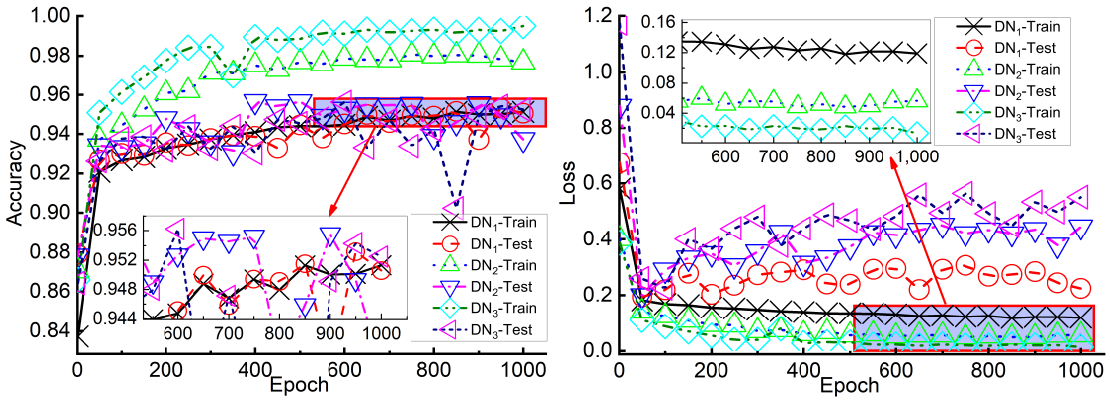
Table 6-4 DNN Structures of DLSs.

Sign	Number of layers	Structure (with full connections)
DN_1	3	$64 \rightarrow 128 \rightarrow 25$
DN_2	4	$64 \rightarrow 128 \rightarrow 256 \rightarrow 25$
DN_3	5	$64 \rightarrow 128 \rightarrow 256 \rightarrow 512 \rightarrow 25$

For DLS, we establish several deep models with full connections as Table 6-4. Tasking 100000 training data and 10000 testing data, the accuracy and loss in training progress of these DLSs are plotted in Fig 6-4. From Fig 6-4, the test accuracies of these three DLSs are about 95% after training. Without losing generality, we choose the structure of DN_3 as the network structure of subsequent DRLSs and use well-trained DN_3 as their pre-trained model.

6.4.4 DRLS: DRL-based Selector of Scheduling Algorithms

DLS depends on the establishment of the labeled dataset marking the current best algorithm. During the operation of the actual Cloud computing system, it is unknown in advance which resource scheduling algorithm is most suitable for the dynamic scenarios. Therefore, the other one method able to act as algorithms selector is Deep Reinforcement



(a) Accuracy. (b) Loss.

Figure 6-4 The Train Process for DL-based Selectors of Table 6-4.

Learning-based selector (DRLS). Based on the decision between exploitation and exploration, the advantages of reinforcement learning are to support non-supervised learning or semi-supervised learning, as well as to support lifelong learning. The utilization of DNN in RL is conducive to adapting to the continuous input space of SFSSA. In this chapter, the base components of DRLS are defined as:

(1) Environment: The environment consisting of a simulated system receives the actions from a agent (or agents) of DRLS, executes the selected scheduling algorithms, then calculates the cost according to the performance of selected algorithms and feedback the reward of selected algorithms.

(2) State space: The state space refers to the current status of each subsystem including the objective of subproblems, weights of cost and tasks set, which are the input of DRLS.

(3) Action space: The action space corresponding to the algorithms pool means selecting a specific algorithm to address the corresponding subproblem.

(4) Policy: The policy is a decision maker and also the selection policy for resource scheduling algorithms in this chapter.

(5) Reward and Loss Function: Training DRLS requires some feedback from the environment. In this chapter, we use two types of feedback to train DRLS with random alternation to accelerate the training process: one is the reward function equal to the reciprocal of the cost i.e. $1/U_k(t)$ and train DRLS with policy gradient shown in Algorithm 6-5; the other is using the best selection among some given strategies and a target net as the label to get the cross-entropy loss and train DRLS by gradient descent. The principle of the combination of two types of feedback is two different gradient

routes can jump out of the local convergence of a single route.

Algorithm 6-5 Update parameters by policy gradient

```
Initialize parameters of networks θ arbitrary or inherit them from DL-based selector
for the experience of each episode ⟨s(1), a(1), r(1)⟩, ..., ⟨s(t), a(t), r(t)⟩ π_θ do
    for i = 1 → t do
        θ → θ + α∇_θ log π_θ (s_i, a_i) v_i
return θ
```

For HCCMS, the policy gradient is as:

$$g = \sum_{k=1}^{q} \left\{ q_\pi \left(s_{tk}, a_{tk} \right) \nabla_\theta \sum_{i=0}^{t} \log \pi \left(a_{ik} | s_{ik}; \theta \right) \right\} \tag{6-16}$$

and the updated parameters are as:

$$\theta = \theta + \alpha g \tag{6-17}$$

where $q_\pi \left(s_{tk}, a_{tk} \right)$ is the score for subsystem S_k based on policy π_θ, $\pi \left(a_{ik} | s_{ik}; \theta \right)$ is the conditional probability of action a_{ik} under status of s_{ik} of subsystem S_k based on current policy π_θ, and $\sum_{k=1}^{q} \sum_{i=0}^{t}$ means using accumulate feedback of all subsystems to update parameters of DRLS.

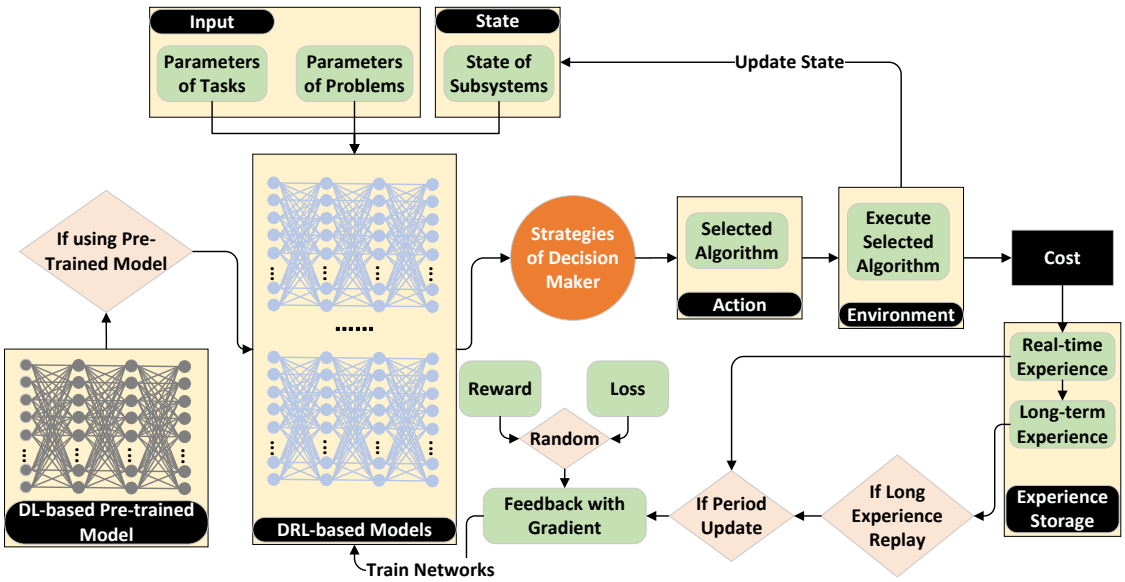


Figure 6-5 A framework of DRL-based selector with various strategies.

Algorithm 6-6 Training of DRLS with multi strategies

```
Initialize parameters of DRL-based networks (randomly initialize or inherit
  parameters from DL-based selector)
for time = 0 → t do
    Generate the objective of subproblem and the weights w_ω^(k)(time) and
      w_τ^(k)(time)
    Generate parameters of tasks
    Input and obtain selection results of DL-based selector, delayed network of
      trained selector, random selector, fixed selector and DRL-based selector at
      time respectively
    Execute selected algorithms of above selectors and calculate costs based on
      w_ω^(k)(time) and w_τ^(k)(time)
    Get the best selection result or ranking according to costs
    Get the reward of the selection result according to the ranking or get the label
      according to the best selection result
    Store reward or label into experience pool
    if replay long-term experience then
        Randomly selected long-term experience from experience pool
    if period update then
        Train the networks of DRL-based selector according to real-time
          experience and long-term experience
    if use delayed network and reach the periodic save point then
        Save current network as delayed network
```

The other method to get the gradient of training DRLS is target labels of action. Combining the preponderance of DDQN and DL, we choose the best action of the following strategies as the target action.

(1) DLS: a well-trained DLS based on the labeled dataset;

(2) Delayed network of DRLS: the DRLS saved regularly during the training process;

(3) Random selector: randomly selecting one algorithm;

(4) Fixed selector: fixedly selecting one or several algorithms;

(5) DRLS itself: One or several DRLS with their own results.

In order to verify and select the appropriate DRLS model, we train multiple selectors based on different strategies. The strategies of DRLS considered in this chapter contain:

(1) Pre-trained model: using the model trained by DLS with labeled dataset as the initial network of DRLS;

(2) Long term experience replay & periodical update (LTPU): the environment stores the experience $e(t) = \langle s(t), a(t), r(t) \rangle$ into replay memory $D(t) = \langle e(1), e(2), \ldots, e(t) \rangle$,

then randomly selects the experiences in the experience memory at a specific time to participate in the training of DRLS;

(3) Joint training: multi-agents of DRL participate in training and generation of target value simultaneously.

Table 6-5 DRL-based Models Combining Various Strategies Trained in This Chapter.

Models	Pre-trained	LTPU	Joint Training
$Model_1$	✗	✗	✗
$Model_2$	✗	✓	✗
$Model_3$	✓	✗	✗
$Model_4$	✗	✓	✓
$Model_5$	✓	✓	✓

With these components, a framework of DRL-based selector can be shown as Fig 6-5. The framework of Fig 6-5 contains multiple components of strategies and the algorithm of training DRLS is shown in Algorithm 6-6. However, a DRL-based model may only utilize some of them and different combinations of them have distinctness in performance. Leveraging various training strategies and their combinations, we train and obtain five DRL-based models as Table 6-5.

6.5 Experimental Results and Analysis

6.5.1 Experiment Setting

Currently, there is limited literature exploring the selector of the resource scheduling algorithm in Cloud computing and some algorithm selection strategies in other research directions contain Random Selector (RS) [236], Greedy Selector (GS) [236], Single Best Solver (SBS) [237,238], Virtual Best Solver (VBS) [226,237], Optimal Decision Tree [239] and Machine Learning [240]. Among these, RS, GS, VBS and SBS are most commonly used as baselines of algorithms selection. Adding two more selection strategies including Round-Robin Selector and SFS (select the fastest single algorithm), we construct the compared baselines of this chapter as Table 6-6.

With baselines in Table 6-6, DLSs in 6-4 and DRLSs in 6-5 to evaluate the performances of DLS and DRLS, we carry out four sets of comparative experiments respectively between:

(1) EX_0: Cloud system without subsystem and HCCMS;
(2) EX_1: Well-trained DLSs in Table 6-4 and baseline strategies in Table 6-6;
(3) EX_2: Dynamically trained DRLS ($Model_1$) and baseline strategies;
(4) EX_3: Dynamically trained $Model_1$ to $Model_5$ in Table 6-5 (various strategies of DRLSs).

These sets of experiments are all based on the variable-controlling approach. Among them, EX_0 fixes specific scheduling problems and changes the number of subsystems to observe the influence of the number of subsystems on the computational complexity and optimization results of several scheduling algorithms. Fixing HCCMS structure, parameters of tasks and server nodes, and parameters of cost model, EX_1, EX_2 and EX_3 are to change algorithm selection strategies so as to observe the impact of different algorithm selection strategies in various scenarios. EX_1 is to evaluate the performance of DLSs compared to baselines in the scenarios with relative static ranges of parameters and simultaneously to test the adaptability of DLSs for the scenarios outside the training dataset. EX_2 is to evaluate the performance of DRLS compared to baselines in the relative dynamic scenarios. And then, EX_3 is to evaluate the performance of different strategy combinations in DRLS. The combination of these four sets of experiments illustrates the advantages of HCCMS structure in resource scheduling, illustrates the advantages of DLS and DRLs in specific scenarios, and finally provides a reference for the choice of algorithm selectors.

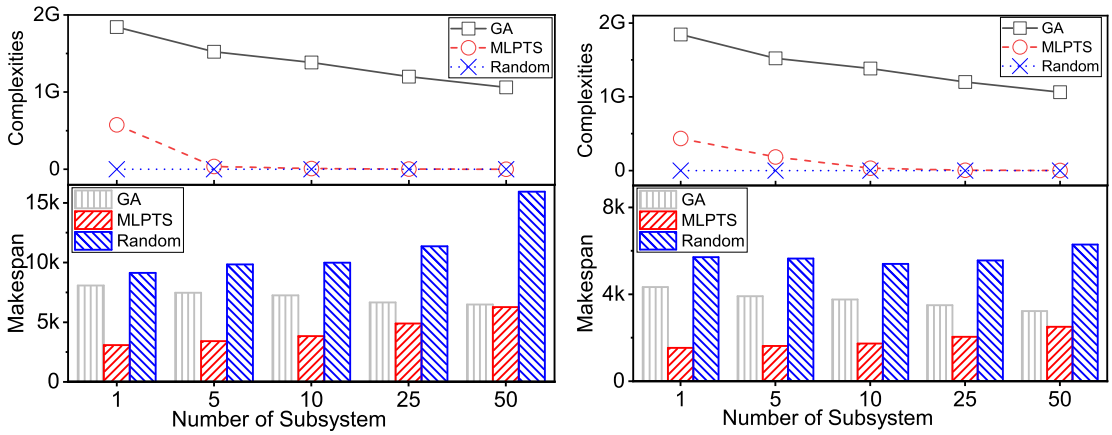
Subsystem type often determines its user type, optimization objective and characteristic parameter. Additionally, different characteristic parameters have different degrees of effects on the cost model. Therefore, we set different parameter ranges according to the characteristic parameters of subsystems for various objectives, which can be seen in Table 6-7.

Table 6-6 Compared Baseline Strategies.

Strategies	Description
RS [236]	Random selector: randomly select an algorithm
GS [236]	Greedy selector: greedily select the algorithm with solution quality or complexity
RRS	Round-robin selector: select each algorithm in turn
SBS [237,238]	Single best selector: select the single algorithm with best average performance
VBS [237,238]	Virtual Best Selector: perfectly select the algorithm with the best previous statistical cost
SFS	Single fast selector: select the fastest algorithm

Table 6-7 Parameter Setting in Experiments.

Objective	Subsystem Type	User Type	Parameter	Parameter Value	Constraint
$\min \omega_k^{(1)}$	Homo. full	Specific users	Processing time ET_{kji}	U(30, 120)min	$RT_{ki} \leq$ 1day
$\min \omega_k^{(2)}$	Heter. full	Specific users	Processing time ET_{kji}	U(30, 120)min	$RT_{ki} \leq$ 1day
$\min \omega_k^{(3)}$	Homo. partial	DS Users	Disk storage v_{kji}^{DS}	U(10^3, 50K)MB	$L_{ki}^{DS} \leq 10^5$MB
$\min \omega_k^{(4)}$	Homo. partial	CPU Users	CPU capacity v_{kji}^{CPU}	U(10, 100)MIPS	$L_{ki}^{CPU} \leq 10^3$MIPS
$\min \omega_k^{(5)}$	Heter. partial	CPU Users	CPU capacity v_{kji}^{CPU}	U(10, 100)MIPS	$L_{ki}^{CPU} \leq 10^3$MIPS



(a) 100 server nodes.

(b) 200 server nodes.

Figure 6-6 The makespan and computational complexities of various system organizations for 10000 tasks.

Then, we simulate the HCCMS through a Python-based simulation environment and launch the experiments on a desktop computer with configurations as follows:

- CPU: Intel(R) Core(TM) i5-8400 CPU @ 2.8GHZ.
- SSD: KINGSTON SA400S37 240GB.
- GPU: NVIDIA GeForce GTX 1060 6GB.
- Program version: Python 3.6, Tensorflow.

6.5.2 Results and Discussion

6.5.2.1 EX$_0$: Single-layer System vs. Multi-layer System

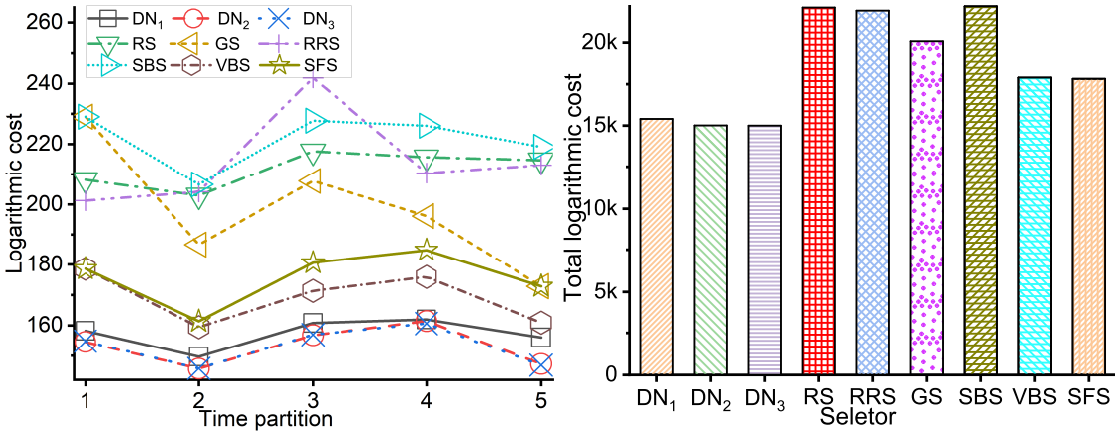
Before experiments of verifying algorithms selectors, we carry out a set of experiments to compare the traditional single-layer system and multi-layer system (HCCMS). To facilitate comparison in this set of experiments, we set the scheduling problem as minimizing makespan for heterogeneous resources (i.e. $\min \omega^{(2)}$). We use three algorithms that Random algorithm, MLPTS and GA (with fixed 600 generations and 100 populations)

for verification of complexities and optimization results. We experiment in five types of system organizations for the same tasks set with 10^4 tasks waiting to be allocated. The parameters of tasks are consistent with those of $\min \omega^{(2)}$ in Table 6-7 and not subject to the constraints, which means each task is randomly generated by the uniform distribution $ET_{kji} \sim \mathrm{U}(30, 120)$min. Using the number of subsystems as the main variable, these five types of system organizations are respectively no subsystem (corresponding to abscissa 1), 5 subsystems, 10 subsystems, 25 subsystems and 50 subsystems. In the system organizations with subsystems, we divide the tasks into the number of subsystems and each subsystem processes the same number of tasks. Then, the results of two experiments respectively with 100 server nodes and 200 server nodes are plotted in Fig 6-6.

Fig 6-6(a) plots the complexities and makespan for the system with 100 server nodes and Fig 6-6(b) for that with 200 server nodes. From Fig 6-6, the complexities of MLPTS and GA both decrease with the increase of the number of subsystems. This validates the relationship of Eq (6-2) that dividing the Cloud system into multi-layer systems with multiple subsystems can significantly reduce the complexities of resource scheduling. The computational complexity of the random algorithm does not vary with the change of system organizations, which is because the random algorithm does not consider the characteristics of the task itself, and its computational complexity is equal to the total number of tasks. The makespan of MLPTS increases with the increase of the number of subsystems, while that of GA decreases slightly and that of random algorithm overall increases but with instability. The increase of makespan using MLPTS algorithm is caused by HCCMS structure where the solution of MLPTS algorithm is close to the global optimal solution but the uneven assignment of tasks to each subsystem leads to the increase of global optimal makespan. This reason can also explain the overall increasing trend of the makespan of random algorithm. Additionally about the decreasing trend of GA, the solution of GA is not close to the global optimal solution and the solution search space of multiple subsystems is far smaller than that of no subsystem, so GA with fixed generations and populations can find a relatively better solution in the scenario of multiple subsystems than that of no subsystem. Specifically discussing 100 server nodes in Fig 6-6(a), the complexity of MLPTS in no subsystem is 575M and that in 5 subsystems is 35.9M, as well as the makespan of MLPTS in no subsystem is 3076 and that in 5 subsystems is 3212, which indicates increasing subsystems from 0 to 5 can reduce the complexity of MLPTS by 93.8% with increase of makespan by 4.4%. This demonstrates that HCCMS provides

a way to significantly reduce the computational complexity in exchange for a small loss of optimization.

On the whole, this set of experiments demonstrates the obvious benefits of multiple subsystems to the resource management of the Cloud computing system. The experiments in the following subsections will also be carried out in the HCCMS structure.



(a) The cost per time partition for 5 time partitions.

(b) The total cost for 100 time partitions.

Figure 6-7 The performance comparison between baseline strategies and DL-based selectors with 25 subsystems for $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

Table 6-8 The comparison of total cost for 100 time partitions with 25 subsystems between DN_3 and baselines in the scenarios of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

Items	DN_3	RS	RRS	GS	SBS	VBS	SFS
Total Cost	15001	22114	21924	20084	22188	17902	17823
Improvement ($I_{baseline-\mathrm{DN}_3}$)	–	7112	6923	5083	7187	2901	2822
$I_{baseline-\mathrm{DN}_3}/\mathrm{DN}_3$ (%)	–	47.4	46.1	33.9	47.9	19.3	18.8

6.5.2.2 EX_1: Baseline Strategies vs. DL-based Selectors

In this set of experiments, we compare baseline strategies corresponding to Table 6-6 with DLSs corresponding to Table 6-4 to evaluate the optimality and adaptability of DLS in various scenarios.

We experiment in the simulated HCCMS with $q = 25$ subsystems, where numbers of each type of problems are all 5. The cost weights of subsystems are also in uniform distribution where $w_{\omega}^{(k)}(t) \sim \mathrm{U}(0, 100)/100$ and $w_{\tau}^{(k)}(t) \sim \mathrm{U}(0, 100)/100$ where $\mathrm{U}(0, 100)$

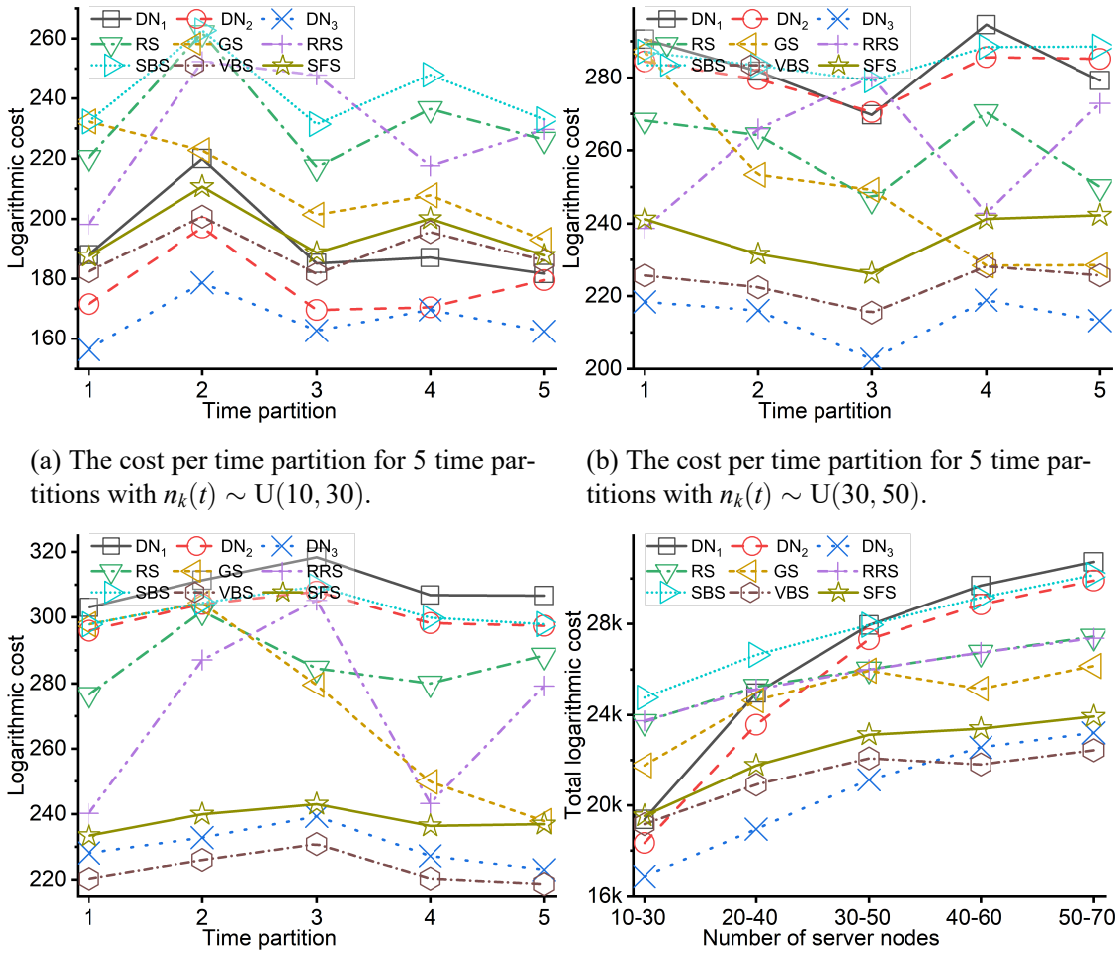
means randomly generating an integer between $[0, 100]$ subject to uniform distribution. As the management of HCCMS in this chapter mainly concerns the cost of the system, thus we only use $\log U_k(t)$ (logarithmic cost) calculated by solution quality and complexity as Eq (6-12) to evaluate the performance of selectors instead of additionally present optimization results in detail such as makespan and standard deviation for each subproblem. Dividing time into multi partitions, the number of the arriving tasks and the available resources in each time partition are generated by the uniform distribution. Additionally, the three DLSs, i.e. DN_1, DN_2 and DN_3 with different structures of networks, have been well trained in dataset for the distributions of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

We carry out experiments in the scenarios of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$: we randomly generate a set of tasks through these probability distributions, execute the selected algorithms using different algorithm selection strategies to schedule of generated tasks to meet the specified optimization objectives, record the execution time and optimization results of their selected scheduling algorithms, and then calculate the logarithmic cost according to the cost model as the index of the performance evaluation. Then, we plot the experiment results of logarithmic cost per time partition and total logarithmic cost in Fig 6-7.

Fig 6-7(a) plots the cost of the system at each time partition resulted from DLSs and the baseline strategies including random selector (denoted as RS in figures), greedy selector (GS), RR selector (RRS), single best selector (SBS), virtual best selector (VBS) and single fast selector (SFS). Each selector processes the same task under the same configuration of subsystems. In Fig 6-7(a), the three DLSs that DN_1, DN_2 and DN_3 obtain less cost than baselines, where DN_3 performs best among all the DLSs. The results in Fig 6-7(a) show that DLS can be used to reduce the cost of resource scheduling and increasing the number of neural network layers can improve the performance of algorithm selection. Fig 6-7(b) plots the total cost for 100 time partitions where the tasks and subsystems for each selector are also the same. In terms of the total cost over a long time span, DLSs still outperform compared baseline strategies.

First of all, Fig 6-7 shows that it is difficult to adapt to all scenarios with only one algorithm or some fixed algorithms, which proves once again the significance of selecting appropriate scheduling algorithms for different scenarios. Moreover, the performance of randomly or greedily selecting an algorithm to solve the resource scheduling problem is unstable. Then, Table 6-8 lists the total costs of Fig 6-7(b) to observe the relative improve-

ment of DN_3 compared with the baseline algorithm. Overall from Table 6-8, the total cost of DN_3 is improved by 47.4%, 46.1%, 33.9%, 47.9%, 19.3% and 18.8% respectively compared to RS, GS, RRS, SBS, VBS and SFS for 100 time partitions with 25 subsystems in the scenarios of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.



(a) The cost per time partition for 5 time partitions with $n_k(t) \sim \mathrm{U}(10, 30)$.

(b) The cost per time partition for 5 time partitions with $n_k(t) \sim \mathrm{U}(30, 50)$.

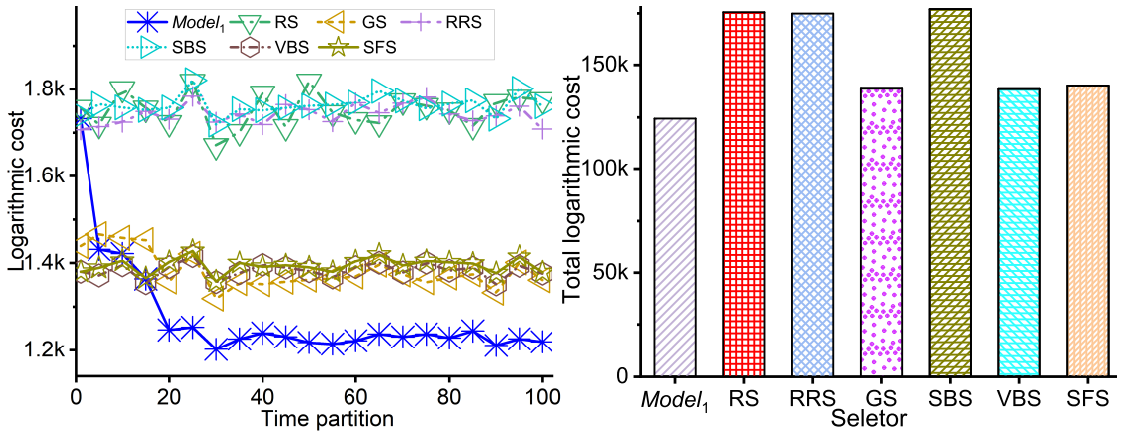
(c) The cost per time partition for 5 time partitions with $n_k(t) \sim \mathrm{U}(50, 70)$.

(d) The total cost for 100 time partitions with varying ranges of server nodes.

Figure 6-8 The performance comparison between baseline strategies and DL-based selectors with 25 subsystems with varying ranges of server nodes and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

In the experiments of Fig 6-7 where DN_1, DN_2 and DN_3 significantly outperform baselines, the ranges of server nodes and tasks are that corresponding to the training dataset, i.e. $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$. This implies DLS can perform well in the parameters they are repeatedly trained. However, models based on deep learning may not be adaptable to scenarios other than their training dataset. Considering this, we continue to evaluate the performance and validity of DLS for untrained parameter ranges.

We changed the ranges of server nodes, retain the ranges of tasks as $m_k(t) \sim \mathrm{U}(n_k(t), 100)$ and carried out several groups of experiments to observe the logarithmic cost of the DN_1, DN_2 and DN_3 which are trained in $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.



(a) The cost per time partition. (b) The total cost.

Figure 6-9 The performance comparison between the DRLS ($Model_1$ being trained) and baseline strategies with 200 subsystems for 100 time partitions in the scenarios of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

Table 6-9 The comparison of total cost for 100 time partitions with 200 subsystems between $Model_1$ and baselines in the scenarios $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

Items	$Model_1$	RS	RRS	GS	SBS	VBS	SFS
Total Cost	124426	175572	174959	138977	177077	138744	140024
Improvement ($I_{baseline-Model_1}$)	–	51146	50533	14551	52651	14318	15598
$I_{baseline-Model_1}/Model_1$ (%)	–	41.1	40.6	11.7	42.3	11.5	12.5

The results are plotted in Fig 6-8, where Fig 6-8(a), Fig 6-8(b) and Fig 6-8(c) plot the logarithmic cost per time partition for 5 time partitions respectively for the ranges $n_k(t) \sim \mathrm{U}(10, 30)$, $n_k(t) \sim \mathrm{U}(30, 50)$ and $n_k(t) \sim \mathrm{U}(50, 70)$. Then, Fig 6-8(d) plots the total cost for 100 time partitions for the ranges from $n_k(t) \sim \mathrm{U}(10, 30)$ to $n_k(t) \sim \mathrm{U}(50, 70)$. In Fig 6-8(a) where $n_k(t) \sim \mathrm{U}(10, 30)$ slightly deviates from the range of training $n_k(t) \sim \mathrm{U}(2, 20)$, DN_1, DN_2 and DN_3 still outperform baselines. However, DN_1, DN_2 and DN_3 differ greater in performance than that of Fig 6-7. DN_3 has significantly lower cost than DN_2 and DN_1, as well as DN_1 has close cost with VBS. In Fig 6-8(b) where $n_k(t) \sim \mathrm{U}(30, 50)$, the differences of cost between DN_1, DN_2 and DN_3 further increase, where DN_3 still outperform baselines but DN_1 and DN_2 achieve worse cost than other

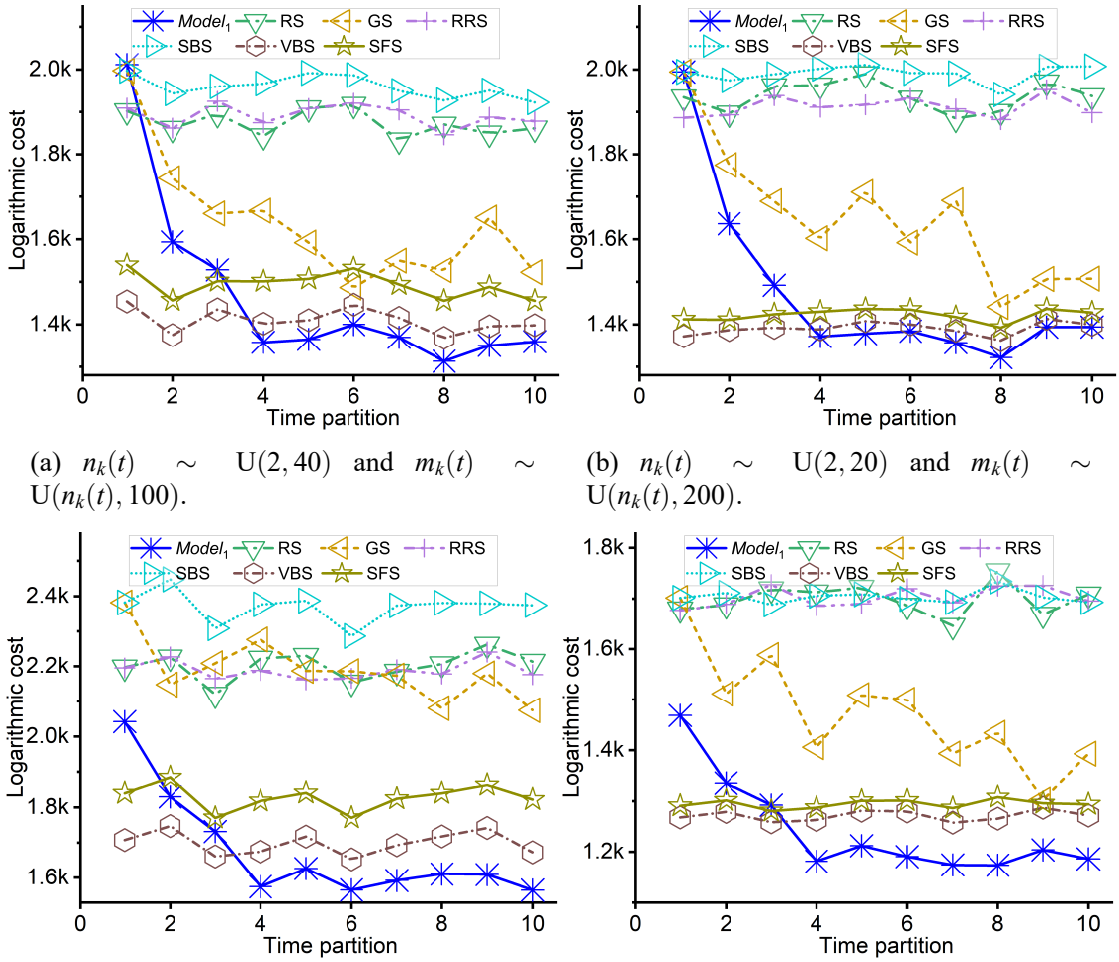
baselines except SBS. When in Fig 6-8(c), performances of the three DLSs continue to decline so that VBS exceeds DN_3 although DN_3 still performs better than other baselines. Fig 6-8(d) provides an overall trend of performance changing with the ranges of server nodes. In Fig 6-8(d), the performance degradation speeds of DN_2 and DN_1 are faster than that of DN_3. When the range of server nodes reaches $U(20, 40)$, the costs of DN_1 and DN_2 become larger than that of VBS and SFS in baselines, as well as until that range reaches or exceeds $U(40, 60)$, the costs of DN_3 become greater than that of VBS and still less than SFS.

Analyzing the overall results of experiments in Fig 6-8 can gain that the performance of DLS, although with certain adaptability, will decline with the ranges of server nodes far away from that trained by DLS. This also means that DLS can maintain performance in static scenes, but cannot guarantee adaptability in dynamic scenes. Comparing DN_1, DN_2 and DN_3 in Fig 6-8 and Fig 6-7, we can see DN_3 is least affected by the range of server nodes in terms of performance followed by DN_2. In addition to the differences in network structure, these three DLSs have the same training strategy and dataset, as well as have approximate accuracies when achieving convergence. Therefore, this reveals a possible reason that a neural network-based DLS with more neurons or connection layers may have stronger adaptability in algorithm selection for varying ranges of parameters, consistent with that the better performing DN_3 has more layers and neurons than DN_2 and DN_1.

6.5.2.3 EX_2: Baseline Strategies vs. $Model_1$

The premise of using DLS is enough historical data in advance to support training. From the above results of Fig 6-7 and Fig 6-8, after repeatedly training within the specified parameter range, DLS did learn the optimal selection within this parameter range, but its performance is still limited for the parameter range outside the training. Therefore, although DLS has better performance than the comparison strategies in static selection, we still need to explore additional selectors to tackle the dynamic selection of scheduling algorithms, which indicates the necessity of DRLS.

In this set of experiments, we compare baseline strategies corresponding to Table 6-6 with a DRLS ($Model_1$) to evaluate the optimality and adaptability of DRLS. We experiment in a simulated HCCMS with $q = 200$ subsystems, where numbers of each type of problems are all 40. The numbers of the arriving tasks and the available resources in each time partition are generated by uniform distribution or other distribution (in some



(a) $n_k(t) \sim \mathrm{U}(2, 40)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

(b) $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 200)$.

(c) Randomly selecting $n_k(t) \sim \mathrm{U}(2, 20)$, $n_k(t) \sim \mathrm{U}(20, 40)$, $n_k(t) \sim \mathrm{U}(30, 70)$, $n_k(t) \sim \mathrm{U}(50, 90)$ with equal probability in each time partition and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

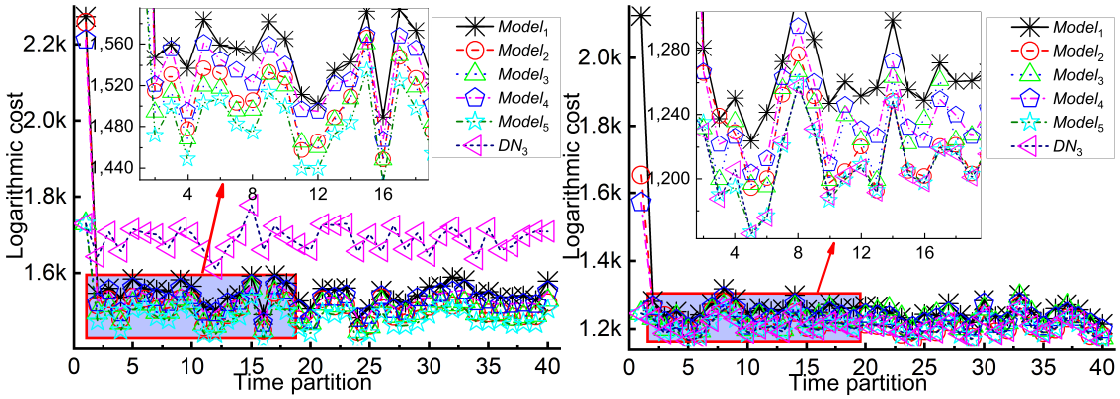
(d) $n_k(t) \sim \mathrm{Tr}(2, 20)$ and $m_k(t) \sim \mathrm{Tr}(n_k(t), 100)$.

Figure 6-10 The cost per time partition of $Model_1$ being trained and baseline strategies with 200 subsystems for 10 time partitions and various distributions of numbers of server nodes and tasks.

experiments). The cost weights of subsystems are also in uniform distribution where $w_{\omega}^{(k)}(t) \sim \mathrm{U}(0, 100)/100$ and $w_{\tau}^{(k)}(t) \sim \mathrm{U}(0, 100)/100$. The above parameter settings are the same as those of the comparative experiments EX_1. The difference is that in the process of training DRLS, it is unknown which scheduling algorithm is the best in advance. DRLS finds a better selection through exploration and exploitation. Additionally, we choose $Model_1$ (without additional training strategies) as the instance of DRLS to observe the inherent advantages of DRL-based algorithm selector.

Since DRLS does not have the ability to optimize decision-making at the beginning

before training which needs enough training to participate in decision-making, we set a long time period with 100 time partitions to observe the logarithmic cost and carry out experiments in the scenarios of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$. Then, the results of the logarithmic costs corresponding to the comparison baseline strategies and DRLS ($Model_1$) are plotted in Figure 6-9.



(a) $n_k(t) \sim \mathrm{U}(2, 50)$, $m_k(t) \sim \mathrm{U}(n_k(t), 200)$. (b) $n_k(t) \sim \mathrm{U}(2, 20)$, $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

Figure 6-11 The performance comparison in train process of the $Model_1$ to $Model_5$ with 200 subsystems for 40 time partitions.

Fig 6-9(a) plots the cost of the system at each time partition. As shown in Figure 6-9(a), before the 10-th time partition, the costs of $Model_1$ are not obviously smaller than the best baseline, which is because $Model_1$ selector has not been well trained. After the 10-th time partition, $Model_1$ can maintain better than comparison strategies with adequate training. This is because DRLS can learn the best selection strategy in the current scenario after enough time partitions. When time goes on, DRLs with lifelong learning will continue to evolve itself. Fig 6-9(b) plots the total cost for 100 time partitions of the training process. In Fig 6-9(b), $Model_1$ achieves the minimum total cost much less than baselines, although the cost of $Model_1$ is large in the initial few time partitions.

This set of experiments has validated the feasibility of DRLS for the scenarios without labeled data. Additionally, using DRLS can reduce the cost evidently for dynamic resource scheduling. Table 6-9 lists the total costs of Fig 6-9(b) to observe the relative improvement of $Model_1$ compared with the baseline algorithm. Overall from Table 6-9, the total cost of $Model_1$ is improved by 41.1%, 40.6%, 11.7%, 42.3%, 11.5% and 12.5% respectively compared to RS, GS, RRS, SBS, VBS and SFS for 100 time partitions with 200 subsystems in the scenarios of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

Similar to the experiments of DLS in Section 6.5.2.2, we carry out multi experiments

to test the validity of DRLS in various ranges of parameters. We respectively change the distributions of server nodes and tasks, then plot the cost per time partition of $Model_1$ and baselines in the scenarios with 200 subsystems for 10 time partitions in Fig 6-10. Comparing to Fig 6-9, experiments in Fig 6-10(a) change the range of server nodes to $n_k(t) \sim \mathrm{U}(2, 40)$; that in Fig 6-10(b) change the range of tasks to $m_k(t) \sim \mathrm{U}(n_k(t), 200)$; that in Fig 6-10(c) select $n_k(t) \sim \mathrm{U}(2, 20)$, $n_k(t) \sim \mathrm{U}(20, 40)$, $n_k(t) \sim \mathrm{U}(30, 70)$, $n_k(t) \sim \mathrm{U}(50, 90)$ with equal probability; as well as that in Fig 6-10(d) change the distribution of server nodes and tasks to triangular distributions i.e. $n_k(t) \sim \mathrm{Tr}(2, 20)$ and $m_k(t) \sim \mathrm{Tr}(n_k(t), 100)$ where the probability of $\mathrm{Tr(a, b)}$ is

$$P_{\mathrm{Tr(a,b)}}(x) = \frac{4 \min\left(|x-a|, |x-b|\right)}{(b-a)^2 - ((b-a) \mod 2)} \tag{6-18}$$

where $a \leq x \leq b \in \mathbb{N}^+$.

As shown in Fig 6-10, DRLS can reach better performance than the baseline within 5 time partitions in various distributions of server nodes and tasks. Compared to that in Fig 6-10(a) and Fig 6-10(b), DRLS (i.e. $Model_1$) in Fig 6-10(c) and Fig 6-10(d) has greater advantages than baselines, which indicates that DRLS has stronger adaptability to more complex dynamic scenes than baselines. Differentiating from DLS, DRLS retains better performance than baselines with the change of parameters. This is because DRLS is continuously trained by the experience in each time partition, which makes DRLS adaptable to the dynamic varying scenarios. In addition, DRLS based on deep reinforcement learning not only learns the decision of algorithm selection, but also learns the properties of parameter distribution of tasks and server nodes to a certain extent, which is also helpful to improve the performance of DRLS in dynamic scenes.

6.5.2.4 EX_3: Comparison between Model_1 to Model_5

Previous experiments have verified the availability and superiority of DRLS. In this set of experiments, we further consider the strategies of DRLS including pre-trained model of DLS, LTPU and joint training corresponding to Table 6-5.

We carry out experiments in the simulated HCCMS with $q = 200$ subsystems. The cost weights of subsystems are also in uniform distribution where $w_{\omega}^{(k)}(t) \sim \mathrm{U}(0, 100)/100$ and $w_{\tau}^{(k)}(t) \sim \mathrm{U}(0, 100)/100$. The numbers of the arriving tasks and the available resources in each time partition are generated by the uniform distribution. In order to test the adaptability of DRLS for dynamic algorithms selection, we experiment with two groups

of parameters generation:

- $n_k(t) \sim \mathrm{U}(2, 50)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 200)$.
- $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

The parameters generation of the first group is the same as the training dataset of DLS such as DN_3. Therefore, we also add the cost of DN_3 to participate in the comparison.

The pre-trained model being used in this set of experiments is the DN_3 in Subsection 6.5.2.2 which is trained under the parameters $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$. We observe the training process of $Model_1$ to $Model_5$ in 40 time partitions, and plot the results of the costs for each time partition in Fig 6-11. Fig 6-11(a) plots the results for the parameters generation of $n_k(t) \sim \mathrm{U}(2, 50)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 200)$, and Fig 6-11(b) for that of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$.

From Fig 6-11(a), $Model_5$, simultaneously leveraging pre-trained model, LTPU and joint training, has the fastest convergence speed and the lowest cost. Ranking from lowest to highest logarithmic cost gains $Model_5 < (Model_3, Model_2) < Model_4 < Model_1 < \mathrm{DN}_3$. $Model_2 < Model_4 < Model_1$ illustrates joint training can improve the optimization of decision-making with a small range, but not better than strategy of LTPU. $Model_3$, combining LTPU and pre-trained model, is better than $Model_2$ in the first 15 time partitions, but their results are very close after 15 partitions. This demonstrates usage of pre-trained model can improve convergence speed however cannot evidently improve optimization of decision-making for the model after sufficient training. Additionally, the DN_3 without re-training has the higher costs than all of DRLS. This demonstrates DLS can not adapt to dynamic selection of algorithms once the scenario is different from the trained dataset.

However in Fig 6-11(b), DN_3 is close to $Model_5$ and outperforms than all of DRLS. The reason is that: during training DN_3 under the parameters of $n_k(t) \sim \mathrm{U}(2, 20)$ and $m_k(t) \sim \mathrm{U}(n_k(t), 100)$, the trained labels of dataset correspond to the best algorithms, i.e., DN_3 has learned the best selection; however in training of DRLS, the best algorithms are unknown and DRLS only learned a relatively high score selection through the exploration for better selections. Thus, the choose of the algorithms selectors depends on the scenarios.

In general, each strategy of pre-trained model, LTPU and joint training can improve the training or decision-making performance of the DRLS. And DRLS is more appropriate to resolve the dynamic selection in the real-time scheduling process when the parameters are unknown in advance. The $Model_5$ using all strategies simultaneously can achieve the best performance in the process of algorithm selection.

6.5.3 Overall Summary

Through the multiple sets of experiments from different sights in this section, we can observe both DLS and DRLS outperform than baseline strategies in various resource scheduling scenarios of HCCMS. Among these experiments:

- EX_0 demonstrates the multi-layer system structure of HCCMS can greatly reduce the computational complexity of resource scheduling with little loss of optimization.
- EX_1 demonstrates DLS can reduce the whole cost significantly compared with baselines in the scenarios with stable parameter ranges where DN_3 reduces the cost by 47.4%, 46.1%, 33.9%, 47.9%, 19.3% and 18.8%, respectively compared to RS, GS, RRS, SBS, VBS and SFS. Additionally, EX_1 also demonstrates the performance of DLS will decline with the ranges of server nodes far away from that trained by DLS, as well as DLS with more layers or neurons may have stronger adaptability in algorithm selection for varying ranges of parameters.
- EX_2 demonstrates DRLS can obtain far better cost than baselines in the dynamic scheduling scenarios without labeled data where $Model_1$ reduces the cost by 41.1%, 40.6%, 11.7%, 42.3%, 11.5% and 12.5% respectively compared to RS, GS, RRS, SBS, VBS and SFS. Additionally, EX_2 also demonstrates DRLS retains better performance than baselines with the change of parameters.
- Finally, EX_3 demonstrates the effect of different strategies of DRLS where the simultaneous usage of DL-based pre-trained model, LTPU and joint training performs the best among all the DRLSs, as well as validates DRLS have stronger adaptability to dynamic scenes than DLS.

6.6 Summary of this Chapter

In this chapter, we formulate the joint scheduling problem of HCCMS combining five types of subproblems in four types of subsystems, which always cannot be addressed by a single scheduling algorithm. Focusing on this issue, we propose the scheduling framework to select the scheduling algorithms (SFSSA) to meet the resource management of complex HCCMS. To concretize SFSSA, we proposed DLS and DRLS, which can learn algorithm selection decisions in various scenarios to tackle the challenging joint scheduling problem of HCCMS. To improve the optimality and convergence of DRLS, we further apply various strategies including pre-trained model, long experience reply and joint training. Then, we carry out four sets of experiments to evaluate the performance of DLS and

DRLS in the joint scheduling problem of reducing the cost of HCCMS.

From extensive experiments in multiple sights, we can conclude not only HCCMS structure can significantly improve the speed of resource management, but also our proposed SFSSA, with DLS and DRLS as instances both outperforming baselines, can address the joint scheduling problem in HCCMS. Concretely, DLS reduced the cost by 47.4%, 46.1%, 33.9%, 47.9%, 19.3% and 18.8% in the scenarios with stable parameter ranges; DRLS reduced the cost by 41.1%, 40.6%, 11.7%, 42.3%, 11.5% and 12.5% in the dynamic scenarios compared to RS, GS, RRS, SBS, VBS and SFS respectively. We can also conclude that DRLS has stronger adaptability to dynamic resource scheduling scenarios than DLS. Additionally, the strategies, i.e. pre-trained model, long-term experience replay & period update and joint training, are all conducive to improving the performance of DRLS.

The obvious significance of DLS and DRLS is that they not only demonstrate the great potential of DL and DRL as algorithms selectors, but also prove that *resource scheduling algorithms can also be regarded as the resources to be scheduled*. Compared with directly using DL and DRL as scheduling algorithms, DLS and DRLS (using DL and DRL as algorithm selectors) have much lower computational complexity, which also provides more possibilities for their application in realistic complex systems. It also puts forward a novel considerable solution to the challenge that *no scheduling algorithm is suitable for all scenarios*, where SFSSA with DLS and DRLS implies that *since there is no such algorithm suitable for all scenarios, we will choose the most appropriate algorithm for different scenarios*. In SFSSA, all scheduling algorithms are regarded as useful "treasures", because we could always find a certain scenario for each algorithm, making this algorithm superior to all other algorithms in some aspects. On other counts, this chapter also illustrates the potential of hierarchical management of Cloud computing systems using multiple subsystems.

In the future work, we plan to explore more structures and strategies of DLS and DRLS to adapt to more complex resource scheduling scenarios, as well as we plan to continue to explore the dynamic configuration and access control policy of resources and server nodes of HCCMS where we consider that the same server node can actually belong to different subsystems at different time.

Chapter 7 Conclusion and Prospect

7.1 Conclusion

In order to explore methods or strategies that can enhance the resource management capabilities of distributed computing systems such as cloud computing, this dissertation selected five scheduling scenarios in cloud environments for exploration and research, and multiple optimization algorithm architectures (multiple optimization algorithm series) were proposed:

(1) For the scenario of single dimensional cloud computing resource scheduling, this dissertation proposes a series of multi-path local search algorithms using heuristic algorithms as search paths. Through theoretical deduction, it has been proven that the theoretical approximation ratio of the proposed search algorithm in the classic NP hard problem $P||C_{max}$ is $5/4$ (currently, there are few other public reports on the theoretical proof of the approximation ratio of search algorithms), which is superior to the existing baseline algorithm LPT's approximation ratio of $4/3$. The experiment verified its superiority in both homogeneous and heterogeneous scenarios.

(2) For the problem of heterogeneous multi-dimensional cloud computing resource scheduling, this dissertation introduces the concept of stages to reconstruct the traditional genetic algorithm architecture, and proposes a series of scalable genetic algorithms (a new genetic algorithm architecture, first seen in public materials) to improve the optimization performance and convergence speed of optimization algorithms in solving NP hard problems considering multi-dimensional resource scheduling. Experiments have shown that the performance of the growth genetic algorithm is superior to that of the baseline evolutionary algorithm.

(3) For the parallel training workflow of deep learning models in cloud distributed systems, this dissertation derives the analytical expression of a theoretical loss model that considers both computation and communication time, and takes into account the nonlinear relationship between these times and data volume, with pipeline model partitioning and micro batch data partitioning schemes as independent variables (filling the gap in the theoretical loss model under this scenario). The joint optimization problem is theoretically modeled, and a multidimensional improved binary method and a cross search algorithm based on the improved binary method are proposed. Through theoretical deduction, it has

been proven that the theoretical approximation ratio of the proposed multidimensional improved binary method can approach 1 and the computational complexity only increases linearly (there are currently no other public reports on the theoretical proof of the approximation ratio of the multidimensional improved binary method). The experiment verified the optimization of the parallel training scheme using the cross search algorithm based on the improved binary method.

(4) For parallel training of deep learning models in cloud distributed systems, this dissertation further proposes a novel architecture: UMPIPE, a pipeline parallel mode based on non-uniform data partitioning, which considers unequal data partitioning in parallel training of deep learning models. The general recursive expression of its theoretical loss model and the matrix based recursive expression are derived. Theoretical evidence has demonstrated the optimization capability of this architecture. Propose a two-level improved dual chromosome genetic algorithm based on matrix form recursion to quickly solve the optimization training scheme of UMPIPE. The experiment verified the optimization of the new parallel architecture and the optimization algorithm for solving the training scheme.

(5) Based on the practical application scenarios of cloud systems, this dissertation introduces a multi-level and multi subsystem cloud architecture, and models the dynamic management problem of complex cloud resources as a joint optimization problem of multiple sub problems. Combining algorithm pools containing multiple algorithms (first seen in publicly available materials), a cloud resource scheduling algorithm selection architecture and a reinforcement learning based algorithm selector were proposed to dynamically optimize cloud system resource management. The experiment shows that the proposed algorithm selector can improve the optimization of multi-level and multi subsystem cloud scheduling.

This dissertation explores the adaptability of cloud systems and algorithm systems by expanding their application breadth and theoretical depth. Among them:

(1) **Existence of Changes in Dimensions：** The research progress from single dimension (Chapter 2) to multiple dimensions (Chapters 3, 4, 5, and 6) has improved the adaptability of cloud systems and algorithm systems to changes in resource dimensions;

(2) **Existence of Changes in Relevance：** The research progress from independent task sets (Chapters 2 and 3) to associated workflow task sets (Chapters 4 and 5) has improved the adaptability of cloud systems and algorithm systems to changes in task corre-

lation;

(3) **Existence of Changes in Granularity**：The research progress from the equal data partitioning workflow (Chapter 4) to the unequal data partitioning workflow (Chapter 5) has improved the adaptability of cloud systems and algorithm systems to scheduling granularity changes;

(4) **Existence of Changes in the numbers of Optimization Objectives and Problems**：The research progress from the equal data partitioning workflow (Chapter 4) to the unequal data partitioning workflow (Chapter 5) has improved the adaptability of cloud systems and algorithm systems to scheduling granularity changes;

(5) **Existence of Changes in the Dynamism of Scheduling**：The research progress from static allocation (Chapter 2) to dynamic scheduling (Chapters 3, 4, 5, and 6) has improved the adaptability of cloud systems and algorithm systems to dynamic scheduling changes;

(6) **Existence of Changes in System Hierarchy and Architecture**：The research progress from single center and single level (Chapters 2, 3, 4, and 5) to multi center, multi level, and multi subsystem (Chapter 6) has improved the joint utility of cloud architecture and algorithm system for system hierarchy and scheduling scenarios;

(7) **Existence of Changes in Scheduling Scenarios and Types of Algorithm Series**：The research involves multiple resource scheduling scenarios with different configuration forms. The proposed optimization algorithm series involves multiple types of algorithms, including heuristic, local search (multi-path local search series), metaheuristic global search (expandable genetic algorithm series, multi chromosome genetic algorithm series), multi algorithm hybrid cross search, machine learning (deep learning selector and deep reinforcement learning selector), etc.

This dissertation focuses on solving optimization problems in various resource scheduling scenarios in cloud environments. Its research results include architecture design and theoretical exploration of multiple optimization algorithm series, which have theoretical and practical significance.

7.2 Prospect

Regarding the five algorithm series proposed in the five scheduling scenarios studied in this dissertation, from the perspectives of theoretical depth and application exploration, future work includes the following possibilities:

(1) This dissertation proves the theoretical approximation ratio of the search algorithm, and the approximation ratio of the multi-path search algorithm proposed in this dissertation is less than 5/4, which is better than the existing 4/3 baseline under the minimum completion time ($P||C_{max}$) problem of isomorphic parallel machines. Future work considers decomposing it into the form of control variables and attempting to derive approximate theoretical expressions or intervals for heterogeneous node univariate systems. In addition, other algorithms can also serve as the basis for multi-path search algorithms to adapt to different optimization scheduling problems and derive their corresponding theoretical approximations.

(2) The infrastructure of the growable genetic algorithm proposed in this dissertation is also applicable to other evolutionary algorithms and swarm algorithms. The future work plan combines other algorithms such as particle swarm optimization, ant colony optimization, etc. to explore high-performance algorithms suitable for many objective problems. The theoretical derivation of convergence speed or optimization performance based on the combination of growth path and genetic algorithm is also worth exploring. In addition, various population strategies and partial growth strategies can be used to further improve the convergence speed of growable genetic algorithms, and future work will also conduct in-depth research on them

(3) For the joint optimization problem of parallel training workflow for deep learning models, future work plans to further introduce tensor model parallel mode to derive theoretical loss models and joint optimization problems under the combination of three parallel modes (pipeline model parallel, data parallel, tensor model parallel), and consider further attempts to improve the cross search algorithm architecture.

(4) For the training architecture of non-uniform partitioning in the parallel training workflow problem of deep learning models in cloud systems, future work considers introducing three types of non-uniform partitioning simultaneously (inter layer inequality, intra layer inequality, and inter Minibatch inequality), and theoretically deriving and experimentally testing their loss models. In addition, the dual chromosome genetic algorithm proposed in this dissertation can be combined with local search algorithms to form a scalable dual chromosome genetic algorithm for this new training architecture, which can attempt to further improve the optimization of convergence solutions.

(5) This dissertation considers the dynamic joint scheduling problem of algorithm selectors in cloud systems containing multiple layers and subsystems. Future work con-

siders introducing micro partitioning of tasks and dynamic attribution of service nodes to improve the granularity of resource scheduling and further explore new architectures for cloud distributed systems and other distributed systems. In addition, there may be multi-level subsystems in a multi-level and multi subsystem architecture, and their optimization scheduling and permission management issues also need to be further studied.

On the basis of the research in this dissertation, the basic strategies or algorithms in the algorithm series architecture can also be adjusted to adapt to other application scenarios or research fields. The algorithm series or architectures proposed in this dissertation can also be applied to improve other algorithms. The scheduling of cloud computing resources is related to bandwidth, task delay ratio, the ratio of computing and communication, and the hardware and software used to run programs. How to apply, deploy or build the research content of this dissertation in real large-scale distributed systems, such as comprehensively considering software program behavior and characteristics, and how to balance and optimize the computational, communication and latency of resources in cloud platforms, is also an important future work.

Acknowledgements

At this point, in addition to bidding farewell to my graduate studies, I would like to take this opportunity to express my gratitude. The last time I wrote my graduation thesis was in 2016 when I wrote my master's thesis. In the past eight years, I have walked from the crabapple flowers in Beiyang Garden to the ginkgo path of Chengdian, from the magnificent ships and oceans to the high-performance computing that casts light from the clouds. Along the way, tossing and turning, looking around, asking questions, feeling lost, and stumbling. I dare not falsely claim to have experienced great storms, but I have come to realize even more today that good intentions are worth remembering seriously.

Sincerely thank my supervisor, Teacher Tian Wenhong. Thank you for accepting me as a doctoral student in your research team, allowing me to learn and grow in a united and supportive atmosphere; Thank you for providing me with academic guidance and cultivation, carefully guiding my research and paper writing work; Thank you for trusting me and giving me the opportunity to participate in scientific research projects; Thank you for helping me overcome the setbacks and failures during the research process; Thank you for creating the opportunity for me to seek advice and learn from other teachers. Teacher Tian not only teaches students professional knowledge and research methods, but also pays attention to our physical and mental health, tolerates our shortcomings, and carefully teaches us the principles of establishing a world.

Sincerely thank my co-supervisor, Professor Rajkumar Buyya; Thank you for your guidance on my research methods and paper writing; Thank you for pointing out and encouraging my research direction and ideas. Thank you to Professor Wu Kui for providing me with a lot of guidance and help. Thank you for patiently and meticulously guiding my research and paper. Thank you to Teacher Liu Yongguo for preparing my recommendation letter for pursuing a PhD. Thank you for your continuous care and assistance. Thank you, Senior Brother Xu Minxian, for your patient guidance and guidance. Thank you for your careful planning and assistance. Thank you to Teacher Liu Qiao for giving me the opportunity to participate in the teaching assistant work of the course "Statistical Machine Learning". Thank you to the teachers Yao Yuanzhe, Rao Yunbo, Guo Jiandong and others in the team for their guidance. Thank you to teachers such as Chen Anlong, Chen Dajiang, Li Xiaoyu, Liu Qihe, Lei Hang, She Kun, Zhang Fengli for their guidance Thank you to

the teachers in the college, including Professor Zhong, Professor Zhou, Professor Zhang, and Professor Chen, for their help. Thank you for the companionship and assistance of fellow disciples such as Ma Tingsong, Xie Yuanlun, Wang Ji'an, Lan Haocheng, Ou Jie, and Wang Zhaokun. Thank you to friends such as Zhang Yun, Fu Yiqin, Qu Xiaoqi, Deng Rui, Zhu Lei, and Luo Hao for their help and support. Thank you to my master's supervisor, Mr. Tang Yougang, for your concern and concern. Thank you to professors such as Yu Jianxing, Guo Zhenbang, and Lin Weixue.

Thank you to the experts who reviewed this article, thank you for ensuring the end of my academic career, and thank you for planning and guiding my future work. Thank you to the predecessors in the relevant disciplines; It is through your exploration in the long river of time that I can find my way forward today. Thank you for the cultivation of our country, thank you for the training of the University of Electronic Science and Technology of China, and thank you for the training of the School of Information and Software Engineering. Thank you to every mountain that has come, and to the fallen moon and withered snow below; Thank you for the sea when you were young, and the white leaf flying butterflies by the seaside; Thanks to the later cities and the small alleys and green steps in the city.

References

[1] Mei J, Li K, Tong Z, et al. Profit maximization for cloud brokers in cloud computing[J]. IEEE Trans. Parallel Distributed Syst., 2019, 30(1): 190-203.

[2] Guo W, Tian W, Ye Y, et al. Cloud resource scheduling with deep reinforcement learning and imitation learning[J]. IEEE Internet Things J., 2021, 8(5): 3576-3586.

[3] Cong P, Xu G, Wei T, et al. A survey of profit optimization techniques for cloud providers[J]. ACM Comput. Surv., 2020, 53(2): 26:1-26:35.

[4] Adhikari M, Amgoth T, Srirama S. N. A survey on scheduling strategies for workflows in cloud environment and emerging trends[J]. ACM Comput. Surv., 2019, 52(4): 68:1-68:36.

[5] Zhan Z, Liu X. F, Gong Y, et al. Cloud computing resource scheduling and a survey of its evolutionary approaches[J]. ACM Comput. Surv., 2015, 47(4): 63:1-63:33.

[6] Guo S, Liu J, Yang Y, et al. Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing[J]. IEEE Trans. Mob. Comput., 2019, 18(2): 319-333.

[7] Rjoub G, Bentahar J, Wahab O. A. Bigtrustscheduling: Trust-aware big data task scheduling approach in cloud computing environments[J]. Future Gener. Comput. Syst., 2020, 110: 1079-1097.

[8] Arunarani A. R, Manjula D, Sugumaran V. Task scheduling techniques in cloud computing: A literature survey[J]. Future Gener. Comput. Syst., 2019, 91: 407-415.

[9] Laili Y, Lin S, Tang D. Multi-phase integrated scheduling of hybrid tasks in cloud manufacturing environment[J]. Robotics and Computer-Integrated Manufacturing, 2020, 61: 101850.

[10] Kumar M, Sharma S. C, Goel A, et al. A comprehensive survey for scheduling techniques in cloud computing[J]. J. Netw. Comput. Appl., 2019, 143: 1-33.

[11] Caron E, Desprez F, Loureiro D, et al. Cloud computing resource management through a grid middleware: A case study with DIET and eucalyptus[C]. IEEE International Conference on Cloud Computing, CLOUD 2009, 21-25 September, 2009, Bangalore, India, 151-154.

[12] Tuli S, Ilager S, Ramamohanarao K, et al. Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks[J]. IEEE Transactions on Mobile Computing, 2020, PP(99): 1-1.

[13] Fiandrino C, Kliazovich D, Bouvry P, et al. Performance and energy efficiency metrics for communication systems of cloud computing data centers[J]. IEEE Trans. Cloud Comput., 2017, 5(4): 738-750.

[14] Zhan Z, Liu X. F, Gong Y, et al. Cloud computing resource scheduling and a survey of its evolutionary approaches[J]. ACM Comput. Surv., 2015, 47(4): 63:1-63:33.

[15] Tian W, He M, Guo W, et al. On minimizing total energy consumption in the scheduling of virtual machine reservations[J]. J. Netw. Comput. Appl., 2018, 113: 64-74.

[16] Foster I, Zhao Y, Raicu I, et al. Cloud computing and grid computing 360-degree compared[C]. 2008 grid computing environments workshop, 1-10.

[17] Merwe Jvan der , Dawoud D. S, McDonald S. A survey on peer-to-peer key management for mobile ad hoc networks[J]. ACM Comput. Surv., 2007, 39(1): 1.

[18] Fu Y, Chase J. S, Chun B. N, et al. SHARP: an architecture for secure resource peering[C]. Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003, 133-148.

[19] Miao Y, Wu G, Li M, et al. Intelligent task prediction and computation offloading based on mobile-edge cloud computing[J]. Future Gener. Comput. Syst., 2020, 102: 925-931.

[20] Liu J, Xiong K, Ng D. W. K, et al. Max-min energy balance in wireless-powered hierarchical fog-cloud computing networks[J]. IEEE Trans. Wirel. Commun., 2020, 19(11): 7064-7080.

[21] Cappanera P, Gangi L. D, Lapucci M, et al. Integrated task scheduling and personnel rostering of airports ground staff: A case study[J]. Expert Syst. Appl., 2024, 238(Part C): 121953.

[22] Valencia A, Malikopoulos A. A. On safety of passengers entering a bus rapid transit system from scheduled stops[C]. IEEE Conference on Control Technology and Applications, CCTA 2023, Bridgetown, Barbados, August 16-18, 2023, 620-625.

[23] Fox A, Griffith R, Joseph A, et al. Above the clouds: A berkeley view of cloud computing[J]. Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, 2009, 28(13): 2009.

[24] Klems M, Nimis J, Tai S. Do clouds compute? A framework for estimating the value of cloud computing[C]. Designing E-Business Systems. Markets, Services, and Networks - 7th Workshop on E-Business, WEB 2008, Paris, France, December 13, 2008, Revised Selected Papers, 110-123.

[25] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing[J]. Commun. ACM, 2010, 53(4): 50-58.

[26] Buyya R, Yeo C. S, Venugopal S, et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility[J]. Future Gener. Comput. Syst., 2009, 25(6): 599-616.

[27] Monge D. A, Pacini E, Mateos C, et al. CMI: an online multi-objective genetic autoscaler for scientific and engineering workflows in cloud infrastructures with unreliable virtual machines[J]. J. Netw. Comput. Appl., 2020, 149.

[28] Ragmani A, Elomri A, Abghour N, et al. FACO: a hybrid fuzzy ant colony optimization algorithm for virtual machine scheduling in high-performance cloud computing[J]. J. Ambient Intell. Humaniz. Comput., 2020, 11(10): 3975-3987.

[29] Sakila V. S, Manohar S. Real-time air quality monitoring in bull trench kiln-based brick industry by calibrating sensor readings and utilizing the serverless computing[J]. Expert Syst. Appl., 2024, 237(Part B): 121397.

[30] Soldani J, Brogi A. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey[J]. ACM Comput. Surv., 2023, 55(3): 59:1-59:39.

[31] Cui L, Qu Z, Zhang G, et al. A bidirectional DNN partition mechanism for efficient pipeline parallel training in cloud[J]. J. Cloud Comput., 2023, 12(1): 22.

[32] Kennedy J, Sharma V, Varghese B, et al. Multi-tier GPU virtualization for deep learning in cloud-edge systems[J]. IEEE Trans. Parallel Distributed Syst., 2023, 34(7): 2107-2123.

[33] Ghalami L, Grosu D. Scheduling parallel identical machines to minimize makespan: A parallel approximation algorithm[J]. J. Parallel Distributed Comput., 2019, 133: 221-231.

[34] Xu M, Cui L, Wang H, et al. A multiple qos constrained scheduling strategy of multiple workflows for cloud computing[C]. IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2009, 10-12 August 2009, Chengdu, Sichuan, China, 629-634.

[35] Praveenchandar J, Tamilarasi A. Dynamic resource allocation with optimized task scheduling and improved power management in cloud computing[J]. Journal of Ambient Intelligence and Humanized Computing, 2020, 1-13.

[36] Gokuldhev M, Singaravel G, Mohan N. R. R. Multi-objective local pollination-based gray wolf optimizer for task scheduling heterogeneous cloud environment[J]. J. Circuits Syst. Comput., 2020, 29(7): 2050100:1-2050100:24.

[37] Mishra S. K, Manjula R. A meta-heuristic based multi objective optimization for load distribution in cloud data center under varying workloads[J]. Clust. Comput., 2020, 23(4): 3079-3093.

[38] Sardaraz M, Tahir M. A parallel multi-objective genetic algorithm for scheduling scientific workflows in cloud computing[J]. International Journal of Distributed Sensor Networks, 2020, 16(8): 1550147720949142.

[39] Natesan G, Chokkalingam A. Multi-objective task scheduling using hybrid whale genetic optimization algorithm in heterogeneous computing environment[J]. Wirel. Pers. Commun., 2020, 110(4): 1887-1913.

[40] Dong T, Xue F, Xiao C, et al. Task scheduling based on deep reinforcement learning in a cloud manufacturing environment[J]. Concurr. Comput. Pract. Exp., 2020, 32(11).

[41] Pandiyan S, Lawrence T. S, Sathiyamoorthi V, et al. A performance-aware dynamic scheduling algorithm for cloud-based iot applications[J]. Comput. Commun., 2020, 160: 512-520.

[42] Belgacem A, Bey K. B, Nacer H, et al. Efficient dynamic resource allocation method for cloud computing environment[J]. Clust. Comput., 2020, 23(4): 2871-2889.

[43] Zhang L, Zhou L, Salah A. Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments[J]. Inf. Sci., 2020, 531: 31-46.

[44] Haytamy S. S, Omara F. A. A deep learning based framework for optimizing cloud consumer qos-based service composition[J]. Computing, 2020, 102(5): 1117-1137.

[45] Ghasemi A, Haghighat A. T. A multi-objective load balancing algorithm for virtual machine placement in cloud data centers based on machine learning[J]. Computing, 2020, 102(9): 2049-2072.

[46] Adhikari M, Amgoth T, Srirama S. N. Multi-objective scheduling strategy for scientific workflows in cloud environment: A firefly-based approach[J]. Appl. Soft Comput., 2020, 93: 106411.

[47] Li C, Bai J, Chen Y, et al. Resource and replica management strategy for optimizing financial cost and user experience in edge cloud computing system[J]. Inf. Sci., 2020, 516: 33-55.

[48] Lu H, Gu C, Luo F, et al. Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning[J]. Future Gener. Comput. Syst., 2020, 102: 847-861.

[49] Ding D, Fan X, Zhao Y, et al. Q-learning based dynamic task scheduling for energy-efficient cloud computing[J]. Future Gener. Comput. Syst., 2020, 108: 361-371.

[50] Gabi D, Ismail A. S, Zainal A, et al. Cloud customers service selection scheme based on improved conventional cat swarm optimization[J]. Neural Comput. Appl., 2020, 32(18): 14817-14838.

[51] Priya V, Kumar C. S, Kannan R. Resource scheduling algorithm with load balancing for cloud service provisioning[J]. Appl. Soft Comput., 2019, 76: 416-424.

[52] Wang W, Liang B, Li B. Multi-resource fair allocation in heterogeneous cloud computing systems[J]. IEEE Trans. Parallel Distributed Syst., 2015, 26(10): 2822-2835.

[53] Nouri S. M. R, Li H, Venugopal S, et al. Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications[J]. Future Gener. Comput. Syst., 2019, 94: 765-780.

[54] Zhang X, Jia M, Gu X, et al. An energy efficient resource allocation scheme based on cloud-computing in H-CRAN[J]. IEEE Internet Things J., 2019, 6(3): 4968-4976.

[55] Devaraj A. F. S, Elhoseny M, Dhanasekaran S, et al. Hybridization of firefly and improved multi-objective particle swarm optimization algorithm for energy efficient load balancing in cloud computing environments[J]. J. Parallel Distributed Comput., 2020, 142: 36-45.

[56] Tian W, Li G, Yang W, et al. Hscheduler: an optimal approach to minimize the makespan of multiple mapreduce jobs[J]. J. Supercomput., 2016, 72(6): 2376-2393.

[57] Tian W, Xiong Q, Cao J. An online parallel scheduling method with application to energy-efficiency in cloud computing[J]. J. Supercomput., 2013, 66(3): 1773-1790.

[58] Liu X. F, Zhan Z, Deng J. D, et al. An energy efficient ant colony system for virtual machine placement in cloud computing[J]. IEEE Trans. Evol. Comput., 2018, 22(1): 113-128.

[59] Guan Z, Melodia T. The value of cooperation: Minimizing user costs in multi-broker mobile cloud computing networks[J]. IEEE Trans. Cloud Comput., 2017, 5(4): 780-791.

[60] Lin W, Wu W, He L. An on-line virtual machine consolidation strategy for dual improvement in performance and energy conservation of server clusters in cloud data centers[J]. IEEE Trans. Serv. Comput., 2022, 15(2): 766-777.

[61] Hong Z, Chen W, Huang H, et al. Multi-hop cooperative computation offloading for industrial iot-edge-cloud computing environments[J]. IEEE Trans. Parallel Distributed Syst., 2019, 30(12): 2759-2774.

[62] Lin W, Wang W, Wu W, et al. A heuristic task scheduling algorithm based on server power efficiency model in cloud environments[J]. Sustain. Comput. Informatics Syst., 2018, 20: 56-65.

[63] A S. C, Sudhakar C, Ramesh T. Energy efficient VM scheduling and routing in multi-tenant cloud data center[J]. Sustain. Comput. Informatics Syst., 2019, 22: 139-151.

[64] Abualigah L, Diabat A. A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments[J]. Cluster Computing, 2020, 1-19.

[65] Deb K, Agrawal S, Pratap A, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II[J]. IEEE Trans. Evol. Comput., 2002, 6(2): 182-197.

[66] Liu Q, Cai W, Shen J, et al. A speculative approach to spatial-temporal efficiency with multi-objective optimization in a heterogeneous cloud environment[J]. Secur. Commun. Networks, 2016, 9(17): 4002-4012.

[67] Seada H, Deb K. U-NSGA-III: A unified evolutionary optimization procedure for single, multiple, and many objectives: Proof-of-principle results[C]. Evolutionary Multi-Criterion Optimization - 8th International Conference, EMO 2015, Guimarães, Portugal, March 29 -April 1, 2015. Proceedings, Part II, 34-49.

[68] Miriam A. J, Saminathan R, Chakaravarthi S. Non-dominated sorting genetic algorithm (NSGA-III) for effective resource allocation in cloud[J]. Evol. Intell., 2021, 14(2): 759-765.

[69] Xu X, Liu Q, Luo Y, et al. A computation offloading method over big data for iot-enabled cloud-edge computing[J]. Future Gener. Comput. Syst., 2019, 95: 522-533.

[70] Jiang H, Yi J, Chen S, et al. A multi-objective algorithm for task scheduling and resource allocation in cloud-based disassembly[J]. Journal of Manufacturing Systems, 2016, 41: 239-255.

[71] Li H, Zhu G, Zhao Y, et al. Energy-efficient and qos-aware model based resource consolidation in cloud data centers[J]. Clust. Comput., 2017, 20(3): 2793-2803.

[72] Jena R. Multi objective task scheduling in cloud environment using nested pso framework[J]. Procedia Computer Science, 2015, 57: 1219-1227.

[73] Midya S, Roy A, Majumder K, et al. Multi-objective optimization technique for resource allocation and task scheduling in vehicular cloud architecture: A hybrid adaptive nature inspired approach[J]. J. Netw. Comput. Appl., 2018, 103: 58-84.

[74] Reddy G. N, Kumar S. P. Multi objective task scheduling algorithm for cloud computing using whale optimization technique[C]. International Conference on Next Generation Computing Technologies, 286-297.

[75] Sanaj M, Prathap P. J. Nature inspired chaotic squirrel search algorithm (cssa) for multi objective task scheduling in an iaas cloud computing atmosphere[J]. Engineering Science and Technology, an International Journal, 2020, 23(4): 891-902.

[76] Ramezani F, Lu J, Taheri J, et al. Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments[J]. World Wide Web, 2015, 18(6): 1737-1757.

[77] Khan T, Tian W, Zhou G, et al. Machine learning (ml)-centric resource management in cloud computing: A review and future directions[J]. J. Netw. Comput. Appl., 2022, 204: 103405.

[78] Rodrigues T. K, Suto K, Nishiyama H, et al. Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective[J]. IEEE Commun. Surv. Tutorials, 2020, 22(1): 38-67.

[79] Chen X, Zhu F, Chen Z, et al. Resource allocation for cloud-based software services using prediction-enabled feedback control with reinforcement learning[J]. IEEE Trans. Cloud Comput., 2022, 10(2): 1117-1129.

[80] Chen G, Qi J, Sun Y, et al. A collaborative scheduling method for cloud computing heterogeneous workflows based on deep reinforcement learning[J]. Future Gener. Comput. Syst., 2023, 141: 284-297.

[81] Wang X, Zhang L, Liu Y, et al. Logistics-involved task scheduling in cloud manufacturing with offline deep reinforcement learning[J]. J. Ind. Inf. Integr., 2023, 34: 100471.

[82] Zhou G, Tian W, Buyya R. Multi-search-routes-based methods for minimizing makespan of homogeneous and heterogeneous resources in cloud computing[J]. Future Gener. Comput. Syst., 2023, 141: 414-432.

[83] M M, T J. Combined particle swarm optimization and ant colony system for energy efficient cloud data centers[J]. Concurr. Comput. Pract. Exp., 2021, 33(10).

[84] Chaudhary D, Kumar B. Cost optimized hybrid genetic-gravitational search algorithm for load scheduling in cloud computing[J]. Appl. Soft Comput., 2019, 83.

[85] Mansouri N, Zade B. M. H, Javidi M. M. Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory[J]. Comput. Ind. Eng., 2019, 130: 597-633.

[86] Kayalvili S, Selvam M. Hybrid SFLA-GA algorithm for an optimal resource allocation in cloud[J]. Clust. Comput., 2019, 22(Supplement): 3165-3173.

[87] Zhou G, Tian W, Buyya R, et al. Growable genetic algorithm with heuristic-based local search for multi-dimensional resources scheduling of cloud computing[J]. Appl. Soft Comput., 2023, 136: 110027.

[88] Chen X, Zhang J, Lin B, et al. Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments[J]. IEEE Trans. Parallel Distributed Syst., 2022, 33(3): 683-697.

[89] Chen Z, Zheng H, Zhang J, et al. Joint computation offloading and deployment optimization in multi-uav-enabled MEC systems[J]. Peer-to-Peer Netw. Appl., 2022, 15(1): 194-205.

[90] Muthulakshmi B, Somasundaram K. A hybrid ABC-SA based optimized scheduling and resource allocation for cloud environment[J]. Clust. Comput., 2019, 22(5): 10769-10777.

[91] Ibrahim G. J, Rashid T. A, Akinsolu M. O. An energy efficient service composition mechanism using a hybrid meta-heuristic algorithm in a mobile cloud environment[J]. J. Parallel Distributed Comput., 2020, 143: 77-87.

[92] Alla H. B, Alla S. B, Touhafi A, et al. A novel task scheduling approach based on dynamic queues and hybrid meta-heuristic algorithms for cloud computing environment[J]. Clust. Comput., 2018, 21(4): 1797-1820.

[93] Helft M. Google confirms problems with reaching its services[J]. The New York Times, 2009.

[94] Markoff J. Software via the internet: Microsoft in cloud computing[J]. New York Times, 2007, 3.

[95] Chase J, Niyato D. Joint optimization of resource provisioning in cloud computing[J]. IEEE Trans. Serv. Comput., 2017, 10(3): 396-409.

[96] Yang R, Zhang Y, Garraghan P, et al. Reliable computing service in massive-scale systems through rapid low-cost failover[J]. IEEE Trans. Serv. Comput., 2017, 10(6): 969-983.

[97] Welsh T, Benkhelifa E. On resilience in cloud computing: A survey of techniques across the cloud domain[J]. ACM Comput. Surv., 2020, 53(3): 59:1-59:36.

[98] Laili Y, Tao F, Wang F, et al. An iterative budget algorithm for dynamic virtual machine consolidation under cloud computing environment[J]. IEEE Trans. Serv. Comput., 2021, 14(1): 30-43.

[99] Sofia A. S, Ganeshkumar P. Multi-objective task scheduling to minimize energy consumption and makespan of cloud computing using NSGA-II[J]. J. Netw. Syst. Manag., 2018, 26(2): 463-485.

[100] Li M, Yu F. R, Si P, et al. Resource optimization for delay-tolerant data in blockchain-enabled iot with edge computing: A deep reinforcement learning approach[J]. IEEE Internet Things J., 2020, 7(10): 9399-9412.

[101] Mao J, Pan Q, Miao Z, et al. An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance[J]. Expert Syst. Appl., 2021, 169: 114495.

[102] Kim Y. J, Jang J. W, Kim D. S, et al. Batch loading and scheduling problem with processing time deterioration and rate-modifying activities[J]. International Journal of Production Research, 2021, 1-21.

[103] Croce F. D, Scatamacchia R. The longest processing time rule for identical parallel machines revisited[J]. J. Sched., 2020, 23(2): 163-176.

[104] Bitsakos C, Konstantinou I, Koziris N. DERP: A deep reinforcement learning cloud system for elastic resource provisioning[C]. 2018 IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2018, December 10-13, 2018, Nicosia, Cyprus, 21-29.

[105] Xu C, Rao J, Bu X. URL: A unified reinforcement learning approach for autonomic cloud management[J]. J. Parallel Distributed Comput., 2012, 72(2): 95-105.

[106] Lolos K, Konstantinou I, Kantere V, et al. Elastic management of cloud applications using adaptive reinforcement learning[C]. 2017 IEEE International Conference on Big Data, BigData 2017, December 11-14, 2017, Boston, MA, USA, 203-212.

[107] Feng J, Yu F. R, Pei Q, et al. Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach[J]. IEEE Internet Things J., 2020, 7(7): 6214-6228.

[108] Karthiban K, Raj J. S. An efficient green computing fair resource allocation in cloud computing using modified deep reinforcement learning algorithm[J]. Soft Comput., 2020, 24(19): 14933-14942.

[109] Liu N, Li Z, Xu J, et al. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning[C]. 37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, June 5-8, 2017, Atlanta, GA, USA, 372-382.

[110] Tong Z, Chen H, Deng X, et al. A scheduling scheme in the cloud computing environment using deep *Q*-learning[J]. Inf. Sci., 2020, 512: 1170-1191.

[111] Li C, Zhang Y, Luo Y. Neighborhood search-based job scheduling for iot big data real-time processing in distributed edge-cloud computing environment[J]. J. Supercomput., 2021, 77(2): 1853-1878.

[112] Luo C, Qiao B, Xing W, et al. Correlation-aware heuristic search for intelligent virtual machine provisioning in cloud systems[C]. Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, 12363-12372.

[113] Somu N, Raman M. R. G, Kaveri A, et al. IBGSS: an improved binary gravitational search algorithm based search strategy for qos and ranking prediction in cloud environments[J]. Appl. Soft Comput., 2020, 88: 105945.

[114] Kumar K. R. P, Kousalya K. Amelioration of task scheduling in cloud computing using crow search algorithm[J]. Neural Comput. Appl., 2020, 32(10): 5901-5907.

[115] Chen C, Hung L, Hsieh S, et al. Heterogeneous job allocation scheduler for hadoop mapreduce using dynamic grouping integrated neighboring search[J]. IEEE Trans. Cloud Comput., 2020, 8(1): 193-206.

[116] Diallo M, Quintero A, Pierre S. A tabu search approach for a virtual networks splitting strategy across multiple cloud providers[J]. Int. J. Metaheuristics, 2020, 7(3): 197-238.

[117] Zhang M, Peng Y, Yang M, et al. A discrete pso-based static load balancing algorithm for distributed simulations in a cloud environment[J]. Future Gener. Comput. Syst., 2021, 115: 497-516.

[118] Vazirani V. V. Approximation algorithms[M]. Springer, 2001.

[119] Zhang W, Wen Y. Energy-efficient task execution for application as a general topology in mobile cloud computing[J]. IEEE Trans. Cloud Comput., 2018, 6(3): 708-719.

[120] Kumar A. M. S, Venkatesan M. Multi-objective task scheduling using hybrid genetic-ant colony optimization algorithm in cloud environment[J]. Wirel. Pers. Commun., 2019, 107(4): 1835-1848.

[121] Yang Y, Yang B, Wang S, et al. A dynamic ant-colony genetic algorithm for cloud service composition optimization[J]. The International Journal of Advanced Manufacturing Technology, 2019, 102(1-4): 355-368.

[122] Ismayilov G, Topcuoglu H. R. Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing[J]. Future Gener. Comput. Syst., 2020, 102: 307-322.

[123] Tian W, Xu M, Chen Y, et al. Prepartition: A new paradigm for the load balance of virtual machine reservations in data centers[C]. IEEE International Conference on Communications, ICC 2014, Sydney, Australia, June 10-14, 2014, 4017-4022.

[124] Zhou X, Zhang G, Sun J, et al. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT[J]. Future Gener. Comput. Syst., 2019, 93: 278-289.

[125] Kardani-Moghaddam S, Buyya R, Ramamohanarao K. ADRL: A hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds[J]. IEEE Trans. Parallel Distributed Syst., 2021, 32(3): 514-526.

[126] Domanal S. G, Guddeti R. M. R, Buyya R. A hybrid bio-inspired algorithm for scheduling and resource management in cloud environment[J]. IEEE Trans. Serv. Comput., 2020, 13(1): 3-15.

[127] Bolaji A. L, Okwonu F. Z, Shola P. B, et al. A modified binary pigeon-inspired algorithm for solving the multi-dimensional knapsack problem[J]. J. Intell. Syst., 2021, 30(1): 90-103.

[128] Abdel-Basset M, Mohamed R, Sallam K. M, et al. BSMA: A novel metaheuristic algorithm for multi-dimensional knapsack problems: Method and comprehensive analysis[J]. Comput. Ind. Eng., 2021, 159: 107469.

[129] Goudarzi H, Pedram M. Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems[C]. IEEE International Conference on Cloud Computing, CLOUD 2011, Washington, DC, USA, 4-9 July, 2011, 324-331.

[130] Dhaenens C, Jourdan L. Metaheuristics for big data[M]. , 2016, .

[131] Hadary O, Marshall L, Menache I, et al. Protean: VM allocation service at scale[C]. 14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020, 845-861.

[132] Roy A, Midya S, Majumder K, et al. Distributed resource management in dew based edge to cloud computing ecosystem: A hybrid adaptive evolutionary approach[J]. Trans. Emerg. Telecommun. Technol., 2020, 31(8).

[133] Chen T, Marqués A. G, Giannakis G. B. DGLB: distributed stochastic geographical load balancing over cloud networks[J]. IEEE Trans. Parallel Distributed Syst., 2017, 28(7): 1866-1880.

[134] Li Q, Yao H, Mai T, et al. Reinforcement-learning- and belief-learning-based double auction mechanism for edge computing resource allocation[J]. IEEE Internet Things J., 2020, 7(7): 5976-5985.

[135] Li J, Han Y. A hybrid multi-objective artificial bee colony algorithm for flexible task scheduling problems in cloud computing system[J]. Clust. Comput., 2020, 23(4): 2483-2499.

[136] Xia W, Shen L. Joint resource allocation at edge cloud based on ant colony optimization and genetic algorithm[J]. Wirel. Pers. Commun., 2021, 117(2): 355-386.

[137] Xie K, Wang X, Xie G, et al. Distributed multi-dimensional pricing for efficient application offloading in mobile cloud computing[J]. IEEE Trans. Serv. Comput., 2019, 12(6): 925-940.

[138] Bao N, Chai Y, Zhang Y, et al. More space may be cheaper: Multi-dimensional resource allocation for nvm-based cloud cache[C]. 38th IEEE International Conference on Computer Design, ICCD 2020, Hartford, CT, USA, October 18-21, 2020, 565-572.

[139] Pan Y, Gao L, Luo J, et al. A multi-dimensional resource crowdsourcing framework for mobile edge computing[C]. 2020 IEEE International Conference on Communications, ICC 2020, Dublin, Ireland, June 7-11, 2020, 1-7.

[140] Yu H, Zhou Z, Jia Z, et al. Multi-timescale multi-dimension resource allocation for noma-edge computing-based power iot with massive connectivity[J]. IEEE Trans. Green Commun. Netw., 2021, 5(3): 1101-1113.

[141] Gopu A, Venkataraman N. Optimal VM placement in distributed cloud environment using MOEA/D[J]. Soft Comput., 2019, 23(21): 11277-11296.

[142] Nurcahyadi T, Blum C. Negative learning in ant colony optimization: Application to the multi dimensional knapsack problem[C]. ISMSI 2021: 2021 5th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence, Victoria, Seychelles, April 10-11, 2021, 22-27.

[143] Yu M, Wu C, Ji B, et al. A sum-of-ratios multi-dimensional-knapsack decomposition for DNN resource scheduling[C]. 40th IEEE Conference on Computer Communications, INFOCOM 2021, Vancouver, BC, Canada, May 10-13, 2021, 1-10.

[144] Aktar M. S, De M, Mazumder S. K, et al. Multi-objective green 4-dimensional transportation problems for breakable incompatible items with different fixed charge payment policies[J]. Comput. Ind. Eng., 2021, 156: 107184.

[145] Chen J, Wu H, Lyu F, et al. Multi-dimensional resource allocation for diverse safety message transmissions in vehicular networks[C]. ICC 2021 - IEEE International Conference on Communications, Montreal, QC, Canada, June 14-23, 2021, 1-6.

[146] Ehrgott M. Multicriteria optimization (2. ed.)[M]. Springer, 2005.

[147] . Multi-objective optimization - evolutionary to hybrid framework[M]. Springer, 2018.

[148] Yang J, Zhu H, Liu T. Secure and economical multi-cloud storage policy with NSGA-II-C[J]. Appl. Soft Comput., 2019, 83.

[149] Shang K, Ishibuchi H. A new hypervolume-based evolutionary algorithm for many-objective optimization[J]. IEEE Trans. Evol. Comput., 2020, 24(5): 839-852.

[150] Maree S. C, Alderliesten T, Bosman P. A. N. Uncrowded hypervolume-based multiobjective optimization with gene-pool optimal mixing[J]. Evol. Comput., 2022, 30(3): 329-353.

[151] Srinivas N, Deb K. Multiobjective optimization using nondominated sorting in genetic algorithms[J]. Evol. Comput., 1994, 2(3): 221-248.

[152] Deb K, Agrawal S, Pratap A, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II[J]. IEEE Trans. Evol. Comput., 2002, 6(2): 182-197.

[153] Zitzler E, Brockhoff D, Thiele L. The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration[C]. Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007, Matsushima, Japan, March 5-8, 2007, Proceedings, 862-876.

[154] Shang K, Ishibuchi H, He L, et al. A survey on the hypervolume indicator in evolutionary multiobjective optimization[J]. IEEE Trans. Evol. Comput., 2021, 25(1): 1-20.

[155] Liu R, Ren R, Liu J, et al. A clustering and dimensionality reduction based evolutionary algorithm for large-scale multi-objective problems[J]. Appl. Soft Comput., 2020, 89: 106120.

[156] Tan Z, Wang H, Liu S. Multi-stage dimension reduction for expensive sparse multi-objective optimization problems[J]. Neurocomputing, 2021, 440: 159-174.

[157] Brockhoff D, Zitzler E. Dimensionality reduction in multiobjective optimization: The minimum objective subset problem[C]. Operations Research, Proceedings 2006, Selected Papers of the Annual International Conference of the German Operations ResearchSociety (GOR), Jointly Organized with the Austrian Society of Operations Research (ÖGOR) and the Swiss Society of Operations Research (SVOR), Karlsruhe, Germany, September 6-8, 2006, 423-429.

[158] Zhang Q, Li H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition[J]. IEEE Trans. Evol. Comput., 2007, 11(6): 712-731.

[159] Xu H, Zeng W, Zhang D, et al. MOEA/HD: A multiobjective evolutionary algorithm based on hierarchical decomposition[J]. IEEE Trans. Cybern., 2019, 49(2): 517-526.

[160] Cao J, Zhang J, Zhao F, et al. A two-stage evolutionary strategy based MOEA/D to multi-objective problems[J]. Expert Syst. Appl., 2021, 185: 115654.

[161] Li H, Deb K, Zhang Q, et al. Comparison between MOEA/D and NSGA-III on a set of novel many and multi-objective benchmark problems with challenging difficulties[J]. Swarm Evol. Comput., 2019, 46: 104-117.

[162] Li H, Zhang Q. Multiobjective optimization problems with complicated pareto sets, MOEA/D and NSGA-II[J]. IEEE Trans. Evol. Comput., 2009, 13(2): 284-302.

[163] Shao L, Ehrgott M. Discrete representation of non-dominated sets in multi-objective linear programming[J]. Eur. J. Oper. Res., 2016, 255(3): 687-698.

[164] Liu L, Wang T. An evolvable hardware method based on elite partheno-genetic algorithm[J]. Appl. Soft Comput., 2021, 113(Part): 107904.

[165] Yang J, Hu Y, Zhang K, et al. An improved evolution algorithm using population competition genetic algorithm and self-correction BP neural network based on fitness landscape[J]. Soft Comput., 2021, 25(3): 1751-1776.

[166] Pal K. S, Wang P. P. Genetic algorithms for pattern recognition[J]. CRC Press, Inc., 1996.

[167] Blank J, Deb K. Pymoo: Multi-objective optimization in python[J]. IEEE Access, 2020, 8: 89497-89509.

[168] Wang H, Qu Z, Zhou Q, et al. A comprehensive survey on training acceleration for large machine learning models in iot[J]. IEEE Internet Things J., 2022, 9(2): 939-963.

[169] Ouyang S, Dong D, Xu Y, et al. Communication optimization strategies for distributed deep neural network training: A survey[J]. J. Parallel Distributed Comput., 2021, 149: 52-65.

[170] Wei J, Bosma M, Zhao V. Y, et al. Finetuned language models are zero-shot learners[C]. The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022, .

[171] Brown T. B, Mann B, Ryder N, et al. Language models are few-shot learners[C]. Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, .

[172] Rae J. W, Borgeaud S, Cai T, et al. Scaling language models: Methods, analysis & insights from training gopher[J]. CoRR, 2021, abs/2112.11446.

[173] Wang S, Sun Y, Xiang Y, et al. ERNIE 3.0 titan: Exploring larger-scale knowledge enhanced pre-training for language understanding and generation[J]. CoRR, 2021, abs/2112.12731.

[174] Li Z, Chang V, Hu H, et al. Optimizing makespan and resource utilization for multi-dnn training in GPU cluster[J]. Future Gener. Comput. Syst., 2021, 125: 206-220.

[175] Sun P, Wen Y, Han R, et al. Gradientflow: Optimizing network performance for large-scale distributed DNN training[J]. IEEE Trans. Big Data, 2022, 8(2): 495-507.

[176] Fu H, Tang S, He B, et al. HGP4CNN: an efficient parallelization framework for training convolutional neural networks on modern gpus[J]. J. Supercomput., 2021, 77(11): 12741-12770.

[177] Xu J, Wang J, Qi Q, et al. Effective scheduler for distributed DNN training based on mapreduce and GPU cluster[J]. J. Grid Comput., 2021, 19(1): 8.

[178] Ye X, Lai Z, Li S, et al. Hippie: A data-paralleled pipeline approach to improve memory-efficiency and scalability for large DNN training[C]. ICPP 2021: 50th International Conference on Parallel Processing, Lemont, IL, USA, August 9 - 12, 2021, 71:1-71:10.

[179] Romero J, Yin J, Laanait N, et al. Accelerating collective communication in data parallel training across deep learning frameworks[C]. 19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022, 1027-1040.

[180] Li Z, Zhuang S, Guo S, et al. Terapipe: Token-level pipeline parallelism for training large-scale language models[C]. Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event, 6543-6552.

[181] Narayanan D, Shoeybi M, Casper J, et al. Efficient large-scale language model training on GPU clusters using megatron-lm[C]. SC '21: The International Conference for High Performance Computing, Networking, Storage and Analysis, St. Louis, Missouri, USA, November 14 - 19, 2021, 58:1-58:15.

[182] Huang Y, Cheng Y, Bapna A, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism[C]. Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 103-112.

[183] Narayanan D, Harlap A, Phanishayee A, et al. Pipedream: generalized pipeline parallelism for DNN training[C]. Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019, 1-15.

[184] Fan S, Rong Y, Meng C, et al. DAPPLE: a pipelined data parallel approach for training large models[C]. PPoPP '21: 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Virtual Event, Republic of Korea, February 27- March 3, 2021, 431-445.

[185] Ye X, Lai Z, Li S, et al. Hippie: A data-paralleled pipeline approach to improve memory-efficiency and scalability for large DNN training[C]. ICPP 2021: 50th International Conference on Parallel Processing, Lemont, IL, USA, August 9 - 12, 2021, 71:1-71:10.

[186] Zeng Z, Liu C, Tang Z, et al. Training acceleration for deep neural networks: A hybrid parallelization strategy[C]. 58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021, 1165-1170.

[187] Dryden N, Maruyama N, Moon T, et al. Channel and filter parallelism for large-scale CNN training[C]. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, Denver, Colorado, USA, November 17-19, 2019, 10:1-10:20.

[188] He J, Zhai J, Antunes T, et al. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models[C]. PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022, 120-134.

[189] Jain A, Awan A. A, Aljuhani A. M, et al. GEMS: gpu-enabled memory-aware model-parallelism system for distributed DNN training[C]. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020, 45.

[190] Dong J, Cao Z, Zhang T, et al. EFLOPS: algorithm and system co-design for a high performance distributed training platform[C]. IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020, 610-622.

[191] Zheng L, Li Z, Zhang H, et al. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning[C]. 16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022, 559-578.

[192] Zeng Z, Liu C, Tang Z, et al. Training acceleration for deep neural networks: A hybrid parallelization strategy[C]. 58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021, 1165-1170.

[193] Bai Y, Li C, Zhou Q, et al. Gradient compression supercharged high-performance data parallel DNN training[C]. SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021, 359-375.

[194] Elango V. Pase: Parallelization strategies for efficient DNN training[C]. 35th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2021, Portland, OR, USA, May 17-21, 2021, 1025-1034.

[195] Zhao S, Li F, Chen X, et al. vpipe: A virtualized acceleration system for achieving efficient and scalable pipeline parallel DNN training[J]. IEEE Trans. Parallel Distributed Syst., 2022, 33(3): 489-506.

[196] Elango V. Pase: Parallelization strategies for efficient DNN training[C]. 35th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2021, Portland, OR, USA, May 17-21, 2021, 1025-1034.

[197] Han Z, Qu G, Liu B, et al. Exploit the data level parallelism and schedule dependent tasks on the multi-core processors[J]. Inf. Sci., 2022, 585: 382-394.

[198] Li Y, Zeng Z, Li J, et al. Distributed model training based on data parallelism in edge computing-enabled elastic optical networks[J]. IEEE Commun. Lett., 2021, 25(4): 1241-1244.

[199] Beaumont O, Eyraud-Dubois L, Shilova A. Madpipe: Memory aware dynamic programming algorithm for pipelined model parallelism[C]. IEEE International Parallel and Distributed Processing Symposium, IPDPS Workshops 2022, Lyon, France, May 30 - June 3, 2022, 1063-1073.

[200] Zhao S, Li F, Chen X, et al. Naspipe: high performance and reproducible pipeline parallel supernet training via causal synchronous parallelism[C]. ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022, 374-387.

[201] Narayanan D, Phanishayee A, Shi K, et al. Memory-efficient pipeline-parallel DNN training[C]. Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event, 7937-7947.

[202] Li J, Wang Y, Zhang J, et al. Pipepar: A pipelined hybrid parallel approach for accelerating distributed DNN training[C]. 24th IEEE International Conference on Computer Supported Cooperative Work in Design, CSCWD 2021, Dalian, China, May 5-7, 2021, 470-475.

[203] Zhang Z, Chen J, Hu B. The optimization of model parallelization strategies for multi-gpu training[C]. IEEE Global Communications Conference, GLOBECOM 2021, Madrid, Spain, December 7-11, 2021, 1-6.

[204] Lee Y, Chung J, Rhu M. Smartsage: training large-scale graph neural networks using in-storage processing architectures[C]. ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022, 932-945.

[205] Gu R, Chen Y, Liu S, et al. Liquid: Intelligent resource estimation and network-efficient scheduling for deep learning jobs on distributed GPU clusters[J]. IEEE Trans. Parallel Distributed Syst., 2022, 33(11): 2808-2820.

[206] Md V, Misra S, Ma G, et al. Distgnn: scalable distributed training for large-scale graph neural networks[C]. SC '21: The International Conference for High Performance Computing, Networking, Storage and Analysis, St. Louis, Missouri, USA, November 14 - 19, 2021, 76:1-76:14.

[207] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[C]. 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, .

[208] Xue F, Wang Q, Guo G. Transfer: Learning relation-aware facial expression representations with transformers[C]. Proceedings of the IEEE/CVF International Conference on Computer Vision, 3601-3610.

[209] Zini J. E, Awad M. On the explainability of natural language processing deep models[J]. ACM Comput. Surv., 2023, 55(5): 103:1-103:31.

[210] Lee Y, Chung J, Rhu M. Smartsage: training large-scale graph neural networks using in-storage processing architectures[C]. ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022, 932-945.

[211] Rao T, Li J, Wang X, et al. Facial expression recognition with multiscale graph convolutional networks[J]. IEEE Multim., 2021, 28(2): 11-19.

[212] Raffel C, Shazeer N, Roberts A, et al. Exploring the limits of transfer learning with a unified text-to-text transformer[J]. J. Mach. Learn. Res., 2020, 21: 140:1-140:67.

[213] Fang J, Zhu Z, Li S, et al. Parallel training of pre-trained models via chunk-based dynamic memory management[J]. IEEE Trans. Parallel Distributed Syst., 2023, 34(1): 304-315.

[214] Romero J, Yin J, Laanait N, et al. Accelerating collective communication in data parallel training across deep learning frameworks[C]. 19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022, 1027-1040.

[215] Lai Z, Li S, Tang X, et al. Merak: An efficient distributed DNN training framework with automated 3d parallelism for giant foundation models[J]. IEEE Trans. Parallel Distributed Syst., 2023, 34(5): 1466-1478.

[216] Wang M, Huang C, Li J. Supporting very large models using automatic dataflow graph partitioning[C]. Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, March 25-28, 2019, 26:1-26:17.

[217] Li F, Zhao S, Qing Y, et al. Fold3d: Rethinking and parallelizing computational and communicational tasks in the training of large DNN models[J]. IEEE Trans. Parallel Distributed Syst., 2023, 34(5): 1432-1449.

[218] Cui L, Qu Z, Zhang G, et al. A bidirectional DNN partition mechanism for efficient pipeline parallel training in cloud[J]. J. Cloud Comput., 2023, 12(1): 22.

[219] Guan L, Yang Z, Li D, et al. pdladmm: An admm-based framework for parallel deep learning training with efficiency[J]. Neurocomputing, 2021, 435: 264-272.

[220] Kahira A. N, Nguyen T. T, Bautista-Gomez L, et al. An oracle for guiding large-scale model/hybrid parallel training of convolutional neural networks[C]. HPDC '21: The 30th International Symposium on High-Performance Parallel and Distributed Computing, Virtual Event, Sweden, June 21-25, 2021, 161-173.

[221] Zhang J, Niu G, Dai Q, et al. Pipepar: Enabling fast DNN pipeline parallel training in heterogeneous GPU clusters[J]. Neurocomputing, 2023, 555: 126661.

[222] Zhang Z, Ji Z, Wang C. Momentum-driven adaptive synchronization model for distributed DNN training on HPC clusters[J]. J. Parallel Distributed Comput., 2022, 159: 65-84.

[223] Zheng S, Chen R, Jin Y, et al. Neoflow: A flexible framework for enabling efficient compilation for high performance DNN training[J]. IEEE Trans. Parallel Distributed Syst., 2022, 33(11): 3220-3232.

[224] Wan B, Dang J, Li Z, et al. Modeling analysis and cost-performance ratio optimization of virtual machine scheduling in cloud computing[J]. IEEE Trans. Parallel Distributed Syst., 2020, 31(7): 1518-1532.

[225] Hu H, Li Z, Hu H, et al. Multi-objective scheduling for scientific workflow in multicloud environment[J]. J. Netw. Comput. Appl., 2018, 114: 108-122.

[226] Nguyen T. T, Ha V. N, Le L. B, et al. Joint data compression and computation offloading in hierarchical fog-cloud systems[J]. IEEE Trans. Wirel. Commun., 2020, 19(1): 293-309.

[227] Li J. Resource optimization scheduling and allocation for hierarchical distributed cloud service system in smart city[J]. Future Gener. Comput. Syst., 2020, 107: 247-256.

[228] Czako Z, Sebestyen G, Hangan A. Automaticai - A hybrid approach for automatic artificial intelligence algorithm selection and hyperparameter tuning[J]. Expert Syst. Appl., 2021, 182: 115225.

[229] Huerta I. I, Neira D. A, Ortega D. A, et al. Improving the state-of-the-art in the traveling salesman problem: An anytime automatic algorithm selection[J]. Expert Syst. Appl., 2022, 187: 115948.

[230] Duc T. L, Leiva R. A. G, Casari P, et al. Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey[J]. ACM Comput. Surv., 2019, 52(5): 94:1-94:39.

[231] Li L. An optimistic differentiated service job scheduling system for cloud computing service users and providers[C]. 2009 Third International Conference on Multimedia and Ubiquitous Engineering, MUE 2009, June 4-6, 2009, Qingdao, China, 295-299.

[232] Pradhan P, Behera P. K, Ray B. Modified round robin algorithm for resource allocation in cloud computing[J]. Procedia Computer Science, 2016, 85: 878-890.

[233] Narwal A, Dhingra S. Enhanced task scheduling algorithm using multi-objective function for cloud computing framework[C]. International Conference on Next Generation Computing Technologies, 110-121.

[234] Al-Mahruqi A. A. H, Morison G, Stewart B. G, et al. Hybrid heuristic algorithm for better energy optimization and resource utilization in cloud computing[J]. Wirel. Pers. Commun., 2021, 118(1): 43-73.

[235] Iranmanesh A, Naji H. R. DCHG-TS: a deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing[J]. Clust. Comput., 2021, 24(2): 667-681.

[236] Gudu D, Hardt M, Streit A. Combinatorial auction algorithm selection for cloud resource allocation using machine learning[C]. Euro-Par 2018: Parallel Processing - 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings, 378-391.

[237] Seiler M, Pohl J, Bossek J, et al. Deep learning as a competitive feature-free approach for automated algorithm selection on the traveling salesperson problem[C]. Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I, 48-64.

[238] Deshpande N, Sharma N, Yu Q, et al. R-CASS: using algorithm selection for self-adaptive service oriented systems[C]. 2021 IEEE International Conference on Web Services, ICWS 2021, Chicago, IL, USA, September 5-10, 2021, 61-72.

[239] Boas M. G. V, Santos H. G, Campos Merschmann L. Hde , et al. Optimal decision trees for the algorithm selection problem: integer programming based approaches[J]. Int. Trans. Oper. Res., 2021, 28(5): 2759-2781.

[240] Muñoz M. A, Kirley M. Sampling effects on algorithm selection for continuous black-box optimization[J]. Algorithms, 2021, 14(1): 19.

Research Results Obtained During the Study for Doctoral Degree

[1] **Zhou G**, Wen R, Tian W, et al. Deep Reinforcement Learning-based Algorithms Selectors for the Resource Scheduling in Hierarchical Cloud Computing[J]. Journal of Network and Computer Applications, 2022, 208: 103520.

[2] **Zhou G**, Tian W, Buyya R, et al. Growable Genetic Algorithm with Heuristic-based Local Search for Multi-dimensional Resources Scheduling of Cloud Computing[J]. Applied Soft Computing, 2023, 136: 110027.

[3] **Zhou G**, Tian W, Buyya R. Multi-search-routes-based Methods for Minimizing Makespan of Homogeneous and Heterogeneous Resources in Cloud Computing[J]. Future Generation Computer Systems, 2023, 141: 414-432.

[4] **Zhou G**, Tian W, Buyya R, et al. Deep Reinforcement Learning-based Methods for Resource Scheduling in Cloud Computing: A Review and Future Directions[J]. Artificial Intelligence Review, 2024, 57(5): 124.

[5] **Zhou G**, Xie Y, Lan H, et al. Information Interaction and Partial Growth-based Multi-Population Growable Genetic Algorithm for Multi-Dimensional Resources Utilization Optimization of Cloud Computing[J]. Swarm and Evolutionary Computation, 2024, 87: 101575.

[6] **Zhou G**, Lan H, Xie Y, et al. CSIMD: Cross-Search Algorithm with Improved Multi-Dimensional Dichotomy for Micro-batch-based Pipeline Parallel Training in DNN[C]. 30th International European Conference on Parallel and Distributed Computing, Euro-Par 2024. （**Accepted**）

[7] **Zhou G**, Tian W, Buyya R. LPT-One and BFD-One Search Algorithms for Load Balance and Bin-Packing of Cloud Computing[C]. 9th IEEE International Conference on Cloud Computing and Intelligent Systems, CCIS 2023, Dali, China, August 12-13, 2023, 521-525.

[8] **Zhou G**, Tian W, Buyya R, et al. UMPIPE: Unequal Microbatches-based Pipeline Parallelism for DNN Training[J]. IEEE Transactions on Parallel and Distributed Systems. （**Under Review**）

[9] Kadhim M. R, **Zhou G**, Tian W. A Novel Self-directed Learning Framework for Cluster Ensemble[J]. Journal of King Saud University - Computer and Information Sciences, 2022, 34(10 Part A): 7841-7855.

[10] Khan T, Tian W, **Zhou G**, et al. Machine Learning (ML)-centric Resource Management in Cloud Computing: A Review and Future Directions[J]. Journal of Network and Computer Applications, 2022, 204: 103405.

[11] Tian W, Xu M, **Zhou G**, et al. Prepartition: Load Balancing Approach for Virtual Machine Reservations in a Cloud Data Center[J]. Journal of Computer Science and Technology, 2023, 38(4): 773-792.

[12] Kadhim M. R, Tian W, **Zhou G**, et al. A Novel Side-Information for Unsupervised Cluster Ensemble[C]. 18th International Computer Conference on Wavelet Active Media Technology and Information Processing, ICCWAMTIP 2021, 2021, 200-207.