QoS-based Scheduling of Workflows on Global Grids

by

Jia Yu

B.E (Beijing Union University) and M.E (Monash University)

Submitted in total fulfilment of
the requirements for the degree of

Doctor of Philosophy

Department of Computer Science and Software Engineering
The University of Melbourne, Australia

Oct 2007

QoS-based Scheduling of Workflows on Global Grids

Jia Yu

Supervisors: Assoc. Prof. Rajkumar Buyya, Prof. Rao Kotagiri

Abstract

Grid computing has emerged as a global cyber-infrastructure for the next-generation of e-Science applications by integrating large-scale, distributed and heterogeneous resources. Scientific communities are utilizing Grids to share, manage and process large data sets. In order to support complex scientific experiments, distributed resources such as computational devices, data, applications, and scientific instruments need to be orchestrated while managing the application workflow operations within Grid environments. This thesis investigates properties of Grid workflow management systems, presents a workflow engine and algorithms for mapping scientific workflow applications to Grid resources based on specified QoS (Quality of Service) constraints.

To address the field of Grid computing of workflow application scheduling, the thesis has made the following contributions:

- proposed a taxonomy of workflow management systems for Grid computing.
- developed a workflow engine which leverages tuple spaces to provide event-based execution management.
- developed deadline and budget distribution strategies based on the workload and dependency of tasks.
- developed algorithms for scheduling workflows with QoS constraints using genetic algorithms.
- leveraged multi-objective evolutionary algorithms (MOEAs) for workflow execution planning to generate a set of trade-off alternative scheduling solutions.

This is to certify that

(i) the thesis comprises only my original work,

(ii) due acknowledgement has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of table, maps, bibliographies, appendices and footnotes.

Signature _____

Date _____

Acknowledgments

I would like to thank my advisors, Associate Professor Rajkumar Buyya and Professor Raomamohanarao Kotagiri for guiding me to complete my PhD thesis successfully. I am grateful to them for motivating and inspiring me to go deeply into the field of Grid computing and workflows and supporting me throughout the life cycle of PhD studies.

I want to thank present and past members of GRIDs Lab who have provided me with help during my PhD study time. Thanks to Chee Shin Yeo, Hussein Gibbins, Anthony Sulistio, Srikumar Venugopal, Tianchi Ma, Sushant Goel, Krishna Nadiminti, Kyong Hoon Kim, Suraj Pandey for proof-reading my research papers and giving advices on my research. Thanks to Marcos Assunção, Marco A. S. Netto, Rajiv Ranjan, Al-Mukaddim Khan Pathan, and Md Mustafizur Rahman for their great company in playing squash which made my PhD study period more enjoyable. Thanks to Xingchen Chu and Luyao Wu for sharing their skills on building the workflow engine.

Thanks to Michael Kirley for the discussion of multi-objective algorithms and Robert Stewart for providing scripts for statistical analysis. Thanks to James Dobson for discussion and providing data for executing neuro-science workflow applications.

I want to offer my special thanks to Chen Khong Tham (National University, Singapore), Wolfram Schiffmann (FernUniversitaet in Hagen, Germany), Ivona Brandic (University of Vienna, Austria), Soonwook Hwang (National Institute of Informatics, Japan), Ewa Deelman (University of Southern California, USA), Chris Mattmann (NASA Jet Propulsion Laboratory, USA), Henan Zhao (University of Manchester, UK), Bertram Ludaescher (University of California, Davis), Thomas Fahringer (University of Innsbruck, Austria), Gregor von Laszewski (Argonne National Laboratory, USA), Ken Kennedy, Anirban Mandal, and Chuck Koelbel (Rice University, USA), Pramod Kumar Konugurthi (Indian Space Re-

search Organisation, India) for providing information of their projects and comments on my research work.

I would like to thank all sponsors of research projects undertaken at the Grid Computing and Distributed Systems (GRIDS) Lab especially the Australian Research Council as their funding for the GRIDS Lab has enabled me to carry out the research report in this thesis.

I would like thank my parents and sister for their unreserved support and understanding during my research studies.

I would like to thank all other staff members in the Department and the University especially Julien Reid, Binh Phan, Pinoo Bharucha and technical staff for their enthusiastic support.

Jia Yu
Melbourne, Australia
Oct 15, 2007

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the context of the research presented in this thesis. It firstly provides a high-level overview of workflow management systems. Then, it briefly presents the inspiration of QoS-based workflow scheduling and the primary contributions. This chapter ends with a discussion on the organization for the rest of this thesis.

1.1 Grid Workflow Management Systems

Grids [65] have emerged as global cyber-infrastructure for the next-generation of e-Science applications by integrating large-scale, distributed and heterogeneous resources. More recently, Grid computing has progressed towards a service-oriented paradigm, which defines a new way of service provisioning based on utility computing models [166]. Within utility Grids, each resource is represented as a service to which consumers can negotiate their usage and quality of service.

Scientific communities, such as high-energy physics, gravitational-wave physics, geophysics, astronomy and bioinformatics, are utilizing Grids to share, manage and process large data sets [162]. In order to support complex scientific experiments, distributed resources such as computational devices, data, applications, and scientific instruments need to be orchestrated while managing the application operations within Grid environments [115]. A workflow expresses an automation of procedures wherein files and data are passed between procedures applications according to a defined set of rules, to achieve an overall goal [44]. A workflow management system defines, man-

ages and executes workflows on computing resources. The use of the workflow paradigm for application composition on Grids offers several advantages [153] such as:

- Ability to build dynamic applications and orchestrate the use of distributed resources.
- Utilization of resources that are located in a suitable domain to increase throughput or reduce execution costs.
- Execution spanning multiple administrative domains to obtain specific processing capabilities.
- Integration of multiple teams involved in managing different parts of the experiment workflow – thus promoting inter-organizational collaborations.

Realizing workflow management for Grid computing requires a number of challenges to be overcome. They include workflow application modeling, workflow scheduling, resource discovery, information services, data management, and fault management. However, from the user's perspective, two important barriers that need to be overcome are: (1) the complexity of developing and deploying workflow applications; and (2) their scheduling on heterogeneous and distributed resources to enhance the utility of resources and meet user QoS (Quality of Service) demands. This thesis presents a software framework for composing and managing scientific workflow applications, and scheduling algorithms based on users' requirements.

1.2 QoS Workflow Scheduling

Workflow scheduling is one of the key issues in the management of workflow execution. Scheduling is a process that maps and manages execution of inter-dependent tasks on distributed resources. It introduces allocating suitable resources to workflow tasks so that the execution can be completed to satisfy objective functions specified by users. Proper scheduling can have significant impact on the performance of the system. In general, the problem of mapping tasks on distributed resources belongs to a class of problems known as NP-hard problems [62]. For such problems, no known algorithms

are able to generate the optimal solution within polynomial time. Even though the workflow scheduling problem can be solved by using exhaustive search, the time taken for generating the solution is very high. Scheduling decisions must be made in the shortest time possible in Grid environments, because there are many users competing for resources, so time slots desired by one user could be taken by another user at any moment.

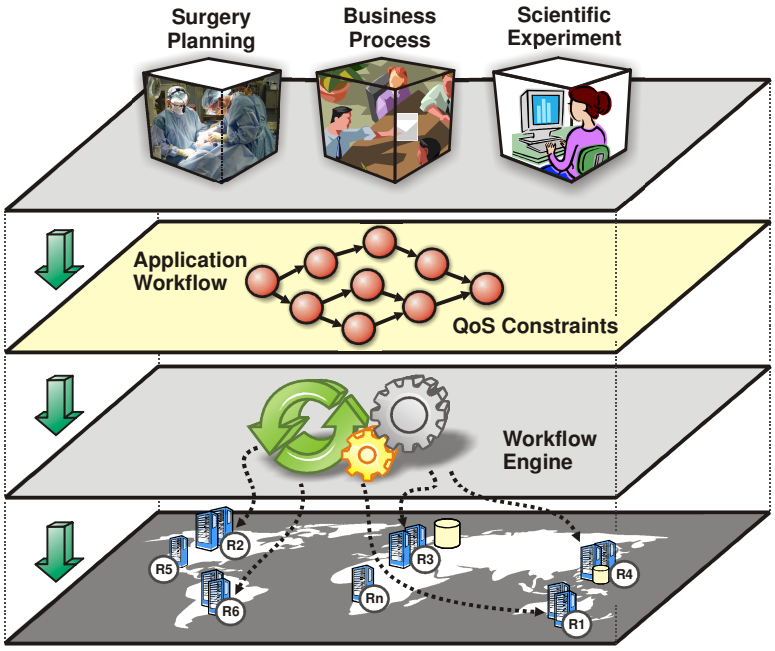


Figure 1.1: A high-level view of a Grid workflow execution environment.

A number of *best-effort* scheduling heuristics [21][109][181] such as Min-Min and HEFT (Heterogeneous Earliest Finish Time) have been developed and applied to schedule Grid workflows. These best-effort scheduling algorithms attempt to complete execution within the shortest time possible. They neither have any provision for users to specify their QoS requirements nor any specific support to meet them. However, many workflow applications in both scientific and business domains require some certain assurance of QoS (see Figure 1.1). For example, a workflow application for maxillo-facial surgery planning [79] needs results to be delivered before a certain time. For these applications, the workflow scheduling applied should be able to ana-

lyze users' QoS requirements and map workflow tasks onto suitable resources such that the workflow execution can be completed to satisfy their requirements.

Several new challenges are presented while scheduling workflows with QoS constraints for Grid computing. A Grid environment consists of large number of resources owned by different organizations or providers with varying functionalities and able to guarantee differing QoS levels. Unlike best-effort scheduling algorithms, which only consider one factor (e.g. execution time), multiple criteria must be considered to optimize the execution performance of QoS constrained workflows. In addition to execution time, monetary execution cost is also an important factor that determines quality of scheduling algorithms, because service providers may charge differently for different levels of QoS [26]. Therefore, a scheduler cannot always assign tasks onto services with the highest QoS levels. Instead, it may use cheaper services with lower QoS that is sufficient enough to meet the requirements of the users.

Also, completing the execution within a required QoS not only depends on the global scheduling decision of the workflow scheduler, but also depends on the local resource allocation model of each execution site. If the execution of every single task in the workflow cannot be completed as expected by the scheduler, it is impossible to guarantee QoS levels for the entire workflow. Therefore, schedulers should be able to interact with service providers to ensure resource availability and QoS levels. It is required that the scheduler can determine QoS requirements for each task and negotiate with service providers to establish a Service Level Agreement (SLA) [106], which is a contract specifying the minimum expectations and obligations between service providers and consumers.

This thesis, therefore, is focused on developing and evaluating a number of workflow scheduling algorithms based on QoS constraints such as deadline and budget while taking into account the costs and capabilities of Grid services. It also presents a workflow execution framework to facilitate deployment of scheduling strategies and interaction with Grid resources.

1.3 Contributions

This thesis makes several contributions towards enhancing the understanding of workflow management in Grid environments and advancing the area of QoS-aware workflow scheduling as specified below:

1. This thesis provides comprehensive taxonomies of various aspects of workflow design, information retrieval, workflow scheduling, fault tolerance and data movement. The taxonomies not only provide a basis for differentiating various Grid workflow systems but also identify strengths and weaknesses of the state-of-art in Grid workflows and offer directions for future work.

2. This thesis presents the design and development of a workflow enactment engine for composing and deploying workflows onto Grid resources. The engine utilizes functions offered by the Gridbus resource broker to support interaction with different Grid middleware such as globus, Condor [158] and Sun Grid Engine [157]. It provides a decentralized architecture to facilitate deployment of multiple scheduling strategies for workflow tasks. It uses an event-driven mechanism to trigger workflow task execution in order to support complex workflow structures. The tuple space paradigm has been leveraged to simplify implementation of the workflow engine. Parameterization is also supported at both workflow language level and execution level in order to support large scientific experiments. The capabilities of the engine have been demonstrated using real-world applications by creating a neuroscience application and executing it on global Grid resources.

3. This thesis presents heuristics to optimize execution performance while meeting users' specified time or monetary cost constraints. These heuristics provide strategies to distribute the overall deadline and budget of the entire workflow over single workflow tasks based on their workload and dependen-

cies. The sub-deadline and sub-budget assigned to each task are important for negotiating and monitoring SLAs between the workflow engine and service providers for task execution.

4. This thesis develops Genetic Algorithm (GA) - inspired techniques for scheduling workflows with QoS constraints. The goal is either to minimize execution time while meeting the user's budget constraint or to minimize execution cost while meeting the user's deadline constraint. The algorithm has been evaluated by comparing with non-GA algorithms for various constraint levels and workflow structures.

5. This thesis presents a multi-objective planning approach for workflow execution. It introduces Multi-Objective Evolutionary Algorithms (MOEAs) for generating a set of trade-off scheduling solutions according to users' QoS requirements. The goal is to simultaneously minimize two conflicting objectives - execution time and execution price while meeting users' maximum time constraint (deadline) and cost constraint (budget). Simulation studies have been conducted using the GridSim Toolkit [155] to demonstrate the ability of MOEAs in optimally scheduling workflows derived from neuroscience applications and astronomy applications.

1.4 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 presents an overview of Grid workflow management systems and proposes a taxonomy that characterizes and classifies approaches of scientific workflow systems in the context of Grid computing. This thesis then presents the design of the workflow enactment engine in Chapter 3. The engine supports parameterization and decentralized scheduling architecture. Chapter 4 presents the formalization of scheduling problems for utility Grids, application models and evaluation simulation environments. It also introduces overall deadline and budget distribution strategies for deadline and budget constrained heuris-

tics. Genetic algorithms-based scheduling algorithms for solving the QoS constrained problems are presented in Chapter 5. Chapter 6 introduces and evaluates multi-objective evolutionary algorithms (MOEAs) for workflow execution planning. Finally, this thesis concludes and presents future work in Chapter 7.

The core chapters are derived from various research papers published during the course of the PhD. candidature as detailed below:

Chaper 2 is derived from:

- **Jia Yu** and Rajkumar Buyya, A Taxonomy of Workflow Management Systems for Grid Computing, *Journal of Grid Computing*, Springer Press, New York, USA. September 2005.

- **Jia Yu** and Rajkumar Buyya, A Taxonomy of Scientific Workflow Systems for Grid Computing, Special Issue on Scientific Workflows, *SIGMOD Record*, 34(3):44-49, ACM Press, New York, USA, September 2005.

Chapter 3 is derived from:

- **Jia Yu** and Rajkumar Buyya, A Novel Architecture for Realizing Grid Workflow using Tuple Spaces, Proceedings of *the 5th IEEE/ACM International Workshop on Grid Computing* (Grid 2004, IEEE CS Press, Los Alamitos, CA USA), November 8, 2004, Pittsburgh, USA.

- **Jia Yu**, Srikumar Venugopal, Rajkumar Buyya, A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Services. *Journal of Supercomputers*, Kluwer Academic Publishers, USA, January 2006.

Chapter 4 is derived from:

- **Jia Yu**, Rajkumar Buyya and Chen Khong Tham, Cost-based Scheduling of Workflow Applications on Utility Grids, Proceedings of the *1st IEEE International Conference on e-Science and Grid Computing* (eScience2005, IEEE CS Press, Los Alamitos, CA, USA), December 5-8, 2005, Melbourne, Australia.

- **Jia Yu**, Rajkumar Buyya and Ramamohanarao Kotagiri, Workflow Scheduling Algorithms for Grid Computing, Fatos Xhafa and Ajith Abraham (ed.), *Meta-heuristics for Scheduling*, Springer, 2008. (in press)

Chapter 5 is derived from:

- **Jia Yu** and Rajkumar Buyya, A Constrained Scheduling of Workflow

Applications on Utility Grids using Genetic Algorithms, *Workshop on Workflows in Support of Large-Scale Science*, HPDC 2006, June 19-23, 2006, Paris, France.

- **Jia Yu** and Rajkumar Buyya, Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms, *Scientific Programming Journal*, 14(3-4):217-230, IOS Press, Amsterdam, The Netherlands, December 2006.

Chapter 6 is derived from:

- **Jia Yu**, Michael Kirley and Rajkumar Buyya, Multi-objective Planning for Workflow Execution on Utility Grids, Proceedings of *the 8th IEEE/ACM International Conference on Grid Computing* (Grid 2007, IEEE CS Press, Los Alamitos, CA USA), Austin, USA, September 19-21, 2007.

Chapter 2

A Taxonomy of Workflow Management Systems

This chapter provides a general model and taxonomy of Grid workflow systems. The taxonomy covers the topics such as workflow design, information retrieval, workflow scheduling, fault tolerance and data movement. This chapter also classifies mechanisms developed and applied in several representative Grid workflow systems according to the taxonomy.

2.1 Grid Workflow Management Systems

Figure 2.1 shows the architecture and functionalities supported by various components of the Grid workflow system based on the workflow reference model [44] proposed by Workflow Management Coalition (WfMC) [184] in 1995. At the highest level, functions of Grid workflow management systems could be characterized into *build-time* functions and *run-time* functions. The build-time functions are concerned with defining, and modeling workflow tasks and their dependencies; while the run-time functions are concerned with managing workflow executions and interactions with Grid resources for processing workflow applications. Users interact with workflow modeling tools to generate a workflow specification, which is submitted to a run-time service called the workflow enactment service for execution. Major functions provided by the workflow enactment service are scheduling, fault management and data movement. Workflow scheduling discovers resources and allocates tasks on

suitable resources to meet users' requirements, while data movement manages data transfer between selected resources and fault management provides mechanisms for failure handling during execution. In addition, the enactment engine provides feedback to a monitor so that users can view the workflow process status through a Grid workflow monitor.

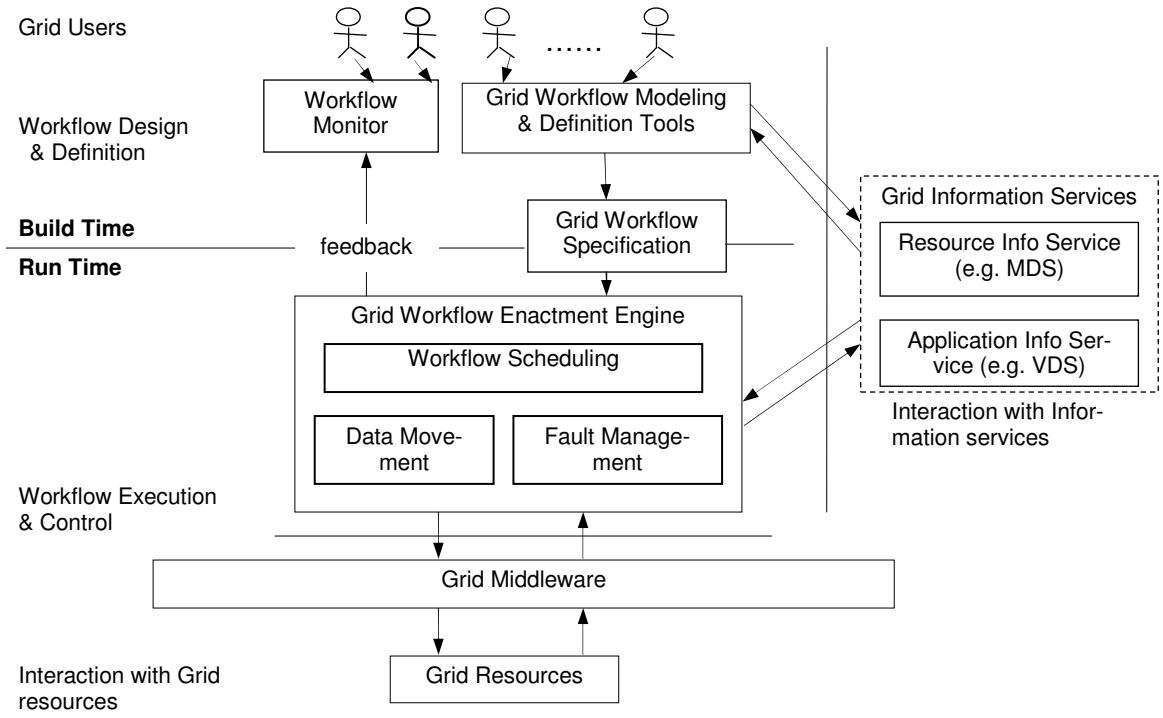


Figure 2.1: Workflow management system.

The workflow enactment service may be built on the top of low level Grid middleware (e.g. Globus toolkit [69], UNICORE [169] and Alchemi [107]), through which the workflow management system invokes services provided by Grid resources. At both the build-time and run-time stages, the information about resources and applications may need to be retrieved using Grid information services.

In the recent past, several Grid workflow systems have been proposed and developed for defining, managing and executing scientific workflows. In order to enhance our understanding of the field, a taxonomy is proposed to primarily (a) capture archi-

tectural styles and (b) identify design and engineering similarities and differences between them. The taxonomy provides an in-depth understanding of building and executing workflows on Grids. It compares different approaches and also helps users to decide on minimum subset of features required for their systems.

2.2 Taxonomy

The taxonomy characterizes and classifies approaches of workflow management in the context of Grid computing. As shown in Figure 2.2, it consists of five elements of a Grid workflow management system: (a) workflow design, (b) information retrieval, (c) workflow scheduling, (d) fault tolerance and (e) data movement. In this section, we look at each element and its taxonomy in detail.

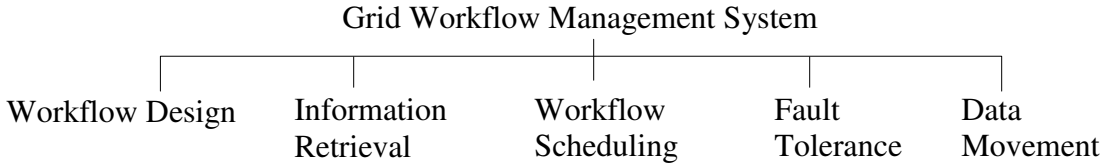


Figure 2.2: Elements of a Grid workflow management system.

2.2.1 Workflow Design

As shown in Figure 2.3, workflow design includes four key factors, namely (a) workflow structure, (b) workflow model/specification, (c) workflow composition system, and (d) workflow QoS (Quality of Service) constraints.

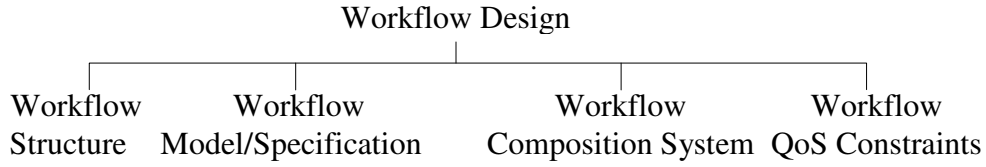


Figure 2.3: Workflow design taxonomy.

2.2.1.1 Workflow Structure

A workflow is composed by connecting multiple tasks according to their dependencies. The workflow structure, also referred as workflow pattern [2][3][6], indicates the temporal relationship between these tasks. Figure 2.4 shows the workflow structure taxonomy. In general, a workflow can be represented as a *Directed Acyclic Graph (DAG)* [140] or a *non-DAG*.

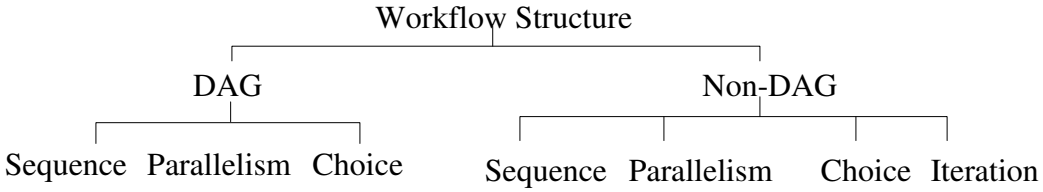


Figure 2.4: Workflow structure taxonomy.

In DAG-based workflow, workflow structure can be classified as *sequence*, *parallelism*, and *choice*. Sequence is defined as an ordered series of tasks, with a task starting after previous one has been completed. Parallelism represents tasks which are performed concurrently, rather than serially. In choice control pattern, a task is selected to execute at run-time when its associated conditions are true.

In addition to all patterns contained in a DAG-based workflow, a non-DAG workflow also includes the *iteration* structure in which sections of workflow tasks in an iteration block are allowed to be repeated. Iteration is also known as *loop* or *cycle*. The iteration structure is quite frequently used in scientific applications, where one or more tasks need to be executed repeatedly [114]. For example, in a promoter identification workflow [105] as shown in Figure 2-5, step 5 to step 8 are executed iteratively to create and refine a promoter model.

These four types of workflow structure, namely sequence, parallelism, choice and iteration, can be used to construct many complex workflows. Moreover, sub-workflows can also use these types of workflow structure as building blocks to form a large-scale workflow.

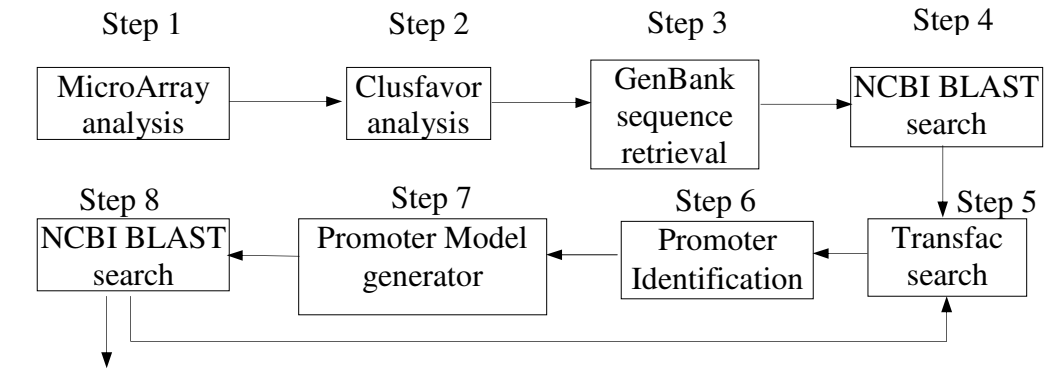


Figure 2.5: Promoter identification workflow.

2.2.1.2 Workflow Model/Specification

Workflow Model (also called workflow specification) defines a workflow including its task definition and structure definition. As shown in Figure 2.6, there are two types of workflow models, namely *abstract* and *concrete*. They are also referred to as abstract workflows and concrete workflows [50][52]. In some literature (e.g. [104]), concrete models are referred to as executable workflows.

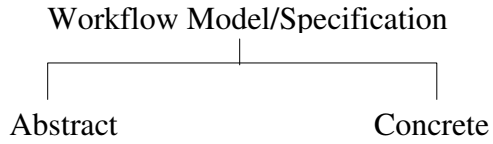


Figure 2.6: Workflow model taxonomy.

In an abstract model, a workflow is described in an abstract form in which the workflow is specified without referring to specific Grid resources for task execution. An abstract model provides a flexible way for users to define workflows without being concerned about low-level implementation details. Tasks in an abstract model are portable and can be mapped onto any suitable Grid services at run-time by using suitable discovery and mapping mechanisms. Using abstract models also eases the

sharing of workflow descriptions between Grid users [52]; in particular it benefits the participants of Virtual Organizations (VOs) [66].

In contrast, a concrete model binds workflow tasks to specific resources. In some cases, a concrete model may include tasks acting as data movement to transfer data in and out of the computation and data publication to publish newly derived data into VO [52]. In other situations, tasks in a concrete model may also include necessary application movement to transfer computational code to a data site for large scale data analysis.

Given the dynamic nature of the Grid environment, it is more suitable for users to define workflow applications in abstract models. A full or partial concrete model can be generated just before or during workflow execution according to the current status of resources. Additionally, in some systems [192], every task in a workflow is concretized only at the time of task execution. However, concrete models may be used by some end users who want to control the execution sequence [95].

2.2.1.3 Workflow Composition System

Workflow composition systems are designed for enabling users to assemble components into workflows. They need to provide a high level view for the construction of Grid workflow applications and hide the complexity of underlying Grid systems. Figure 2.7 shows the taxonomy for the workflow composition systems. *User-directed* composition systems allow users to edit workflows directly, whereas *automatic* composition systems generate workflows for users automatically. In general, users can use workflow languages for *language-based modeling* and the tools for *graph-based modeling* to compose workflows.

Within language-based modeling, users may express workflow using a *markup* language such as Extensible Markup Language (XML) [176] (e.g. GridAnt [95], WSFL [99], XLANG [164], BPEL4WS [15], W3C XML-Pipeline language [179], and Gridbus Workflow [192]) or other formats (e.g. Condor DAGman [158]). Language-based modeling may be convenient for skilled users, but they require users to memorize a lot of language-specific syntax. In addition, it is impossible for users to

express a complex and large workflow by scripting workflow components manually. However, workflow languages are more appropriate for sharing and manipulation, whereas the graphical representations are intuitive but they require to be converted into other forms for manipulation. So in most Grid systems, workflow languages are designed to bridge the gap between the graphical clients and the Grid workflow execution engine [76]. XML-based languages are used widely for workflow specification as it facilitates information description in a nested structure. Moreover, many tools are provided to validate XML syntax and verify XML documents against XML schema [178] or DTD (Document Type Definition) [176]. Furthermore, many XML parsing tools (e.g. JDOM [87] and dom4j [54]) are widely available.

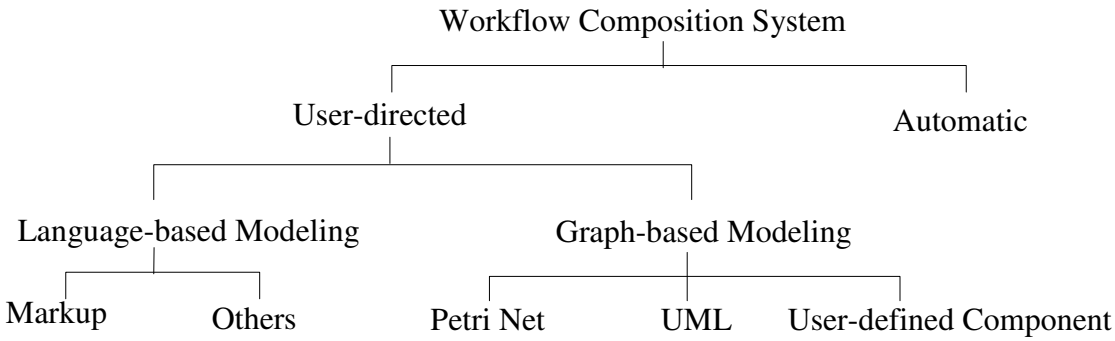


Figure 2.7: Workflow composition system taxonomy.

Graph-based modeling allows graphical definition of an arbitrary workflow through a few basic graph elements. It allows users to work with a graphical representation of the workflow. Users can compose and review a workflow by just clicking and dropping the components of interest. It avoids low-level details and hence enables users to focus on higher levels of abstraction at application level [78]. The major modeling approaches are *Petri Nets* [129], *UML* (Unified Modeling Language) [124] and *user-defined component*. Graph-based modeling is preferred by users as opposed to language-based modeling.

Petri Nets are a special class of directed graphs that can model sequential, parallel, loops and conditional execution of tasks [76][80]. They have been used in many

workflow management systems such as Grid-Flow [76], FlowManager [98], and XRL/Flower [175]. UML activity diagrams [125] have also been extended and applied as a workflow specification language [18][56][131]. Compared with UML activity diagrams, Petri Nets have formal semantics and have been used widely for constructing several workflows [1][57]. A vast number of algorithms and tools for Petri Nets analysis have been developed along the years [111]. However, Eshuis et al. [57] argue that Petri Nets may be unable to model workflow activities accurately without extending its semantics and this drawback has been addressed in UML activity diagrams. Rather than following the standard syntax and semantics of Petri Nets and UML, many workflow editors for Grid workflow tools create their own graphical representation of workflow components. For example, Triana [161] allows users to predefine software components and reuse them to design DAG-based workflows. Kepler [14] provides graphical environment and a framework that supports the design and reuse of grid workflows. These tools are more convenient for users to manipulate their workflow applications, as they provide a more user-friendly programming environment. They have also been integrated into underlying local applications, Grid middleware and monitoring systems. For example, P-GRADE [89][103] interoperates with a wide range of parallel applications in addition to Condor and Globus based Grid middleware. It also allows users to access and modify program code of a workflow task through a graphical editor. However, lack of standards hinders the collaboration between these projects. Many works are thus replicated such as different user interfaces developed by different projects for the same functionality. Moreover, workflow structures supported by most of them are limited to only sequence and parallelism.

Graph-based modeling is very intuitive and can be handled easily even by a non-expert user. However, the layout of workflow components on a display screen can become very huge and difficult to manage [126]. One of the solutions to overcome this limitation is to use hierarchical graph definition [80]. Another solution is to have a system which composes workflows automatically. Pegasus [52] is one such automatic composition system for Grid computing; it has to be adapted to particular

applications, because the composition is based on application-dependent metadata. It receives a metadata description of desired data products and initial input values from users. The tasks are then composed automatically to form a workflow by querying a virtual data catalog [67] that contains information for data derivation of application components. Compared with user-directed systems, automatic composition systems are ideal for large scale workflows which are very time consuming to compose manually. However, the automatic composition of application components is challenging because it is difficult to capture the functionality of components and data types used by the components [33] [126].

2.2.1.4 Workflow QoS Constraints

In a Grid environment, there are a large number of similar or equivalent resources provided by different parties. Grid users can select suitable resources and use them for their workflow applications. These resources may provide the same functionality, but optimize different QoS measures. In addition, different users or applications may have different expectations and requirements. Therefore, it is not sufficient for a workflow management system to only consider functional characteristics of the workflow. QoS requirements such as time limit (deadline) and expenditure limit (budget) for workflow execution also need to be managed by workflow management systems. Users must be able to specify their QoS expectations of the workflow at the design level. Then, the actions conducted by workflow systems using run-time must be chosen according to the initial QoS requirements.

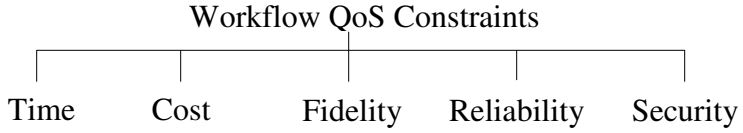


Figure 2.8: Workflow QoS constraints taxonomy.

Figure 2.8 shows the taxonomy of Grid workflow QoS constraints based on a QoS model for Web services based workflow provided by Cardoso et al. [34] and QoS of Web services [110][128]. It includes five dimensions: *time*, *cost*, *fidelity*, *reliability*

and *security*. Time is a basic measure of performance. For workflow systems, it refers to the total time required for completing the execution of a workflow. Cost represents the cost associated with the execution of workflows including the cost for managing workflow systems and usage charge of Grid resources for processing workflow tasks. Fidelity refers to the measurement related to the quality of the output of workflow execution. Reliability is related to the number of failures for execution of workflows. Security refers to confidentiality of the execution of workflow tasks and trustworthiness of resources.

As indicated in Figure 2.9, there are two different ways to assign QoS constraints in a workflow model. One way is to allow users to assign QoS constraints at *task-level*. The overall QoS can be assessed by computing all individual tasks. For example, a user assigns desired execution time for every task in a workflow. The deadline for the entire workflow execution can be calculated by a workflow reduction algorithm (e.g. SWR(w) algorithm [32]). Another way is to assign QoS constraints at *workflow-level*, allowing users to define the overall workflow QoS requirements. However, QoS constraints for each task may be required by schedulers for resource allocation at run-time. For the time dimension, users are likely to specify a deadline for the entire workflow execution rather than for every single task. In order to fulfill the deadline for the entire workflow, the scheduler needs to decide how fast each task has to be processed using a deadline assignment approach (e.g. Ultimate Deadline, Effective Deadline, Equal Slack, and Equal Flexibility strategies in [90]).

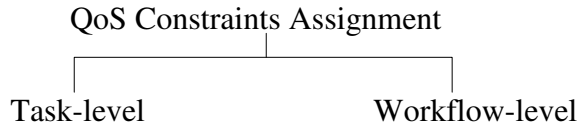


Figure 2.9: QoS constraints assignment taxonomy.

2.2.2 Information Retrieval

A Grid workflow management system does not execute the tasks itself, but it merely coordinates the execution of the tasks by the Grid resources. To map tasks onto suit-

able resources, information about the resources has to be retrieved from appropriate sources [189]. As indicated in Figure 2.10, there are three dimensions of information retrieval: *static information*, *historical information* and *dynamic information*.

Static information refers to information that does not vary with time. It may include *infrastructure-related* (e.g. the number of processors), *configuration-related* (e.g. operating system, libraries), *QoS-related* (e.g. flat usage charge), *access-related* (e.g. service operations), and *user-related* information (e.g. authentication ID). Generally, static information is utilized by Grid workflow management systems to pre-select resources during the initiation of the workflow execution.

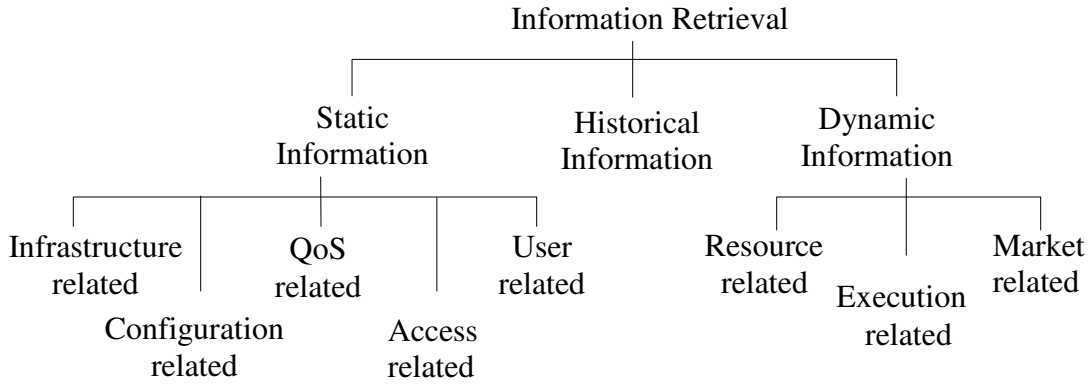


Figure 2.10: Information retrieval taxonomy.

As Grid resources are not dedicated to the owners of the workflow management systems, the Grid workflow management system also needs to identify dynamic information such as resource accessibility, system workload, and network performance during execution time. Unlike static information, dynamic information reflects the status of the Grid resources, such as load average of a cluster, available disk space, CPU usage, and active processes. It also includes task execution information and market related information such as dynamic resource price.

Historical information is obtained from previous events that have occurred such as performance history and execution history of Grid resources and application components. Generally, workflow management systems can analyze historical information to predict the future behaviors of resources and application components on a given set

of resources. Historical information can also be used to improve the reliability of future workflow execution. For example, the user can correct the logic of a failed workflow according to the log of the workflow system.

Several information services are available for accessing static and dynamic information about Grid resources. For example, Monitoring and Discovery System (MDS) [139] provides static hardware information such as CPU type, memory size and software information such as operating system information, and some dynamic information such as CPU load snapshot. Network Weather Service (NWS) [183] provides additional dynamic information about availability of CPU, memory, and bandwidth. An object oriented model for publication and retrieval of electronic resources is given in [40].

2.2.3 Workflow Scheduling

Casavant et al. [36] categorized task scheduling in distributed computing systems into 'local' task scheduling and 'global' task scheduling. Local scheduling involves handling the assignment of tasks to time-slices of a single resource whereas global scheduling involves deciding where to execute a task. According to this definition, workflow scheduling is a kind of global task scheduling as it focuses on mapping and managing the execution of inter-dependent tasks on shared resources that are not directly under its control.

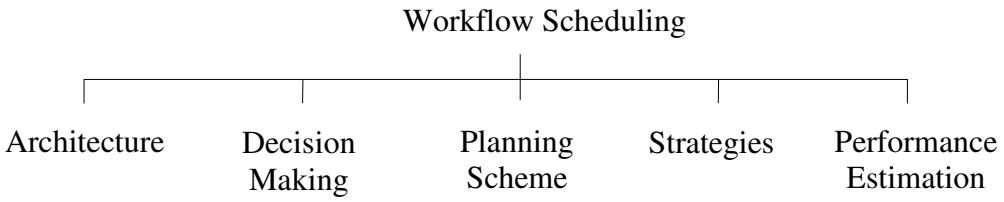


Figure 2.11: Workflow scheduling taxonomy.

The workflow scheduler needs to coordinate with diverse local management systems as Grid resources are heterogeneous in terms of local configuration and policies. Taking into account users' QoS constraints is also important in the scheduling process so as to satisfy user requirements. In this section, we discuss workflow scheduling

taxonomy from the view of (a) scheduling architecture, (b) decision making, (c) planning scheme, (d) scheduling strategy, and (e) performance estimation as shown in Figure 2.11.

2.2.3.1 Scheduling Architecture

The architecture of the scheduling infrastructure is very important for scalability, autonomy, quality and performance of the system [77]. Three major categories of workflow scheduling architecture as shown in Figure 2.12 are centralized, hierarchical and decentralized scheduling schemes.

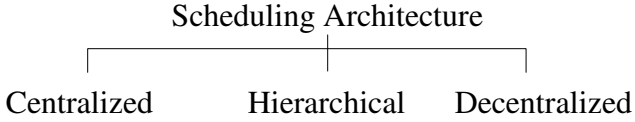


Figure 2.12: Scheduling architecture taxonomy.

In a *centralized* workflow enactment environment, one central workflow scheduler makes scheduling decisions for all tasks in the workflow. The scheduler has the information about the entire workflow and collects information of all available processing resources. It is believed that the centralized scheme can produce efficient schedules because it has all necessary information [77]. However, it is not scalable with respect to the number of tasks, the classes and number of Grid resources. It is thus only suitable for a small scale workflow or a large scale workflow in which every task has the same objective (e.g. same class of resources).

Unlike centralized scheduling, both *hierarchical* and *decentralized* scheduling allow tasks to be scheduled by multiple schedulers. Therefore, one scheduler only maintains the information related to a sub-workflow. Thus, compared to *centralized* scheduling, they are more scalable since they limit the number of tasks managed by one scheduler. However, the best decision made for a partial workflow may lead to sub-optimal performance for the overall workflow execution. Moreover, conflict problems are more severe [112]. One example of conflict is that tasks from different sub-workflows scheduled by different schedulers may compete for the same resource.

For *hierarchical* scheduling, there is a central manager and multiple lower-level sub-workflow schedulers. This central manager is responsible for controlling the workflow execution and assigning the sub-workflows to the low-level schedulers. For example, in GridFlow project [31], there is one workflow manager and multiple lower-level schedulers. The workflow manager schedules sub-workflows onto corresponding lower-level schedulers. Each lower-level scheduler is responsible for scheduling tasks in a sub-workflow onto resources owned by one organization. The major advantage of using the hierarchical architecture is that the different scheduling policies can be deployed in the central manager and lower-level schedulers [77]. However, the failure of the central manager will result in entire system failure.

In contrast, there are multiple schedulers without a central controller in decentralized scheduling. Every scheduler can communicate with each other and schedule a sub-workflow to another scheduler with lower load. Compared to hierarchical scheduling, decentralized scheduling is more scalable but faces more challenges to generate optimal solutions for overall workflow performance and minimize conflict problems.

2.2.3.2 Decision Making

There is no single best solution for mapping workflows onto resources for all workflow applications, since the applications can have very different characteristics. It depends to some degree on the application models to be scheduled. In general, decisions about mapping tasks in a workflow onto resources can be based on the information of the current task or of the entire workflow and can be of two types, namely *local* decision and *global* decision [50] as shown in Figure 2.13. Scheduling decisions made with reference to just the task or sub-workflow at hand are called local decisions whereas scheduling decisions made with reference to the whole workflow are called global decisions.

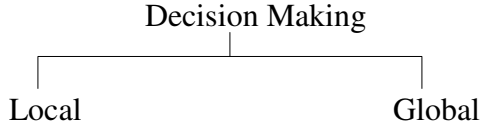


Figure 2.13: Decision making taxonomy.

Local decision based scheduling only takes one task or sub-workflow into account, so it may produce the best schedule for the current task or sub-workflow but could also reduce the entire workflow performance. An example given by Deelman et al. [50] assumes that there is a data-intensive application where the overall run-time is driven by data transfer costs. Consider a situation where the output of a task is very large. If the selection of a resource for a task is based only on a local decision without consideration of data transfer between other resources, when selection of a resource for child tasks need to be made, the initial selection may be found to be a poor choice if latency between the nodes is very high. This would lead to higher data transfer costs for this child task and hence the entire workflow.

Scheduling workflow tasks using global decision improves the performance of the entire workflow. There are some algorithms for scheduling task graphs in parallel systems that could be applied to Grid workflow scheduling. Li et al. [100] developed the Forward-Looking Analysis Method (FLAM). It analyses dependencies of the entire graph to resolve the conflicts of parallel tasks which compete for the same resource. It is believed that global decision based scheduling can provide a better overall result. However, it may take much more time in scheduling decision making. Thus, the overhead produced by global scheduling could reduce the overall benefit and may even exceed the benefits it will produce [50]. Therefore, the choice of decision making for workflow scheduling should not be made without considering balance between the overall execution time and scheduling time. However, for some applications such as a data analysis application where the outputs of tasks in the workflow are always smaller than the inputs, using local decision based scheduling is sufficient.

2.3.3.3 Planning Scheme

A planning scheme is a method for translating abstract workflows to concrete workflows. As shown in Figure 2.14, schemes for the schedule planning of workflow applications can be categorized into either *static* scheme or *dynamic* scheme. In a static scheme, concrete models have to be generated before the execution according to current information about the execution environment and the dynamically changing

state of the resources is not taken into account. In contrast, a dynamic scheme uses both dynamic information and static information about resources to make scheduling decisions at run-time.

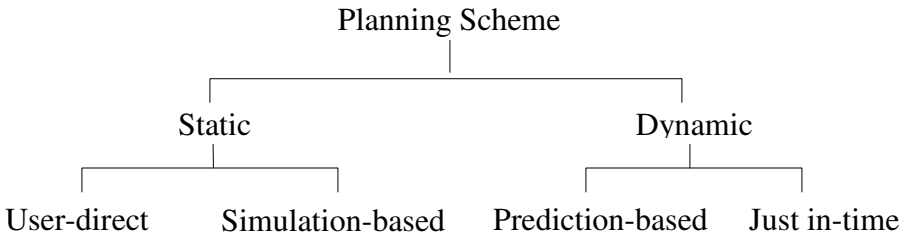


Figure 2.14: Planning scheme taxonomy.

Static schemes, also known as *full-ahead* planning, include *user-directed* and *simulation-based* scheduling. In user-directed scheduling, users emulate the scheduling process and make resource mapping decisions according to their knowledge, preference and/or performance criteria. For example, users prefer to map tasks to resources on which they have not experienced failures. In simulation-based scheduling, the 'best' schedule is achieved by simulating task execution on a given set of resources before a workflow starts execution. The simulation can be processed based on static information or the result of performance estimation. For example, in GridFlow [31], the 'best' resource selected for scheduling a task is based on the predictive task execution time that the resource provides.

Dynamic schemes include *prediction-based* and *just in-time* scheduling. Prediction-based dynamic scheduling uses dynamic information in conjunction with some results based on prediction. It is similar to simulation-based static scheduling, in which the scheduler is required to predict the performance of task execution on resources and generate a near optimal schedule for the task before it starts execution. However, it changes the initial schedule dynamically during the execution. For example, GrADS [39] generates preliminary mapping by using prediction results, but it migrates a task execution to another resource when its initial contract is broken or a better resource is found for execution. Sakellariou et al. [140] developed a low-cost rescheduling policy for the mapping of workflows on Grids. It considers rescheduling

workflow tasks at a few carefully selected points during execution in a dynamically changing Grid environment, since the initial schedule built using inaccurate predictions can affect performance significantly.

Rather than making a schedule ahead, just in-time scheduling [52] only makes scheduling decision at the time of task execution. Planning ahead in Grid environments may produce a poor schedule, since it is a dynamic environment where utilization and availability of resources varies over time and a better resource can join at any time. Moreover, it is not easy to accurately predict the execution time of all application components on Grid resources. However, as the technology of advance reservation [155] for various resources improves, it is believed that the role of static and prediction-based planning will increase [50].

2.3.3.4 Scheduling Strategy

In general, scheduling workflow applications in a distributed system is an NP-complete problem [62]. Therefore, many heuristics have been developed to obtain near-optimal solutions to match users' QoS constraints. As shown in Figure 2.15 we categorize strategies of major scheduling approaches into *performance-driven*, *market-driven* and *trust-driven*.

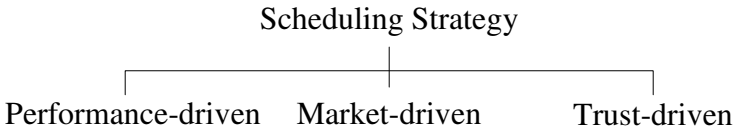


Figure 2.15: Scheduling strategy taxonomy.

Performance-driven strategies try to find a mapping of workflow tasks onto resources that achieves optimal execution performance such as minimize overall execution time. Most of Grid workflow scheduling systems falls in this category. GrADS [39] optimizes DAG-based workflows using Min-Min, Max-Min and Suffrage heuristics, hoping to obtain minimum completion times. Prodan et al. [132] use

classical genetic algorithms with cycle elimination techniques to minimize non-DAG based workflow execution on Grids.

Market-driven strategies employ market models to manage resource allocation for processing workflow tasks. They apply computational economy principle and establish an open electronic marketplace between workflow management systems and participating resource providers. Workflow schedulers act as consumers buying services from the resource providers and pay some notion of electronic currency for executing tasks in the workflow. The tasks in the workflow are dynamically scheduled at run-time depending on resource cost, quality and availability, to achieve the desired level of quality for deadline and budget. Unlike the performance-driven strategy, market-driven schedulers may choose a resource with later deadline if its usage price is cheaper. Market-driven strategies have been applied to several Grid systems such as Nimrod-G [9] and Gridbus data resource broker [173]. One example of the market-driven workflow scheduling proposed by Geppert et al. [72] utilizes market mechanisms during the task assignment. In the system, bids are collected from eligible resource providers for each task. The optimal bid is selected by computing the amount of time and cost saved or overdrawn up to the point. If the execution time has been minimized at the expense of an overdrawn cost, a bid with lower price will be chosen as the optimal bid. Consequently, scheduler assigns the task to the resource whose provider offers the optimal bid. A recent work on cost-based scheduling of workflow tasks on Grids is reported in [22].

Recently, trust-driven scheduling approaches (e.g. CCOF project in [196] and GridSec project in [150][151]) in distributed systems are emerging. Trust-driven schedulers select resources based on their trust levels. For example, within GridSec, the scheduler accesses the trust level of Grid sites. It maps tasks onto resources whose trust level is higher than users' demand. Trust model of resources is based on attributes such as security policy, accumulated reputation, self-defense capability, attack history, and site vulnerability. By using trust-driven approaches, workflow management systems can reduce the chance of selecting malicious hosts, and non-reputable

resources [196]. Therefore, overall accuracy and reliability of workflow execution will be increased.

2.3.3.5 Performance Estimation

In order to produce a good schedule, estimating the performance of tasks on resources is crucial, especially for constructing a preliminary workflow schedule. By using performance estimation techniques, it is possible for workflow schedulers to predict how tasks in a workflow or sub-workflow will behave on distributed heterogeneous resources and thus make decisions on how and where to run them. As indicated in Figure 2.16, there are several performance estimation approaches: *simulation*, *analytical modeling*, *historical data*, *on-line learning*, and *hybrid*.

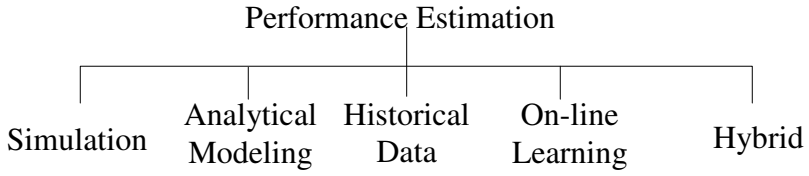


Figure 2.16: Performance estimation taxonomy.

Simulation approaches [53][199] provide resource simulation environments to emulate the execution of tasks in the workflow prior to its actual execution. In analytical modeling [39][46][123], a scheduler predicts the performance of tasks in workflow on a given set of resources based on an analytic metric. For example, in GrADS [39], two types of performance models are developed, namely memory hierarchy performance model and computational model. By using these models, one can predict memory requirements and the execution time of an application component for a resource according to the associated problem size. The historical data approach [86][114][148] relies on historical data to predict the task's execution performance. The historical data related to a particular user's application performance or experience can also be used in predicting the share of available of resources for that user while making scheduling decisions based on QoS constraints. The on-line learning approach predicts task execution performance from on-line experience without prior knowledge of the environment's dynamics. For example, Buyya et al. [27][28] and Galstyan et

al. [71] map a job onto a 'best' Grid resource by learning the completion time of most recent jobs submitted to resources. As historical and on-line learning approaches use experimental data, they can be broadly termed as *empirical modeling* approaches for performance estimation.

In certain conditions, these approaches could be used together in a hybrid approach for generating performance evaluation of workflow tasks. For instance, Bacigalupo et al. [17] use both layered queuing modeling and historical performance data to predict the performance of dynamic e-Commerce systems on heterogeneous servers. In addition, GrADS constructs computational models semi-automatically by emulating the execution of workflow components on small data sets. That is, it uses a combination of historical and analytical approaches for performance estimation.

2.2.4 Fault Tolerance

In a Grid environment, workflow execution failure can occur for various reasons: the variation in the execution environment configuration, non-availability of required services or software components, overloaded resource conditions, system running out of memory, and faults in computational and network fabric components. Grid workflow management systems should be able to identify and handle failures and support reliable execution in the presence of concurrency and failures.

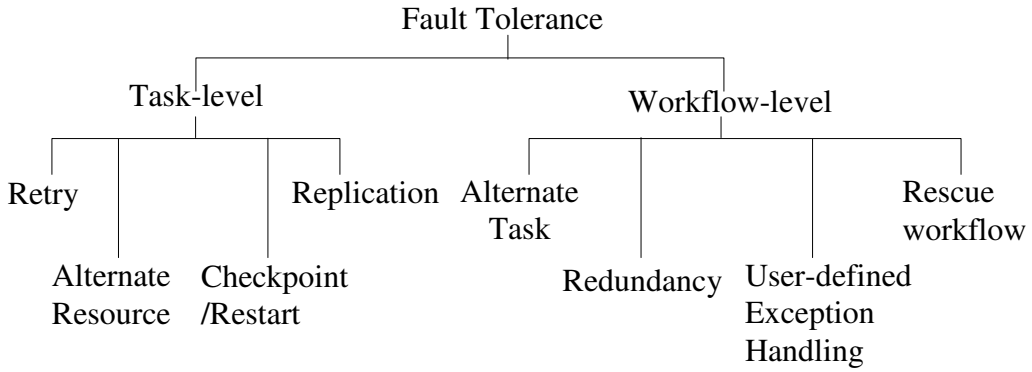


Figure 2.17: Fault tolerance taxonomy.

As shown in Figure 2.17, Hwang et al. [82] divided workflow failure handling techniques into two different levels, namely *task-level* and *workflow-level*. *Task-level* techniques mask the effects of the execution failure of tasks in the workflow, while *workflow-level* techniques manipulate the workflow structure such as execution flow to deal with erroneous conditions.

Task-level techniques have been widely studied in parallel and distributed systems. They can be cataloged into *retry*, *alternate resource*, *checkpoint/restart* and *replication*. The retry technique [159] is the simplest failure recovery technique, as it simply tries to execute the same task on the same resource after failure. The alternate resource technique [159] submits the failed task to another resource. The checkpoint/restart technique [45] moves failed tasks transparently to other resources, so that the task can continue its execution from the point of failure. The replication technique [7][82] runs the same task simultaneously on different Grid resources to ensure task execution provided that at least one of the replicas does not fail.

Workflow-level techniques include *alternate task*, *redundancy*, *user-defined exception handling* and *rescue workflow*. The first three approaches proposed by Hwang and Kesselman [82] assume there is more than one implementation for a certain computation with different execution characteristics. The alternate task technique executes another implementation of a certain task if the previous one failed, while the redundancy technique executes multiple alternative tasks simultaneously. The user-defined exception handling allows the users to specify a special treatment for a certain failure of a task in workflow. The rescue workflow technique developed in Condor DAGMan system [45] ignores the failed tasks and continues to execute the remainder of the workflow until no more forward progress can be made. Then, a rescue workflow description called rescue DAG, which indicates failed nodes with statistical information, is generated for later submission.

2.2.5 Intermediate Data Movement

For Grid workflow applications, the input files of tasks need to be staged to a remote site before processing the task. Similarly, output files may be required by their chil-

dren tasks which are processed on other resources. Therefore, the intermediate data has to be staged out to the corresponding Grid sites. Some systems require users to manage intermediate data transfer in the workflow specification, rather than providing *automatic* mechanisms to transfer intermediate data. As indicated in Figure 2.18, we categorize approaches of *automatic* intermediate data movement into *centralized*, *mediated* and *peer-to-peer*.

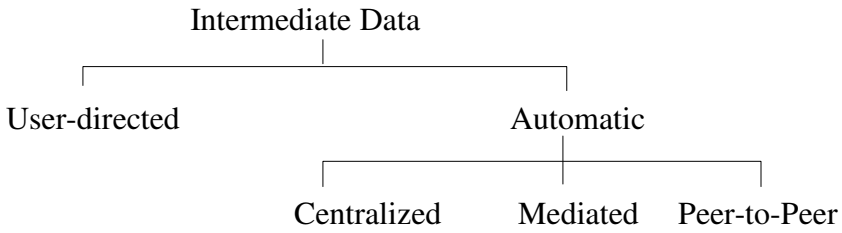


Figure 2.18: Intermediate data movement.

Basically a centralized approach transfers intermediate data between resources via a central point. For example, a central workflow execution engine can collect the execution results after task completion and transfer them to the processing entities of corresponding successors. Centralized approaches are easy to implement and suit workflow applications in which large-scale data flow is not required.

In a mediated approach, rather than using a central point, the locations of the intermediate data are managed by a distributed data management system. For example, in Pegasus system, the intermediate data generated at every step is registered in a replication catalog service [37], so that input files of every task can be obtained by querying the replication catalog service. Mediated approaches are more scalable and suitable for applications which need to keep intermediate data for later use.

A peer-to-peer approach transfers data between processing resources. Since data is transmitted from the source resource to the destination resource directly without involving any third-party service, peer-to-peer approaches save the transmission time and reduce the bottleneck problem caused by the centralized and mediated approaches. Thus, they are suitable for large-scale intermediate data transfer. However, there are more difficulties in deployment because they require every Grid node to be capable of

providing both data management and movement service. In contrast, centralized and meditated approaches are more suitable to be used in applications such as bio-applications, in which users need to monitor and browse intermediate results. In addition, they also need to record them for future verification purposes.

2.3 Survey

In this section, we present a detailed survey of existing Grid workflow systems in addition to mapping the proposed taxonomy. Table 2.1 shows the summary of selected Grid workflow management projects. A comparison of various Grid workflow systems and their categorization based on the taxonomy is shown in Table 2.2, Table 2.3, and Table 2.4.

Table 2.1: Summary of Grid workflow management projects.

Name	Organization	Grid Integration	Applications
DAGMan [158]	University of Wisconsin-Madison, USA. http://www.cs.wisc.edu/condor/dagman/	Condor which can run on top of Globus Toolkit version 2 (GT2)	Compute-intensive
Pegasus [51]	University of Southern California, USA. http://pegasus.isi.edu	Condor and Globus.	Targeted for data-intensive, but supports other types.
Triana [161]	Cardiff University, UK. http://www.trianacode.org/	GAT (JXTA, Web services, Globus)	Compute-intensive
ICENI [116]	London e-Science Centre, UK. http://www.lesc.ic.ac.uk/iceni/	Jini, JXTA, Globus	Compute-intensive
Taverna [125]	Collaboration between several European Institutes and industries. http://taverna.sourceforge.net/	Web services, Soaplab, local processor, BioMoby, etc.	Service Grids
GridAnt [95]	Argonne National Laboratory, USA. http://www.cogkit.org/	GT2, GT3, GT4	Client controllable workflow applications
GrADS [19]	Collaboration between several American Universities. http://www.hipersoft.rice.edu/grads/	Globus, Parallel Systems (e.g. MPI)	Compute-intensive and communication-intensive applications with MPI components

Name	Organization	Grid Integration	Applications
GridFlow [31]	University of Warwick, UK. http://www.dcs.warwick.ac.uk/research/hpsg/workflow/workflow.html	Parallel Systems (e.g. MPI and PVM)	MPI and PVM based components
Unicore [13]	Collaboration between German research institutions and industries http://www.unicore.org	Unicore	Computational-intensive and MPI components
Gridbus workflow [192]	The University of Melbourne, Australia. http://www.gridbus.org	GT2	Computational-and Data-intensive
Askalon [59]	University of Innsbruck http://dps.uibk.ac.at/askalon	GT2, GT4, WSRF, Web services	Performance- oriented applications
Karajan [96]	Argonne National Laboratory http://www.cogkit.org	GT2, GT3, GT4, Condor, runtime exec, ssh, WebDAV	Those required to access Grid middleware
Kepler [14]	A cross-project collaboration. http://kepler-project.org	Globus, Storage Resource Broker(SRB), EcoGrid, Web services	Scientific workflow applications

Table 2.2: Workflow design taxonomy mapping.

Project Name	Structure	Model	Composition Systems	QoS Constraints
DAGMan	DAG	Abstract	User-directed • Language-based	User specified rank expression for desired resources
Pegasus	DAG	Abstract	User-directed • Language-based Automatic	N/A
Triana	Non-DAG	Abstract	User-directed • Graph-based	N/A
ICENI	Non-DAG	Abstract	User-directed • Language-based • Graph-based	Metrics specified by users
Taverna	DAG	Abstract/ Concrete	User-directed • Language-based • Graph-based	N/A
GridAnt	Non-DAG	Concrete	User-directed • Language-based	N/A
GrADS	DAG	Abstract	User-directed • Language-based	Estimated application execution time
GridFlow	DAG	Abstract	User-directed	Application exe-

Project Name	Structure	Model	Composition Systems	QoS Constraints
			• Graph-based Language-based	cution time
Unicore	Non-DAG	Concrete	User-directed • Graph-based	N/A
Askalon	Non-DAG	Abstract	User-directed • Graph-based • Language-based	Constrains and properties specified by users or pre-defined
Karajan	Non-DAG	Abstract	User-directed • Language-based • Graph-based	N/A
Kepler	Non-DAG	Abstract/ Concrete	User-directed • Graph-based	N/A

Table 2.3: Workflow scheduling taxonomy mapping.

Project Name	Architecture	Decision Making	Planning Scheme	Strategies	Performance Estimation
DAGMan	Centralized	Local	Just in-time	Performance-driven	N/A
Pegasus	Centralized	Local/ Global	User-directed/ Just in-time	Performance-driven	Historical data, Analytical modeling
Triana	Decentralized	Local	Just in-time	Performance-driven	N/A
ICENI	Centralized	Global	Prediction-based	Performance/ Market-driven	Historical data
Taverna	Centralized	Local	Just in-time	Performance-driven	N/A
GridAnt	Centralized	User-defined*	User-directed	User-defined*	N/A
GrADS	Centralized	Local/ Global	Prediction-based	Performance-driven	Historical data (empirical) , Analytical modeling
GridFlow	Hierarchical	Local	Simulation-based	Performance-driven	Analytical modeling
Unicore	Centralized	User-defined*	User-directed	User-defined*	N/A
Askalon	Decentralized	Global	Just in-time/ Prediction-based	Performance/ Market-driven	Analytical modeling, Historical data
Karajan	Centralized	User-	User-defined*	User-defined*	N/A

Project Name	Architecture	Decision Making	Planning Scheme	Strategies	Performance Estimation
Kepler	Centralized	defined* User-defined*	User-defined*	User-defined*	N/A

***user-defined -** the architecture of the system has been explicitly designed for user extension.

Table 2.4: Information retrieval, fault-tolerance and data movement.

Project Name	Information Retrieval	Fault-tolerance	Data Movement
DAGMan	Resource information is retrieved by Condor *Matchmaker* that manages resource and task info advertisement and notification.	Task Level • Migration • Retrying Workflow Level • Rescue workflow	User-directed
Pegasus	Resource information retrieved through Globus MDS and RLS. Application component information is retrieved from the GriPhyN Transformation Catalog.	Based on DAGMan	Mediated
Triana	Based on GAT protocol	Based on GAT manger	Peer-to-Peer
ICENI	Application component information is retrieved by the component metadata service and performance repository service.	Based on middleware	Mediated
Taverna	Service information is retrieved through DAML-S web service ontology, domain ontology information service, and UDDI.	Task Level • Retry • Alternate Resource	Centralized
GridAnt	Resource information is retrieved through Globus MDS.	User-defined*	User-directed
GrADS	Resource information is retrieved through Globus MDS and GrADS information service (GIS). Dynamic information is retrieved by NWS. Autopilot is used for provide performance contract information.	Task Level in rescheduling work in GrADS, but not in workflows.	Peer-to-Peer
GridFlow	Resource information is re-	Task Level	Peer-to-Peer

Project Name	Information Retrieval	Fault-tolerance	Data Movement
	trieved through Titan	• Alternate resource	
Unicore	Unicore information service	Based on Unicore middleware	Mediated
Askalon	Static information • Infrastructure-related • Configuration-related • QoS-related Dynamic information • Resource-related • Execution-related	Task Level • Retry • Alternate resource Workflow level • Rescue workflow	Centralized User-directed
Karajan	User-defined*	Task Level • Retry • Alternate resource Workflow Level • User-defined exception handling	User-directed
Kepler	User-defined*	Task Level • Alternative resource Workflow Level • User-defined exception handling • Workflow rescue	Centralized Mediated Peer-to-Peer

***user-defined -** the architecture of the system has been explicitly designed for user extension.

2.3.1 Condor DAGMan

Condor [101][158][163] is a specialized Resource Management System (RMS) developed at the University of Wisconsin-Madison for compute-intensive jobs. Condor provides a High Throughput Computing (HTC) environment based on large collections of distributed computing resources ranging from desktop workstations to super computers. Condor-G, a component within Condor, utilizes Globus GRAM serving as a uniform interface to heterogeneous batch systems, thus enabling large scale computational Grids. *Matchmaking* within Condor, matches jobs and available resources according to their job and resource classified advertisement. When more than one resource satisfies the job requirement, the resource with higher value of rank expression, which expresses the desirability of a match, is preferred.

The Directed Acyclic Graph Manager (DAGMan) [45][158] is a meta-scheduler for Condor jobs. While Condor aims to discover available machines for the execution of jobs, DAGMan handles the dependencies between the jobs. DAGMan uses DAG as the data structure to represent job dependencies. Each job is a node in the graph and the edges identify their dependencies. Each node can have any number of "parent" or "children" nodes. Children cannot run until their parents have completed. Cycles, where two jobs are both descended from one another, are prohibited, because it would lead to deadlock. DAGMan does not support automatic intermediate data movement, so users have to specify data movement transfer through pre-processing and post-processing commands associated with processing job.

The individual job execution is managed by Condor scheduler. Hence if a job fails due to the nature of the distributed system, such as loss of network connection, it will be recovered by Condor while DAGMan is unaware of such failures. However, DAGMan is responsible for reporting errors for the set of submitted jobs, and generates a rescue DAG. In the case of a job failure, the remainder of the DAG continues until no more progress can be made. A failed node can be retried a configurable number of times. The rescue DAG indicates the uncompleted portions of the DAG with detail of failures. Users can correct the errors of failed jobs and resubmit the rescue DAG.

2.3.2 Pegasus in GriPhyN

GriPhyN [75] aims to support large-scale data management in physics experiments such as high-energy physics, astronomy, and gravitational wave physics. Pegasus [50][51][52] (Planning for Execution in Grids) is a workflow manager in GriPhyN developed by the University of Southern California.

Pegasus performs a mapping from an abstract workflow to the set of available Grid resources, and generates an executable workflow. An abstract workflow can be constructed by querying Chimera [67], a virtual data system, or provided by users in DAX (DAG XML description). An abstract workflow describes the computation in terms of logical files and logical application components and indicates their depend-

encies in the form of Directed Acyclic Graph (DAG). Before mapping, Pegasus reduces the abstract workflow by reusing a materialized dataset which is produced by other users within a VO. Reduction optimization assumes that it is more costly to produce a dataset than access the processing results. The reduction algorithm removes any antecedents of the redundant jobs that do not have any unmaterialized descendents in order to reduce the complexity of the executable workflow.

Pegasus consults various Grid information services to find the resources, software, and data that are used in the workflow. A Replica Location Service (RLS) [37] and Transformation Catalog (TC) [49] are used to locate the replicas of the required data, and to find the location of the logical application components respectively. Pegasus also queries Globus Monitoring and Discovery Service (MDS) [41] to find available resources and their characteristics.

There are two methods used in Pegasus for resource selection, one is through random allocation, the other is through a performance prediction approach. In the latter approach, Pegasus interacts with Prophesy [86][187], which serves as an infrastructure for performance analysis and modeling of parallel and distributed applications. Prophesy is used to predict the best site to execute an application component by using performance historical data. Prophesy gathers and stores the performance data of every application. The performance information can provide insight into the performance relationship between the application and hardware and between the application, compilers, and run-time systems. An analytical model is produced based on the performance data and is used by the prediction engine to predict the performance of the application on different platforms. It is required that Pegasus send the request associated with information such as the component name, the semantic parameter names and their values, and the list of available resources. The ranking of the given resources is returned by Prophesy after the query is received.

For ease of use, Pegasus is able to generate a workflow from a metadata description of the desired data product with the aid of artificial intelligence planning techniques. Although, the workflow execution of Pegasus is based on static planning and its executable workflow is transformed into Condor jobs for execution manage-

ment by Condor DAGMan, it has been recently extended to support just in-time scheduling [52] and pluggable task scheduling strategies.

2.3.3 Triana

Triana [160][161] is a visual workflow-oriented data analysis environment developed at Cardiff University. In 2002, Triana was extended to implement a consumer Grid [160] by using a peer-to-peer approach. Recently, Triana has been redesigned and integrated with Grids via GridLab GAT (Grid Application Toolkit) interface [12]. GAT defines a high level API for core Grid service access through JXTA [88], Web services [177], and OGSA [68][168].

Triana provides a visual programming interface with functionality represented by units. Applications are written by dragging the required units onto the workplace and connecting them to construct a workflow. Apart from many implemented tool units, Triana also provides a custom user interface to allow users to build their own units. Several control units (e.g. loop) and logic units (e.g. if) are also provided for users to control the logic of workflow execution. Since control and logic units are implemented as a standard Triana unit, it is easy to introduce new flow patterns. Interconnected units can also be grouped into a group unit, which has the same properties as normal unit.

Triana clients such as Triana GUI can log into a Triana Controlling Service (TCS), remotely build and run a workflow and then visualize the result on their device (e.g. PC, PDA, etc). Each TCS interacts with the Triana engine and every engine provides a service and is capable of executing complete or partial task-graphs locally, or by distributing the code to other servers based on the specified distribution policy for the supplied task-graph. The distribution policy is based on the concept of group units and two distribution policies have been implemented, namely parallel and peer-to-peer. Both policies distribute every unit in the group to separated hosts, however while the peer-to-peer mechanism relies on intermediate data being passed between hosts, there is no such host-based communication with the parallel policy. Since a distributed

task-graph is not fixed to a specific set of resources, it can be dynamically allocated to available services in the most effective way.

2.3.4 Workflow Management in ICENI

The ICENI (Imperial College e-Science Network Infrastructure) [116][117] developed at London e-Science Centre provides component-based Grid middleware. Within ICENI, users construct an abstract workflow, which is a collection of components, and then submit this to ICENI environment for execution.

Each ICENI component is described in terms of meaning, control flow and implementation. The workflow components are primarily composed based on a spatial view, in which all units are represented concurrently, with details of how they relate and interact with each other. Then a temporal view is derived from the spatial view by the system. In the temporal view, workflow information is attached to each component that consists of a graph in which the directed arcs contain the partnership according to the temporal dependence. Within ICENI, the workflow model is similar to that of the YAWL (Yet Another Workflow Language) [4], although simplified in certain respects. The workflow language includes all basic workflow structure such as sequence, parallelism, choice and iteration.

The scheduling service [116][117] within ICENI is responsible for concretizing the abstract workflow. The scheduling task includes matching component meaning with component implementation and mapping these qualified components onto a suitable subset of the available resources. Several scheduling algorithms used to determine resource mapping have been implemented. They include random, best of *n* random, simulated annealing and game theory. Most schedulers implemented within ICENI aim to provide approximate optimal solutions to map the abstract workflow to a combination of component implementations and resources in terms of execution time and cost. The schedulers take into account all components in applications rather than standalone components. The scheduling framework also allows third-party scheduling algorithms to be plugged in.

ICENI has developed a performance repository system [114] which is able to monitor running applications and obtain and store performance data for the components within the applications. This data is stored within a repository with meta-data about the resource the component was executed on, the implementation of the component used, and the number of other components concurrently running on the same resource. This data can be used by schedulers for future runs of applications to estimate the execution times of each component within the workflow.

Two scheduling schemes [116] are considered within ICENI, namely lazy scheduling and advanced reservation. The metadata of the component implementation indicates which scheme the component can benefit from. Non-reservation component is scheduled to a resource just before it is required, while reservation component has been allocated to a resource and has made a reservation in advance. The schedulers can interrogate the performance repository to predict execution in order to produce accurate reservation. The reservation negotiation protocol is based on WS-Agreement [74].

2.3.5 Taverna in myGrid

Taverna [125] is the workflow management system of the myGrid [154] project, which aims to exploit Grid technology to develop high-level middleware for supporting personalized in silico experiments in biology. Taverna is a collaboration among several European universities, institutes and industries. The purpose of Taverna is used to assist scientists with the development and execution of bioinformatics workflows on the Grid. Taverna provides data models, enactor task extensions, and graphical user interfaces. FreeFluo [125] is also integrated into Taverna as a workflow enactment engine to transfer intermediate data and invoke services.

In Taverna, data models can be represented in either a graphical format or in an XML based language called Simple Conceptual Unified Flow Language (SCUFL). The data model consists of inputs, outputs, processors, data flow and control flow. In addition to specifying execution order, the control flow can also be trigged by state transitions during the execution of parent processors. Compared to other workflow

languages, such as the Business Process Execution Language for Web Services (BPEL4WS) [15] , SCUFL allows implicit iteration over incoming data sets based on a specified strategy. At the execution level, the workflow enactor also provides a multithreading mechanism to speed up the iteration process. Users are allowed to set the *Thread* property to specify how many concurrent instances will send parallel requests to the iteration processor. It is especially suitable for services that are capable of handling significant simultaneous processing, for example, a service that is backed by a cluster. It also can reduce service waiting time since workflow engine can send the next input data at the same time as the service is working on the current input.

Taverna also provides a user-friendly multi-window environment for users to manipulate workflows, validate and select available resources, and then execute and monitor these workflows. The enactment status panel [159] of Taverna shows the current progress of a workflow invocation. It also allows the users to browse the intermediate and final results. Through the enactment panel, users can handle storage of those results on local or remote data stores in a variety of formats.

Fault tolerance [159] in the workflow management of myGrid is achieved by setting configuration for each processor in the workflow, for example, the number of retries, time delay and alternate processors. It also allows users to specify the critical level for faults on each processor. If the processor is set as *Critical*, after all retries and alternates have failed, entire workflow execution will be terminated, otherwise, the workflow will continue but children nodes of the failed processor will never be invoked.

myGrid follows service-oriented grid architecture and supports several different types of services within the workflow management system, including WSDL-based [185] single operation web services, soaplab bio-services [142] and local services such as programs coded as java classes. In addition, information services such as UDDI (the Universal Description, Discovery and Integration) [169] and ontology directory [186] are adopted for service discovery.

2.3.6 GridAnt

The GridAnt [95] is an extensible client-side workflow management system developed by Argonne National Laboratory. It has been designed for Grid end-users as a convenient tool to express and control the execution sequence without having any expertise in sophisticated workflow systems. GridAnt focuses on distributed process management rather than the aggregation of services which is the concern of most other Grid-enabled workflow frameworks.

GridAnt consists of four major components, namely workflow engine, run-time environment, workflow vocabulary and workflow monitoring. The workflow engine is the central controller that handles task dependencies, failure recoveries, performance analysis, and process synchronization. GridAnt workflow engine extends Ant [16], an existing commodity tool for controlling build process in Java, by adding additional components to support workflow orchestration and composition. GridAnt also provides an environment for inter-task communication, so that individual GridAnt tasks can read and write intermediate data by using a globally accessible whiteboard-style communication model. Several important constructs such as constants, arithmetic expressions, global variables, array references, and literals are supported by the run-time environment. GridAnt extends Ant's vocabulary in the Grid domain with the addition of the tags such as grid-copy, grid-authenticate and grid-query. These new tags are used by users to predefine the Grid tasks and construct complex workflows at compile time. It uses a control construct provided by Ant container for expressing parallel and sequential tasks. Furthermore, users are allowed to monitor the progress of the execution by means of graphical visualization tool.

In addition to mapping complex client-side workflows, GridAnt can be used for testing the functionality of different Grid services. It has been developed to support version 2 and version 3 of the Globus toolkit [69] by using the Java CoG kit [94]. It has been applied for Position-Resolved Diffraction [14], which is a new experimental technique for the study of nanoscale structures as part of the Argonne National Laboratory's advanced analytical electron microscope.

2.3.7 Workflow Management in GrADS

The Grid Application Development Software (GrADS) project [19] aims to provide programming tools and execution environments for ordinary scientific users to develop, execute, and tune applications on the Grid. GrADS is a collaboration between several American Universities. GrADS supports application development either by assembling domain-specific components from a high-level toolkit or by creating a module by relatively low-level (e.g., MPI) code [39].

GrADS provides application-level scheduling to map workflow application tasks to a set of resources. New Grid scheduling and rescheduling methods [39] are introduced in GrADS. These scheduling methods are guided by an objective function to minimize the overall job completion time (*makespan*) of the workflow application. The scheduler obtains resource information by using services such as MDS [139] and NWS [183] and locates necessary software on the scheduled node by querying GrADS Information Service (GIS). The workflow scheduler ranks each qualified resource for each application component. A rank value is calculated by using "a weighted sum of the expected execution time on the resource and the expected cost of data movement for the component." After ranking, a performance matrix is constructed and used by the scheduling heuristics to obtain a mapping of components onto resources. Three heuristics have been applied in GrADS; those are Min-Min, Max-Min, and Sufferage heuristics [108].

GrADS has built up an architecture-independent model of the workflow component from individual component models. It employs analytical models that are constructed semi-automatically from empirical models (historical data/sample execution data), in order to estimate the performance of a workflow component on a single Grid node. It uses hardware performance counters to collect operation counts from several executions of the workflow components with different, small-size input problems, and then it performs a least-squares fit to the data to construct computational models. In addition, GrADS reuses distance data on small inputs to predict the faction of cache hits and misses on the given data and cache configuration by its memory-hierarchy performance models.

GrADS utilizes Autopilot [135] to monitor performance of the agreement between the application demands and resource capabilities. Once the contract is violated, the rescheduler [39] of the GrADS takes corrective actions. It has been implemented using two rescheduling approaches for MPI applications, the stop/restart approach and process swapping. In the former approach, an executing application component is suspended and migrated to a new resource if better resources are found for improving the execution performance [172]. As a migration event can involve large data transfers, expensive startup costs and significant application code modifications, process swapping provides a lightweight, but less flexible, alternative approach. In process swapping more machines than will actually be used for the computation are launched for an MPI application component, and slower machines in the active set are swapped with faster machines in the inactive set periodically, according to the performance of machines.

2.3.8 GridFlow

GridFlow [31] is a Grid workflow management system developed at the University of Warwick. This work is built on the top of an Agent-based Resource Management System for Grid computing (ARMS) [30]. Rather than focusing on workflow specification and the communication protocol, GridFlow is more concerned about service-level scheduling and workflow management.

There are three layers of Grid resource management within the GridFlow system: the Grid resource, the local Grid and the global Grid. A Grid resource is simply just a particular Grid resource; local Grid consists of multiple Grid resources that belong to one organization; and a global Grid consists of all local Grids. Global Grid also provides a portal for compose the workflow.

A workflow in GridFlow is represented as a flow of several different activities, each activity represented by a sub-workflow. Each sub-workflow is a flow of closely related tasks that is to be executed in a local Grid. A portal has been developed by GridFlow as graphical user interface for users to compose workflow elements.

The workflow management within GridFlow is conducted by a hierarchical scheduling system including global Grid workflow manager and local Grid sub-workflow scheduling. Global Grid workflow manager receives requests from the GridFlow portal with the workflow description in the format of XML, and then simulates workflow execution to find a near-optimal schedule. After the users accept the simulated result, GridFlow schedules the workflow onto different local Grids through ARMS. Within ARMS, each agent represents a local Grid at a global level of Grid resource management, and conducts local Grid sub-workflow scheduling. In contrast to the global Grid workflow management, the local Grid schedulers handle conflicts since scheduled sub-workflows may belong to different workflows.

ARMS has integrated Titan [152], which utilizes performance data obtained from PACE [123], a toolset for resource performance and usage analysis, with iterative heuristic algorithms to minimize the makespan and idle time of a Grid resource. PACE can exact control flow, and use an analytical model approach based on queuing theory, to predict application performance on a given set of resources such as time, scalability and system resource usage. Titan also provides Grid resource information.

2.3.9 Workflow Management in Unicore Plus

Unicore plus [171] provides seamless and secure access to distributed resources of the German high performance computing centers. Unicore plus is a follow-on project of Unicore (Uniform Interface to Computing Resources) [13], started in 1997 to improve uniform interfaces to distributed High Performance Computing and data resources using the mechanisms of the World Wide Web. Unicore plus provides a programming environment for users to design and execute job flow.

Within Unicore, one job or job group that can be executed on any Unicore site may contains other jobs and/or job groups. The original Unicore job model supports jobs that are constructed as a set of directed acyclic graphs with temporal dependencies. Since Unicore version 4, advanced flow controls have been added, which include conditional execution (e.g. if-then-else), repeated execution (e.g. do-*n*), conditional repeated execution (e.g. do-repeat), and conditional suspend action (e.g. hold-job). In

addition, three types of run-time conditions are implemented for supporting conditional checking; these are based on the return code of a previous executed task, existence or properties of a file and whether a given time and date have passed.

Unicore plus provides graphical tools that allow users to create a job flow and convert it into an Abstract Job Object (AJO) which is a serialized java object. The AJO is submitted from a user client to a Unicore server. The server translates the job specification into a number of batch jobs and dispatches them to the target resource. The server also makes sure that a successor is executed if its predecessors are finished and all necessary data is available at the executing site.

Unicore allows users to specify jobs and different parts of job group onto multiple resources. The output of individual jobs may be needed by its successors. Therefore, a temporary Unicore space is created for each job group for transferring data sets. Unicore also allows users to explicitly specify the transfer function as a task through GUI; it is also able to perform the necessary data movement function without user intervention.

2.3.10 Askalon

Askalon [60] is a Grid application development and computing environment developed by the University of Innsbruck, Austria. The main objective of Askalon is to simplify the development and optimization of mostly Grid workflow applications that can harness the power of Grid computing.

Askalon comes with two separate composition systems, AGWL (Abstract Grid Workflow Language) [58] and Teuta [59], that support the development of Grid workflow applications. AGWL is an XML-based language. It provides a rich set of constructs to express sequence, parallelism, choice, and iteration workflow structure. In addition, programmers can specify high-level constraints and properties defined over functional and non-functional parameters for tasks and their dependencies which can be useful for a runtime system to optimize the workflow execution. Teuta supports the graphical specification of Grid workflow applications based on the UML activity diagram which is a graphical interface to AGWL.

Askalon provides a new hybrid approach for scheduling workflow applications on the Grid through dynamic monitoring and steering combined with a static optimization. Static scheduling maps entire workflows onto the Grid using genetic algorithms based on user-defined QoS parameter. A dynamic scheduling algorithm takes into consideration the dynamic nature of the Grid resources such as machine crashes or external CPU and network load. Performance contracts are defined for every task and monitor whether tasks execute properly or whether they should be migrated. Askalon also develops a fault tolerant execution engine that supports reliable workflow execution in the presence of resource failures through checkpointing and migration techniques.

In Askalon, the performance of workflow components is estimated based on a training phase which measures the actual execution time of tasks for different loads and problem sizes on a variety of Grid sites. The performance estimation of the workflow is conducted based on a combination of historical data obtained from a training phase and analytical modeling.

2.3.11 Karajan

Karajan [96][97], developed by Argonne National Laboratory, aims to provide an integrated approach of exposing workflow to the Grid community. It is an extensible workflow framework and can be easily utilized by third parties to provide workflow solutions for a variety of users. It is derived from GridAnt and provides additional capabilities such as scalability, workflow structure and error handling.

Karajan is part of Java CoG Kit. Java CoG Kit is based on modular design and provides mechanisms for fast application development and easy integration of the variety of Grid middleware. It provides a number of programming abstractions for job executions and file transfers. The concept of *Grid providers* is introduced to facilitate different middleware to be used as part of an instantiation of Grid abstractions. As a result, it is easy to integrate Karajan to any middleware. To date, it has been integrated into various versions of Globus, Condor, runtime exec, ssh, and some data transfer techniques such as WebDAV [38] and SCP. Karajan leverages lower-level program-

ming abstractions in Java CoG Kit to access the Grid, and at the same time it provides programming interfaces for higher level applications such as workflow schedulers and application portlets to develop users' strategies.

In addition to sequence and parallelism, Karajan supports choices and loops of workflow structures. It also provides a user friendly XML-based workflow language. Elements used for the description of workflow tasks are user-definable. Thus, the user can define names and parameters along with annotations and descriptions for a new element. A number of standard operators including mathematical and boolean operators are defined for integration within execution control statements. It also provides advanced data structures such as list, range, and map (or hash tables) for repetitive tasks (e.g. parameter studies) as part of the workflow.

A number of fault handling methods are supported in Karajan. *Error handling* allows users to integrate strategies for errors and exceptions into the workflow. Checkpointing enables users to store intermediate states of the workflow execution for later roll back when a problem occurs.

2.3.12 Kepler

Kepler [14][105] is one of the popular workflow systems with advanced features for composing scientific applications. It is derived from Ptolemy II system [102] and currently under development across a number of scientific data management projects. In addition to a user-friendly graphical user-interface and an extendable open source platform, Kepler also inherits the actor-oriented feature from Ptolemy II. It models a workflow system as a composition of independent components (actors) that communicate through well-defined interfaces. An actor is an encapsulation of parameterized operations performed on input to produce output data. An execution model of a workflow, which can be defined in a *director* object, imposes an execution order and communication mechanisms on the usable actors of the workflow. This modular design approach allows different execution models or machineries to be implemented and easily plugged into workflows without changing any of the components of workflows.

Kepler has been extended to support seamless access to remote resources and services. A web service HARVESTER component can retrieve all service description files in a web page or service repository to create instantiations of web services actors in the user's local actor library. Each web services actor can be instantiated for any particular operation specified in its service description. A number of fault-tolerant methods have been developed to make workflows with web services more reliable. Instead of associating a service operation with a fixed URL, a list of services is allowed to provide the alternative invocation during service failure. It is also able to produce partial results even when the entire workflow fails. Advanced failure handling can also be supported through extensions of exception-catching actors. In addition, Kepler has defined a set of Grid actors for access authentication, file copy, job execution, job monitoring, execution reporting, storage access, data discovery, and service discovery.

2.4 Summary

A taxonomy for Grid workflow management systems has been presented in this chapter. The taxonomy focuses on topics including workflow design, workflow scheduling, fault management and data movement. Several workflow management systems for Grid computing have been surveyed and classified into different categories using the taxonomy. This chapter thus helps to understand key workflow management approaches and current state of Grid workflow systems.

Many Grid workflow-enabled systems have developed graph-based editing environments. They allow users to compose a workflow by dragging and dropping components on a composition panel. A workflow abstract specification or concrete specification is then generated by these visual tools and passed to the workflow enactment engine. These processes are transparent to users to aid usability. In addition, a number of web-services based workflow languages have been developed and supported in the workflow systems.

Several performance information services are utilized in the Grid workflow projects to predict performance prediction to optimize workflow execution. Some simple

fault handling techniques such as retry and alternative resources have been developed and implemented in most systems.

However, Quality of Service (QoS) issues have not been addressed very well in most Grid workflow management systems due to their focus on the use of system centric policies in resource allocation. When workflow management systems are used in commercial or production environments, supporting QoS at both specification and execution level becomes increasingly critical. At the specification level, workflow languages need to allow users to express their QoS requirements. At the execution level, the workflow system must be able to negotiate with resource providers and map the workflow onto Grid resources to meet users' QoS requirements. Therefore, the requirement for QoS based workflow specification and execution is increasingly important, though it is currently ignored by most existing Grid workflow management systems.

In the following chapters, a workflow execution engine and several QoS-based workflow scheduling algorithms are presented.

Chapter 3

Workflow Enactment Engine

With the advent of Grid technologies, scientists and engineers are building more and more complex applications to manage and process large data sets, and execute scientific experiments on distributed Grid resources. Building complex workflows requires means for composing and executing distributed applications. Even though efforts have been made by several projects as described in Chapter 2, capabilities such as support for parameterization and run-time data-driven mechanism are required by many application domains.

This chapter presents a novel workflow enactment engine. It first presents the architecture of the system and its event-driven execution management mechanism. Then the details of service discovery, parameterization-enabled workflow language and fault handling are described. The chapter also briefly provides the implementation of the engine and it finishes with a case study in neuroscience applications.

3.1 Workflow Engine

Executing a Grid workflow is a complex endeavor. Workflow tasks are expected to be executed on heterogeneous resources which may be geographically distributed. Different resources may be involved in the execution of one workflow. For example, in a scientific experiment, one needs to acquire data from an instrument, and analyze it on resources owned by other organizations, in sequence or in parallel with other tasks.

Therefore, discovery and selection of resources for executing workflow tasks could be quite complicated. In addition, a large number of tasks may be required to be executed and monitored in parallel and the location of intermediate data may be known only at run-time.

Given the complexity of Grid workflow execution environments, a decentralized architecture is developed in the Workflow Enactment Engine (WFEE) to support various middleware access and resource allocation strategies. The details of the architecture are presented in the following subsections.

3.1.1 Entities

The primary components of WFEE and their relationship with other services in the Grid infrastructure are shown in Figure 3.1. Workflow applications, such as scientific application portals, submit task definitions along with their dependencies, expressed in a workflow language, as well as associated QoS requirements to WFEE. WFEE schedules tasks through Grid middleware on the Grid resources.

The key components of WFEE are: workflow submission, workflow language parser, resource discovery, dispatcher, data movement and workflow executor.

- Workflow submission accepts workflow enactment requests from planner level applications.
- Workflow language parser converts workflow description from XML format into Java objects, *Task*, *Parameter* and *DataConstraint* (workflow dependency) which can be accessed by workflow scheduler.
- Resource discovery is carried out by querying Grid information services such as Globus MDS [69], directory service and replica catalogs, to locate suitable resources for the tasks.
- Dispatcher is used to access middleware. Resources may be Grid-enabled by different middleware such as Globus [69] or Web services [15]. WFEE had been designed to support different middleware by creating dispatchers for each middleware to support interaction with resources.
- Data movement system enables data transfer between Grid nodes by using HTTP

and GridFTP [11] protocols.

- Workflow executor is the central component in WFEE. It interacts with *resource discovery* to find suitable Grid resources at run time; it locates a task on resources by using the dispatcher component; it controls input data transfer between task execution nodes through *data movement*.

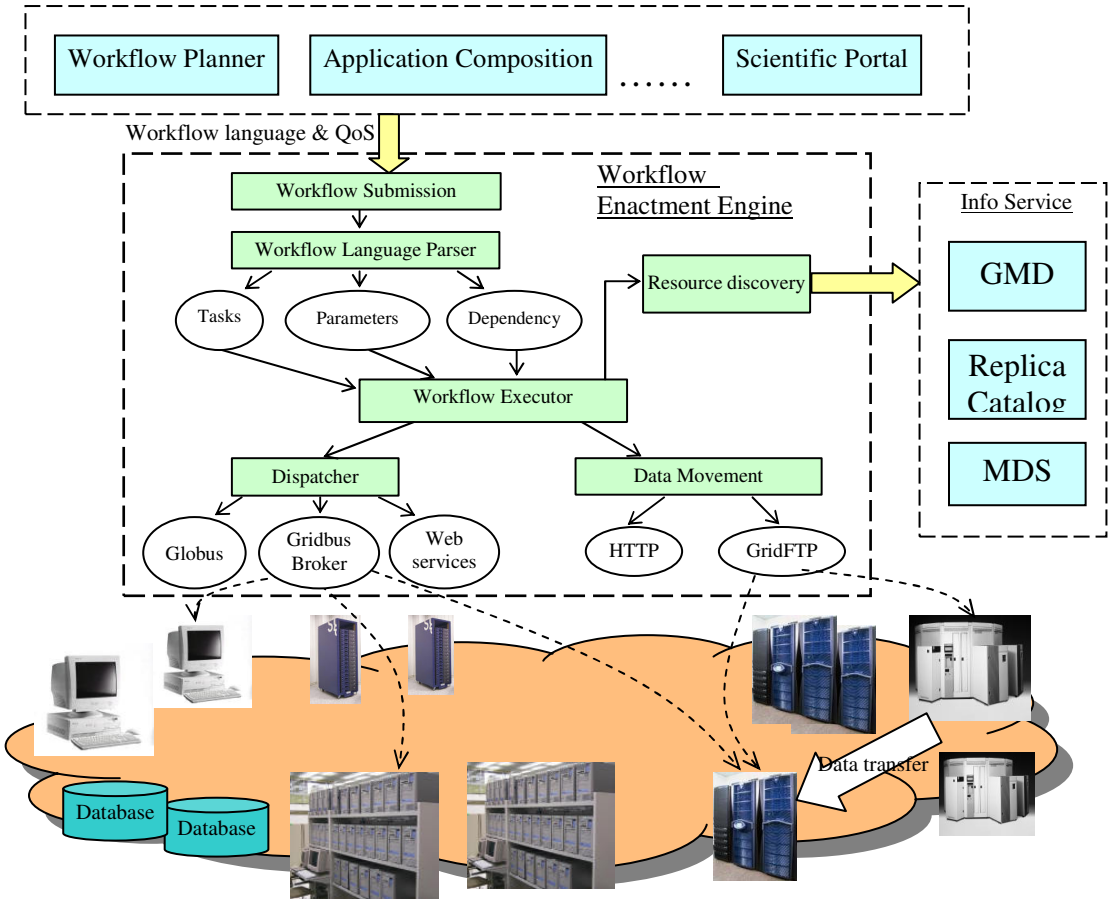


Figure 3.1: Architecture of WFEE.

3.1.2 Workflow Execution Management

The workflow execution is managed using a decentralized architecture. Instead of a central scheduler for handling whole workflow execution, a task manager is created for handling the processing of a task or a group of tasks, including resource discovery and allocation, task dispatcher and failure processing. Different scheduling strategies

can be deployed in different Task Managers (TMs) for resource selection, QoS negotiation and data transmission optimization. The lifetimes of TMs, as well as the whole workflow execution, are controlled by a Workflow Coordinator (WCO).

As shown in Figure 3.2, dedicated TMs are created by WCO for each task group. Each TM has its own monitor which is responsible for monitoring the health of the task execution on the remote node. Every TM maintains a resource group which is a set of resources that provides services required for the execution of an assigned task. TMs and WCO communicate through an Event Service Server (ESS).

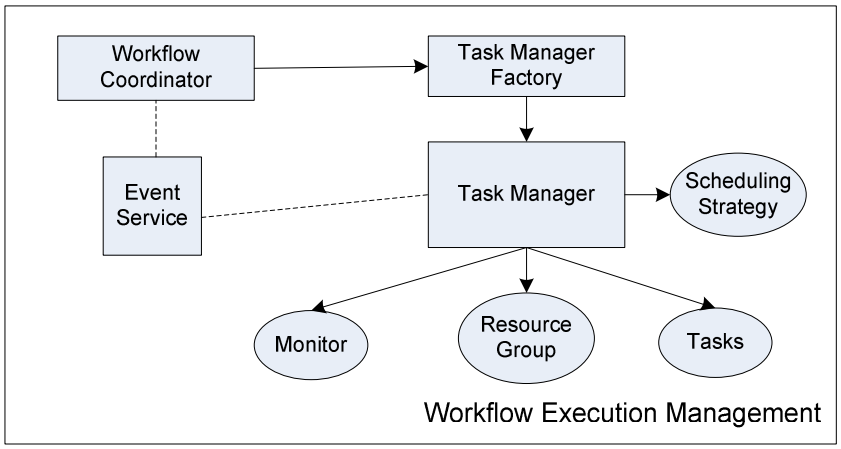


Figure 3.2: Execution management.

3.1.3 Communication Approach

A communication approach is needed for task managers. On one hand, every task manager is an independent thread of execution and they can be run in parallel. On the other hand, the behavior of each task manager may depend on the processing status of other task managers according to the task dependencies. For example, a task manager should not execute the task on a remote node if the input generated by its parent tasks is not available for any reason.

In addition, in a workflow, a task may have more than one input that comes from different tasks. Furthermore, the output of these tasks may also be required by other task managers as well. Hence the communication model between the task managers is

not just one-to-one or one-to-many, but it could be many-to-many depending on task dependencies of the workflow.

Given this motivation, an event-driven mechanism with subscription-notification model has been developed to control and manage execution activities. In the system, the behaviors of task managers and workflow coordinator are driven by events. A task manager is not required to handle communication with others and only generates events according to a task's processing status. At the same time, the task managers take actions only depending on the events occurred without concern for details of other task managers. The benefit of this event-driven mechanism is that it provides loosely-coupled control; hence the design and development of the system is very flexible and additional components can be easily plugged in.

.

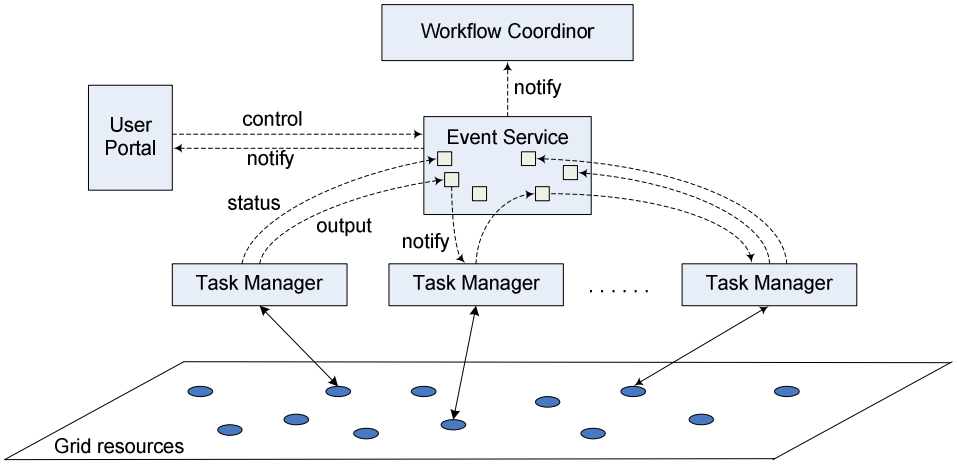


Figure 3.3: Event-driven mechanism.

The event notification is based on subscription-notification model. WCO and TMs just subscribe to events of interest after activation, and then are informed immediately when a subscribed event occurs. There are three basic types of events, *status events*, *output events* and *control events*. Status events are sent by the TMs to provide information on the status of task execution. Output events are sent by TMs to announce the task output is ready along with the location of its storage. Control events

are used to send control messages, such as to *pause* and *resume* the execution, to task managers

As illustrated in Figure 3.3, TMs inform each other and communicate with the WCO through the ESS. For example, TMs put their task execution status (e.g. executing, done, failure) into the ESS, which notifies the WCO. If the output of a task is required by its child tasks, the task managers of the child tasks can subscribe to output events of the task. Once the task generates the required output, an output event is sent to the ESS, which notifies immediately, the child TMs that have subscripted to the output event. A user can control and monitor the workflow execution by subscribing to status events and sending control evens through a visual user interface.

3.1.4 State Transition

The state transition of WCO is illustrated in Figure 3.4. WCO registers with the ESS and start TMs of first level tasks, and then monitors activated TMs. Upon receiving execution status from a TM, WCO starts the TMs of its child tasks. If the WCO receives a status *done* event, it checks whether other TMs are still running. If so, WCO goes back to *monitoring*, otherwise it exits. If WCO receives a failed event from a TM, it proceeds to *failure processing*, and then ends.

The state transition of TMs is illustrated in Figure 3.5. The TM registers events, such as output events, status events, generated by its parent tasks and waits for the events to occur; when an event occurs, the TM goes to the *event processing* state. If all input data is available, it starts a new thread to process execution for a job; otherwise, the TM goes back to *wait* state. A job is a unit of work that a TM sends to a Grid node and one task may create more than one jobs. The job execution is started from *resource matching*, in which a suitable resource is selected from the resource group created by querying a directory service (see Section 3.3). If a suitable resource is available, the TM submits a job to the resource and then monitors the status of job execution on the remote resource. If the execution has failed, the TM goes back to *resource matching* and selects an alternative resource and then submits the job to it. If all parent tasks and execution jobs are completed, the TM ends.

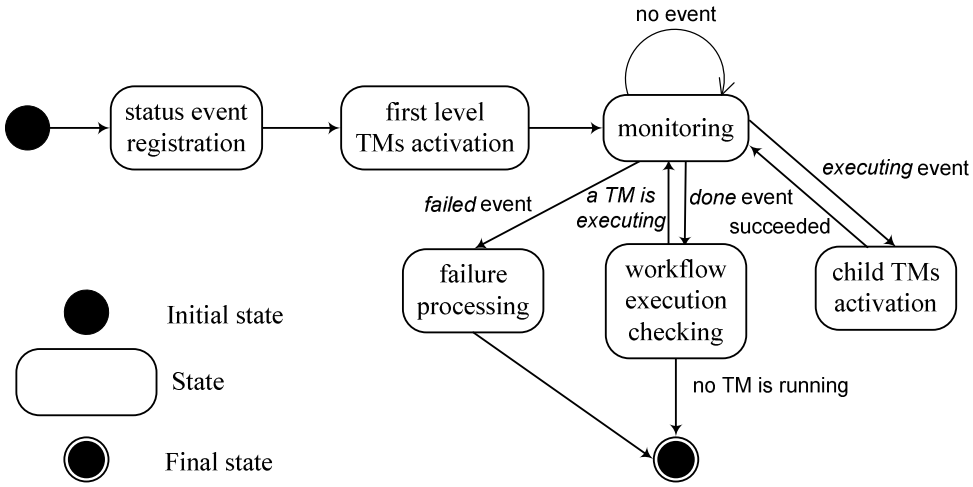


Figure 3.4: State transition of WCO.

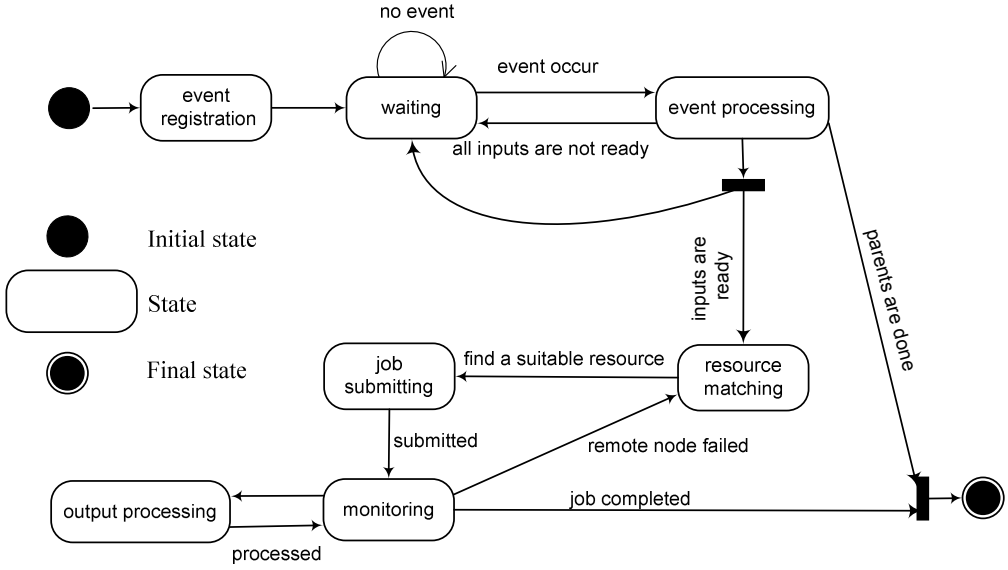


Figure 3.5: State transition of TM.

3.1.5 Interaction

The interactions between the WCO, TMs, ESS and remote resources are illustrated in Figure 3.6. First, the WCO needs to register to the ESS and subscribe to task status events. Then, the WCO activates task managers of first level tasks of which, in this

example, there is only one TM1. After TM1 finishes the preprocessing for the task execution, it sends a message to ESS saying "*I am executing the task*". ESS informs the WCO and WCO activates TMs of the child tasks of TM1, namely TM2 and TM3, in this example.

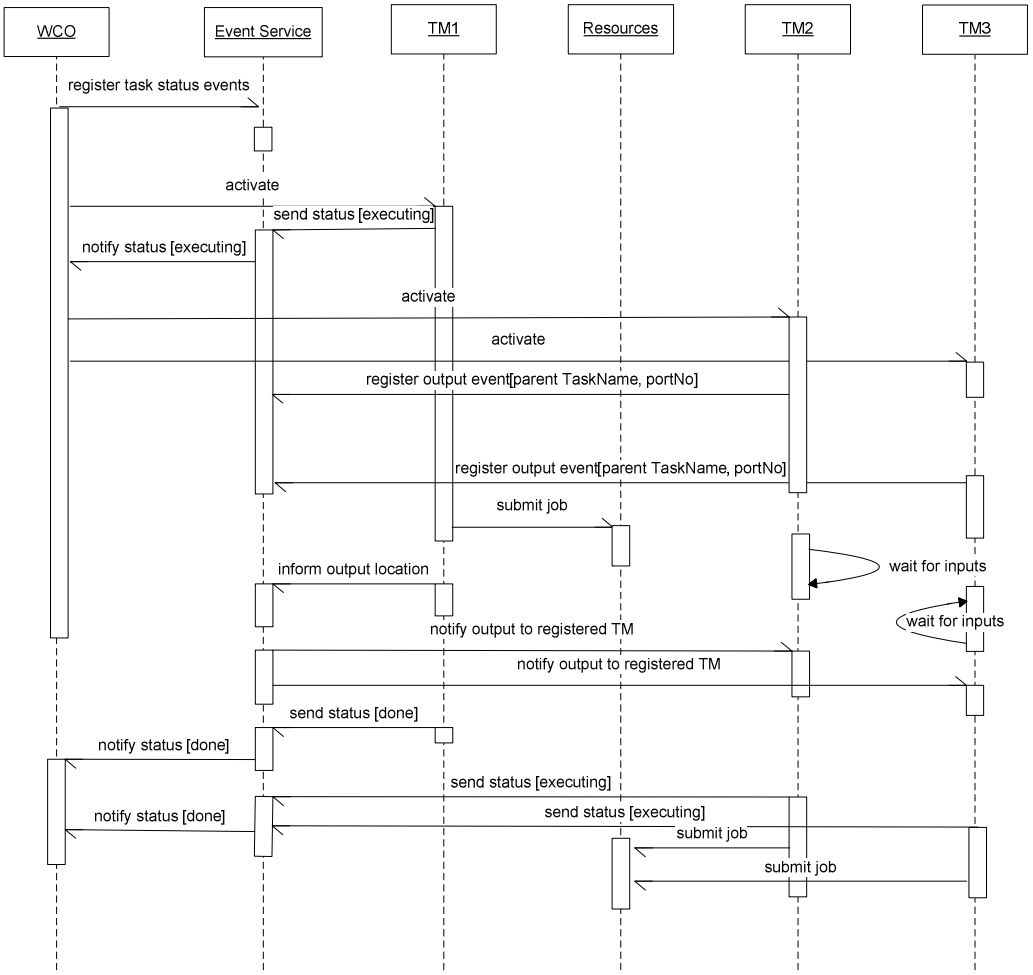


Figure 3.6: Interaction sequence diagram the WCO, TMs and ESS.

The inputs of the task managed by TM2 and TM3 rely on the output of the task of TM1, so TM2 and TM3 register to ESS and listen to its output events. Once TM1 identifies a suitable resource, it submits task to that resource. As soon as TM1 knows the output of the task, it informs TM2 and TM3 through ESS, saying *"my output of*

port No. x is ready and its location is xxxx". If all input data for TM2 and TM3 are ready, TM2 and TM3 reports execution status to ESS, and then proceeds to initialize the execution of their tasks. After WCO receives the notification of the execution of the tasks in TM2 and TM3, WCO will activate their child task managers, so that they can prepare for task execution. This process will be continued until the end of workflow execution.

3.2 Service Discovery

In a Grid environment, many services having same functionality and user interaction, can be provided by different organizations. In addition, a service may be replicated and deployed in many locations. From the user's point of view, it is better to use a service that offers a higher performance at a lower price. Therefore, a method is required to allow users to find replicated services easily.

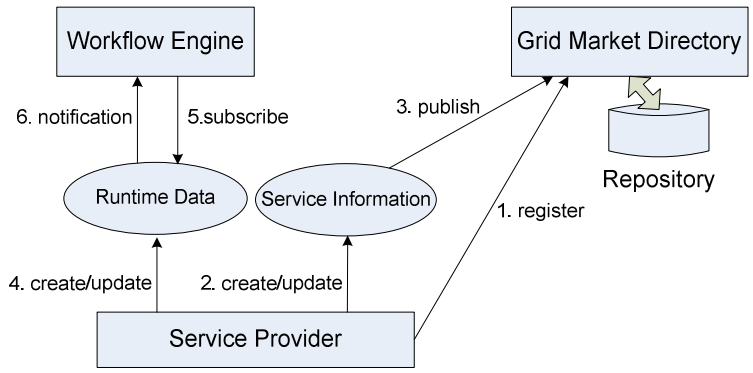


Figure 3.7: Service discovery using GMD.

A directory service called *Grid Market Directory* (GMD) has been developed to support service publishing and discovery. GMD is an infrastructure that allows (a) the creation of one or more registries for service providers; (b) the service providers to register their resources/application services that they wish to provide; (c) users such as workflow engine to discover resources/services and their attributes (e.g., access price, location and usage constraints) that meet their QoS requirements.

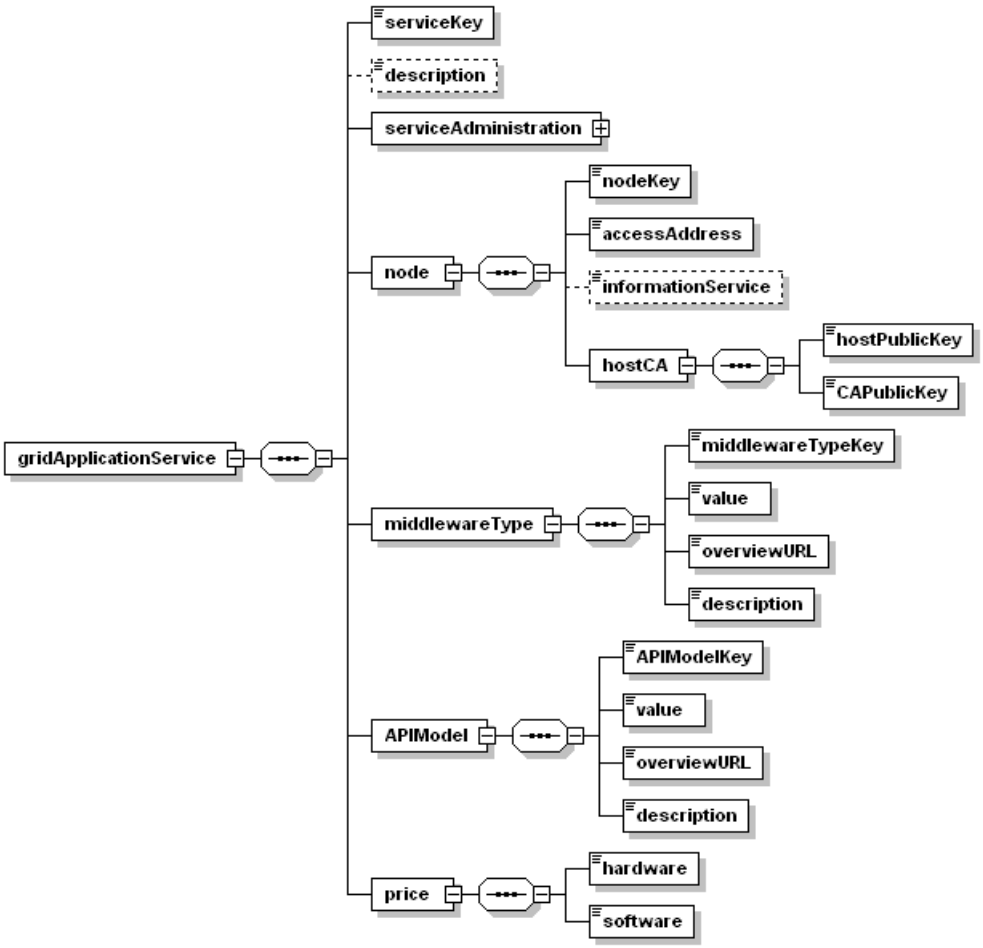


Figure 3.8: Grid application service schema.

Figure 3.7 illustrates service publishing and discovery in a Grid environment through GMD. Service providers first register with the GMD and publish their static information such as location, service capability and access methods. A Grid user such as the workflow engine can query GMD to find a suitable service. After that, the user can also query and subscribe to the service provider directly to obtain more dynamic information such as service execution status.

A provider can provide their specialist applications for others to access remotely. Figure 3.8 shows an application service schema. An application service provider may also provide hosted machine information such as the host name and host public key

for remote secure access. Service providers also need to indicate middleware through which Grid users can access the service.

Grid Application Model (GAM) is developed for application identification. GAM is a set of specifications and APIs for a Grid application. The GAM can be published by service providers within the GMD, and the users can search GMD for services conforming to a particular GAM. The applications with the same GAM name provide the same function and API. In the case of the workflow system, if users do not specify a particular service for a task in the workflow description, the scheduler uses the GAM name associated with the executable of the task to query the GMD. The GMD will return a list of services. These services are all able to execute the task.

3.3 Workflow Language

In order to allow users to describe tasks and their dependencies, a XML-based workflow language (xWFL) has been defined. The workflow language provides the means to build new applications by linking standalone applications.

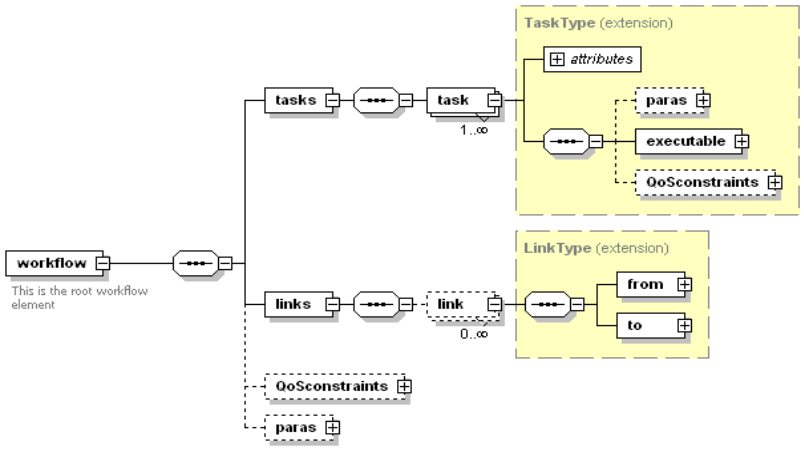


Figure 3.9: Structure of workflow language.

Figure 3.9 shows the basic structure of the workflow language. It consists of two parts: task definitions defined in <*tasks*>, data dependencies defined in <*links*> and QoS constraints defined in <*QoSconstraints*>.

3.3.1 Tasks

The element <*tasks*> is a set of definitions of tasks that are to be executed. Figure 3.10 shows the schema of task definition. A task can be a single task or a parameter sweep task. A parameter sweep task is able to process a set of parameters. The parameters are defined in <*paras*>. The detailed design of parameter tasks is introduced in Section 3.4.3. The element <*executable*> is used to define the information about the application, input and output data of the task. The workflow language supports both abstract and concrete workflows. The user or higher workflow planner can either specify the location of a particular service providing a required application in <*service*> or leave it to the engine to identify their providers dynamically at run-time. The middleware of the application is identified through a service information file by the GMD when dispatching tasks.

In the example that follows, task A executes dock.exe program on host *bellegrid.com* in the directory */services* and the executable *dock* has two input I/O ports[1]: port 0 (a file) and port 1 (a parameter value. The representation of task *A* shown below has a single output port.

```
<task name= "A">
  <executable>
    <name>dock</name>
    <service>
       <hostname="bellegrid.com" />
     <accesspoint value="/services/dock.exe" />
    </service>
    <input>
       <port num=0 type="file" url=http://www.gridbus.org/dock.in
        value="dock.in"/>
      <port num=1 type="msg" value=1/>
    </input>
    <output>
      <port num=2 type="file" value="dock.out"/>
    </output>
  </executable>
</task>
```

[1] Ports play a role of data links to or from a node in the workflow. The data on the link can be standard data items or files.

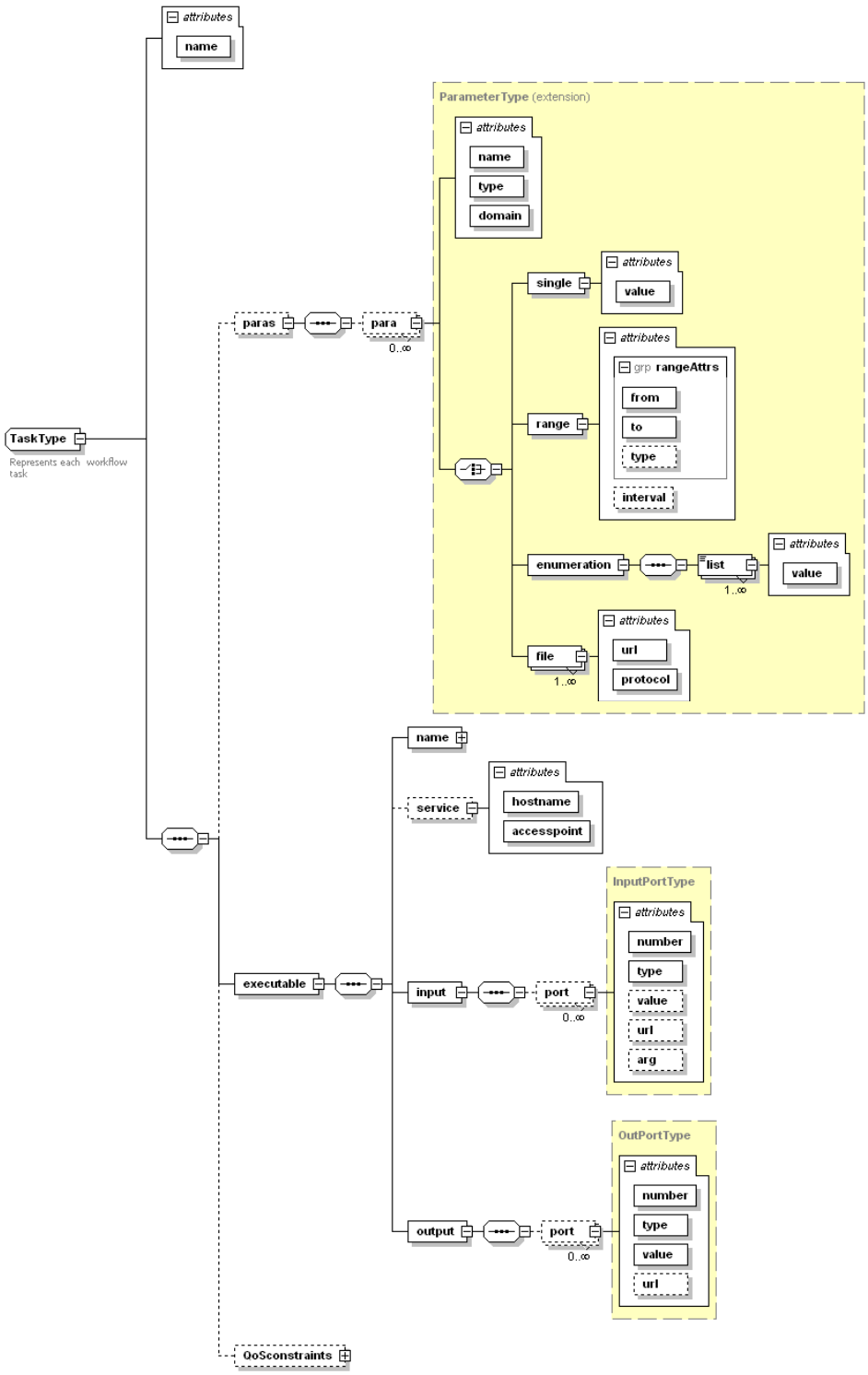


Figure 3.10: Schema of task definition.

3.3.2 Data Dependencies

A data link is used to specify the data flow between two tasks. The schema of the data link is defined in Figure 3.9. Figure 3.11 shows the example of a data flow description. The inputs of task *B* and task *C* rely on the output of task *A*. The output of task *A* needs to be transferred to the node on which tasks *B* and *C* are executed. Input could be a file, parameter value or data stream.

```
<workflow>
  <tasks>
    <task name= "A">
      ….
    </task>
    <task name= "B">
      ……
    </task>
  <task name= "C">
   ……
  </task>
  <task name= "D">
   ……
  </task>
 </tasks>
 <links>
  <link>
   <from task="A" port=2 />
   <to task="B" port=0 />
  </link>
  <link>
   <from task="A" port=2 />
   <to task="C" port=0 />
  </link>
  <link>
   <from task="B" port=1 />
   <to task="D" port=0 />
  </link>
  <link>
   <from task="C" port=2 />
   <to task="D" port=1 />
  </link>
 </links>
</workflow>
```

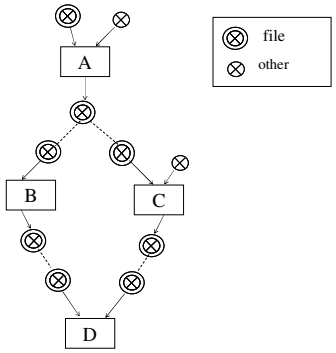


Figure 3.11: Flow diagram of task A, B, C and D.

3.3.3 Parameterization

Supporting parameterization in the workflow language is very important for scientific applications. It enables scientists to perform experiments across a range of different parameters without being concerned about the detailed workflow description. A parameter defined in each task type is called a *local parameter* ; when it is defined for the entire workflow, it is called a *global parameter*. As shown in Figure 3.9, multiple

parameters types such as single, range, file and enumeration are supported. An example for a single parameter type and a range parameter type is given in Figure 3.12.

```
<paras>
  <para type= “single”>
      <name>X</name>
      <value type=integer>10</value>
    </para>
  <para type= “range”>
      <name>Y</name>
      <min>1</min>
      <max>20</max>
      <step>2</step>
  </para>
</paras>
```

Figure 3.12: Single parameter and range parameter.

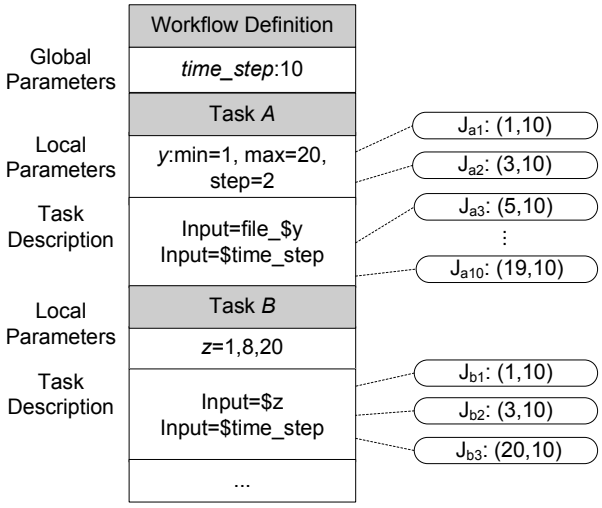


Figure 3.13: Illustration of workflow parameters.

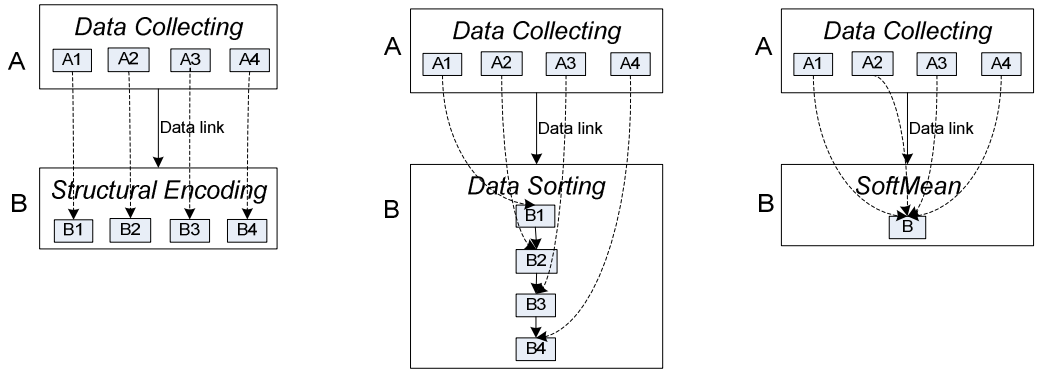
Among these parameter types, range and enumeration types are used to define the range or a list of parameters which the task is required to be executed with. This type of task is called as a *parameter-sweep* task [8] and is structured as a set of multiple execution jobs, each of which is executed with a distinct set of parameters. Figure 3.13 illustrates parameter sweep tasks. There are two parameter sweep tasks: task *A* and task *B*. Each task is required to input a global parameter (named *time_step*) and a local parameter. The local parameter of task *A* is a range type parameter (named *y*) while the local parameter of task *B* (named *z*) is an enumeration type. At execution

time, the global parameter value is combined with each local parameter value and generates 10 sub-jobs ($J_{a1} - J_{a10}$) for task *A* and 3 sub-jobs ($J_{b1} - J_{b3}$) for task *B*.

3.3.4 I/O Models

As shown in Figure 3.11, a task receives output data from its parent as its input through a data link. However, for some tasks, more than one output could be generated by one output port. For example, such a task could be a data collecting task that continues to read information from a sensor device and generate corresponding output data or a parameter-sweep task that generates multiple data based on various sets of parameters. These outputs can be produced at different times. Some successor tasks may not require to be processed until all these output data are generated. It can process the output once it is available. However, ancestor tasks can process the output data generated from a single parent differently, depending on their requirements.

Three I/O models have been developed in the workflow system to provide data-handling capabilities. These models are: *many-to-many*, *many-to-one* and *synchronization*. In Figure 3.13, there are two tasks: task *A* and task *B*. They are connected by a data link. There are multiple sub-jobs in *A* and each job produces an output. For the many-to-many model, task *B* starts to process data and generates an output once there is an input available on the data link. As shown in Figure 3.14a, four outputs generated by four sub-jobs of *A* are processed individually by four sub-jobs of *B*. For the many-to-one model, task *B* starts to process data once there is an input available, however, the result is calculated based on the result generated by earlier sub-jobs. As shown in Figure 3.14b, sub-job *B1* processes the output generated by sub-job *A1*. Once the output of *A2* is available, sub-job *B2* is created and processes the output of *A2* based on the output generated by *B1*. For the synchronization model, task *B* does not start processing until all the output is available on the data link. As shown in Figure 3.14c, there is only one sub-job in task *B* and it processes all outputs generated by sub-jobs of *A* at one time.



a) Many-to-Many b) Many-to-One c) Synchronization

Figure 3.14: Input/output models.

3.4 Fault Handling

Two fault handling mechanism are developed in the system, *retry submission* and *critical task replication*.

The retry submission mechanism basically combines the fault handling methods of retry and alternative resource which are described in Chapter 2. It reschedules a failed job onto a current available resource, and also records the number of failed jobs for each resource. Once the failed job number exceeds a *warning threshold*, the scheduler decreases the number of jobs submitted to these resources. If the number of failed jobs exceeds a *critical threshold*, the scheduler terminates submission of jobs onto this resource.

The critical task replication mechanism replicates a task execution on more than one resource. The result produced earliest is then used for the rest of the workflow. This mechanism is designed to execute a long running task when there are multiple spare resources.

3.4 Implementation

The WFEE has been implemented by leveraging the following key technologies: (1) IBM TSpaces [188] for supporting subscription-notification based event exchange; (2) Gridbus broker for deploying and managing job execution on various middleware; (3)

XML parsing tools including JDOM [87]. The detailed class diagram and event server implementation are presented as follows:

3.4.1 Design Diagram

The class design diagram of the WFEE is shown in Figure 3.15. *XMLParsingToModel* parses XML formatted workflow description into Java objects which are instances of class of *Task*, *Port*, *DataConstraint*. These objects are passed on to *WorkflowModel*. *WorkflowModelToDiGraph* converts *WorkflowModel* into a directed graph represented by class *DiGraph* which encompasses many *GraphNode* objects. An instance of *GraphNode* contains a workflow task and the references of *GraphNodes* of its parent and child tasks. *WorkflowCoordinator* creates and controls the instantiation of *TaskManager* according to the graph node dependencies. *Job* class represents a unit of work assigned to a Grid resource. Every job has a monitor implemented by *JobMonitor* to monitor job execution status on the remote node. In order to extend WFEE to support multiple Grid middleware, we abstract class *Resource* and *Dispatcher* which provides interfaces that interact with Grid resources.

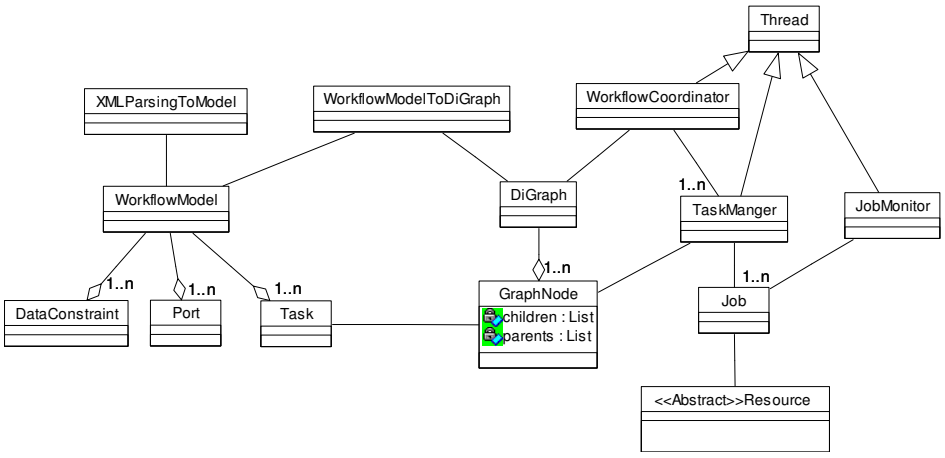


Figure 3.15: Class diagram of WFEE.

3.4.2 Event Messages

We have utilized tuple spaces to exchange events. The tuple space model is originated at Yale with the Linda system [35]. A tuple is simply a vector of typed values (Fields). A tuple space is a collection of tuples that can be shared by multiple parties by using operations such as read, write and delete. In our work, we have leveraged IBM's recent implementation of tuple spaces called TSpaces [188] to be the event server.

Table 3.1: Format of events.

Event name	Field1	Field2	Field3	Tuple template for registration
task status event	task No.	"status"	value	new Tuple(new Field(String), status, new Field(String))
output event	task No.	port No.	value	new Tuple(taskNo, portNo, new Field(String))
job status event	job No.	task No.	value	new Tuple(new Field(String), taskNo, new Field(String))

There are three types of event tuples whose format is shown in Table 3.1, *task status event*, *output event* and *job status event*. Task status event is sent by TMs and WCO use it to control TMs activation. The first field is the ID of the task, the second field is string "status" to indicate the type of tuple for the registration purposes, and the third field gives the value of status.

Child TMs need output events sent by the parent TMs to be informed if their inputs are available. The events have three fields: the task ID is given in the first field, the second field is port numbers, and the third field is the location of output.

One task can have multiple jobs. Job status events are sent by the job monitor. Every job status event provides a job ID and its task ID with status value. TMs make decisions according to the job events. For example, when a job has failed, the TM can reschedule it on another resource in the resource group.

The tuple templates are used for subscribing to the corresponding event. For example, task status events can be received by a tuple template with the second field as a String called "status".

3.5 A Case Study in fMRI Data Analysis

Magnetic Resonance Imaging (MRI) [113] uses radio waves and a strong magnetic field to provide detailed images of internal organs and tissues. Functional MRI (fMRI) [81] is a procedure that uses MRI to measure the signal changes in an active part of the brain. fMRI is becoming a major diagnostic method for learning how a normal, or a diseased brain is working. fMRI images obtained by scanning the brains of subjects as they perform cognitive tasks. A typical study of fMRI data consist of multiple-stage processes that begin with pre-processing of raw data and conclude with a statistical analysis. Such analysis procedures often require upon hundreds or even thousands of images [198].

3.5.1 Population-based Atlas Workflow

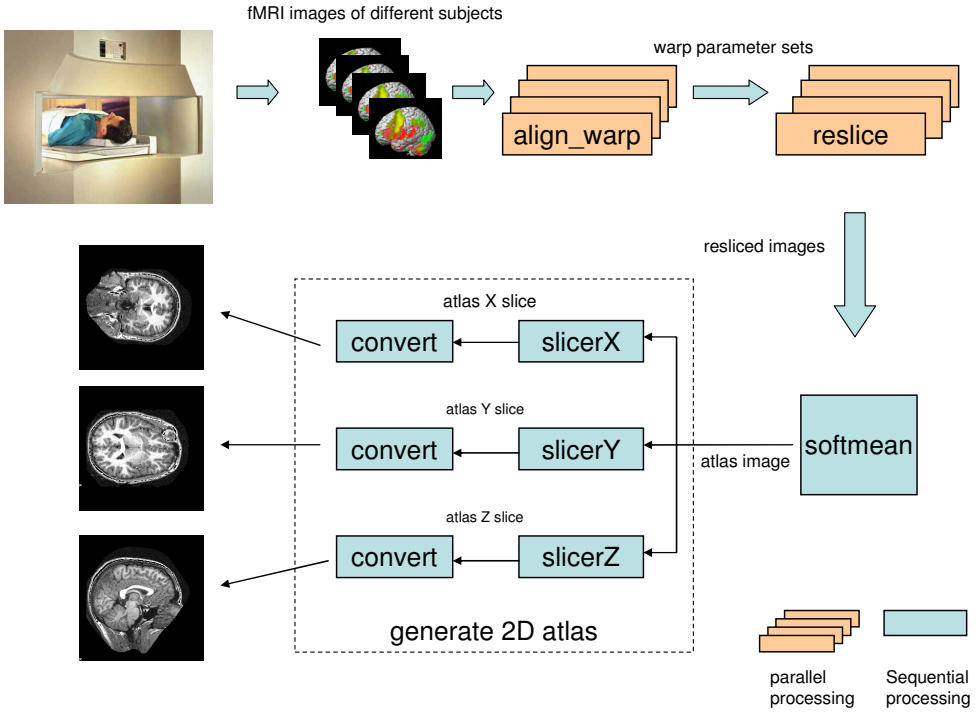


Figure 3.16: Population-based atlas workflow.

Population-based atlas [165] creation is one of the major fMRI research activities. These atlases combine anatomy imaging data from healthy and diseased populations. These describe how the brain varies with age, gender, and demographics. They can be used for identifying systematic effects on brain structure. For instance, they provide a comprehensive approach for studying a particular subgroup, with a specific disease, receiving different medications, or neuropsychiatric disorder. Population-based atlases contain anatomical models from many subjects. They store population templates and statistical maps to summarize features of the population. They also average individual images together so that common features of the subgroup are reinforced.

Figure 3.16 shows a workflow that employs the Automated Image Registration (AIR) [182] and FSL [149] suite for creating population-based brain atlases from high resolution anatomical data. The stages of this workflow are follows:

a) The inputs to the workflow are a set of brain images which are 3D brain scans of population with varying resolutions and a reference brain image. For each brain image, *align_warp* adjusts the position and shape of each image to match the reference brain. The output of each process is a *warp parameter set* defining the spatial transformation to be performed.
b) For each warp parameter set, *reslice* creates a new version of the original brain image according to the configuration parameters defined in the warp parameter set. The output of each reslice procedure is a resliced image.
c) *softmean* averages all the resliced images into one single atlas image.
d) The averaged image is sliced using *slicer* to give a 2D atlas along a plane in three dimension (x, y and z), taken through the centre of the 3D image. The output is an atlas data set.
e) Finally, each atlas data set is converted into a graphical atlas image using *convert*.

3.5.2 Experiment

Table 3.2: Applications configuration of Grid sites.

Node / Details	Applications	Location
Manjra.cs.mu.oz.au	AIR FSL Convert	University of Melbourne, Australia
vgtest.vpac.org	AIR	VPAC, Australia
Vgdev.vpac.org	AIR	VPAC, Australia
Brecca-1.vpac.org	AIR	VPAC, Australia
Brecca-2.vpac.org	AIR	VPAC, Australia
karwendel.dps.uibk.ac.at	FSL	University of Innsbruck, Austria
uuuu.maekawa.is.uec.ac.jp	FSL	University of Elec-tro-Communications, Japan
walkure.maekawa.is.uec.ac.jp	FSL	University of Elec-tro-Communications, Japan

* AIR package includes software for executing the task

Table 3.3: Resource attributes.

Node / Details	CPU (type/#/GHz)	Middleware
manjra.cs.mu.oz.au	4/Intel Xeon/2.00GHz	SSH/GT2
vgtest.vpac.org	1/Intel Xeon/3.20GHz	SSH/GT4
ngdev.vpac.org	1/Intel Xeon/3.20GHz	SSH/GT4
brecca-1.vpac.org	Intel Xeon/4/2.80GHz	SSH
brecca-2.vpac.org	4/Intel Xeon/2.80GHz	SSH
karwendel.dps.uibk.ac.at	2/AMD Opteron 880/2.39 GHz	SSH/SGE
uuuu.maekawa.is.uec.ac.jp	1/Intel Xeon/2.80 GHz	SSH/GT4
walkure.maekawa.is.uec.ac.jp	1/Intel Xeon/2.80 GHz	SSH/GT4

The experiment was conducted using the testbed provided by the University of Melbourne (Australia), Victorian Partnership for Advanced Computing (VPAC) (Australia), University of Electro-Communications (Japan), and University of Innsbruck (Austria). The configuration of all resources is listed in Table 3.2 and Table 3.3. Table 3.2 shows the application software available on every resource. All application software cannot be installed on every resource, due to their varied capability and administration policy. The AIR application required for executing procedure *align_warp*, *reslice* and *softmean* is installed on the sites of VPAC and the University of Melbourne, while the FSL application required for executing procedure *slicer* is installed on other sites. The Convert application required to execute procedure *convert* is only available on the site of the University of Melbourne. Table 3.3 shows processor cabability and supporting middleware of each resource.

In the first experiment, the impact of the number of Grid sites is investigated for the various numbers of subjects. Figure 3.17 shows the total execution times using 1-5 Grid sites for generating atlas of 25, 50 and 100 subjects. The size of image file associated with each subject is around 16 to 22 MB. We can see that the total execution time increases as the number of subjects increases. Additionally, the larger the number of Grid sites, the faster execution time is achieved. For example, the total execution time of generating an atlas of 100 subjects using one Grid node is 95 minutes; however, it only takes 45 minutes using five Grid nodes. The speedup rate is over 50%. It shows the performance of conducting fMRI data analysis can be significantly improved by using the Grid.

Figure 3.18 shows the execution progress for processing 50 subjects. At the beginning of the workflow execution, 50 *align_warp* jobs are generated for the first step, and each job processes one subject image. Once a job in the step one is completed, the task manager of the step two is notified by the output event of this job. It then generates a new *reslice* job of the step two. Therefore, the number of waiting jobs does not continuously reduce when *align_warp* jobs are completed. In Figure 3.18, we can observe that the number of waiting jobs remains around 50 until the 50 jobs of the step one are completed. All the results of the step two are processed once by the *softmean*

task, and the completion of *softmean* generates three *slicer* jobs to produce 2D images along three dimensions. Therefore, there is only small number of waiting jobs after 600 seconds.

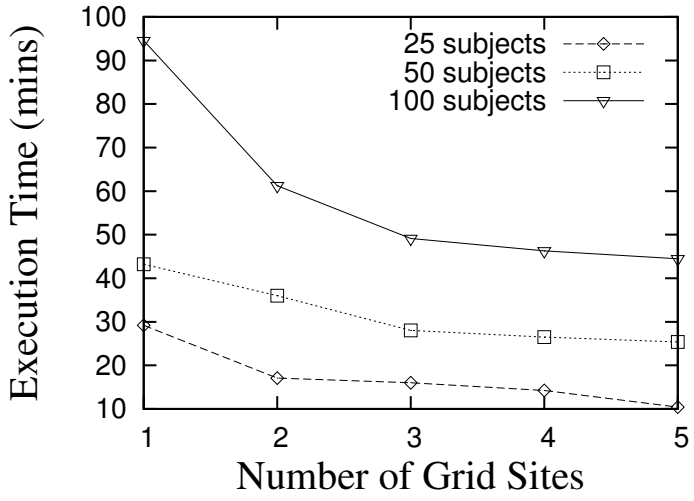


Figure 3.17: Total execution times of processing 25, 50 and 100 subjects over various Grid sites.

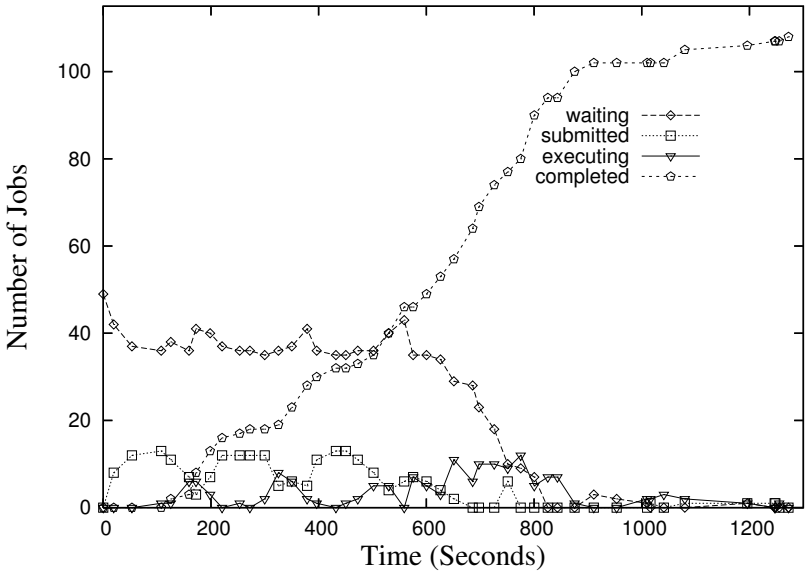


Figure 3.18: Execution progress for processing 50 subjects.

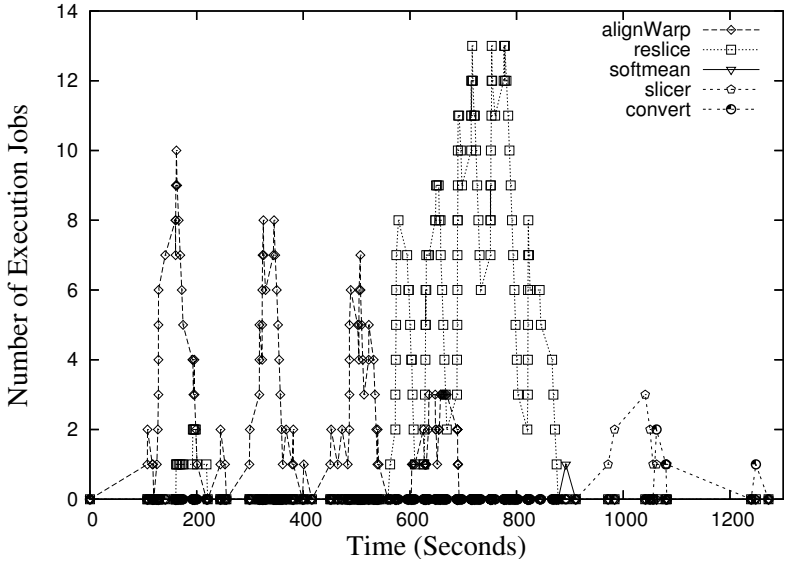


Figure 3.19: Execution tasks for processing 50 subjects.

Figure 3.19 shows the number of jobs of each task running over the execution. Available jobs of the step one and step two can be executed in parallel. However, the jobs of the step one have a higher priority than those of the step two, when they are compete for resources. As we can obverse the most execution jobs during 0-580 seconds are produced by *alignWarp* and only a few of the jobs of *reslice* were executed.

Table 3.4: Detailed execution times of the tasks for processing 100 subjects.

Task	**Start Time (min)**	**End Time (min)**	**Duration (min)**
align_warp	0	22.82	22.82
reslice	2.65	28.50	25.85
softmean	28.92	43.08	14.17
sliceX	43.1	44.1	1
sliceY	43.1	44.13	1.03
sliceZ	43.1	44.07	0.97
convertX	44.4	44.72	0.32
convertY	44.42	44.75	0.33
convertZ	44.07	44.43	0.37

Table 3.4 shows the start and end time for each task in the workflow for 100 subjects. The start time we measured is the time when the stage-in of input data to the

remote resource is started and the end time is the completion time of a task. As we can see from Figure 3.16, there are multiple sub-tasks in task *align_wrap* and *reslice*. The task *reslice* process was started after *align_wrap* process, once an output produced by a sub-task of *align_wrap* was available. However, the task *softmean* is started to process after all sub-tasks of align_wrap have been comeleted, because it requires all results produced by the sub-task of *align_wrap* to generate a mean image. Theoretically, after task *softmean* finishes, its child tasks should be submitted immediately, however, there are some time intervals between the parent tasks' end time and child tasks' start time. That gap can be attributed to the overhead of running WFEE, including time involved in processing event notifications, resource discovery and remote resource submission. However, compared to the running time of tasks, this gap is insignificant and less than 2%.

3.6 Related Work

The workflow engine presented in this chapter is an independent workflow execution system and takes advantage of various middleware services such as security, Grid resource access, file movement and replica management services provided by the Globus middleware [64][69], and multiple middleware dispatchers provided by the Gridbus Broker.

Many efforts toward grid workflow management have been made. DAGMan [163] was developed to schedule jobs to the Condor system in an order represented by a DAG and to process them. With the integration of Chimera [67], Pegasus [52] maps and executes complex workflows based on full-ahead-planning. In Pegasus, a workflow can be generated from metadata description of the desired data product using AI-based planning technologies. The Taverna project [125] has developed a tool for the composition and enactment of bioinformatics workflow for the life science community. The tool provides a graphical user interface for the composition of workflows. Other workflow projects in the Grid context include UNICORE [136], ICENI [116], Karajan [97], Triana [161] and ASKLON [60].

Compared with the work listed above, the workflow engine provides a decentralized scheduling system by using tuple spaces model, which facilitates deployment of different scheduling strategies to each task. It also enables resources to be discovered and negotiated at run-time.

A number of workflow languages have been developed. AGWL (Abstract Grid Workflow Language) [58] is an XML-based language which allows users to define a graph of activities that refer to computational tasks or user interactions without concerning the complexity of underlying technologies. QoWL (QoS-aware Grid Workflow Language) [23] allows users to define their preferences regarding the execution location affinity for activities with specific security and legal constraints. Some languages such as WSBPEL [15] and GSFL [93] address workflows for web services. The workflow language proposed in this chapter is middleware independent and support both abstract and concrete models. It also supports parameterization [8], which is important to scientific applications.

3.7 Summary

In this chapter, a workflow enactment engine is introduced to facilitate composition and execution of workflows in a user-friendly manner. The engine supports different Grid middleware as well as run-time service discovery. It is capable of linking geographically distributed standalone applications and takes advantage of distributed computational resources to achieve high throughput.

The event-driven and subscription-notification mechanisms developed using the tuple spaces model make the workflow execution loosely-coupled and flexible. Supporting parameterization in the workflow language allows users to easily define a range and list of parameters for scientific experiments to generate a set of multiple parallel execution jobs. The engine has been successfully applied to an fMRI analysis application.

The engine proposed in this chapter facilitates users to build workflows to solve their domain problems and provides a basic infrastructure to schedule workflows in Grid environments. The next chapter presents scheduling problems posed by intro-

ducing the service-oriented paradigm in Grid computing. A number of heuristics are then developed to optimize execution performance while meeting users' QoS requirements.

Chapter 4

Cost-Aware Workflow Scheduling

Grid technologies provide the basis for creating a service-oriented paradigm that enables a new way of service provisioning based on utility computing models. For typical utility computing-based services, users are charged for consuming services based on their usage and QoS level required. Therefore, while scheduling workflows on utility Grids, service price must be considered while optimizing the execution performance.

In this chapter, the characteristics of utility Grids and the corresponding scheduling problems are discussed followed by two scheduling heuristics based on two QoS constraints, deadline and budget. Two different workflow structures and experiment settings are also presented for the evaluation of the proposed scheduling heuristics.

4.1 Utility Grids

Utility computing [166] has emerged as a new service provisioning model [43] and is capable of supporting diverse computing services such as servers, storage, network and applications for e-Business and e-Science over a global network. For utility computing-based services, users consume required services, and pay only for what they use. With economic incentive, utility computing encourages organizations to offer their specialized applications and other computing utilities as services so that other

individuals/organizations can access these resources remotely. Therefore, it facilitates individuals/organizations to develop their own core activities without maintaining and developing fundamental compute infrastructure. In the recent past, providing utility computing services has been reinforced by service-oriented Grid computing, that creates an infrastructure for enabling users to consume services transparently over a secure, shared, scalable, sustainable and standard world-wide network environment.

Table 4.1: Community Grids vs. Utility Grids.

Type of Grids / Attributes	Community Grids	Utility Grids
Availability	Best effort	Advanced reservation
QoS	Best effort	Contract/Service Level Agreement (SLA)
Pricing	Not considered or free access	Usage, QoS level, Market supply and demand

Table 4.1 shows different aspects between community Grids and utility Grids in terms of availability, Quality of Services (QoS) and pricing. In utility Grids, users can make a reservation with a service provider in advance to ensure service availability, and users can also negotiate service level agreements with service providers for the required QoS. Compared with utility Grids, service availability and QoS in community Grids may not be guaranteed. However, community Grids provide free access based on mutual agreement driven by partnership (LHCGrid [84]) or free access (e.g., SETI@Home [156]), whereas users need to pay for service access in utility Grids. In general, the service pricing is based on the QoS level, and current market supply and demand.

Typically, service providers charge higher prices for higher QoS. Therefore users do not always need to complete workflows earlier than they require. They sometimes prefer to use cheaper services with a lower QoS that is sufficient to meet their requirements. Given this motivation, cost based workflow scheduling is developed to

schedule workflow tasks on utility Grids according to users' QoS constraints such as deadline and budget.

4.2 Problem Overview

4.2.1 Problem Description

A workflow application can be modeled as a Directed Acyclic Graph (DAG). Let Γ be the finite set of tasks $T_i (1 \le i \le n)$. Let Λ be the set of directed edges. Each edge is denoted by (T_i, T_j), where T_i is called an immediate parent task of T_j, and T_j the immediate child task of T_i. A child task cannot be executed until all of its parent tasks have been completed. There is a transmission time and cost associated with each edge.

In a workflow graph, a task which does not have any parent task is called an *entry task*, denoted as T_{entry} and a task which does not have any child task is called an *exit task*, denoted as T_{exit}. In this thesis, we assume there is only one T_{entry} and T_{exit} in the workflow graph. If there are multiple entry tasks and exit tasks in a workflow, we can connect them to a zero-cost pseudo entry or exit task.

The execution requirements for tasks in a workflow could be heterogeneous. A service may be able to execute some of workflow tasks. The set of services capable of executing task T_i is denoted as S_i, and each task is only assigned for execution on one service. Services have varied processing capability delivered at different prices. The task runtime on each service and input data transfer time are assumed to be known. The estimation of task runtime is presented in Section 4.2.2. The data transfer time can be computed using bandwidth and latency information between the services. t_i^j is the sum of the processing time and input data transmission time, and c_i^j is the sum of the service price and data transmission cost for processing T_i on service $s_i^j (1 \le j \le |S_i|)$.

Let B be the cost constraint (budget) and D be the time constraint (deadline) specified by a user for workflow execution. The budget constrained scheduling problem is to map every T_i onto a suitable service to minimize the execution time of the workflow and complete it with the total cost less than B. The deadline constrained scheduling problem is to map every T_i onto a suitable service to minimize the execution cost of the workflow and complete it before deadline D.

4.2.2 Performance Estimation

Performance estimation is the prediction of performance of task execution on services and is crucial for generating an efficient schedule for advance reservations. Different performance estimation approaches can be applied to different types of utility services. We classify existing utility services as either *resource services*, *application services* or *information service*.

Resource services provide hardware resources such as computing processors, network resources, storage and memory, as a service for remote clients. To submit tasks to resource services, the scheduler needs to determine the number of resources and duration required to run tasks on the discovered services. The performance estimation for resource services can be achieved by using existing performance estimation techniques (e.g. analytical modeling, empirical and historical data) to predict task execution time on every discovered resource service.

Application services allow remote clients to use their specialized applications, while information services provide information for the users. Unlike resource services, an application service and information service are capable of providing estimated service times based on the metadata of users' service requests. As a result, the task execution time can be obtained by the providers.

4.3 Cost-based Workflow Scheduling Heuristics

Workflow scheduling focuses on mapping and managing the execution of inter-dependent tasks on diverse utility services. In this Section, two heuristics are

provided as a baseline for cost-based workflow scheduling problems. The heuristics follow the divide-and-conquer technique and the methodology is listed below:

Step 1. Discover available services and predict execution time for every task.

Step 2. Distribute users' overall deadline or budget into every task.

Step 3. Query available time durations (time slots), generate an optimized schedule plan and make advance reservations based on the local optimal solution of every task partition.

4.3.1 Deadline Constrained Scheduling

The proposed deadline constrained scheduling heuristic is called *Greedy Cost-Time Distribution* (GreedyCost-TD). In order to produce an efficient schedule, Greedy-Cost-TD groups workflow tasks into task partitions and assigns sub-deadlines to each task based on their workload and dependencies. At runtime, a task is scheduled on a service, which is able to complete it within its assigned sub-deadline at the lowest cost.

Workflow Task Partitioning

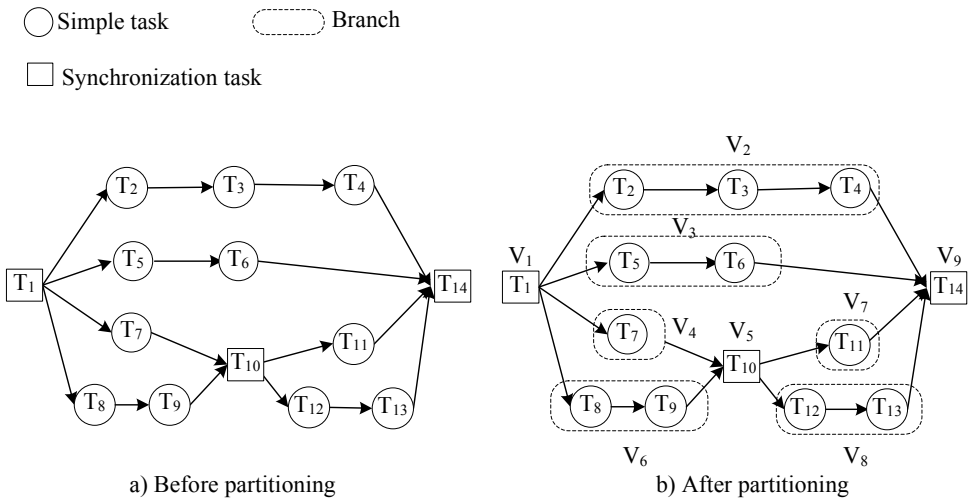


Figure 4.1: Workflow task partition.

In workflow task partitioning, workflow tasks are first categorized as either *synchronization tasks* or *simple tasks*. A synchronization task is defined as a task which

has more than one parent or child task. In Figure 4.1, T_1, T_{10} and T_{14} are synchronization tasks. Other tasks which have only one parent task and child task are simple tasks.

In Figure 4.1, $T_2 - T_9$ and $T_{11} - T_{13}$ are simple tasks. Simple tasks are then clustered into a *branch*. A branch is a set of interdependent simple tasks that are executed sequentially between two synchronization tasks. For example, the branches in Figure 4.1b are $\{T_2, T_3, T_4\}$, $\{T_5, T_6\}$, $\{T_7\}$, $\{T_8, T_9\}$, $\{T_{11}\}$ and $\{T_{12}, T_{13}\}$.

After task partitioning, workflow tasks Γ are then clustered into partitions. As shown in Figure 4.1b, a *partition* $V_i (1 \le i \le k)$ is either a branch or a set of one synchronization task, where k is the total number of branches and synchronization tasks in the workflow. For a given workflow $\Omega(\Gamma, \Lambda)$, the corresponding task partition graph $\Omega'(\Gamma', \Lambda')$ is created as follows:

$$\begin{cases} \Gamma' = \{ V_i \mid V_i \text{ is a partition in } \Omega, \text{ where } 1 \le i \le k \} \\ \Lambda' = \{ (v_i, w) \mid v \in V_i, w \in V_j, \text{and } (v, w) \in \Lambda \} \end{cases}$$

where Γ' is a set of task partitions and Λ' is the set of directed edges of the form (V_i, V_j), and V_i is a parent task partition of V_j and V_j is a child task partition of V_i.

Deadline Distribution

After workflow task partitioning, the overall deadline is distributed over each V_i in Ω'. The deadline assigned to any V_i is called a *sub-deadline* of the overall deadline D. The deadline assignment strategy considers the following facts:

P1. *The cumulative expected execution time of any simple path between two synchronization tasks must be the same.*

A *simple path* in Ω' is a sequence of task partitions such that there is a directed edge from every task partition (in the simple path) to its child, where none of the task

partitions in the simple path is repeated. For example, $\{V_1, V_3, V_9\}$ and $\{V_1, V_6, V_5, V_7, V_9\}$ are two simple paths between V_1 and V_9.

A synchronization task cannot be executed until all tasks in its parent task partitions are completed. Thus, instead of waiting for other simple paths to be completed, a path capable of being finished earlier can be executed on slower but cheaper services. For example, the deadline assigned to $\{T_8, T_9\}$ is the same as $\{T_7\}$ in Figure 4.1. Similarly, deadlines assigned to $\{T_2, T_3, T_4\}, \{T_5, T_6\}$, and $\{\{T_7\}, \{T_{10}\}, \{T_{12}, T_{13}\}\}$ are the same.

P2. *The cumulative expected execution time of any path from* V_i $(T_{entry} \in V_i)$ *to* V_j $(T_{exit} \in V_j)$ *is not greater than the overall deadline* D.

P2 assures that once every task partition is computed within its assigned deadline, the whole workflow execution can satisfy the user's required deadline.

P3. *Any assigned sub-deadline must be greater than or equal to the minimum processing time of the corresponding task partition.*

If the assigned sub-deadline is less than the minimum processing time of a task partition, its expected execution time will exceed the capability that its execution services can provide.

P4. *The overall deadline is divided over task partitions in proportion to their minimum processing time.*

The execution times of tasks in workflows vary; some tasks may only need 20 minutes to be completed, and some others may need at least one hour. Thus, the deadline distribution for a task partition should be based on its execution time. Since there are multiple possible processing times for every task, we use the minimum processing time to distribute the deadline.

The deadline assignment strategy on the task partition graph is implemented by combining Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms with

critical path analysis to compute start times, proportion and deadlines of every task partition. After distributing overall deadline into task partitions, each task partition is assigned a deadline. There are three attributes associated with a task partition V_i: deadline ($dl[V_i]$), ready time ($rt[V_i]$), and expected execution time ($eet[V_i]$). The ready time of V_i is the earliest time when its first task can be executed. It can be computed according to its parent partitions and defined by:

$$rt[V_i] = \begin{cases} 0, & T_{entry} \in V_i \\ \max\limits_{V_j \in PV_i} dl[V_j], & otherwise \end{cases} \tag{4.1}$$

where PV_i is the set of parent task partitions of V_i. The relation between three attributes of a task partition V_i follows that:

$$eet[V_i] = dl[V_i] - rt[V_i] \tag{4.2}$$

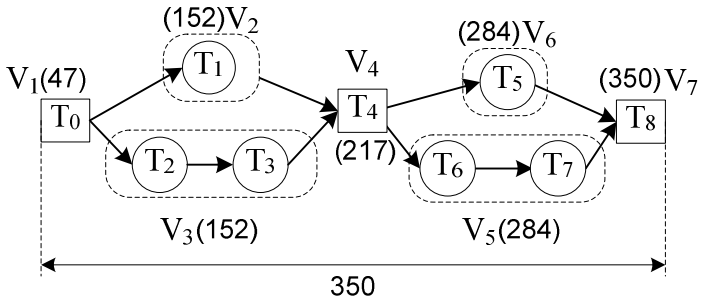


Figure 4.2: Deadline distribution.

After deadline distribution over task partitions, a sub-deadline is assigned to each task. Figure 4.2 shows an example of spreading the overall deadline (350 time units) over partitions (V_1 - V_7). If the task is a synchronization task, its sub-deadline is equal to the deadline of its task partition. As shown in the example, the sub-deadline assigned to T_4 is 217 time units. However, if a task is a simple task of a branch, its sub-deadline is assigned by dividing the deadline of its partition based on its processing time. For example, there are two tasks in V_3 and the sub-deadline of each task

need to be computed based on the sub-deadline of V_3 and the minimum processing time of each task. Let P_i be the set of parent tasks of T_i. The assigned deadline of task T_i in partition V is defined by:

$$dl[T_i] = eet[T_i] + rt[V] \tag{4.3}$$

$$\text{where} \quad eet[T_i] = \frac{\min\limits_{1 \le j \le |S_i|} t_i^j}{\sum\limits_{T_k \in V} \min\limits_{1 \le l \le |S_k|} t_k^l} \times eet[V] \tag{4.4}$$

$$rt[T_i] = \begin{cases} 0, & T_i = T_{entry} \\ \max\limits_{T_j \in P_i} dl[T_j], & otherwise \end{cases} \tag{4.5}$$

Greedy Cost-Time Distribution (TD)

Once each task has its own sub-deadline, a local optimal schedule can be generated for each task. If each local schedule guarantees that their task execution can be completed within their sub-deadline, the whole workflow execution will be completed within the overall deadline. Similarly, the result of the cost minimization solution for each task leads to an optimized cost solution for the entire workflow. Therefore, an optimized workflow schedule can be constructed from all local optimal schedules. The schedule allocates every workflow task to a selected service such that they can meet its assigned sub-deadline at low execution cost. Let $Cost(T_i)$ be the sum of data transmission cost and service cost for processing T_i. The objective for scheduling task T_i is:

$$\text{Minimize} \quad Cost(T_i) = \min_{1 \le j \le |S_i|} c_i^j, \tag{4.6}$$

$$\text{subject to} \quad t_i^j \le eet[T_i]$$

The details of GreedyCost-TD heuristic are presented in Algorithm 4.1. It first partitions workflow tasks and distributes overall deadline over each task partition and then divides the deadline of each partition into each single task. Unscheduled tasks are queued in the ready queue waiting to be scheduled. The order of tasks in the ready

queue is sorted by a ranking strategy which is described in Section 4.3.3. The scheduler schedules tasks in the ready queue one by one. The entry task is the first task which is put into the ready task and scheduled. Once all parents of a task have been scheduled, the task becomes ready for scheduling and is put into the ready queue (Line 15). The ready time of each task is computed at the time it is scheduled (Line 9). After obtaining all available time slots (Line 9) based on the ready time and sub-deadline of current scheduling tasks, the task is scheduled on a service which can meet the scheduling objective. If no such service is available, the service which can complete the task at earliest time is selected to satisfy the overall time constraint (Line 11-12).

Algorithm 4.1: Greedy Cost-Time Distribution Heuristic

Input: *A workflow graph* $\Omega(\Gamma,\Lambda)$*, deadline D*

Output: *A schedule for all workflow tasks*

```
1   Request processing time and price from available services
    for ∀T ∈ Γ
2   Convert Ω into task partition graph Ω'(Γ',Λ')
3   Distribute deadline D over ∀V_i ∈ Γ' and assign a sub-deadline
    to each task
4   Put the entry task into ready task queue Q
5   while there are ready tasks in Q do
6      Sort all tasks in Q
7      T_i ← the first task from Q
8      Compute ready time of T_i
9      Query available time slots during ready time
       and sub-deadline
10     S ← a service which meets Equation 4-6
11     if S = φ then
12        S ← S_i^j such that j = arg min_{1≤j≤|S_i|} t_i^j
13     end if
14     Make advance reservations of T_i on S
15     Put ready child tasks into Q whose parent tasks have been
       scheduled
16  end while
```

4.3.2 Budget Constrained Scheduling

The proposed budget constrained scheduling heuristic is called *Greedy Time-Cost Distribution* (GreedyTime-CD). It distributes portions of the overall budget to each task in the workflow. At runtime, a task is scheduled on a service which is able to complete it with less cost than its assigned sub-budget at the earliest time.

Budget Distribution

The budget distribution process is to distribute the overall budget over tasks. In the budget distribution, both workload and dependencies between tasks are considered when assigning sub-budgets to tasks. There are two major steps:

Step 1. Assigning portions of the overall budget to each task.

In this step, an initial sub-budget is assigned to tasks based on their average execution and data transmission cost. In a workflow, tasks may require different types of services with various price rates, and their computational workload and required I/O data transmission may vary. Therefore, the portion of the overall budget each task obtains should be based on the proportion of their expense requirements. Since there are multiple possible services and data links for executing a task, their average cost values are used for measuring their expense requirements. The expected budget for task T_i is defined by:

$$eec[T_i] = \frac{avgCost[T_i]}{\sum_{T_i \in \Gamma} avgCost[T_i]} \times B \tag{4.7}$$

where $avgCost[T_i] = \frac{\sum_{1 \le j \le |S_i|} c_i^j}{|S_i|}$

Step 2. Adjusting initial sub-budget assigned to each task by considering their task dependencies.

The sub-budget of a task assigned in the first step is only based on its average cost without considering its execution time. However, some tasks could be completed at earliest time using more expensive services based on their local budget, but its child

tasks cannot start execution until other parent tasks have been completed. Therefore, it is necessary to consider task dependencies for assigning a sub-budget to a task. In the second step, the initial assigned sub-budgets of tasks are adjusted. It first computes the approximate execution time based on its initial expected execution budget and its unit time per cost so that the approximate start time and end time of each task partition can be calculated. The approximate execution time of task T_i is defined by:

$$aet[T_i] = eec[T_i] \times \frac{avgTime[T_i]}{avgCost[T_i]} \tag{4.8}$$

$$\text{where} \quad avgTime[T_i] = \frac{\sum_{1 \le j \le |S_i|} t_i^j}{|S_i|}$$

Then it partitions workflow tasks and computes approximate start time and end time of each partition. If the end time of a task partition is earlier than the start time of its child partition, it is assumed that the initial sub-budget of this partition is higher than what it really requires and its sub-budget is reduced. The spare budget produced by reducing initial sub-budgets is calculated and is defined by:

$$spareBudget = B - \sum_{T_i \in \Gamma} eec[T_i] \tag{4.9}$$

Finally, the spare budget is distributed to each task based on their assigned sub-budgets. The final expected budget assigned to each task is:

$$eec[T_i] = eec[T_i] + spareBudget \times \frac{eec[T_i]}{\sum_{T_i \in \Gamma} eec[T_i]} \tag{4.10}$$

Greedy Time-Cost Distribution (CD)

After budget distribution, CD attempts to allocate the fastest service to each task among those services which are able to complete the task execution within its assigned budget. Let $Time(T_i)$ be the completion time of T_i. The objective for scheduling task T_i is:

$$\text{Minimize } Time(T_i) = \min_{1 \le j \le |S_i|} t_i^j, \tag{4.11}$$

$$\text{subject to } c_i^j \le eec[T_i]$$

Algorithm 4.2: Greedy Time-Cost Distribution Heuristic

Input: *A workflow graph* $\Omega(\Gamma,\Lambda)$*, budget B*

Output: *A schedule for all workflow tasks*

```
1   Request processing time and price from available services
    for ∀T_i ∈ Γ
2   Distribute budget B over ∀T_i ∈ Γ
3   plannedCost = 0 ; acturalCost = 0
4   Put the entry task into ready task queue Q
5   while there are ready tasks in Q do
6     Sort all tasks in Q
7     S ← the first task from Q
8     Compute start time of T_i and query available time slots
9     eec[T_i] = plannedCost − acturalCost + eec[T_i]
10    S ← a service which meets Equation 4-11
11    if S = φ then
12       S ← S_i^j such that j = arg min_{1≤j≤|S_i|} c_i^j
13    end if
14    Make advance reservations with of T_i on S
15    acturalCost = acturalCost + c_i^j
16    plannedCost = plannedCost + eec[T_i]
17    Put ready child tasks into Q whose parent tasks have been
      scheduled
18  end while
```

The detail of Greedy Time-Cost Distribution is presented in Algorithm 4.2. It first distributes the overall budget to all tasks. After that, it starts to schedule first level tasks of the workflow. Once all parents of a task have been scheduled, the task is ready for scheduling and then the scheduler put it into a ready queue (Line 17). The order of tasks in the ready queue is sorted by a ranking strategy (see Section 4.3.3). The actual costs of allocated tasks and their planned costs are also computed successively at runtime (Line 15-16). If the aggregated actual cost is less than the aggregated planned cost, the scheduler adds the unspent aggregated budget to sub-budget of the

current scheduling task (Line 9). A service is selected if it satisfies the scheduling objective; otherwise, a service with the least execution cost is selected in order to meet the overall cost constraint.

4.3.3 Ranking Strategy for Parallel Tasks

In a large scale workflow, many parallel tasks could compete for time slots on the same service. For example, in Figure 4.1, after T_1 is scheduled, T_2, T_5, T_7 and T_8 become ready tasks and are put into the ready task queue. The scheduling order of these tasks may impact on performance significantly.

Eight strategies are developed and investigated for sorting ready tasks in the ready queue:

- **MaxMin-Time**: Obtains the minimum execution time and data transmission time of executing each task on all their available services and sets higher scheduling priority to tasks which have longer minimum processing time.
- **MaxMin-Cost:** Obtains the minimum execution cost and data transmission cost of executing each task on all their available services and sets higher scheduling priority to tasks which require more monetary expense.
- **MinMin-Time:** Obtains minimum execution time and data transmission time of executing each task on all their available services and sets higher scheduling priority to tasks which have shorter minimum processing time.
- **MinMin-Cost**: Obtains minimum execution cost and data transmission cost of executing each task on all their available services and sets higher scheduling priority to tasks which require less monetary expense.
- **Upward Ranking:** Sorts tasks based on upward ranking [167]. The higher upward rank value, the higher scheduling priority. The upward rank of task T_i is recursively defined by:

$$rank(T_i) = \varpi_i + \max_{T_j \in succ(T_i)} (\overline{c_{ij}} + rank(T_j))$$

$$rank(T_{exit}) = 0$$

where $\overline{w_i}$ is the average execution time of executing task T_i on all its available services and $\overline{c_{ij}}$ is the average transmission time of transfering intermediate data from T_i to T_j.

- **First Come First Serve (FCFS)**: Sorts tasks based on their available time; the earlier available time, the higher scheduling priority.
- **MissingDeadlineFirst:** Sets higher scheduling priority to tasks which have earlier sub-deadline.
- **MissingBudgetFirst:** Sets higher scheduling priority to tasks which have fewer sub-budgets.

4.4 Workflow Applications

Given that different workflow applications may have a different impact on the performance of the scheduling algorithms, a task graph generator is developed to automatically generate a workflow based on the specified workflow structure, and the range of task workload and the I/O data. Since the execution requirements for tasks in scientific workflows are heterogeneous, the service type attribute is used to represent different types of services. The range of service types in the workflow can be specified. The width and depth of the workflow can also be adjusted in order to generate workflow graphs of different sizes.

According to many Grid workflow projects [21][109][181], workflow application structures can be categorized as either *balanced structure* or *unbalanced structure*. Examples of balanced structure include Neuro-Science application workflows [197] and EMAN refinement workflows [109], while the examples of unbalanced structure include protein annotation workflows [25] and Montage workflows [21]. Figure 4.3 shows two workflow structures, a *balanced-structure application* and an *unbalanced-structure application*, used in our experiments. As shown in Figure 4.3(a), the balanced-structure application consists of several parallel pipelines, which require the same types of services but process different data sets. In Figure 4.3(b), the structure of the unbalanced-structure application is more complex. Unlike the balanced-structure

application, many parallel tasks in the unbalanced structure require different types of services, and their workload and I/O data varies significantly.

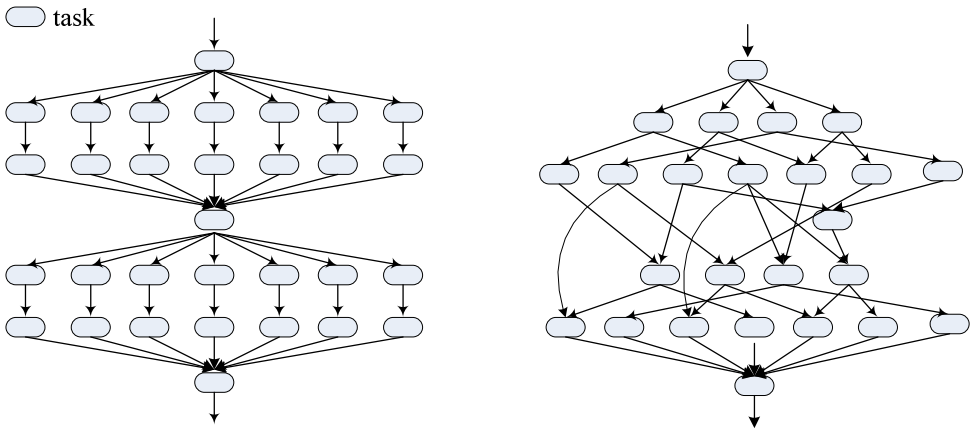


Figure 4.3: Small portion of workflow applications.

4.5 Other Heuristics

In order to evaluate the cost-based scheduling proposed in this chapter, two other heuristics which are derived from existing work are implemented and compared with the TD and CD.

4.5.1 Greedy-Time and Greedy-Cost

Greedy-Time and *Greedy-Cost* are derived from the cost and deadline optimization algorithms in Nimrod-G [9], which is initially designed for scheduling independent tasks on Grids. Greedy-Time is used for solving time optimization problem with a budget. It sorts services by their processing times and assigns as many tasks as possible to the fastest services without exceeding the budget. Greedy-Cost is used for solving the cost optimization problem within the deadline. It sorts services by their processing prices and assigns as many tasks as possible to cheapest services without exceeding the deadline.

4.5.2 Backtracking

Backtracking denoted as BT is proposed by Menascé and Casalicchio [118]. It assigns ready tasks to least expensive computing resources. The heuristic repeats the procedure until all tasks have been mapped. After each iterative step, the execution time of the current assignment is computed. If the execution time exceeds the deadline, it back-tracks to the previous step and removes the least expensive resource from its resource list and reassigns tasks with the reduced resource set. If the resource list is empty the heuristic keeps back-tracking to the previous step. It reduces the corresponding resource list and then reassigns the tasks. The backtracking method is also extended to support optimizing cost while meeting budget constraints. Budget constrained backtracking assigns ready tasks to fastest computing resources.

4.6 Performance Evaluation

4.6.1 Experimental Setup

GridSim [155] is used to simulate a Grid environment for experiments. Figure 4.4 shows the simulation environment, in which simulated services are discovered by querying the GridSim Index Service (GIS). Every service is able to provide free slot query, and handle reservation request and reservation commitment.

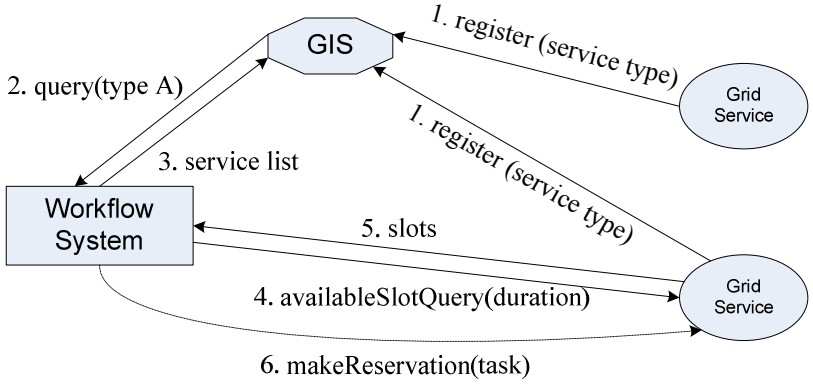


Figure 4.4: A Simulation scenario.

15 types of services with various price rates are modeled to simulate a heterogeneous environment, each of which was supported by 10 service providers with various

processing capability. The topology of the system is such that all services are connected to one another, and the available network bandwidths between services are 100Mbps, 200Mbps, 512Mbps and 1024Mbps.

Table 4.2: Service speed and corresponding price for executing a task.

Service ID	Processing Time (sec)	Cost (G$/sec)
1	1200	300
2	600	600
3	400	900
4	300	1200

Table 4.3: Transmission bandwidth and corresponding price.

Bandwidth (Mbps)	Cost (G$/sec)
100	1
200	2
512	5.12
1024	10.24

For the experiments, the cost that a user needs to pay for a workflow execution comprises of two parts: processing cost and data transmission cost. Table 4.2 shows an example of processing cost, while Table 4.3 shows an example of data transmission cost. It can be seen that the processing cost and transmission cost are inversely proportional to the processing time and transmission time respectively.

In order to evaluate algorithms on reasonable budget and deadline constraints we also implemented a time optimization algorithm, *Heterogeneous-Earliest-Finish Time* (HEFT) [167], and a cost optimization algorithm, *Greedy Cost* (GC). The HEFT algorithm is a list scheduling algorithm which attempts to schedule DAG tasks at minimum execution time on a heterogeneous environment. The GC approach is to minimize workflow execution cost by assigning tasks to services of lowest cost. The deadline and budget used for the experiments are based on the results of these two algorithms. Let C_{min} and C_{max} be the total monetary cost produced by GC and

HEFT respectively, and T_{max} and T_{min} be their corresponding total execution time. Deadline *D* is defined by:

$$D = T_{min} + k(T_{max} - T_{min}) \tag{4.12}$$

and budget *B* is defined by:

$$B = C_{min} + k(C_{max} - C_{min}) \tag{4.13}$$

The value of k varies between 0 and 10 to evaluate the algorithm performance from tight constraint to relaxed constraint. As *k* increases, the constraint is more relaxed.

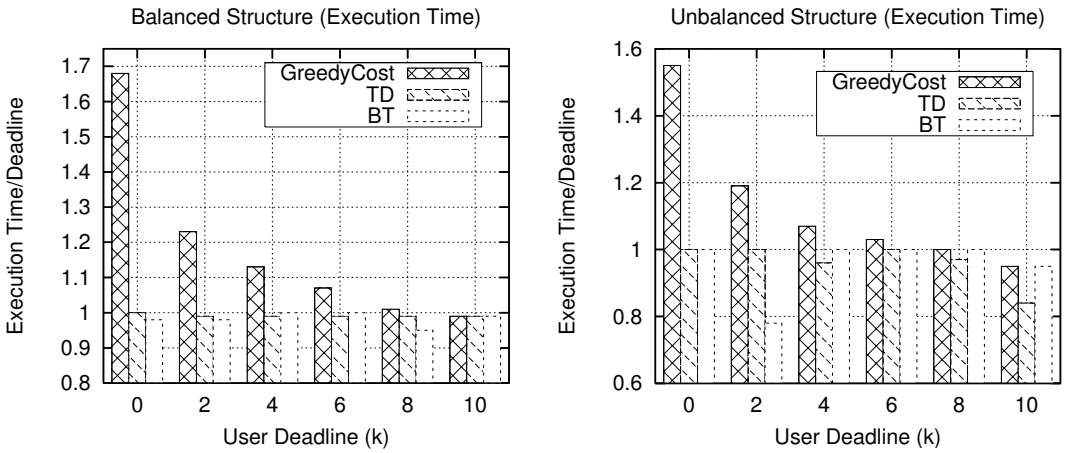
4.6.2 Results

In this section, CD and TD are compared with Greedy-Time, Greedy-Cost and Back-Tracking on the two workflow applications, balanced and unbalanced. In order to show the results more clearly, we normalize the execution time and cost. Let C_{value} and T_{value} be the execution time and the monetary cost generated by the algorithms in the experiments respectively. For the case of budget constrained problems, the execution cost is normalized by using C_{value} / B, and the execution time by using T_{value} / T_{min}. The normalized values of the execution cost should be no greater than one, if the algorithms meet their budget constraints. Therefore, it is easy to recognize whether the algorithms achieve the budget constraints. By using the normalized execution time value, it is also easy to recognize whether the algorithms produce an optimal solution when the budget is high. In the same way, the normalized execution time and the execution cost is normalized for the deadline constraint case by using T_{value} / D and C_{value} / C_{min} respectively.

Cost optimization within a set deadline

A comparison of the execution time and cost results of the three deadline constrained scheduling methods for the balanced-structure application and unbalanced-structure application is shown in Figure 4.5 and Figure 4.6 respectively. The ranking strategy used for TD is FCFS. From Figure 4.5, we can see that it is hard for Greedy-Cost to

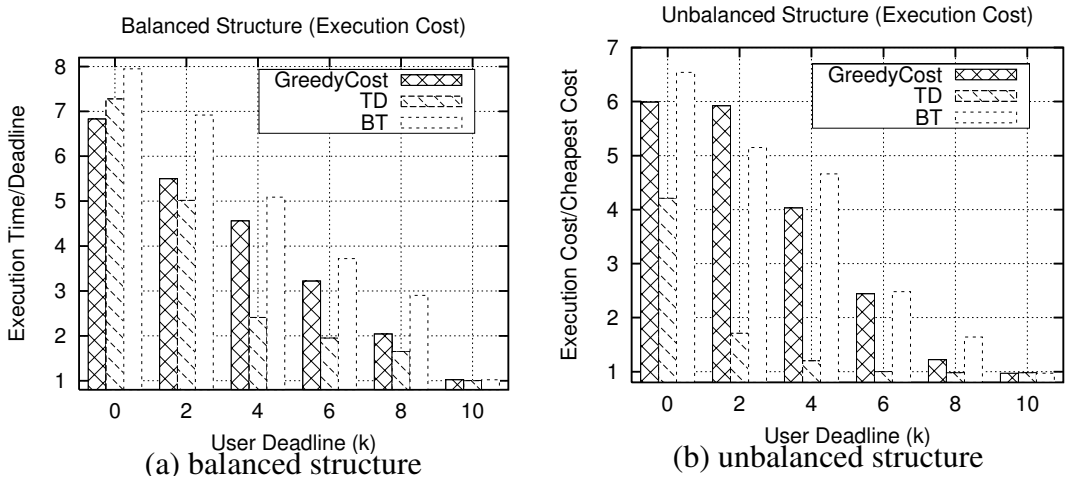
meet deadlines when they are tight, TD slightly exceeds deadline when $k = 0$, while BT can satisfy deadlines each time. For execution cost required by the three approaches shown in Figure 4.6, Greedy-Cost performs worst while TD performs best. Even though the processing time of the Greedy-Cost is longer than TD, its corresponding processing cost is much higher. Compared with BT, TD saves almost 50% execution cost when deadlines are relatively low. However, the three approaches produce similar results when deadline is greatly relaxed.



(a) balanced structure (b) unbalanced structure

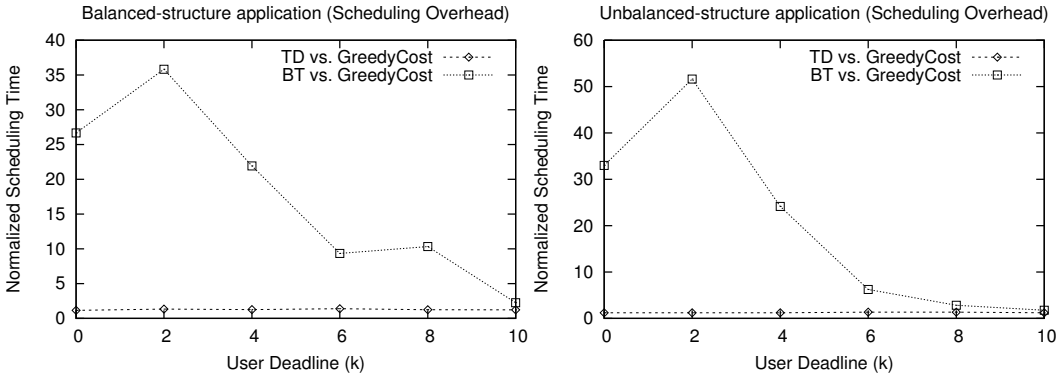
Figure 4.5: Execution time for scheduling balanced- and unbalanced-structure applications.

Figure 4.7 shows scheduling running time for three approaches. In order to show the results clearly, the scheduling time of BT and TD is normalized by the scheduling time of Greedy-Cost. We can observe that Greedy-Cost requires the least scheduling time, since the normalized values of BT and TD are bigger than 1. The scheduling time required by TD is slightly higher than Greedy-Cost but much lower than BT. As the deadline varies, BT requires more running time when deadlines are relatively tight. For example, scheduling times at $k = 0, 2, 4$ are much longer than at $k = 6, 8, 10$. This is because it needs to back-track for more iterations to adjust previous task assignments in order to meet tight deadlines.



(a) balanced structure (b) unbalanced structure

Figure 4.6: Execution cost for scheduling balanced- and unbalanced-structure. applications.



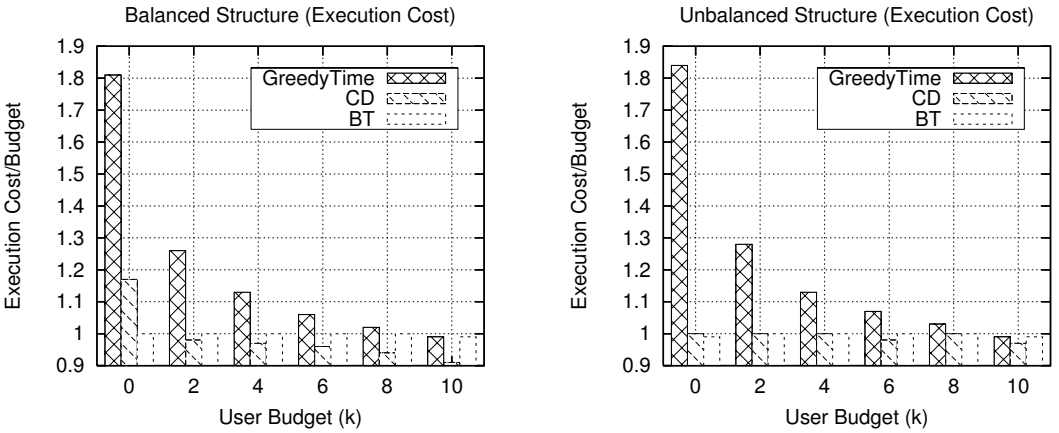
(a) balanced-structure (b) unbalanced-structure

Figure 4.7: Normalized scheduling overhead for deadline constrained scheduling.

Budget constrained heuristics

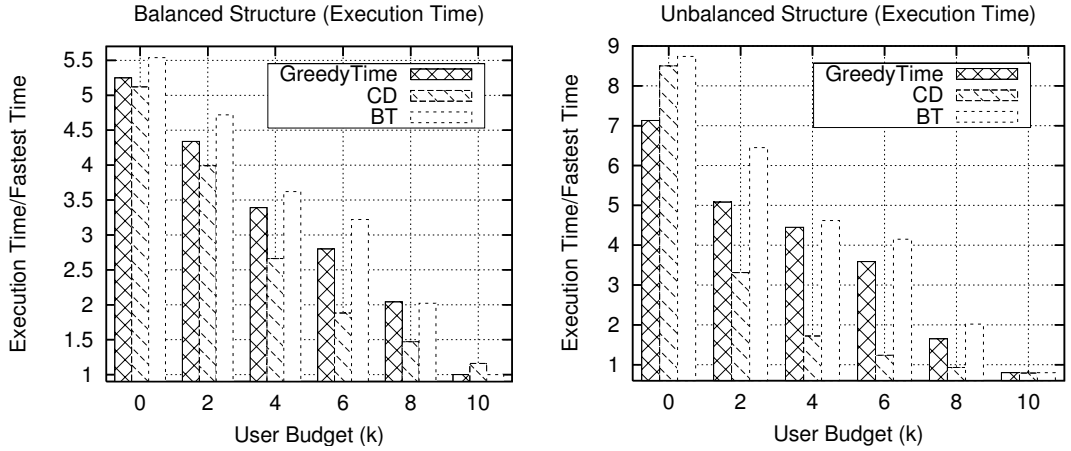
A comparison of the execution cost and time results of three budget constrained scheduling methods for the balanced-structure application and unbalanced-structure application is shown in Figure 4.8 and Figure 4.9 respectively. Greedy-Time can only meet budgets when the budgets are very relaxed. It is also hard for CD to meet budgets when the budgets are very tight (i.e. $k = 10$). However, CD outperforms BT and

Greedy-Time as the budget increases. For scheduling the balanced-structure application (see Figure 4.8a and Figure 4.9a), Greedy-Time and BT also incurs significantly longer execution times even though it uses higher budgets to complete executions. For scheduling the unbalanced-structure application (see Figure 4.8b and 4.9b), CD produces more than 50% faster schedule than that of BT by using similar budgets. However, CD performs worse than BT and Greedy-Time when the budget is very relaxed (i.e. $k = 10$).



(a) balanced structure (b) unbalanced structure

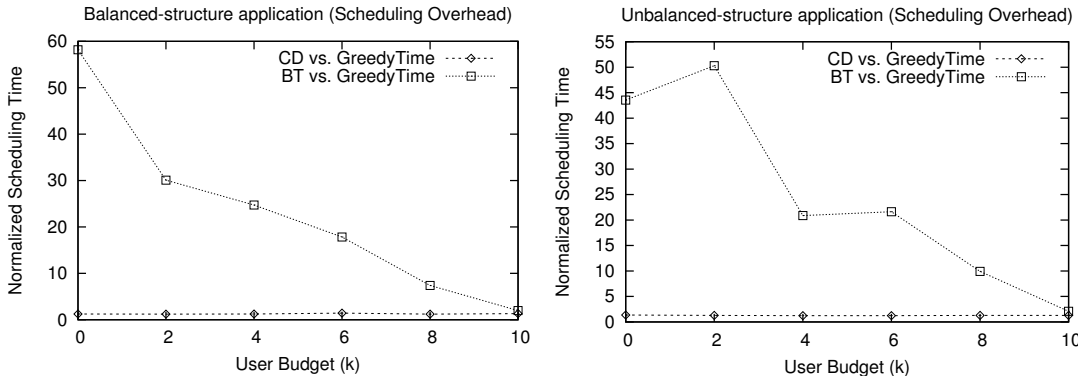
Figure 4.8: Execution cost for scheduling balanced- and unbalanced-structure applications.



(a) balanced structure (b) unbalanced structure

Figure 4.9: Execution time for scheduling balanced- and unbalanced-structure applications.

Figure 4.10 shows scheduling running time for three approaches, Greedy-Time, CD and BT. In order to show the results clearly, we normalize the scheduling time of BT and CD by using the scheduling time of Greedy-Time. We can observe that Greedy-Time requires the least scheduling time, since the normalized values of BT and CD are bigger than 1. The scheduling time required by CD is slightly higher than Greedy-Time but much lower than BT. Similar to the deadline constrained cases, the running time of BT decreases as the budget increases.



(a) balanced structure (b) unbalanced structure

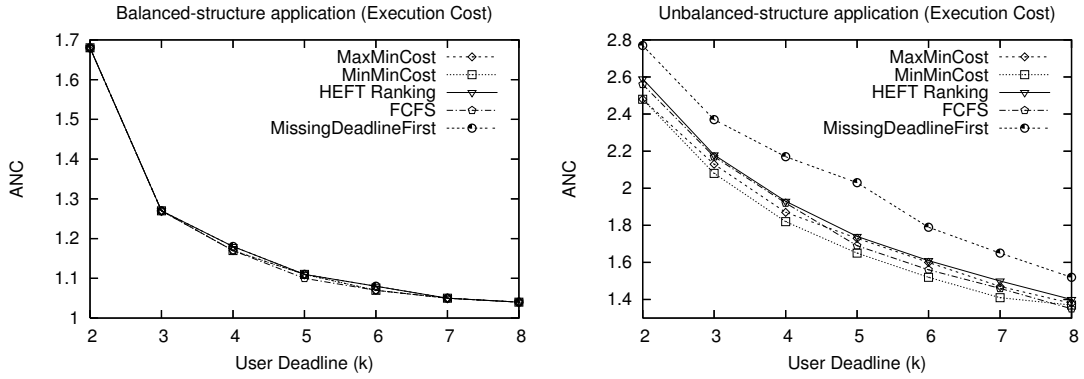
Figure 4-10: Normalized scheduling overhead for budget constrained scheduling.

Impact of Ranking Strategy

FCFS is used as the ranking strategy for the experiments comparing CD and TD with other heuristics. In order to investigate the impact of different ranking strategies, experiments comparing five different ranking strategies for each constrained problem are conducted. In these experiments, each ranking strategy is carried out to schedule 30 random generated balanced-structure and unbalanced-structure workflow applications. The average values are used to report the results.

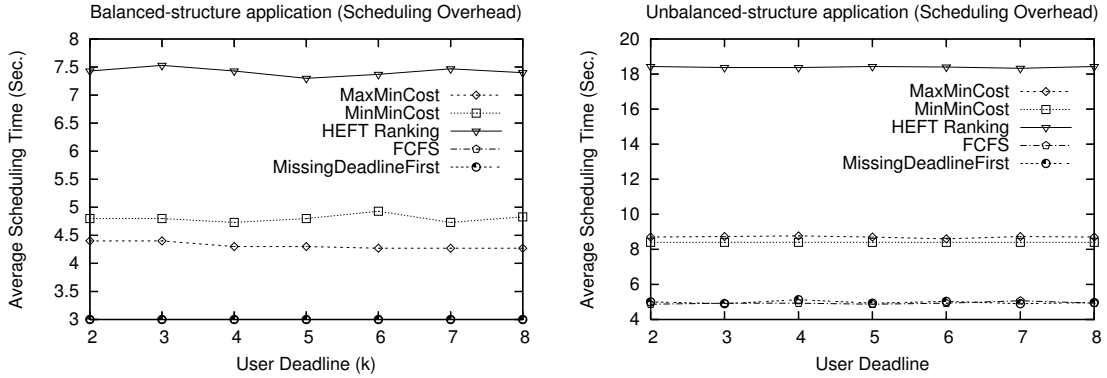
Figure 4.11 shows the total execution costs of TD by employing different ranking strategies: MaxMinCost, MinMinCost, HEFT Ranking, FCFS and MissingDeadline-First. Their cost optimization performances are measured by the Average Normalized Cost (ANC) which is the average value of execution cost for executing 30 different

randomly generated workflow graphs. The execution cost is normalized by the cheapest cost generated by Greedy Cost (GC).



(a) balanced-structure application (b) unbalanced-structure application

Figure 4.11: Comparison of execution costs among five different ranking strategies for scheduling deadline constrained workflow.

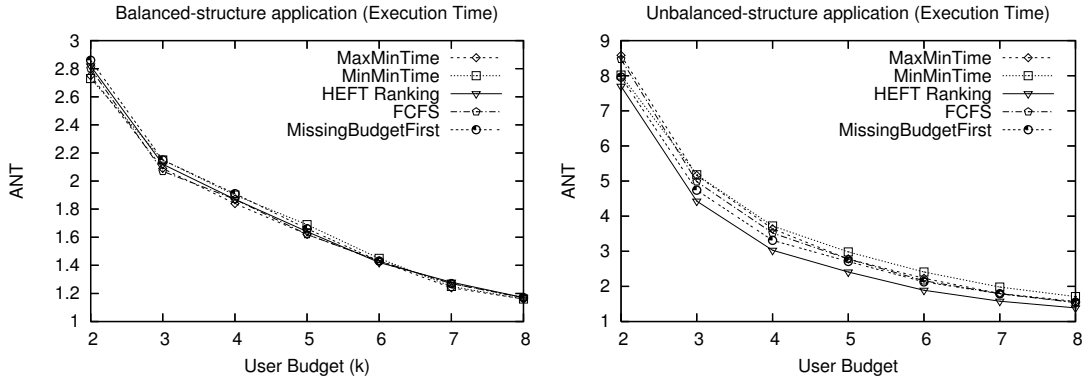


(a) balanced-structure application (b) unbalanced-structure application

Figure 4.12: Scheduling overhead of five deadline constrained ranking strategies.

For the balanced-structure (see Figure 4.11a), five ranking strategies produce similar results. This is because parallel tasks in the balanced-structure application have the same ranking value. However, for the unbalanced-structure (see Figure 4.11b), MinMinCost performs better than others while MissingDeadline incurs significantly higher cost. This is because the Min-MinCost heuristic schedules tasks having minimum execution cost first so that it results in the higher percentage of tasks assigned to their best choice (which can complete the tasks with least cost) than others.

The scheduling running time is also investigated and showed in Figure 4.12. Among five strategies, HEFT Ranking produces highest complexity while FCFS and MissingDeadlineFirst produce lowest complexity. The scheduling running time of MinMinCost is slightly higher than that of MaxMinCost for balanced-structure applications, but they are similar for the unbalanced-structure. Therefore, FCFS and MissingDeadlineFirst are good enough for scheduling balanced-structure applications, since they incur similar execution cost but less scheduling running time.

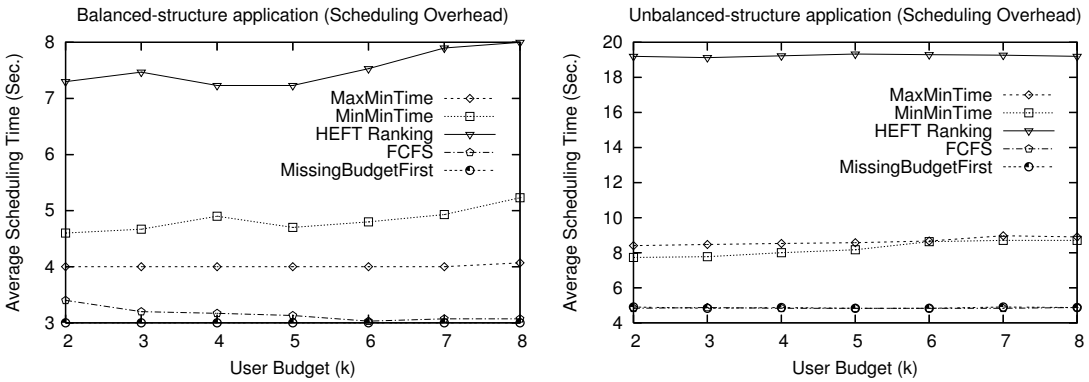


(a) balanced-structure application (b) unbalanced-structure application

Figure 4.13: Comparison of execution costs among five different ranking strategies for scheduling deadline constrained workflow.

Figure 4.13 shows the total execution time of CD employing different ranking strategies: MaxMinTime, MinMinTime, HEFT Ranking, FCFS and MissingBudget-First. Their time optimization performances are measured by the Average Normalized Time (ANT) which is the average value of execution time for executing 30 different random generated workflow applications. The execution time is normalized by the fastest time generated by HEFT.

For the balanced-structure (see Figure 4.13a), five ranking strategies produce similar results. However, for the unbalanced-structure application (see Figure 4.13b), HEFT Ranking produces faster schedules while MinMinTime performs slightly worse than MaxMinTime, FCFS and MissingBudgetFirst. However, as shown in Figure 4.14, it also takes longer for HEFT ranking to return the results.



(a) balanced-structure application (b) unbalanced-structure application

Figure 4.14: Scheduling overhead of five budget constrained ranking strategies.

4.7 Related Work

Many heuristics have been investigated by several projects for scheduling workflows on Grids. The heuristics can be classified as either *task level* or *workflow level*. Task level heuristics make scheduling decisions based only on the information about a task or a set of independent tasks, while workflow level heuristics take into account the information of the entire workflow. *Min-Min*, *Max-Min* and *Sufferage* are three major task level heuristics employed for scheduling workflows on Grids. They have been used by Mandal et al [109] to schedule EMAN bio-imaging applications. Blythe et al [21] developed a workflow level scheduling algorithm based on *Greedy Randomized Adaptive Search Procedure* (GRASP) [61] and compared it with Min-Min in compute- and data-intensive scenarios. Another two workflow level heuristics have been employed by the ASKALON project [132][181]. One is based on *Genetic Algorithms* and the other is a *Heterogeneous-Earliest-Finish-Time* (HEFT) algorithm [167]. Sakellariou and Zhao [140] developed a low-cost rescheduling policy. It intends to reduce the rescheduling overhead by conducting rescheduling only when the delay of a task impacts on the entire workflow execution. However, these works only attempt

to minimize workflow execution time and do not consider other factors such as monetary execution cost.

Several strategies have been proposed to address scheduling problems based on users' deadline and budget constraints. Nimrod-G [9] schedules independent tasks for parameter-sweep applications to meet users' budget. A market-based workflow management system proposed by Geppert et al [72] locates an optimal bid based on the budget of the current task in the workflow. However, the proposed work in this chapter schedule a workflow, which consists of a set of interdependent tasks, according to a specified budget and deadline of the entire workflow. More recently, iterative processing based heuristics such as backtracing [118], and *LOSS* and *GAIN* [141], have been proposed to solve constrained optimization problems. They iteratively amend the schedule optimized for one factor to satisfy the other factor in the way that it can gain maximum benefit or minimum loss. However, they need go through many iterations to modify and recomputed the current schedule to meet the constraint and thus result in large scheduling computation time. In contrast, the work proposed in this chapter makes a scheduling decision for each task once based on a planned sub-deadline.

4.8 Summary

Utility Grids enable users to consume utility services transparently over a secure, shared, scalable and standard world-wide network environment. Users are required to pay for access services based on their usage and the level of QoS provided. In such "pay-per-use" Grids, workflow execution cost must be considered during scheduling based on users' QoS constraints.

This chapter has presented characteristics of utility Grids and modeled its scheduling problem formally. Deadline and budget constrained scheduling heuristics have also been proposed. The deadline constrained scheduling is designed for time critical applications. It attempts to minimize the monetary cost while delivering the results to users before a specified deadline. The budget constrained scheduling is aimed to complete workflow executions based on the budget available for users.

The proposed heuristics distributes overall deadline and budget over each task and then optimizes the execution performance of each task based on their assigned subdeadline and subbudget. The heuristics have been evaluated against others including Greedy-Time, Greedy-Cost, and back-tracking heuristics through simulation in terms of performance and scheduling time. The results show that the proposed approaches provide best performance in short running time.

The heuristics developed in this chapter are also served as a basis for investigating constrained based evolutionary scheduling algorithms developed in following chapters.

Chapter 5

Workflow Scheduling using Genetic Algorithms

As mentioned in Chapter 4, multiple factors such as time, monetary cost, reliability and security need to be considered while making a scheduling decision on utility Grids. A simple heuristic cannot satisfy all cases. Especially as more factors are involved, it may not be even feasible to develop a simple heuristic to solve such complex problems. Therefore, it is necessary to investigate metaheuristic approaches which are capable of being applied to complex domains for workflow scheduling problems.

In this chapter, a genetic algorithm-based workflow scheduling approach is presented. First, the representation of scheduling solutions in the search space is discussed, and then a fitness function is proposed followed by the design of genetic operations. The performance of the genetic algorithm (GA) is observed by comparing against non-GA approaches.

5.1 Genetic Algorithms

Genetic algorithms (GAs) provide robust search techniques that allow a high-quality solution to be derived from a large search space in polynomial time, by applying the principle of evolution [73]. A genetic algorithm combines the exploitation of best solutions from past searches with the exploration of new regions of the solution space. Any solution in the search space of the problem is represented by a set of parameters

called an *individual*. A genetic algorithm maintains a *population* consisting of a set of individuals that evolves over generations. The quality of an individual in the population is determined by a *fitness function*. A fitness function quantifies the optimality of an individual and the value of the fitness function for an individual indicates how good it is, compared to others in the current population. A typical genetic algorithm consists of the following steps:

1. Create an initial population consisting of randomly generated individuals.
2. Generate new offspring by applying genetic operators (e.g. crossover and mutation), one after the other.
3. Evaluate the fitness value of each individual in the population and select individuals into next generation.
4. Repeat steps 2 and 3 until the algorithm meets a termination condition such as when a certain number of generations are reached, or a satisfactory solution to the problem is found.

Using genetic algorithms to solve the workflow scheduling problem requires the determination of the representation of individual in the population, the fitness function and genetic operations. The details of each of these requirements are presented in following sections.

5.2 Individual Representation

For the workflow scheduling problem, a feasible solution is required to meet the following constraints:

- A task can be ready to start only after all its predecessors have completed.
- Every task appears once and only once in the schedule.
- Each task must be allocated to one available time slot of a service capable of executing the task.

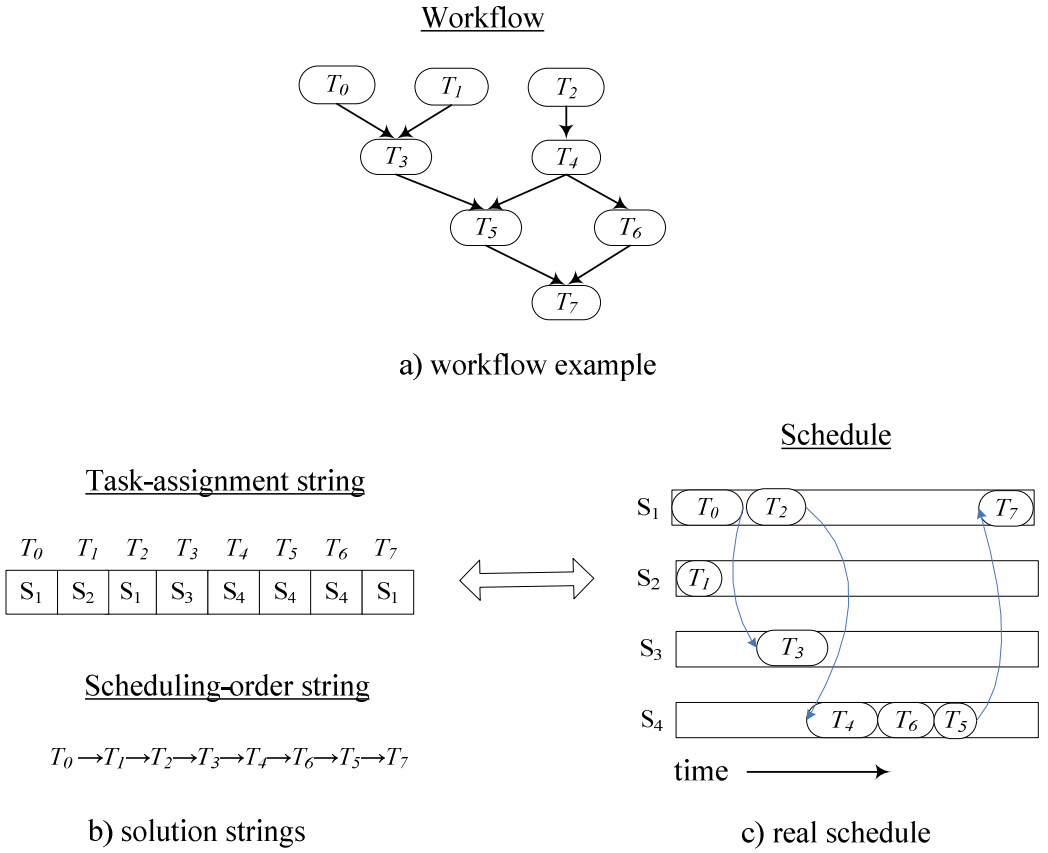


Figure 5.1: Workflow representation in the search space.

Each individual in the search space represents a feasible solution to the problem and consists of a vector of task assignments. Each task assignment includes four elements: *taskID, serviceID, startTime,* and *endTime*. *taskID* and *serviceID* to identify the service to which each task is assigned. *startTime* and *endTime* indicate the time frames allocated on the service for the task execution. However, evolving time frames during the genetic operation may lead to a very complicated situation, because any change made to a task could require adjusting the values of *startTime* and *endTime* of its successive tasks. Therefore, the *operation strings* used for genetic manipulation are simplified by ignoring time frames. The operation strings encode only the mapping of tasks and services. After genetic manipulation, a time slot assignment algorithm is deployed to transfer operation strings to a feasible schedule (see Section 5.7).

The operation strings for the workflow scheduling problem are required to encode the service allocation for each task. In a workflow, the execution order of interdependent tasks is controlled by their dependencies, e.g. a task is always executed after its immediate parent tasks. However, many independent tasks, e.g. T_3 and T_4 in the example workflow shown in Figure 5.1, may compete for the same time slot on a service. Different execution priorities of independent tasks within a workflow may affect the performance of workflow execution significantly. Therefore, encoded operation strings are also required to show the order of task assignments on each service.

Figure 5.1b shows two operation strings: *task-assignment string* and *scheduling-order string*. The task-assignment string encodes the allocation for each task. For example, task T_0 is assigned to service S_1 and T_5 is assigned to S_3. The scheduling-order string encodes the order to schedule tasks. For example, T_1 is scheduled after T_0, T_2 is scheduled after T_1 and so forth. The order in the scheduling-order string must satisfy task dependencies; that means a task should not be placed before its predecessors. However, the main reason for having the scheduling-order string is not only to code task dependencies but also the execution priorities for independent tasks which are assigned to the same service. For example, two independent tasks T_0 and T_2 are assigned on S_1, but T_0 is executed first according to the scheduling-order string.

5.3 Initial Solutions

An initial solution is used for evolutionary algorithms as a starting point to search from. In general, initial solutions are randomly generated. Algorithm 5.1 is the algorithm to generate a random solution, and its corresponding task-assignment and scheduling-order string. It schedules workflow tasks level-by-level in order to generate a scheduling-order string satisfying task dependencies. The scheduler starts from scheduling the entry task since it does not have any parent task. Other tasks become ready tasks to be scheduled once all its parent tasks have been scheduled. All ready tasks are managed by a ready task queue denoted as Q. The scheduler selects a ready task from Q to schedule at random. As shown in Line 1 of the algorithm, the entry

task is the first task which is put in Q to schedule. A task t is selected from Q to be scheduled (Line 3). After t is scheduled, the scheduler put its child tasks into Q if their parent tasks have been scheduled (Line 6-7).

Algorithm 5.1: Random schedule generation
Input: *A workflow graph* $\Omega(\Gamma,\Lambda)$
Output: *Task-assignment string and scheduling-order string*

```
1  Q ← get the entry task
2  while Q is not empty do
3     Randomly choose one task t out of Q
4     Randomly select a service from those that are able to
      run t
5     Update task-assignment string and scheduling-order
      string
6     childT ← get all ready child tasks of t
7     Q = Q ∪ childT
8  end while
```

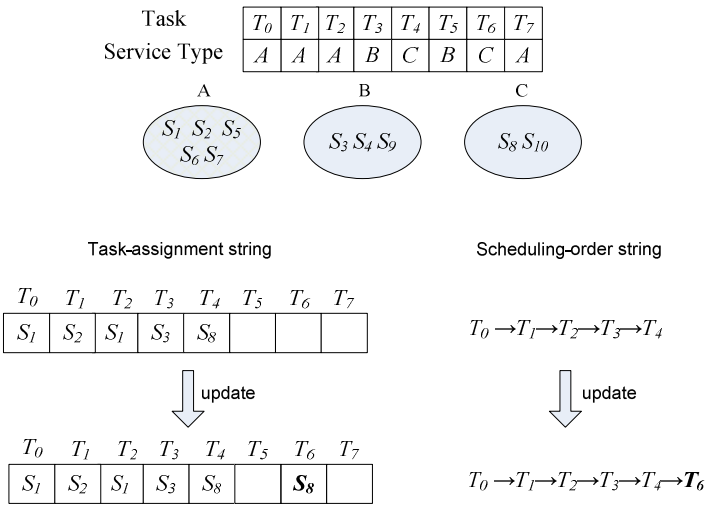


Figure 5.2: Illustration of updating genetic operation strings.

Given the heterogeneous nature of execution environments required by workflow tasks, processing services are classified into groups. Each service group provides a certain type of service that satisfies the execution condition of a task in the workflow. In the example shown in Figure 5.2, different tasks in the workflow require different types of services and all services are grouped together to support service type named *A*, *B*, and *C*. For example, T_0, T_3 and T_4 require services of type *A*, *B* and *C* respectively. A task is selected from *Q* and scheduled on a service which is selected from its service group at random in which each member is capable to execute the task (Line 3-4). After each task assignment, scheduled service allocation and task is appended to the task-assignment string and scheduling-order string (Line 5). Figure 5.2 illustrates the update of operation strings for assigning T_6 to S_8.

In addition to random generated solutions, the results generated by heuristics such as *Greedy Cost - Time Distribution* (TD) and *Greedy Time - Cost Distribution* (CD) proposed in Chapter 4 can also be employed into the initial population. The genetic algorithm can take them as starting points to continue to search better solutions.

5.4 Evolutionary Operations

Evolutionary operators are used to generate new solutions based on solutions found so far. Crossover and mutation are two major operators.

5.4.1 Crossover

Crossovers are used to create new solutions by rearranging parts of the existing solutions in the current population. The idea behind the crossover is that the fittest solution may result from the combination of two of the current fittest solutions. Two-point crossover is implemented for the task-assignment string and illustrated in Figure 5.3. It is implemented as follows:

(1) Two parents are chosen at random in the current population.

(2) Two random points are selected from the task-assignment strings to form a *crossover window*.

(3) All tasks included in the crossover window are chosen as successive crossover points.

(4) The service allocations of all tasks within the crossover window are exchanged.

Figure 5.3: Illustration of crossover operation.

After crossover, two new offspring are generated by combining task assignments taken from the two parents. As shown in Figure 5.3, *offspring 1* inherits tasks assignments of T_0, T_1, T_2, and T_7 from *parent 1*, while the task assignments of the rest tasks are taken from parent 2.

5.4.2 Mutation

In the genetic algorithm, mutations occasionally occur in order to allow a child to obtain features that are not possessed by either of its parents. This process helps the algorithm to explore new and possibly better genetic material than previously considered. Two types of mutations are developed, namely *reordering-mutation* and *replacing-mutation* for the workflow scheduling problem. The mutation operators are applied to the chosen solutions with certain probabilities.

A reordering-mutation aims to change the execution order of independent tasks that compete for a same time slot. It is implemented as follows:

(1) A task in the execution-order string is randomly selected.

(2) An alternative position for this task is randomly selected on its same service from all valid positions on the service.

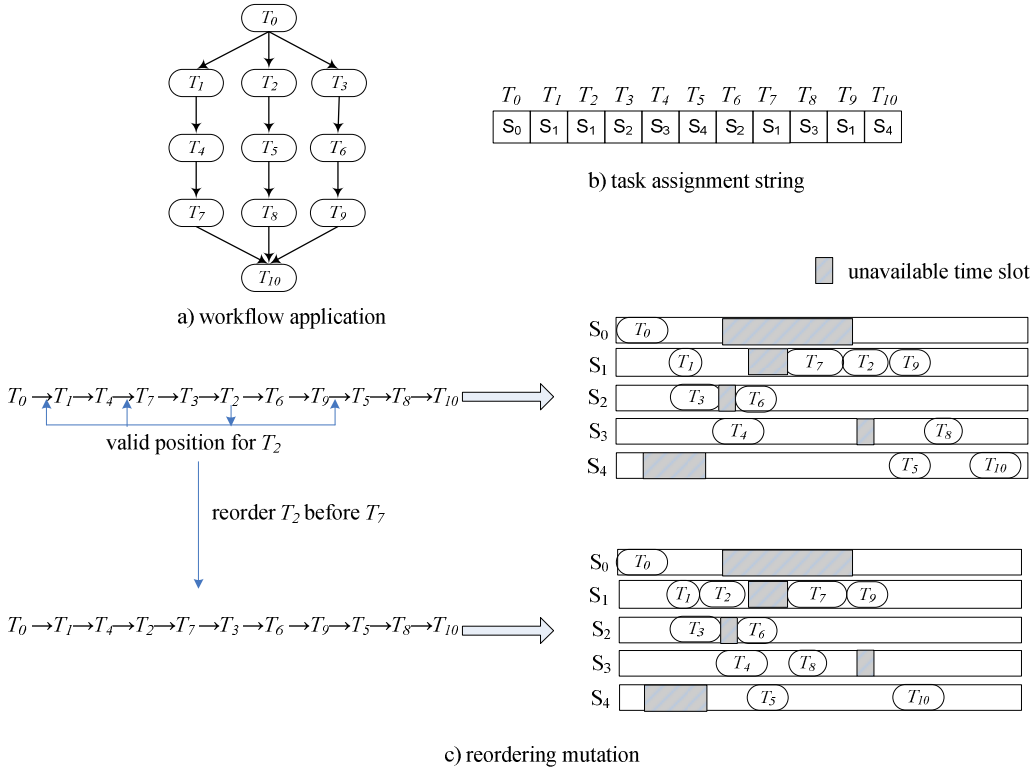


Figure 5.4: Illustration of reordering mutation.

A valid position for a task is a place where the task can be inserted such that its dependencies are satisfied. An example of a reordering-mutation is shown in Figure 5.4. In the example, T_2 is selected. The possible alternative positions for T_2 can only be places where are after its parent task T_0 and before its child task T_5. Among these possible positions, the positions near a task which is not assigned on the same service as that of T_2 may not affect the schedule. For example, if we swap the order of T_2 and T_3 on the original scheduling-order string in Figure 5.4c, the schedule remains same. This is because that they are allocated on different services and they will be assigned to earliest available time slot on their assigned service only. Therefore, valid positions for reordering-mutation are places where are next to tasks assigned on the same service with the mutated task. As shown in Figure 5.4c, the valid positions for T_2 are

places before T_7 and T_1 and after T_9. In the example, the task T_2 is selected and reordered to execute before T_7, such that the execution order of tasks on S_1 is changed, impacting the entire scheduling performance. As shown in Figure 5.3c, the performance of offspring is improved.

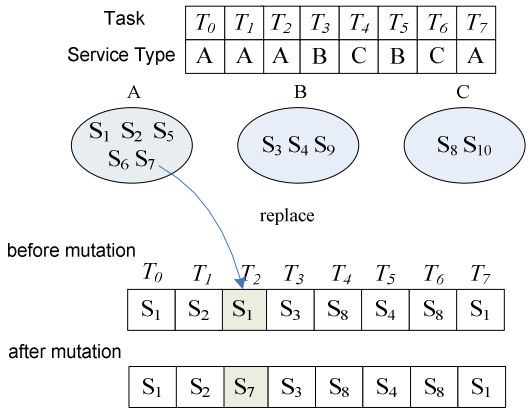


Figure 5.5: Illustration of replacing mutation operation.

A replacing-mutation aims to re-allocate an alternative service to a task in a solution. It is implemented as follows:

(1) A task is randomly selected in the task-assignment string.

(2) An alternative service which is capable of executing the task is randomly selected to replace the current task allocation.

An example of replacing mutation is shown in Figure 5.5. The mutation process randomly selects S_2 in the service group of type A and re-allocates it to T_2.

5.5 Fitness Function

A fitness function is used to measure the quality of the individuals in the population for a given optimization objective. As the goal of the scheduling is to minimize the performance based on two factors, time and monetary cost, the fitness function separates evaluation into two parts: *cost-fitness* and *time-fitness*.

For the budget constrained scheduling, the cost-fitness component encourages the formation of the solutions that satisfy the budget constraint. For the deadline constrained scheduling, it encourages the genetic algorithm to choose individuals with less cost. The cost fitness function of an individual I is defined by:

$$F_{\cos t}(I) = \frac{c(I)}{B^{\alpha}(maxCost^{(1-\alpha)})}, \quad \alpha = \{0,1\} \tag{5.1}$$

where $c(I)$ is total cost of I and *maxCost* is the most expensive solution of the current population, and B is the budget of the workflow. α is a binary variable, where α is set 0 for deadline constrained scheduling and set 1 for budget constrained scheduling.

For the budget constrained scheduling, the time-fitness component is designed to encourage the genetic algorithm to choose individuals with earliest completion time from the current population. For the deadline constrained scheduling, it encourages the formation of individuals that satisfy the deadline constraint. The time fitness function of an individual I is defined by:

$$F_{time}(I) = \frac{t(I)}{D^{\beta}(maxTime^{(1-\beta)})}, \ \beta = \{0,1\} \tag{5.2}$$

where $t(I)$ is the completion time of I, *maxTime* is the largest completion time of the current population, and D is the deadline of the workflow. β is a binary variable, where β is set 0 for budget constrained scheduling and set 1 for deadline constrained scheduling.

For the deadline constrained scheduling problem, the final fitness function combines two parts and it is expressed as:

$$F(I) = \begin{cases} F_{time}(I), & \text{if } F_{time}(I) > 1 \\ F_{cost}(I), & \text{otherwise} \end{cases} \tag{5.3}$$

For the budget constrained scheduling problem, the final fitness function combines two parts and it is expressed as:

$$F(I) = \begin{cases} F_{cost}(I), & \text{if } F_{cost}(I) > 1 \\ F_{time}(I), & \text{otherwise} \end{cases} \quad (5.4)$$

5.6 Selection Scheme

After the fitness evaluation process, the new individuals are compared with the previous generation. The Selection process is then conducted to retain the fittest individuals in the population, as successive generations evolve. The following methods for selecting the fittest individuals are selected for investigating the impact on the workflow scheduling problem.

- Roulette Wheel Selection

  The roulette wheel selection assigns each individual to a slot of a roulette wheel and the slot size occupied by each individual is determined by its fitness. A fitter individual occupies larger slot so that it more likely to be selected. The size of the slot for individual I is defined by:

$$p_I = \frac{F'(I)}{\sum_{i=1}^{n} F'(i)} \quad (5.5)$$

$$\text{and} \quad F'(I) = \frac{\sum_{i=1}^{n} F(i) - F(I)}{\sum_{i=1}^{n} F(i)} \quad (5.6)$$

  where n is the number of all individuals. Since in the workflow scheduling case, an individual with a small value of fitness is fitter than the one with a large value of fitness, the slot size of an individual is reverse proportional to its fitness value.

- Rank Selection

  The rank selection process first sorts all individuals from worst to best according to their fitness and then assigns slots based on their rank. The slot size of an individual I is defined by:

$$p_I = \frac{R(I)}{\sum_{i=1}^{n} R(i)} \tag{5.7}$$

where $R(I)$ is the rank value of I and n is the total number of all individuals for selection.

- Elitism-Roulette Wheel Selection

  The elitism-roulette wheel selection first copies the fittest individual into the next generation and then using the roulette wheel selection to construct the rest of the population.

- Elitism-Rank Selection

  The elitism-rank selection first copies the fittest individual into the next generation and then using the rank selection to construct the rest of the population.

5.7 Time Slot Assignment

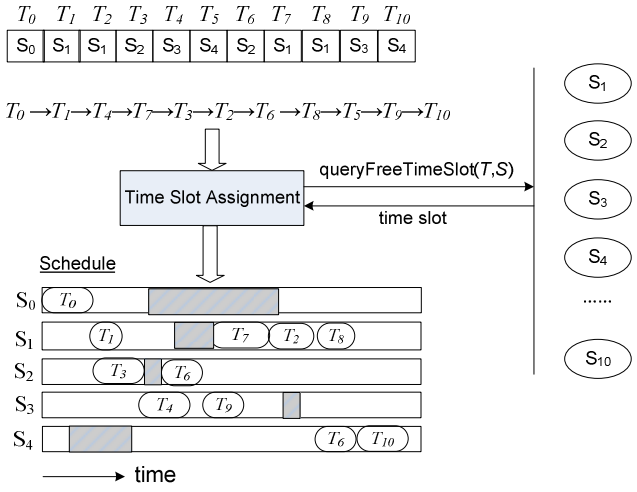


Figure 5.6: Time slot assignment.

The operation strings representing offspring produced by crossover and mutation operators are not a real schedule, since we ignore the time frames during genetic ma-

nipulation. A time slot assignment process is developed to transfer task-assignment string and scheduling-order string to a feasible solution. As illustrated in Figure 5.6, it queries available time slots from services based on the information of resource allocations and task execution orders encoded in strings, and assign a time slot to each task to produce a schedule satisfying the conditions of a feasible solution defined in Section 5.2.

Algorithm 5.2: time slot assignment algorithm
Input: *Scheduling-order string and task-assignment string*
Output: *A feasible schedule*

```
1 T ← get the first task from the scheduling-order string
2 while any task in the scheduling-order string has not been
  assigned do
3     S ← get the service allocation of T from the
      task-assignment string
4     compute the ready time of T on S
5     query and assign a free slot on S for T
6     T ← get the next task from the scheduling-order string
7 end while
```

Algorithm 5.2 shows the pseudo-code of the time slot assignment. It assigns tasks in the order of scheduling-order string (Line 1, 6). The service allocation of tasks is obtained from task-assignment string (Line 3). Before assigning a task on the specified service, it firstly computes the ready time for the execution of the task on the service by considering the input data transmission time and the end time of parent tasks (Line 4). After that, it queries free slots on the service and assign the earliest free slot to the task (Line 5).

5.8 Experiments

In order to evaluate the proposed genetic algorithm, the algorithm is implemented and compared with the non-GA heuristics (i.e. TD and CD) proposed in Chapter 4 on the

two workflow structures, balanced and unbalanced applications as described in Section 4.4.

Table 5.1: Default parameters.

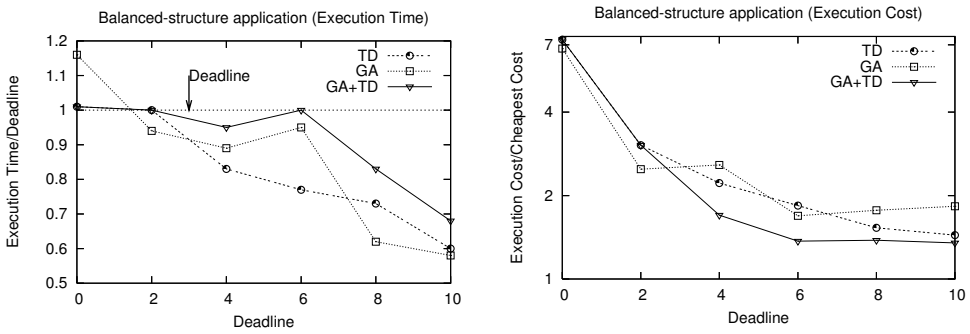
Parameter	*Value/type*
Population size	10
Maximum generation	100
Crossover probability	0.9
Reordering mutation probability	0.5
Replacing mutation probability	0.5
Selection scheme	elitism-rank selection
Initial individuals	randomly generated

The genetic algorithm is investigated by starting with two different initial populations. One initial population consists of randomly generated solutions, while the other initial population consists of a solution produced by either CD or TD together with other randomly generated solutions. The solution produced by CD is used as one of individual in the initial population to solve the budget constrained optimization problem and the solution produced by TD is used for solving deadline constrained problem. In the result presentation, the results generated by GA with a completely random initial population is denoted by GA, while the results generated by GA which include an initial individual produced by the CD and TD heuristics are denoted as GA+CD and GA+TD respectively. The parameter settings used as the default configuration for the proposed genetic algorithm are listed in Table 5.1.

5.8.1 Deadline Constrained Scheduling

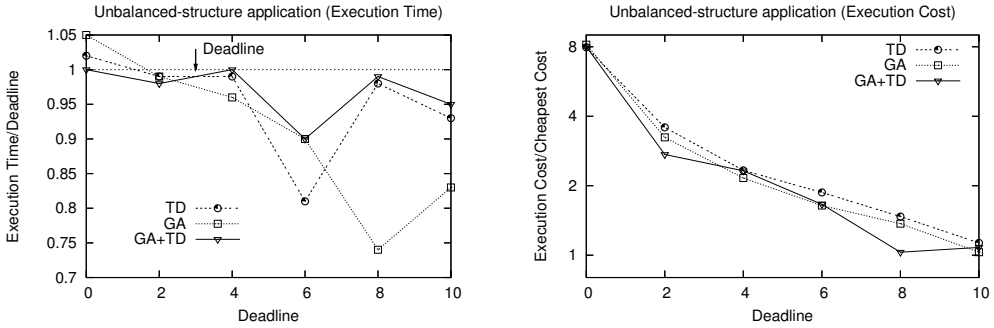
Figure 5.6 and Figure 5.7 compare the execution time and cost of using three scheduling approaches for scheduling the balanced-structure application and unbalanced structure application with various deadlines respectively. The set deadlines here are selected based on Equation 4.12. As the same as in Chapter 4, we use k to represent deadline levels in the result graphs. As k increases, the deadline is more relaxed. In addition, the normalized values are also used in result graphs. The execution time is normalized by the corresponding deadline. Therefore, if the normalized execution time is higher than 1, the deadline is not satisfied. The execution cost is normalized by

the cheapest cost required to execute the workflow on the testbed. The cheapest cost can be computed by a cost optimization scheduling such as *GreedyCost*. As described in Section 4.6, *GreedyCost* is used to find minimum execution cost of scheduling the workflow without considering its execution time.



(a) execution time (b) execution cost

Figure 5.7: Normalized execution time and cost for scheduling balanced-structure application.



(a) execution time (b) execution cost

Figure 5.8: Normalized execution time and cost for scheduling unbalanced-structure application.

We can see that it is hard for both GA and TD to successfully meet the low deadline individually. As shown in Figure 5.7a and 5.8a, the normalized execution times produced by TD and GA exceed 1 at tight deadline ($k = 0$), and GA performs worse than TD since its values is higher than TD, especially for balanced-structure applica-

tion. However, the results are improved when incorporating GA and TD together by putting the solution produced by TD into the initial population of GA. As shown in Figure 5.8a, value of GA+TD is much lower than that of GA and TD at the tight deadline.

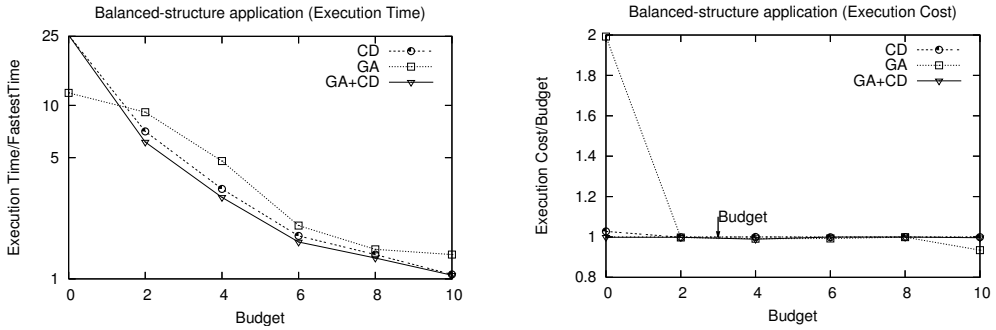
As deadline increases, both GA and TD can meet deadline (see Figure 5.7a and 5.8a) and GA can outperforms TD. For example in Figure 5.7, execution time and cost generated by GA at $k = 2$ are lower than that of TD. However, as shown in Figure 5.7b the performance of GA is reduced and TD can performance better, once the deadline becomes very large. We can observe this in Figure 5.7b when $k = 8$ and 10. In general, GA+TD algorithm performs the best.

5.8.2 Budget Constrained Scheduling

A comparison of the execution time and cost resulted by the three scheduling methods for scheduling the balanced-structure application and unbalanced-structure application with various budgets is shown in Figure 5.9 and Figure 5.10, respectively. The set budgets here are selected based on Equation 4.13 described in Chapter 4. In the result graph we use k to represent budget levels. As k increase the budget is more relaxed. Similar to the deadline constraint experiments, the normalized values are also used in the result graphs. The execution cost is normalized by the corresponding budget (see Figure 5.9b and 5.10b). Therefore, if the normalized execution cost is higher than 1, the budget is not satisfied. The execution time is normalized by the fastest execution time which can be achieved on the testbed. The fastest execution time can be obtained by executing a time optimization scheduling algorithm such as HEFT. As described in Section 4.6, HEFT is used to find the minimum time that can be achieved on the testbed without considering its execution cost.

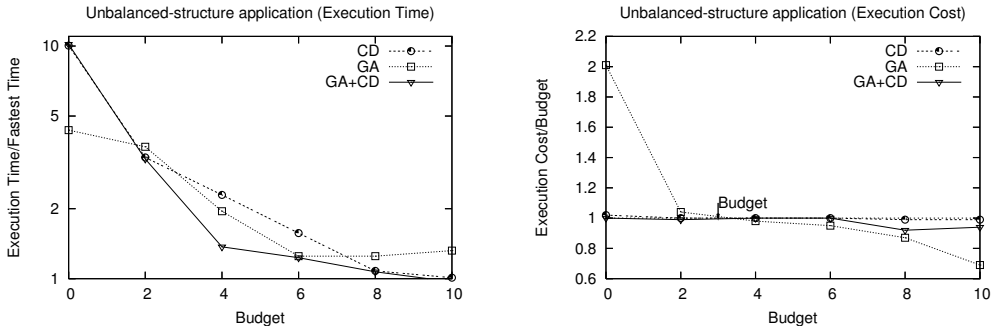
We can see that both GA and CD cannot satisfy the low budget constraint ($k = 0$). CD is slightly over budget whereas GA performs the worst results (see Figure 5.9a and 5.10a). However, the results are improved if we combine GA and CD together by putting the solution produced by CD into the initial population of the GA. As budget

increases, CD performs better for the balanced-structure application (see Figure 5.9b), whereas GA performs better than CD for the unbalanced-structure application (see Figure 5.10b). This shows that budget distribution used by CD is more accurate for balanced-structure. However, its budget constraint distribution problem for the unbalanced-structure application can be released when the budget is very high. As shown in Figure 5.10a, CD performs better than the GA for both applications after $k = 8$. Moreover, by combining the two approaches, GA+CD can improve the performance generated by CD.



(a) execution time (b) execution cost

Figure 5.9: Normalized execution time and cost for scheduling balanced-structure application.



(a) execution time (b) execution cost

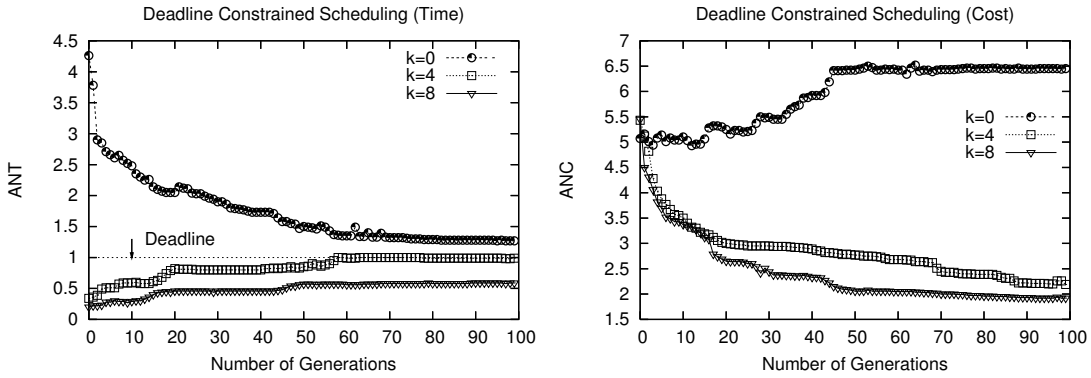
Figure 5.10: Normalized execution time and cost for scheduling unbalanced-structure application.

5.8.3 Effect of the number of generations

We also observe the performance of GA when the number of generation cycles is altered. Figure 5.11 shows the evolution of the average normalized execution time (ANT) and average normalized execution cost (ANC) of the population during 100 generations at tight deadline ($k = 0$), medium deadline ($k = 4$), and high deadline ($k = 8$). For the tight deadline case, ANT of the population as shown in Figure 5.11a is reduced significantly in order to meet the tight deadline as the number of generations increases. Consequently, their corresponding execution cost as shown in Figure 5.11b is increased; this is because individuals with more expensive schedules are selected in order to decrease the execution time. However, GA is still not able to meet such tight deadline within 100 generations. For the medium and high deadline case, the deadlines are easily satisfied even in the initial population since the deadline is relatively relaxed. Therefore, the objective of GA is to reduce the execution cost. As shown in Figure 5.11b, ANC values of medium and relaxed deadline cases are decreased as the number of generation increases, at the meantime, their ANT values are increased but are kept under 1.

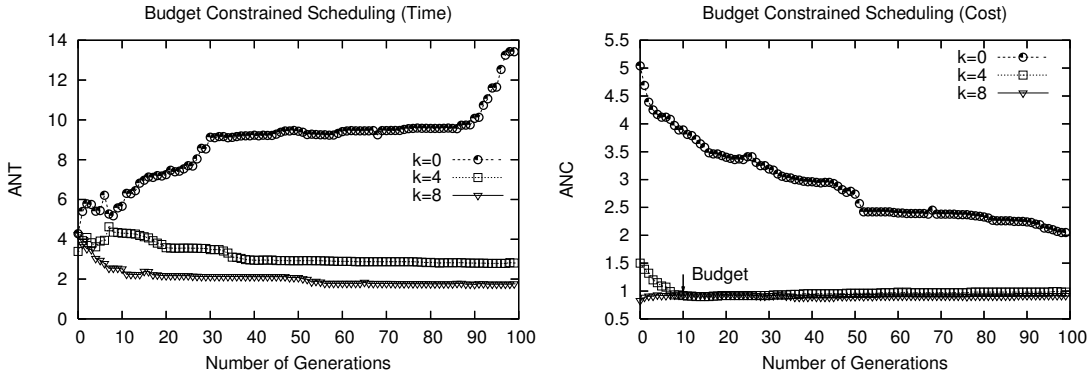
Figure 5.12 shows the evolution of the average normalized execution time (ANT) and average normalized execution cost (ANC) of the population during 100 generations at tight budget ($k = 0$), medium budget ($k = 4$), and high budget ($k = 8$). If the budget constraints cannot be satisfied, GA first to reduce the execution cost to meet the budgets. As shown in Figure 5.12b, ANC values produced by GA at tight budget is significantly reduced as the number of generations increases. Consequently, the execution time as shown in Figure 5.12a is increased; this is because individuals which process slower are selected in order to decrease the execution cost. Similar to the tight deadline case, it is hard to GA to meet the tight budget within 100 generations. For the medium budget case, GA first is to reduce execution cost to meet the budget. As shown in Figure 5.12b, the ANC value is reduced from 1.5 to 1 during generation 0-9 and their corresponding ANT value is slightly increased as shown in Figure 5.12a. However, once it has found individuals which are able to complete the execution within the budgets, it starts to optimize the execution time for successive

generations. For the relaxed budget case, most individuals in the initial population can meet the budget since the budget is very large. Then the objective of GA targets on minimizing the execution time. As shown in Figure 5.12a, its corresponding ANT values are decreased as the number of generation increases.



(a) execution time (b) execution cost

Figure 5.11: Evolution of execution time and cost during 100 generations for deadline constrained scheduling application.



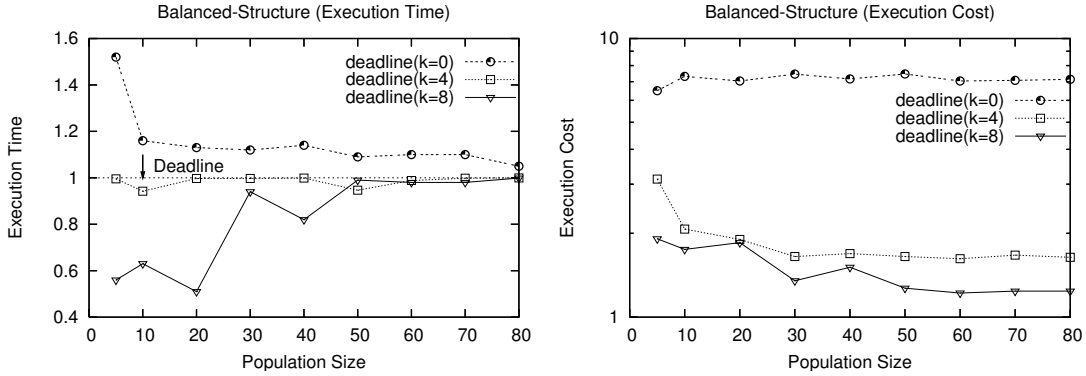
(a) execution time (b) execution cost

Figure 5.12: Evolution of execution time and cost during 100 generations for budget constrained scheduling.

5.8.4 Effect of the size of population

Figure 5.13 shows execution time and cost produced by GA for sizes of the population ranging from 5 to 80 at tight deadline ($k = 0$), medium deadline ($k = 4$), and high

deadline ($k = 8$). For the tight deadline case, we observe from Figure 5.13a, with larger population size, GA could meet a tight deadline. As shown in Figure 5.13a, the completion time is much closer to the deadline as the size of population increases. For the medium and high deadline cases, as shown in Figure 5.13b, large population size gives GA more opportunity to find cheaper solutions while meeting the deadline. However, the scheduling overhead is increased as the size of population increases. Figure 5.14 shows, as the size of the population is increased from 5 to 80, the running time of GA increases from 77 seconds to 1600 seconds.



(a) execution time (b) execution cost

Figure 5.13: Normalized execution time and cost for sizes of the population ranged from 5 to 80 at medium deadline.

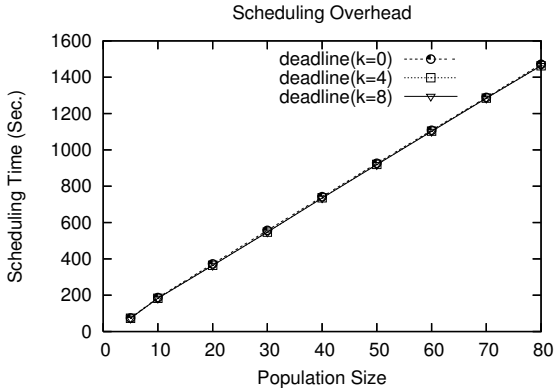
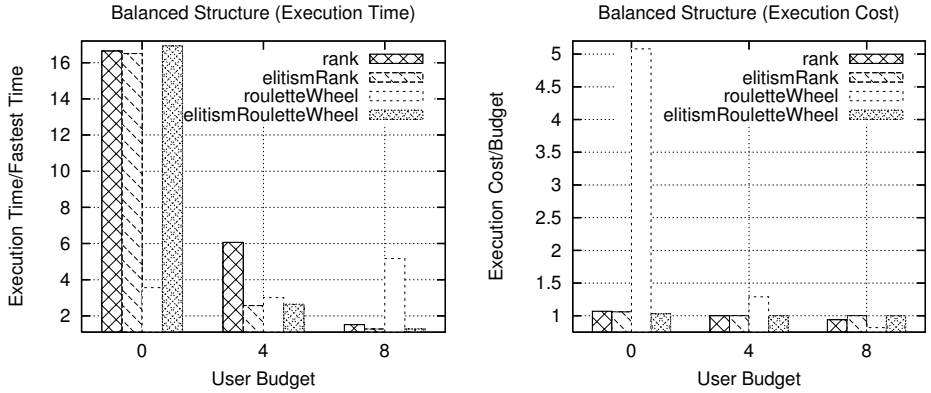


Figure 5.14: Scheduling overhead for the sizes of the population ranged from 5 to 80.

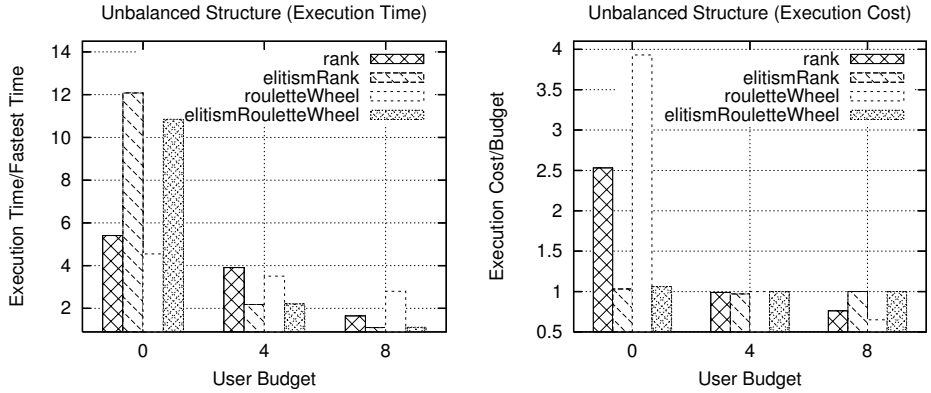
5.8.5 Impact of Selection Scheme

The default selection scheme used for previous experiment is elitism rank selection. Experiments of comparison of different selection scheme are also conducted. In these experiments, each scheme is carried out to schedule 10 random generated balanced- and unbalanced-structure workflows.



(a) execution time (b) execution cost

Figure 5.15: Comparison of execution cost and time among four selection scheme for deadline constrained scheduling on balanced-structure workflows.

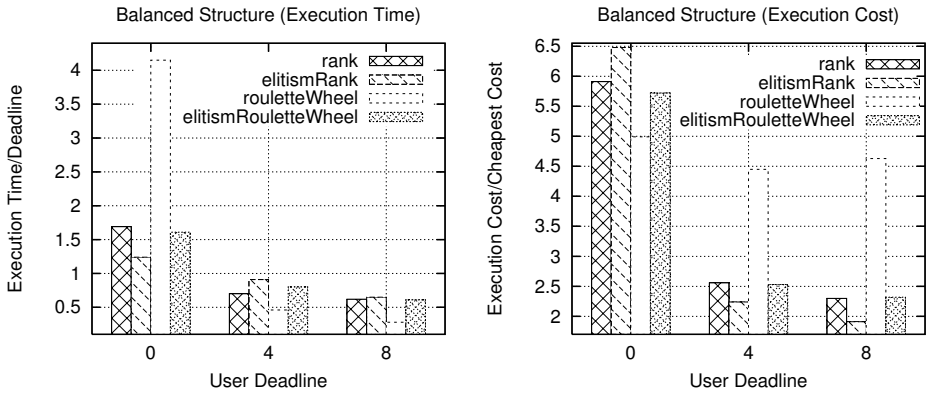


(a) execution time (b) execution cost

Figure 5.16: Comparison of execution cost and time among four selection scheme for deadline constrained problem on unbalanced-structure workflows.

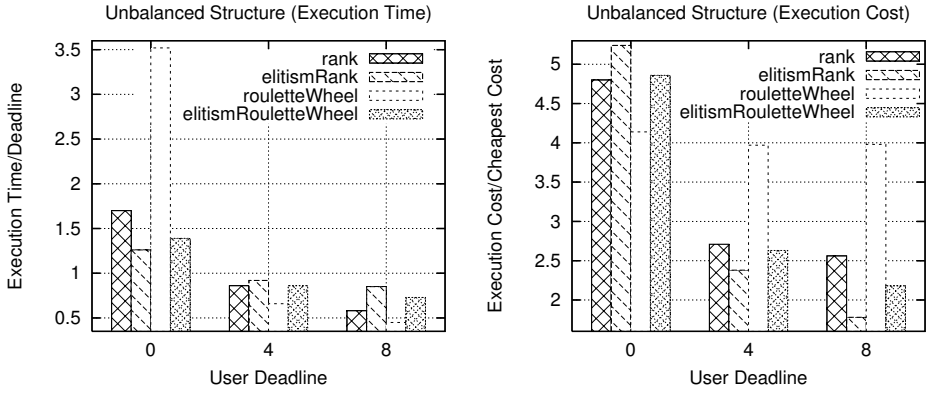
Figure 5.15 and Figure 5.16 show the comparison results for scheduling balanced-structure and unbalanced-structure workflows with various deadlines. We can observe that roulette wheel performance worst. At the tight deadline ($k = 0$), its exe-

cution time is significantly greater than the deadline (see Figure 5.15a and 5.16a). At the medium ($k = 4$) and relaxed deadline ($k = 8$), its execution cost is much higher than others. Comparing with rank selection and elitism rank selection, elitism rank selection is better. It can satisfy tight deadlines and achieve lower execution cost. In addition, it also performs slightly better than elitism roulette wheel.



(a) execution time (b) execution cost

Figure 5.17. Comparison of execution cost and time among four selection scheme for budget constrained problem on balanced-structure workflows.



(a) execution time (b) execution cost

Figure 5.18: Comparison of execution cost and time among four selection scheme for budget constrained problem on unbalanced-structure workflows.

Figure 5.17 and Figure 5.18 show the comparison results for scheduling balanced-structure and unbalanced-structure workflows with various budgets. The results are similar to the deadline constrained scheduling, roulette wheel perform worst while elitism based approaches are better than the non-elitism approaches.

5.9 Related Work

The genetic algorithm approach has been used to solve scheduling problems in existing literatures. Yamada and Nakano [122][190] have proposed the use of genetic algorithms for job-shop scheduling problems. Binary and symbolic representations were developed in this work. The job shop scheduling problem is to schedule a set of jobs and each job is required to process on a set of machines in sequence. In contrast, a workflow is a set of tasks which are required to be executed on a set of resources. Some tasks need to be executed in sequence on same or different resources, while some tasks need to be executed in parallel. Therefore, problem representation developed for job-shop scheduling cannot be directly applied for workflow scheduling problem.

The representation used in the proposed genetic algorithm is similar to the work proposed by Wang et al [180]. Zomaya et al [202] and Hou et al [85] have also applied genetic algorithms for scheduling DAG based task graphs on multiprocessor systems. Two dimensional strings are used to represent the scheduling problem. One dimension represents the number of processors while the other dimension shows the order of tasks on each processor. This two dimensional representation is also suitable for workflow scheduling. Compared with these two approaches, the advantage of the two dimensional strings is that it is intuitive for obtaining the order of tasks on a resource so that it can facilitate the re-ordering mutation operation, while its disadvantage is that it requires extra operations in the crossover in order to maintain the dependencies of the tasks.

In addition, previous work in multiprocessor systems [85][202] assume that all resources are capable of executing all tasks. That is not true for Grid computing envi-

ronments. The execution environments required by different tasks in Grid workflow applications are highly heterogeneous, and different tasks may also require different application services. This feature of Grid workflow applications also distinguishes genetic operations such as crossover and mutation used in this work from the way they are used in multiprocessor systems.

Prodan and Fahringer [132][133] have employed genetic algorithms to schedule WIEN2k workflow [20] on Grids. Spooner et al [153] have also employed genetic algorithms to schedule sub-workflows in a local Grid site. More recently, Singh et al [145] incorporate GA with Min-Min heuristics to optimize execution costs of provisioning resources for application execution. Compared with these works, the proposed work focuses on optimizing workflow execution based on deadline and budget constraints.

5.10 Summary

Genetic algorithms (GAs) provide robust search techniques that allow a high-quality solution to be derived from a large search space. This chapter applies genetic algorithms to solve deadline or budget constrained scheduling problems. The fitness function has been developed to guide the GA to search solutions which meet the scheduling objective. Reservation…

The proposed GA is evaluated by comparing it with non-evolutionary heuristics, on both balanced and unbalanced workflow structures. The results show that the genetic algorithm performances better for unbalanced structures. This proves that the GA can play more important role for scheduling complex workflow structures, since it makes scheduling decisions based on the performance of the entire workflow. The results of the experiments also show that the scheduling performance can be significantly improved by incorporating the GA with the simple heuristics developed in previous chapter.

The work in this chapter demonstrates that the GA is capable of optimizing a single parameter of the performance of workflow execution while meeting a specified QoS constraint such as deadline and budget. The next chapter presents workflow exe-

cution planning approaches using genetic algorithms and local search to optimize multiple performance parameters, and generates a set of alternative optimized solutions based on a range of QoS constraints.

Chapter 6

Multi-objective Planning for Workflow Execution

Cost-based workflow scheduling heuristics are proposed in previous chapters. They either optimize monetary cost while meeting users' time constraint (deadline) or optimize execution time while meeting users' cost constraint (budget). However, they are only suitable for users who have strict deadline or budget requirements and expect to optimize only one of these execution factors. In many situations, users may need more information such as the range of QoS levels available and associated costs before making decisions. Furthermore, QoS levels and prices offered by service providers may be highly diverse and may not be directly correlated with the utility perceived by the users. For example, users may prefer some assignments which have slightly longer execution times but offer large savings in execution cost.

This chapter focuses on a workflow planning method considering multi-objective trade-off within users' requirements. In this chapter, trade-off between two conflicting objectives – execution time and price, while meeting users' maximum deadline and budget requirements, are investigated.

6.1 Multi-objective Workflow Planning

Figure 6.1 shows the system architecture and interaction between the user and system components. The user first submits their workflow applications to the system with maximum QoS requirements. The system estimates workflow execution through the

workflow execution planner and then returns the results to the user, showing possible execution schedules and their trade-off based on user's requirements. The user chooses the most desirable schedule and provides confirmation. The system then starts to execute the workflow according to the specified execution schedule and reschedule some tasks once the initial states of services are changed.

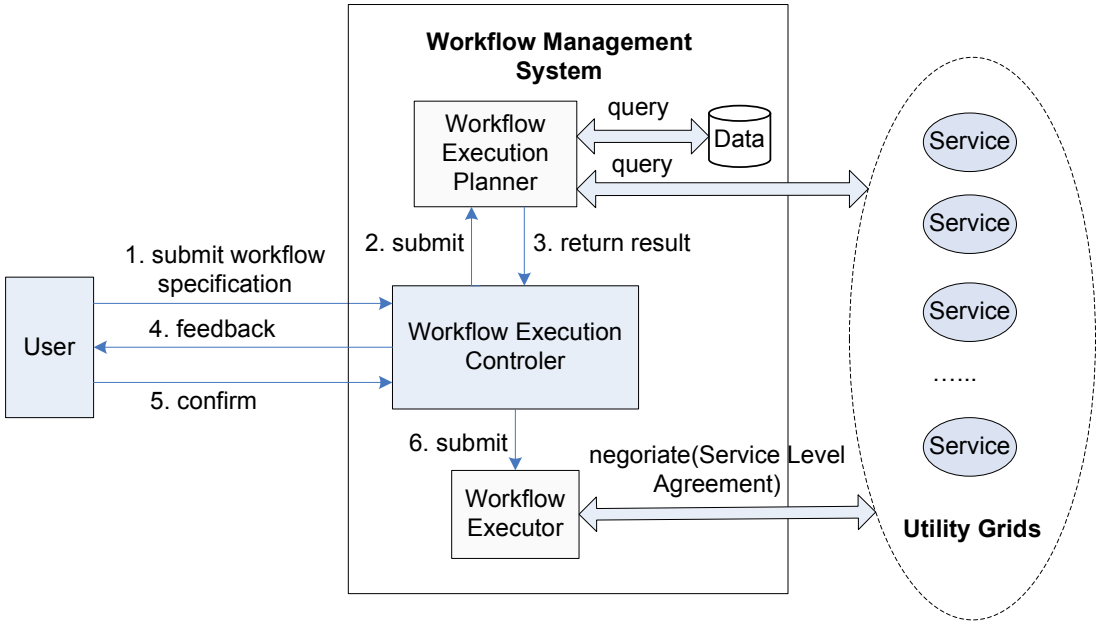


Figure 6.1: Workflow Management System.

Workflow execution planning is a pre-process before workflow execution. It aims to investigate users' execution requirements and generate possible execution schedules. The possible schedules should be optimized execution plans while satisfying users' requirements. Here we formulate the execution optimization problem for deadline and budget constraints during the planning stage. As mentioned in Section 4.2.1, a workflow is modeled as a DAG graph and represented by $\Omega(\Gamma, \Lambda)$, where Γ is the finite set of tasks $T_i\ (1 \le i \le n)$ and Λ is the set of directed edges. We denote that $time(T_i)$ is the completion time of T_i and $cost(T_i)$ is the total cost for processing

T_i. The execution optimization problem is to generate solution I, which maps every T_i onto a suitable service to achieve the multi-objective below:

$$\text{Minimize} \quad Time(I) = \max_{T_i \in \Gamma} \{time(T_i)\} \tag{6.1}$$

$$\text{Minimize} \quad Cost(I) = \sum_{T_i \in \Gamma} \cos t(T_i) \tag{6.2}$$

$$\text{subject to} \quad Cost(I) < B, \tag{6.3}$$

$$Time(I) < D \tag{6.4}$$

where *B* is the cost constraint (budget) and *D* is the time constraint (deadline) required by users for workflow execution.

6.2 Multi-objective Optimization

We formalized the workflow planning problem as a multi-objective optimization problem [42] in which a group of conflicting objectives are simultaneously optimized. As given in Equation 6.1 and 6.2, there are two conflicting objectives: minimize execution time and minimize execution cost. In such problems, there is no single optimal solution but rather a set of potential solutions which are all optimal in some objectives.

6.2.1 Definitions

Mathematically, we can define MOPs as :

$$\text{Minimize} \quad f(x) = \{f_1(x), ..., f_n(x)\} \tag{6.5}$$

where $x \in X$ and *X* is a solution space. We say that the solution is said to dominate another if it is as good as the other solution and better in at least one objective. That is x^* dominates *x*, if and only if

$$\forall i \in [1, ..., n], f_i(x^*) \le f_i(x) \ \wedge \ \exists j \in [1, ..., n], f_j(x^*) < f_j(x) \tag{6.6}$$

Figure 6.2 shows solutions for the minimization problem of two conflicting objectives, $f_1(x)$ and $f_2(x)$. The solution x_3 dominates y_2, because both objective values of x_3 are lower than those of y_2. However, x_3 does not dominate x_4,

since $f_2(x_3) > f_2(x_4)$. We say that x_3 and x_4 are non-dominated solutions. They are optimal in one objective but none of these two solutions is superior to the other with respect to the two objectives.

The set of non-dominated solutions form a non-dominated set *P*, which meets following conditions:

1. Any solution in *P* is non-dominated to any other solution in *P* with respect to all objectives.
2. Any solution in *P* dominates at least one solution not belonging to *P*.

For example, the set of solutions $x_1 - x_4$ is a non-dominated set for the given problem. The image of the non-dominated set is called *non-dominated front*.

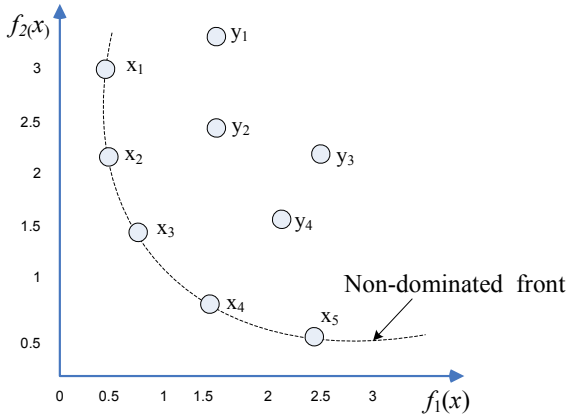


Figure 6.2: Five non-dominated solutions and four dominated solutions for the optimization problem of objectives $f_1(x)$ and $f_2(x)$.

6.2.2 MOP Algorithms

Scheduling of interdependent tasks in distributed heterogeneous computing environments is a well-known NP-hard problem [170]. Many non-evolutionary based heuristics have been proposed to optimize execution time. However, with the increase in the number of parameters and objectives required to be considered, it becomes infeasible to develop a simple heuristic or use a classical method such as linear programming approach to handle the scheduling optimization problem. Evolutionary based algorithms have been widely applied to solve multi-objective optimization

problems in many application domains, such as control systems [63] and network routing [138], and are capable of simultaneously optimizing multiple objectives without combining them into a single scalar objective function.

In this chapter, we focus on applying three well-known algorithms to solve the workflow execution planning problem and compare the algorithms for different workflow structures and users constraint levels. The three algorithms are *elitist Non-dominated Sorting Genetic Algorithm* (NSGAII) [48], *Strength Pareto Evolutionary Algorithm* (SPEA2) [201] and *Pareto Archived Evolution Strategy* (PAES) [91]. They are all evolutionary algorithms and generate better solutions over generations. However, NSGAII and SPEA2 are *population based algorithms* with different evaluation and selection scheme, while PAES is a *local search based algorithm*.

The general high level overview [130] of NSGAII and SPEA2 algorithms is shown in Figure 6.3. Both NSGAII and SPEA2 are genetic algorithm based method and select individuals from a population for reproduction. Both *crossover* and *mutation* genetic operators are performed to generate new offspring. However, these two algorithms are use different evaluation strategies and selection strategies.

NSGAII evaluates solutions based on the values of each objective. It ranks all solutions to form non-dominated fronts according to its values. It first selects non-dominated solutions in the current population as the first level non-dominated front, and selects non-dominated solutions in the rest of the population as next level non-dominated front, the procedures continues until the whole population is classified into non-dominated fronts. It then selects solutions into next generation for reproduction according to their level of non-dominated fronts. The solutions in the first level front have highest priority, and then those in the second level and so forth. Such selection process continues until it finds that the population size would be exceeded if all solutions were added to the next level front. Then, crowding sort is performed on the solutions at this level and finally solutions from the less crowded area are selected.

```
NSGAII and SPEA2 algorithm

1. generate initial population
2. do
3.    perform crossover on individuals in current generation
4.    perform mutation on offspring
5.    evaluate solutions
6.    select individuals to be carried into next generation
7. while(termination condition is not satisfied)
```

Figure 6.3: Generic outline of NSGAII and SPEA2 algorithm.

```
PAES algorithm

 1. generate current solution
 2. create an archive
 2. do
 3.    mutate current solution and generate a candidate
       solution
 4.    evaluate candidate solution
 5.    if candidate solution is not dominated by current
          solution then
 6.       compare candidate solution with archive members
 7.       update archive
 8.       select a new current solution
 9.    end if
10. while(termination condition is not satisfied)
```

Figure 6.4: Overview of PAES algorithm.

On the other hand, SPEA2 uses the degree to which the solution dominates other members in the population and its density estimation to evaluate solutions. The density of a solution in a population is a function of the distance to the k-th nearest solution. Unlike NSGAII, SPEA2 does not select solutions based on their non-dominated levels at each generation. It creates an external *archive* to keep selected individuals for the next generation and first copies non-dominated solutions in current population to the archive. If the size of the archive is exceeded, the solutions in overcrowded areas are removed from the archive; otherwise, it fills the archive with dominated solutions based on their *Euclidean distance* to its nearest neighbor solution.

The overview of PAES is shown in Figure 6.4. PAES uses local search from one current solution to generate a new candidate and compare the current solution with the candidate solution. It continuous to search from the current solution if the candidate is dominated; otherwise it is compared with other archived solutions. In PAES, an archive is maintained to keep the best solutions found so far. Initially there is only one solution in the archive, as the number of generations increases, the archive is updated by adding good candidates and removing dominated archived members. In addition to comparing solutions by using dominance criteria, density estimation is also applied. In PAES, a *d*-dimensional hypercube is used to track the degree of crowding for each solution in the archive, where *d* is the number objectives in the problem. The solution placed in the less crowded hypercube is considered to be a better solution.

6.3 Fitness Functions

A fitness function is used to measure the quality of the solutions according to the given optimization objectives. We separate fitness functions by *objective functions* and *penalty functions*. Objective functions are designed to encourage the algorithms to choose solutions with minimum objective values. The objective functions for solution I are defined as follows:

$$\text{Cost objective function:} \quad f_{cost}(I) = \frac{cost(I)}{B} \tag{6.7}$$

$$\text{Time objective function:} \quad f_{time}(I) = \frac{time(I)}{D} \tag{6.8}$$

Penalty functions are developed to handle constraints and are defined as follows:

$$\text{Penalty function:} \quad P(I) = P_{budget}(I) + P_{deadline}(I) \tag{6.9}$$

where P_{budget} is the budget constraint penalty function defined by Equation 6.10 and $P_{deadline}$ is deadline constraint penalty function defined by Equation 6.11.

$$P_{budget}(I) = \begin{cases} f_{cost}(I), & if\ cost(I) > B \\ 0, & otherwise \end{cases} \quad (6.10)$$

$$P_{deadline}(I) = \begin{cases} f_{time}(I), & if\ time(I) > D \\ 0, & otherwise \end{cases} \quad (6.11)$$

Since satisfying deadline and budget requirements is the primary goal of the scheduling scheme. The overall penalty is added to the objective functions to form the measure functions:

Cost fitness function: $F_{cost}(I) = f_{cost}(I) + P(I)$ (6.12)

Time fitness function: $F_{time}(I) = f_{time}(I) + P(I)$ (6.13)

6.4 Experiments

The parameter settings used as the default configuration for NSGAII, SPEA2 and PAES are listed in Table 6.1. In order to evaluate the proposed genetic algorithm, the three algorithms are implemented and compared on the two workflow structures, balanced and unbalanced. The applications used for the experiments here are same as in Section 4.4. The experiment environment also follow setting descried in Section 4.6.

Table 6.1: Default parameters.

Parameter	Value/Type
Population size	10
Initial population	randomly generated solutions
Maximum generations (NSGAII, SPEA2)	100
Crossover probability (NSGAII, SPEA2)	0.9
Mutation probability	0.5
Archive size (SPEA2, PAES)	10
Depth (PAES)	10
Maximum iterations (PAES)	1000

The behaviors of algorithms are also observed at three constraint levels, namely relaxed constraint, medium constraint, and tight constraint. The relaxed constraint level assumes that users require relatively large deadline and budget, while tight constraint level assumes that users require low deadline and budget. The relaxed/tight

deadlines and budgets of an application are determined by the maximum and minimum time/cost for the workflow execution. The maximum execution time denoted as $T_{\max}$ is the approximate least time required for executing the workflow application on the testbed to achieve the cheapest cost denoted as $C_{\min}$. The maximum cost denoted as $C_{\max}$ is the approximate least cost required for executing the workflow to achieve the fastest completion time denoted as $T_{\min}$. The $T_{\max}$ and $C_{\min}$ can be generated by an cost optimization algorithm such as GreedyCost (see Section 4.6), while $T_{\min}$ and $C_{\max}$ are generated by a time optimization algorithm such as HEFT (see Section 4.6). The deadline and budget are defined by Equation 6.14 and 6.15 respectively.

$$\text{Deadline} \quad D = T_{\max} - k \times (T_{\max} - T_{\min}) \tag{6.14}$$

$$\text{Budget} \quad B = C_{\max} - k \times (C_{\max} - C_{\min}) \tag{6.15}$$

The value of k is varied from 0.2 to 0.8 enable to generate relaxed, medium and tight deadlines and budgets.

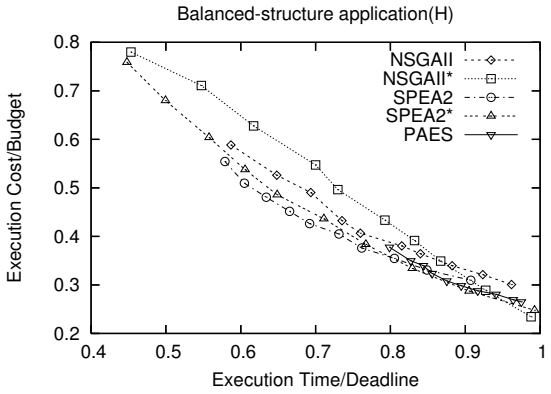
6.4.1 Comparison of different algorithms

In the first experiment, we compare three algorithms, NSGAII, SPEA2 and PAES, using relaxed, medium and tight constraints for unbalanced-and balance-structure applications. NSGAII and SPEA2 were selected with two different initial populations. Two of the members in the initial population were comprised of the solutions produced by two deadline or budget constrained minimization heuristics, TD and CD, which are described in Chapter 4, together with other randomly generated solutions. We denote the results returned by NSGAII and SPEA2 with this type of initial population as NSGAII* and SPEA2* respectively. The other initial population contains only randomly generated solutions and corresponding results returned by NSGAII and SPEA2 are denoted as NSGAII and SPEA2 respectively.

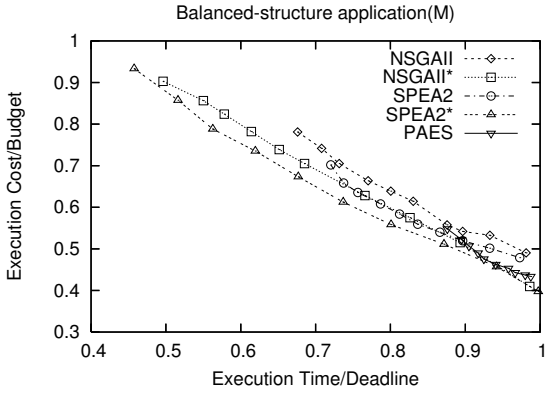
In order to compare the performance of alternative workflow multi-objective scheduling algorithms, we need to examine the extent of minimization of the obtained non-dominated solutions produced by each algorithm for each objective and the spread of their solutions. Figure 6.5 and 6.6 show the non-dominated solutions ob-

tained at the end of simulation trial (average over 10 runs) for the balanced- and unbalanced-structure applications. The performance appears to be dramatically different when the TD and CD heuristics have been used to initialize the population. Compared with SPEA2 and NSGAII, SPEA2* and NSGAII* guarantee the extreme solutions and have a better spread distribution of solutions within the constraints, whereas the solutions obtained by SPEA2 and NSGAII dominate a small subset of solutions produced by SPEA2* and NSGAII*. However, as the constraints are tightened, SPEA2* and NSGAII* significantly outperforms SPEA2 and NSGAII in terms of the minimization and distribution of solutions. As shown in Figure 6.6c, the solutions produced by SPEA2 and NSGAII cannot meet the deadline and budget constraints. This shows that the performance of population-based algorithms can be significantly affected when we initialize the population with solutions generated by TD and CD.

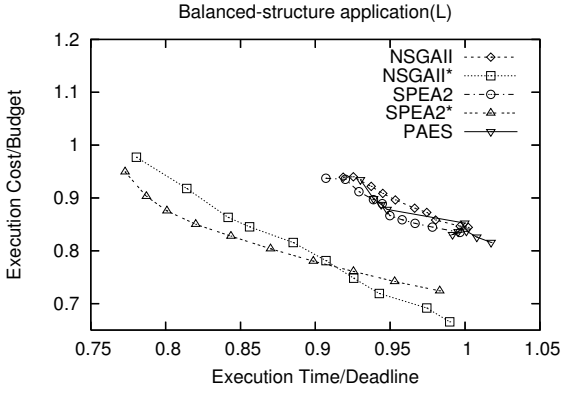
In order to present a comprehensive comparison of the overall quality of these alternative approaches, we have run each algorithm for two different workflow structures at relaxed, medium and tight constraint levels respectively. The experiment for each scenario was repeated 30 times. We have constructed a reference set *R*, by merging all of the archival non-dominated solutions found by each of the algorithms for a given workflow structure and constraint level across 30 runs. We then use the hypervolume difference indicator I_H^- to measure the differences between non-dominated fronts generated by the algorithms and the reference set *R*. I_H^- measures the portion of the objective space that is dominated by *R* [200]. The lower the value of I_H^-, the better the algorithms perform.



(a) non-dominated solutions generated on relaxed constraint

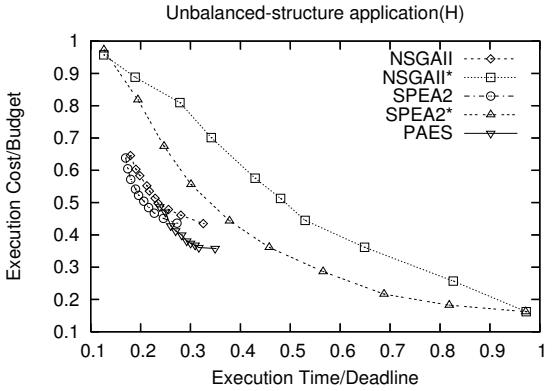


(b) non-dominated solutions generated on medium constraint

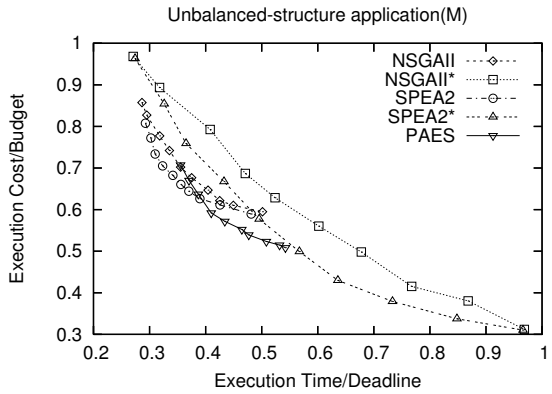


(c) non-dominated solutions generated on tight constraint

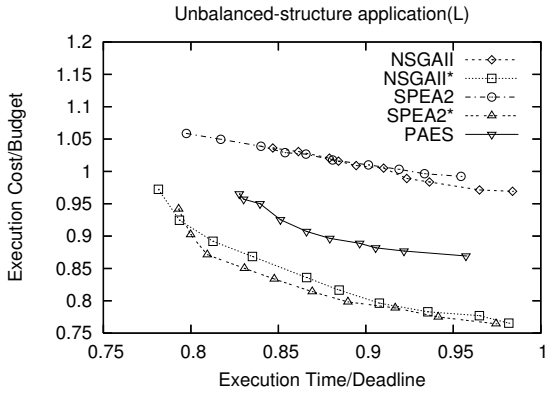
Figure 6.5: Obtained non-dominated solutions with five approaches for the balanced-structure application on different constraint levels.



(a) non-dominated solutions generated on relaxed constraint



(b) non-dominated solutions generated on medium constraint



(c) non-dominated solutions generated on tight constraint

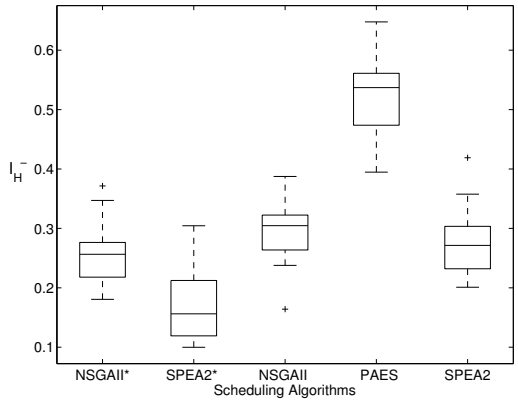
Figure 6.6: Obtained non-dominated solutions with five approaches for the unbalanced-structure application on different constraint levels.

The box plots in Figure 6.7 and 6.8 show statistical significance difference between algorithms for balanced-structure and unbalanced-structure workflow respectively (p-value < 0.05). We can observe that NSGAII and SPEA2 perform similar on both applications, whereas PAES performs differently. For the balanced-structure application, as shown in Figure 6.7 PAES performs worst at relaxed and medium constraint while it performs very similar to SPEA2 and NSGAII at tight constraint. However, it outperforms SPEA2 and NSGAII for the unbalanced-structure application as we can see from Figure 6.8. Therefore, it can be said that the local search-based algorithm is more efficient for the unbalanced-structure application whereas the population based algorithms is better for the balanced-structure application.

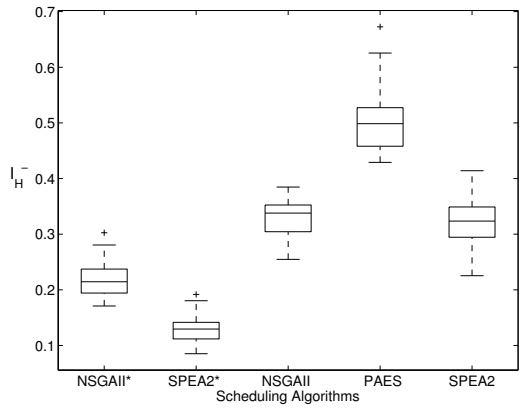
However, a population-based algorithm can be improved by employing solutions optimizing each objective separately as initial individuals. The statistical analysis in Figure 6.7 and 6.8 shows that SPEA2* is significantly better than SPEA2 and outperforms PAES. Even though the behaviors of SPEA2 and NSGAII are similar, NSGAII* does not perform as well as SPEA2*. It even performs worse than NSGAII on the unbalanced-structure application at relaxed constraint (see Figure 6.8a).

The major difference between the SPEA2 algorithm and the NSGAII algorithm is their selection strategies. Different selection strategies mean that NSGAII and SPEA2 perform differently. As described in Section 6.2.2, SPEA2 firstly selects non-dominated solutions in the current population, and then select other solutions which are in less overcrowded area. On the other hand, NSGAII not only selects the non-dominated solutions in the population, but also ranks the rest of the solutions into non-dominated levels and selects solutions belong to higher non-dominated levels. Such an elitist selection strategy could result in less diverse elements contained by individuals and thus leading to premature convergence, since it easily chooses the individuals which inherit the parts from employed initial individuals. However, as the tight degree of the constraint increase, the search space, which contains solutions satisfying constraints decreases. Therefore, the difference between the SPEA2* and NSGAII* decreases as the constraints become tight. As shown in Figure 6.7c and

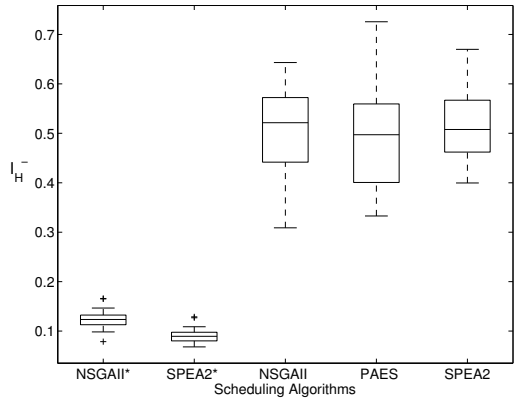
Figure 6.8c, the difference between NSGAII* and SPEA2* is not significant. These results show that with incorporating TD and CD, SPEA2 can be significantly improved and perform best in all the cases, while the performance of NSGAII gets worse for the unbalanced-structure workflow with relaxed and medium constraints. As shown in Figure 6.7 and 6.8, SPEA2* can perform better than NSGAII* for both workflow applications.



(a) relaxed constraint

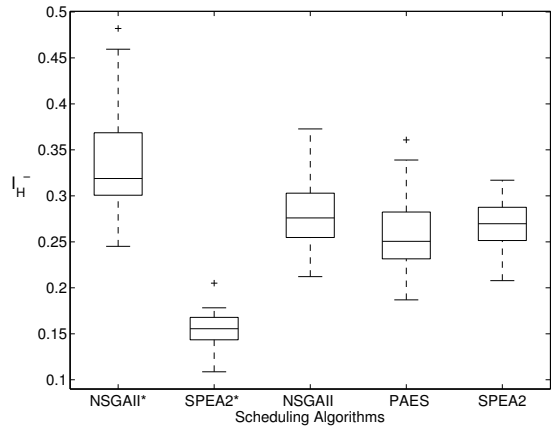


(b) medium constraint

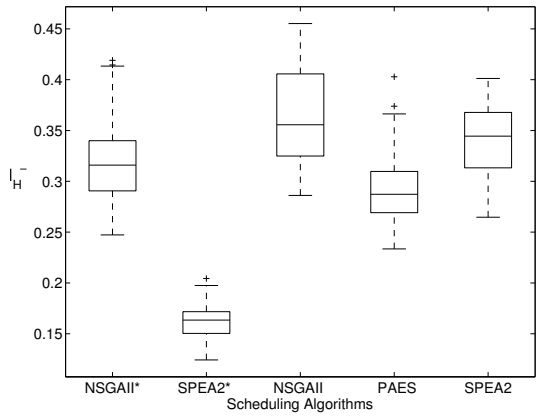


(c) tight constraint

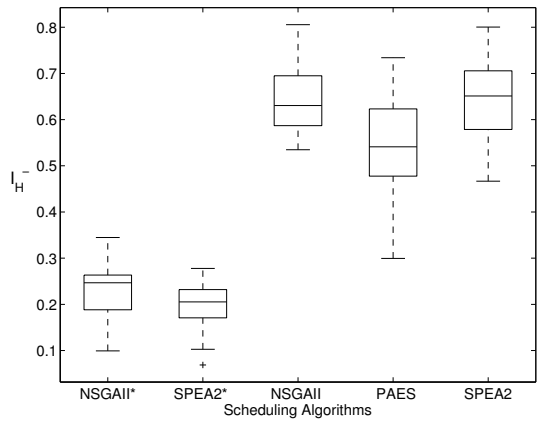
Figure 6.7: Box plot of I_H^- indicator values for the balanced-structure application on different constraint levels.



(a) relaxed constraint



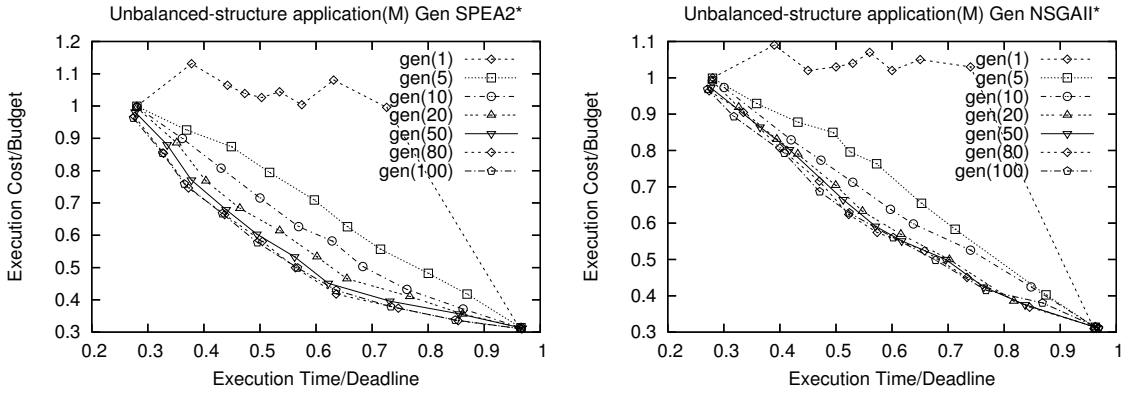
(b) medium constraint



(c) tight constraint

Figure 6.8: Box plot of I_H^- indicator values for the unbalanced-structure application on different constraint levels.

Figure 6.9 shows the execution time and cost of solutions produced by SPEA2* and NSGAII* in the population at generation 1, 5, 10, 20, 50, 80 and 100 on medium constraint case for the unbalanced-structure application. In general, at generation 1, most solutions in the population cannot satisfy the budget constraints. By increasing the generations, they are capable to meet the constraints and minimize both execution time and cost. However, NSGAII* appears to evolve slower than SPEAII* after generation 20.



(a) SPEA2* (b) NSGAII*

Figure 6.9: Evolution of execution time and cost during 100 generations for unbalanced-structure application on the medium constraint.

6.4.2 Effect of altering the archive size of SPEA2

We observe the performance of the SPEA2 when the archive size is varied between 5, 10, 15, 20, 30 and 50 solutions. As observed from Figure 6.10, a large archive size can increase the performance of both minimizing execution time and cost, as well as spread distribution.

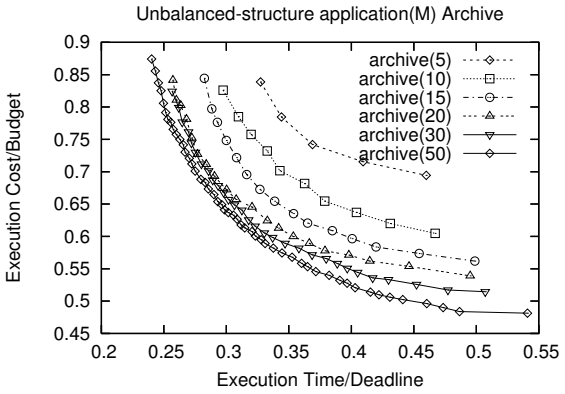
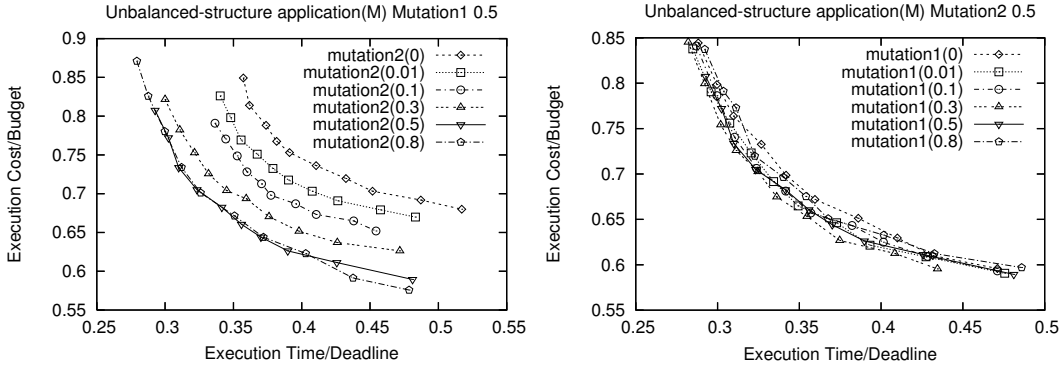


Figure 6.10: Obtained non-dominated solutions of the archive size ranged between 5, 10, 15, 20, 30 and 50 for the unbalanced-structure application on the medium constraint.

6.4.3 Effect of changing mutation possibility

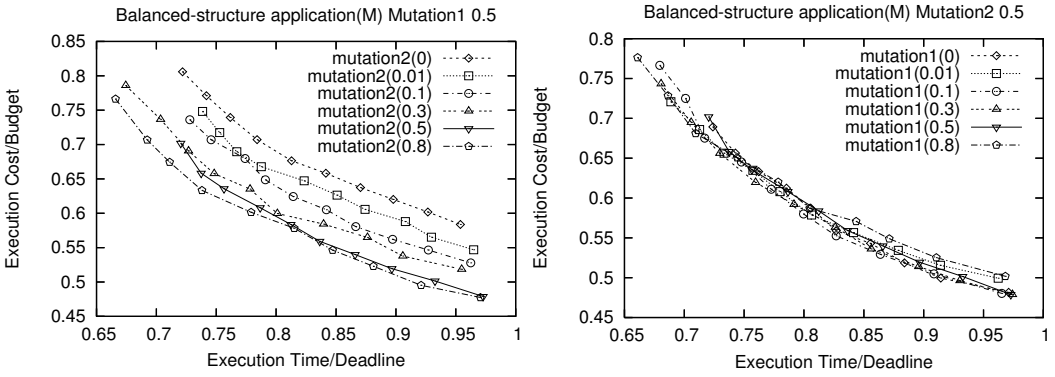


(a) different replacing mutation rates (b) different reordering mutation rates

Figure 6.11: Performances of SPEA2 with different mutation rates for the unbalanced-structure application on the medium constraint.

We also observe the performance of SPEA2 with different mutation probability. Figure 6.11 and 6.12 show the performance of different mutation probability for the unbalanced-structure application and balanced-structure applications respectively. Figure 6.11a and 6.12a show the performances of replacing-mutation (denoted as mutation 2) probability ranged between 0, 0.01, 0.1, 0.3, 0.5 and 0.8 at fixed

reordering-mutation (denoted as mutation 1) probability 0.5. Figure 6.11b and 6.12b show the performances of reordering-mutation probability ranged between 0, 0.01, 0.1, 0.3, 0.5 and 0.8 at fixed replacing-mutation probability 0.5. As shown in the results, replacing-mutation has more impact on performance than reordering mutation in current workflow applications. Results with highest replacing-mutation probability had best performance. This might due to the heterogeneous nature of utility Grids, choosing services with different QoS levels for executing tasks in workflows have significant different impact on performance than the execution order of tasks.



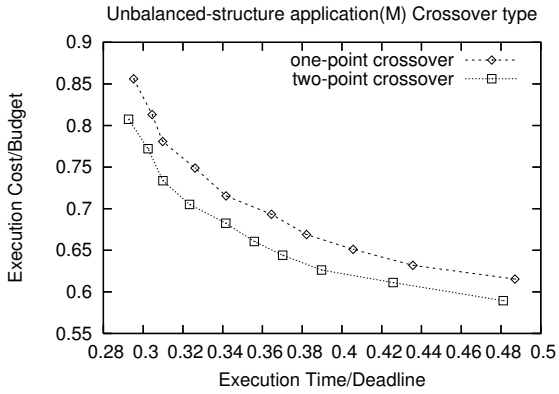
(a) different replacing mutation rates

(b) different reordering mutation rates

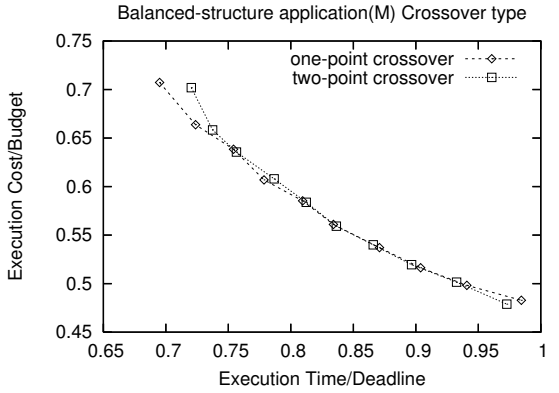
Figure 6.12: Performances of SPEA2 with different mutation rates for the balanced-structure application on the medium constraint.

6.4.4 Effect of changing crossover type

We also compare the crossover operation (denoted as two-point crossover) proposed in Section 5.4.1 with another crossover type denoted as one-point crossover [180]. One-point crossover divides the task-assignment string into top part and bottom part and swaps task assignments of bottom parts of parents are swapped. The results in Figure 6.13a show that the two-point crossover can achieve better performance for the unbalanced-structure application. As shown in Figure 6.13b, the results generated by two crossover types are similar for the balanced-structure application.



(a) unbalance-structure application



(b) balance-structure application

Figure 6.13: Performances of SPEA2 with different crossover types for the balanced-structure application on the medium constraint.

6.5 Related Work

Many efforts have been made for mapping abstract workflows onto resources in a way that optimize the overall workflow execution. Pegasus [52] proposed a reduction algorithm to refine the abstract workflow by using an existing materialized dataset. It also takes into account run-time storage space needed by the workflow during workflow planning [134]. Several clustering and partitioning methods [55][144] have also been developed to reduce the communication overhead of remote resources. Many list scheduling algorithms such as HEFT [181] and meta-heuristics such as GRASP [21]

have been applied to search an optimal schedule in Grid environments. Several efforts [143][195] have also been made toward mapping multiple workflow applications onto heterogeneous computing environments.

The work in this chapter is distinct from the related work because it simultaneously optimizes multiple objectives of workflow execution according to users QoS constraints and is capable of generating a set of alternative trade-off solutions for users' further decisions.

6.6 Summary

Multiple scheduling objectives may need to be considered while scheduling workflows on utility Grids. This chapter proposes a workflow execution planning approaches, which optimize multiple objectives. The planner can generate a set of widespread alternative solutions if the optimization objectives are conflicted.

Multi-objective Evolutionary Algorithms (MOEAs) are introduced in this chapter for solving the workflow execution planning problem. Corresponding fitness functions, which incorporate minimization objectives and penalty functions for the constraints, are developed. This chapter focuses on investigating three benchmark MOEAs, NSGAII, SPEA2 and PAES. Statistical analysis of significance difference between algorithms for various workflow applications and QoS constraint levels is also presented.

The statistical results show that different evolutionary operators and selection strategies employed by the algorithms have an impact on the quality of planned solutions for different workflow structure and constraint levels. The study carried out in this chapter also proposes and proves an initiative of using low complexity heuristics which designed for optimizing a single objective to improve the performance of multi-objective evolutionary algorithms.

Chapter 7

Conclusions and Future Directions

7.1 Summary

Grids provide a global infrastructure for solving large-scale problems in science, engineering and business. They enable the sharing, exchange, discovery, selection and aggregation of geographically distributed, heterogeneous resources, such as computers, data sources, visualization devices and scientific instruments. Workflows have been recognized as an important application for Grid systems. However, composing and deploying such workflows on dynamic and heterogeneous distributed Grid resources to meet users' QoS requirements is a challenging task. To enhance the understanding of Grid workflows and their efficient scheduling on global Grids, this thesis made the following key contributions:

- proposed a taxonomy of workflow management systems for Grid computing.
- developed a workflow engine which leverages tuple spaces to provide event-based execution management.
- developed deadline and budget distribution strategies based on the workload and dependency of tasks.
- developed algorithms for scheduling workflows with QoS constraints using genetic algorithms.
- leveraged multi-objective evolutionary algorithms (MOEAs) for workflow execution planning to generate a set of trade-off alternative scheduling solutions.

- evaluated three benchmark MOEAs based on various QoS constraint levels and workflow structures derived from applications including neuroscience and astronomy.

7.2 Conclusion

This thesis began by introducing a generic Grid workflow system model, and characterizing several aspects of the system including workflow design, information retrieval, scheduling, fault tolerance and data movement. Taxonomies for each of these areas have been developed to provide a basis for comparison of Grid workflow systems. Several representative Grid systems have been selected for comparison according to the taxonomies. This study not only enhances the understanding of design and methodologies adopted in current systems, but also provides a reference to identify areas that need further research.

Based on the study of existing Grid workflow systems, a workflow enactment engine has been presented in this thesis. The major distinction of the engine from related work is its execution model. It utilizes tuple spaces to provide an event-driven mechanism for workflow execution entities. Each entity can subscribe to the events in which it is interested and produces events based on its execution status. The behaviors of the entities are also driven by the events occurred. Each entity is also able to represent users to select resources and negotiate with remote resources for task execution. The benefits of this design include the ease of deployment for various strategies of resource selection and allocation, and supporting complex control and data dependencies of tasks.

Unlike most existing workflow languages which only target web services, the workflow language supported by the engine is independent of any particular middleware. The engine can identify remote application middleware by consulting a service configure file or Grid Market Directory (GMD) at run-time. Parameterization and multiple data flow models which are required by many scientific experiments have also been supported in the engine. The engine has been applied to neuroscience appli-

cations which have workflows similar to those in the areas of astronomy, bioinformatics and geographical information systems are similar [198].

The engine facilitates users to compose and deploy their workflows in Grid environments. However, the performance of workflow execution is determined by scheduling algorithms. Therefore, the majority of this thesis has concentrated on scheduling workflow applications. Most existing scheduling algorithms such as Min-Min and HEFT attempt to optimize execution time without considering other factors. In contrast, the scheduling algorithms proposed in this thesis schedule workflows according to users' QoS constraints such as deadline and budget while taking into account the service price and capabilities of Grid resources.

Deadline and budget constrained heuristics have been developed to optimize a single factor of execution performance while meeting users' specified time and cost constraints. Deadline and budget distribution strategies have also been proposed to distribute an overall deadline and budget over single workflow tasks according to their workload and dependencies. The heuristics have been evaluated against greedy algorithms and back-tracking approach in terms of performance and scheduling time. Results show the proposed heuristics can achieve 50% improvement.

The heuristics used the divide-and-conquer technique to divide workflow scheudling problem into single task scheduling problem in order to generate an optimized solution in a short computation time. However, the heuristics may not achieve a globle optimization solution for all types of workflows. Therefore, genetic algorithms (GAs) are also applied in the thesis to solve the QoS constrained scheduling problem. The advantage of the genetic algorithms is that it produces an optimized scheduling solution based on the performance of entire workflow, rather than the partial of the workflow as considered by the heuristics based approach. Compared with most existing GAs, the proposed approach targets heterogeneous and reservation based service-oriented environments for constrained optimization problems. Comparisons of the GA and the heuristics based on various workflow structures have also been presented. Results show the heuristics performs well for a simple workflow structure such as balanced-structure application, while the GA performs better for handling a

complex workflow structure. Although the GA requires evolving many generations to generate a good quality solution, it can be used to refine the solution generated by the heuristics as the time as long as the time is available. The experiments results show that the GA can significantly improve the solutions returned by the heuristics.

Finally, the thesis focused on optimizing multiple factors of the workflow execution. A multi-objective planning approach has been proposed to generate a set of widespread alternative solutions if the optimization objectives are conflicted. Providing these alternative solutions can offer more flexibility to users to estimate their preferences and choose a desired workflow schedule based on their QoS requirements.

Multi-objective Evolutionary Algorithms (MOEAs) have been applied for the workflow execution planning problem. Our goal was to simultaneously minimize two conflicting objectives - execution time and execution price while meeting users' maximum time constraint (deadline) and price constraint (budget). Corresponding fitness functions, which incorporate minimization objectives and penalty functions for the constraints, have been developed. A comparison of two population-based MOEAs, NSGAII and SPEA2, and one local search-based MOEAs, PAES has also been conducted. A statistical analysis of the result has been presented to show the quality of each algorithm for various workflow structures and constraint levels. The study showed that the local search-based approach overperform the population-based approaches for the unbalanced-structure application, while the population-based approaches perform better for the balanced-structure application. A method has also been proposed to incorporate single objective optimization heuristics and population-based algorithms. Simulation results show that this method can significantly improve the performance of population-based algorithms and find a range of compromise solutions within a short computational time.

The planning and scheduling algorithms proposed in this thesis have taken into account both costs and capabilities of Grid resources while meeting users' QoS requirements. The thesis demonstrated the potential for economy-based scheduling of workflow applications on Utility Grids, and made significant contributions towards advancement of the discipline.

7.3 Future Directions

This thesis improves the understanding of workflow management for Grid environments and contributes a number of deadline and budget constrained scheduling algorithms that take into account the costs and capabilities of distributed resources. Thus, it has laid a foundation for QoS-based workflow management, and scheduling. It has also led to several new future directions.

7.3.1 Dynamic Negotiation Models

This thesis has presented the scheduling problems on service-oriented Grid environments. The proposed scheduling algorithms can produce promising optimized scheduling solutions by using SLAs provided by service providers such as service price, expected scheduling time and available time slots. However, the options currently provided may not satisfy the QoS of the entire workflow. Therefore, the scheduler is required to be flexible to evaluate other available proposed SLAs and negotiate with service providers to establish new SLAs.

It is important for the scheduler to monitor task execution based on the agreed contract including resource capability and service time with service providers. Once the contract has been violated, the scheduler needs to adjust the schedule for the remaining unexecuted tasks in order to meet the specified scheduling objectives. When the execution of one task is affected due to system failure, it produces a cascading effect on the rest of the workflow and therefore SLAs for successing tasks may have to be re-negotiated. The scheduler should be able to find an alternative service and request a SLA for the task execution with respect to its currently accepted set of SLAs and expected return of unscheduled tasks.

7.3.2 Supporting multiple pricing models

The pricing model used in this thesis is based on the commodity market model which specifies service price according to the amount of usage. However, many other pricing models have also been proposed for proposed electronic resource market, including

bargaining model, contract-net model and auction model. The impact of these pricing models on workflow applications needs to be studied.

7.3.3 Supporting multiple scheduling objectives and selection functions

The scheduling algorithms proposed in this thesis focus on time and cost parameters. However, other parameters such as security levels and fidelity could also be required for many applications. For example, confidential applications must be executed on services with high security levels. In addition, the reliability of services needs to be considered during the execution planning stage. Services with high reliability may result in slightly higher cost but reduce the risk of execution failures.

The workflow planning proposed in this thesis serves as an execution adviser for users by recommending a set of alternative solutions when multiple conflicting optimization objectives are provided by users. But instead of multiple solutions, users may prefer a single solution based on their desired trade-off factors [146] of these objectives. This thus requires the development of filtering functions that offers a single solution based on users' preferences. The resulting answer needs to be evaluated by comparing against the results produced by non-evolutionary based methods such as weightage multi-objectives methods and linear programming.

7.3.4 Scheduling non-DAG workflows and multiple workflows

The workflow model of this thesis focuses on deterministic Directed Acyclic Graphs (DAGs). The scheduling decision was based on the assumptions that the dependencies of workflow tasks are well-defined and the workflow does not contain any loop and conditional branch. These assumptions are applied for many scientific applications such as image rendering, fMRI data analysis and montage astronomy. However, supporting loop and condition checking are required by many other applications such as medical, in which the execution of a partial of the workflow is based on the results generated at run-time.

Scheduling multiple workflows is another interesting future research problem. In some cases, multiple workflows are expected to be executed from one organization.

Considering only one workflow at a time could cause bad consequences from the perspective of the overall situation. Enhancing the proposed algorithms to support multiple workflows can improve the total performance of enterprise-level resource utilization.

7.3.5 Benchmarking QoS-based workflow scheduling algorithms

The thesis has developed a number of algorithms for addressing challenges of QoS constrained workflow scheduling problems in the context of Grid computing. They have been evaluated and compared against some existing works based on balanced- and unbalanced structures, and various time and cost constraints. However, as QoS based workflow execution becomes increasingly critical for advancing the developments of Grids, it is necessary to set up a standard set of benchmarks for evaluating and comparing QoS-based workflow scheduling algorithms in terms of different workflow applications, Grid configurations and QoS parameters.

REFERENCES

[1] W.M.P. van der Aalst, K. M. van Hee, and G. J. Houben. Modelling and Analysing Workflow using a Petri-net based approach. *Proceedings of the 2nd Workshop on Computer-supported Cooperative Workshop*, Zaragoza, Spain, June 20-24, 1994.

[2] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. Technical Report, Eindhoven University of Technology, 2000.

[3] W.M.P. van der Aalst, A.H.M ter Hofstede, B. Kiepuszewski, and A. P. Barros. Advanced Workflow Patterns. *Proceedings of the 7th International Conference on Cooperative Information Systems* (*CoopIS 2000*), Lecture Notes in Computer Science (LNCS) 1901, Springer-Verlag, Heidelberg, Germany, 2000; 18-29.

[4] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. Technical Report, Queensland University of Technology, Brisbane, 2002.

[5] W.M.P. van der Aalst and K.M. van Hee, *Workflow Management: models, methods, and Systems*. MIT Press, Cambridge, Mass., USA, 2002.

[6] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski and A.P. Barros, Workflow Patterns. URL http://tmitwww.tm.tue.nl/research/patterns/ [December 2004].

[7] J. H. Abawajy. Fault-Tolerant Scheduling Policy for Grid Computing Systems. *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, Santa Fe, New Mexico, IEEE Computer Society (CS) Press, Los Alamitos, CA, USA, April 26-30, 2004; 238-244.

[8] D. Abramson, J. Giddy, and L. Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, Cancun, Mexico, IEEE CS Press, Los Alamitos, CA, USA, May 1-5, 2000.

[9] D. Abramson, R. Buyya, and J. Giddy. A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker, *Future Generation Computer Systems (FGCS)*, 18(8): 1061-1074, Elsevier Science, The Netherlands, October 2002.

[10] M. Addis, J. Ferris, M. Greenwood, P. Li, D. Marvin, T. Oinn, and A, Wipat. Experiences with e-Science Workflow Specification and Enactment in Bioinformatics, In *UK e-Science All Hands Meeting 2003*, IOP Publishing Ltd, Bristol, UK, 2003; 459-467.

[11] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke. Data Management and Transfer in High Performance Computational Grid Environments, *Parallel Computing Journal*, 28(5):749-771, May 2002.

[12] G. Allen, K. Davis, K. N. Dolkas, N. D. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, and I. Taylor. Enabling Applications on the Grid – A GridLab Overview. *International Journal of High Performance Computing Applications (JHPCA)*, Special Issue on Grid Computing: Infrastructure and Applications, SAGE Publications Inc., London, UK, August 2003.

[13] J. Almond and D. Snelling. Unicore: Secure and Uniform Access to Distributed Resources via the World Wide Web. White Paper, October 1998, http://www.fz-juelich.de/zam/RD/coop/unicore/whitepaper.ps [December 2004].

[14] I. Altintas, A. Birnbaum, K. Baldridge, W. Sudholt, M. Miller, C. Amoreira, Y. Potier, and B. Ludaescher. A Framework for the Design and Reuse of Grid Workflows, *Proceedings of the International Workshop on Scientific Applications on Grid Computing (SAG'04)*, LNCS 3458, Springer, 2005.

[15] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana. Business Process Execution Language for Web Services Version 1.1, 05 May 2003, http://www-128.ibm.com/developerworks/library/ws-bpel/ [Feb 2005]

[16] The Apache Ant Project. http://ant.apache.org/ [December 2004].

[17] D. A. Bacigalupo, S. A. Jarvis, L. He, and G. R. Nudd. An Investigation into the application of different performance techniques to e-Commerce applications. *Proceedings of the IPDPS 2004 Workshop on Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems*, Santa Fe, New Mexico, IEEE CS Press, Los Alamitos, CA, USA, April 26-30, 2004.

[18] R. Bastos, D. Dubugras, and A. Ruiz. Extending UML Activity Diagram for Workflow Modeling in Production Systems. *Proceedings of the 35th Annual Hawaii International Conference on System Sciences* (HICSS'02), Big Island, Hawaii, IEEE CS Press, Los Alamitos, CA, USA, January 07 -10, 2002.

[19] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Computing Applications (JHPCA)*, 15(4):327-344, SAGE Publications Inc., London, UK, Winter 2001.

[20] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, and J. Luitz, WIEN2k: An Augmented Plane Wave plus Local Orbitals Program for Calculating Crystal Properties, Institute of Physical and Theoretical Chemistry, Technical Report, Vienna University of Technology 2001.

[21] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, Task Scheduling Strategies for Workflow-based Applications in Grids, *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*, IEEE Computer Society, 2005.

[22] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt, *Towards Quality of Service Support for Grid Workflows*, First European Grid Conference (EGC 2005), Amsterdam, The Netherlands, Feb 2005.

[23] I. Brandic, S. Pllana, S. Benkner. An Approach for the High-level Specification of QoS-aware Grid Workflows Considering Location Affinity. *Scientific Programming Journal*, 14(3-4):231 - 250 IOS Press, The Netherlands, December 2006.

[24] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao. A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems. *Pro-*

ceedings of the 17th Symposium on Reliable Distributed Systems. West Lafayette, IN. IEEE CS Press, Los Alamitos, CA, October 1998: 330-335.

[25] A. O'Brien, S. Newhouse, and J. Darlington, Mapping of Scientific Workflow within the e-Protein project to Distributed Resources, *UK e-Science All Hands Meeting*, Nottingham, UK, 2004.

[26] R. Buyya, D, Abramson, and J. Giddy. Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid, *Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region* (HPC Asia 2000), May 2000, Beijing, China. IEEE Computer Society Press, USA; 283-289.

[27] R. Buyya, D. Abramson, and J. Giddy. A Case for Economy Grid Architecture for Service-Oriented Grid Computing. *Proceedings of* the *10th IEEE International Heterogeneous Computing Workshop (HCW 2001)*, San Francisco, California, USA , IEEE CS Press, Los Alamitos, CA, USA, April 2001.

[28] R. Buyya, Economic-based Distributed Resource Management and Scheduling for Grid Computing, Ph.D. Thesis, Monash University, Melbourne, Australia, April 12, 2002.

[29] R. Buyya and S. Venugopal. The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report. *Proceedings of the 1st IEEE International Workshop on Grid Economics and Business Models*, GECON 2004, Seoul, Korea, IEEE CS Press, Los Alamitos, CA, USA, April 23, 2004; 19-36.

[30] J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson and G. R. Nudd. ARMS: An Agent-based Resource Management System for Grid Computing. *Scientific Programming*. Special Issue on Grid Computing, 10(2):135-148, IOS Press, Amsterdam, Netherlands, 2002.

[31] J. Cao, S. A. Jarvis, S. Saini, G. R. Nudd. GridFlow: Workflow Management for Grid Computing. *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid)*, Tokyo, Japan, IEEE CS Press, Los Alamitos, May 12-15, 2003.

[32] J. Cardoso. Stochastic Workflow Reduction Algorithm. Technical Report, LSDIS Lab, Department of Computer Science University of Georgia, 2002.

[33] J. Cardoso, and A. Sheth. Semantic E-Workflow Composition. *Journal of Intelligent Information Systems*, 21(3):191-225, Kluwer Academic Publishers, Netherlands, 2003.

[34] J. Cardoso, J. Miller, A. Sheth and J. Arnold. Modeling Quality of Service for Workflows and Web Service Processes. *Web Semantics Journal: Science, Services and Agents on the World Wide Web*, 1(3):281-308, Elsevier Inc, MA, USA, 2004.

[35] N. Carriero and D. Gelernter, Linda in Context, *Communications of the ACM*, 32:444-458, April 1989.

[36] T. L. Casavant and J. G. Kuhl. A Taxonomy of Scheduling in General-purpose Distributed Computing Systems, *IEEE Transactins on Software Engineering*, 14(2):141-154, IEEE CS Press, Los Alamitos, Feb. 1988.

[37] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Lamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Giggle: A Framework for Constructing Scalable Replica Location Services. *Proceed-*

ings of the Supercomputing Conference (SC2002), Baltimore, USA: IEEE Computer Society, Washington, DC, USA, November 16-22, 2002.

[38] G. Clemm, J.F. Reschke, E. Sedlar, J. Whitehead. Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol, *the Internet Society*, May 2004.

[39] K. Cooper, A. Dasgupta, Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan, J. Dongarra. New Grid Scheduling and Rescheduling Methods in the GrADS Project. *Proceedings of the NSF Next Generation Software Workshop*, International Parallel and Distributed Processing Symposium, Santa Fe, IEEE CS Press, Los Alamitos, CA, USA, April 2004.

[40] D. Crichton, J. S. Hughes and S. Kelly. A Science Data System Architecture for Information Retrieval. In *Clustering and Information Retrieval*, Kluwer Academic Publishers, December 2003.

[41] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. Proceedings of the *10th IEEE International Symposium on High Performance Distributed Computing*, San Francisco, USA: IEEE CS Press, Los Alamitos, CA, USA, 7-9 August 2001.

[42] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley and Sons, 2001.

[43] T. Eilam, K. Appleby, J. Breh, G. Breiter, H. Daur, S. A. Fakhouri, G. D. H. Hunt, T. Lu, S. D. Miller, L. B. Mummert, J. A. Pershing, and H. Wagner, Using a utility computing framework to develop utility systems, *IBM System Journal*, 43:97-120, 2004.

[44] D. Hollinsworth. The Workflow Reference Model, Workflow Management Coalition, TC00-1003, 1994.

[45] DAGMan Application. http://www.cs.wisc.edu/condor/manual/v6.4/ 2_11DAGmanApplicaitons.html [December 2004]

[46] H. J. Dail. A Modular Framework for Adaptive Scheduling in Grid Application Development Environments. Master's Thesis, UCSD Technical Report CS2002-0698, University of California at San Diego, March 2002.

[47] H. Dail, H. Casanova, and F. Berman. A Decoupled Scheduling Approach for the GrADS Program Development Environment. *Journal of Parallel Distributed Computing*, 63(5):505-524, Elsevier Inc., MA, USA, 2003.

[48] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II, *Parallel Problems Solving from Nature VI,* pp. 849-858, 2000.

[49] E. Deelman, C. Kesselman, and G. Mehta. Transformation Catalog Design for GriPhyN. Technical Report GriPhN-2001-17, 2001.

[50] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow Management in GriPhyN. *The Grid Resource Management*, Kluwer, The Netherlands, 2003.

[51] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing*, 1:25-39, Kluwer Academic Publishers, Netherlands, 2003.

[52] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. H. Su, K. Vahi, M. Livny. Pegasus: Mapping Scientific Workflow onto the Grid. *Proceedings of the Across Grids Conference 2004,* Nicosia, Cyprus, 2004.

[53] P. A. Dinda. Online Prediction of the Running Time of Tasks. *Cluster Computing*, 5(3):225-236, Kluwer Academic Publishers, Netherlands, 2002.

[54] dom4j. http://www.dom4j.org [December 2004]

[55] R. Duan, R. Prodan, and T. Fahringer. Run-time Optimization for Grid Workflow Applications, *Proceedings of the* 7th *IEEE/ACM International Conference on Grid Computing* Barcelona, Spain, 2006.

[56] M. Dumas and A. H.M. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. In *UML'2001 Conference*, Toronto, Ontario, Canada, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Heidelberg, Germany, October 1-5, 2001.

[57] R. Eshuis and R. Wieringa. Comparing Petri Net and Activity Diagram Variants for Workflow Modelling – A Quest for Reactive Petri Nets. *Advances in Petri Nets: Petri Net Technology for Communication Based Systems*; Lecture Notes in Computer Science (LNCS), 2472:321-351, Springer- Verlag, Heidelberg, Germany, 2003.

[58] T. Fahringer, J. Qin and S. Hainzer. Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language, *Proceedings of IEEE International Symposium on Cluster Computing and the Grid 2005* (CCGrid 2005), Cardiff, UK, May 9-12, 2005.

[59] T. Fahringer, S. Pllana, and J. Testori. Teuta: Tool Support for Performance Modeling of Distributed and Parallel Applications, *Proceedings of the International Conference on Computational Science, Tools for Program Development and Analysis in Computational Science*, Krakow, Poland, Springer-Verlag, Heidelberg, Germany, June 2004.

[60] T. Fahringer, R. Prodan, R.Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, M. Wieczorek. ASKALON: A Development and Grid Computing Environment for Scientific Workflows. Wor*kflows for eScience, Scientific Workflows for Grids*, I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields (ed.), Springer Verlag, ISBN: 978-1-84628-519-6, 2007.

[61] T. A. Feo and M. G. C. Resende, Greedy Randomized Adaptive Search Procedures, *Journal of Global Optimization,* 6:109-133, 1995.

[62] D. Fernández-Baca. Allocating Modules to Processors in a Distributed System. *IEEE Transactions on Software Engineering*, 15(11): 1427-1436, November 1989.

[63] P. J. Fleming and R. C. Purshouse, Evolutiojnary Algorithms in Control Systems Engineering: a Survey, *Control Engineering Practice,* 2002.

[64] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, 11:115-128, 1997.

[65] I. Foster and C. Kesselman (editors). *The Grid: Blueprint for a Future Computing In-*

frastructure, Morgan Kaufmann Publishers, USA, 1999.

[66] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications*, 15 (3), 2001.

[67] I. Foster, J. Vöckler, M. Wilde, Y. Zhao. Chimera: A Virtural Data System for Representing, Querying, and Automating Data Derivation. *Proceedings of the 14th International Conference on Scientific and Statistical Database Management (SSDBM)*, Edinburgh, Scotland, UK: IEEE CS Press, Los Alamitos, CA, USA, July 24-26, 2002.

[68] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The Physiology of the Grid, Technical Report, Globus Project, http://www.globus.org/research/papers/ ogsa.pdf [December 2004]

[69] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems, In *IFIP Internatiaonl Conference on Network and Parallel Computing*, 2006.

[70] D. Gelernter. Generative Communication in Linda, *ACM Computing Surveys*, 7(1):80-112, 1985.

[71] A. Galstyan, K. Czajkowski, and K. Lerman. Resource Allocation in the Grid Using Reinforcement Learning, *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, New York City, New York, USA, IEEE CS Press, Los Alamitos, CA, USA, July 19-23, 2004.

[72] A. Geppert, M. Kradolfer, and D. Tombros. Market-based Workflow Management. *International Journal of Cooperative Information Systems,* World Scientific Publishing Co., NJ, USA, 1998.

[73] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[74] Grid Resource Allocation Agreement Protocol. https://forge.gridforum.org/projects/graap-wg [December 2004].

[75] GriPhyN. http://www.griphyn.org [December 2004].

[76] Z. Guan, F. Hernandez, P. Bangalore, J. Gray, A. Skjellum, V. Velusamy, Y. Liu. Grid-Flow: A Grid-Enabled Scientific Workflow System with a Petri Net-based Interface. Technical Report, http://www.cis.uab.edu/gray/Pubs/grid-flow.pdf [December 2004].

[77] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of Job-Scheduling Strategies for Grid Computing. *Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000),* Berlin, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Heidelberg, Germany, 2000; 191-202.

[78] F. Hernández, P. Bangalore, J. Gray, and K. Reilly. A Graphical Modeling Environment for the Generation of Workflows for the Globus Toolkit. *Proceedings of the Workshop on Component Models and Systems for Grid Applications*, 18th Annual ACM International Conference on Supercomputing (ICS 2004), Saint-Malo, France, ACM Press, New York, NY, USA, June 2004.

[79] T. Hierl, G. Wollny, G. Berti, J. Fingberg, J. G. Schmidt, T. Schulz. Grid-enabled medical simulation services (GEMSS) in oral & maxillofacial surgery. In *Jahrestagung der Deutschen Gesellschaft für Computer- und Roboterassistierte Chirurgie (CURAC 2004),*

Munich, Germany, 2004.

[80] A. Hoheisel. User Tools and Languages for Graph-based Grid Workflows. *Proceedings of the Grid Workflow Workshop*, GGF10, Berlin, Germany, March 9, 2004.

[81] J. van Horn. Online Availability of fMRI Results Images, Journal of Cognitive Neuroscience, 15(6):769-770, 2003.

[82] S. Hwang and C. Kesselman. Grid Workflow: A Flexible Failure Handling Framework for the Grid. *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, Seattle, Washington, USA, IEEE CS Press, Los Alamitos, CA, USA, June 22 - 24, 2003.

[83] I. Horrocks. DAML+OIL: A Reason-able Web Ontology Language. *Proceedings of the International Conference on Extending Database Technology (EDBT 2002)*, Lecture Notes in Computer Science (LNCS), 1091:11-28, Springer-Verlag, Heidelberg, Germany, March 24-28, 2002; 2-13.

[84] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, Data Management in an International Data Grid Project, *Proceedings of the First IEEE/ACM International Workshopon Grid Computing*, (Springer Verlag Press, Germany), India, 2000.

[85] E. S. H. Hou, N. Ansari, and H. Ren. A Genetic Algorithm for Multiprocessor Scheduling, *IEEE Transactions on Parallel and Distributed Systems,* 5:113-120, 1994.

[86] S. Jang, X. Wu, V. Taylor, G. Mehta, K. Vahi, E. Deelman. Using Performance Prediction to Allocate Grid Resources. Technical Report 2004-25, GriPhyN Project, USA.

[87] JDOM. http://www.jdom.org [December 2004]

[88] JXTA v2.0 Protocols Specification, Sun Microsystems Inc, June 22, 2001.

[89] P. Kacsuk, G. Dózsa, J. Kovács, R. Lovas, N. Podhorszki, Z. Balaton, G. Gombás. P-GRADE: a Grid Programming Environment. *Journal of Grid Computing*, 1(2):171-197, Kluwer Academic Publisher, Netherlands, 2003.

[90] B. Kao and H. Garcia-Molina. Deadline Assignment in a Distributed Soft Real-Time System. *IEEE Transactions on Parallel and Distributed Systems*, 8(12):1268-1274, IEEE CS Press, Los Alamitos, CA, USA 1997.

[91] J. D. Knowles and D. W. Corne. The Pareto Archive Evolution Strategy: A New Baseline Algorithm for Multi-Objective Optimization, *Proceedings of the Congress on Evolutionary Computation,* Washington, D.C. USA, 6-9 July 1999.

[92] K. Krauter, R. Buyya, and M. Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Software: Practice and Experience*, 32(2):135-164, John Wiley & Sons, Inc, NJ, USA, February 2002.

[93] S. Krishnan, P. Wagstrom, and G. v. Laszewski. GSFL: A Workflow Framework for Grid Services, Argonne National Laboratory, Technical Report Preprint ANL/MCS-P980-0802, Aug 2002.

[94] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java Commodity Grid Kit, *Concurrency and Computation: Practice and Experience*, 13(8-9): 643-662, John Wiley & Sons, Ltd, Chichester, UK, 2001.

[95] G. von Laszewski, K. Amin, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. GridAnt: A Client-Controllable Grid Workflow System. *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, Big Island, Hawaii: IEEE CS Press, Los Alamitos, CA, USA, January 5-8, 2004.

[96] G. von Laszewski. Java CoG Kit Workflow Concepts for Scientific Experiments. Technical Report, Argonne National Laboratory, Argonne, IL, USA, 2005.

[97] G. von Laszewski, M. Hategan. Java CoG Kit Karajan/GridAnt Workflow Guide. Technical Report, Argonne National Laboratory, Argonne, IL, USA, 2005.

[98] A. Lerina, C. Aniello, G. Pierpaolo, and V. M. Luisa. FlowManager: A Workflow Management System Based on Petri Nets. *Proceedings of the 26th Annual International Computer Software and Applications Conference*, Oxford, England, IEEE CS Press, Los Alamitos, CA, USA, August 2002;1054-1059.

[99] F. Leymann. Web Services Flow Language (WSFL 1.0), May 2001, http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf [December 2004]

[100] D. C. Li and N. Ishii. Scheduling Task Graphs onto Heterogeneous Multiprocessors. *Proceedings of the IEEE Region 10's Ninth Annual International Conference*, Theme: Frontiers of Computer Technology, IEEE CS Press, Los Alamitos, CA, USA, 1994.

[101] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS)*, IEEE CS Press, Los Alamitos, CA, USA, June 1988; 104-111.

[102] X. Liu, J. Liu, J. Eker, and E. A. Lee. Heterogeneous Modeling and Design of Control Systems, *Software-Enabled Control: Information Technology for Dynamical Systems*, Tariq Samad and Gary Balas (eds.), Wiley-IEEE Press, April 2003.

[103] R. Lovas, G. Dózsa, P. Kacsuk, N. Podhorszki, D. Drótos. Workflow Support for Complex Grid Applications: Integrated and Portal Solutions. *Proceedings of the 2nd European Across Grids Conference*, Nicosia, Cyprus, 2004.

[104] B. Ludäscher, I. Altintas, and A. Gupta. Compiling Abstract Scientific Workflows into Web Service Workflows. *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, Cambridge, Massachusetts, USA., IEEE CS Press, Los Alamitos, CA, USA., July 09-11, 2003;241-244.

[105] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the KEPLER System. *Concurrency and Computation: Practice & Experience*, Special Issue on Scientific Workflows, to appear, 2005.

[106] H. Ludwig, A. Keller, A. Dan, R. P. King, R. Franck. Web Service Level Agreement (WSLA) Language Specification, *IBM Corporation*, January 2003.

[107] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal. Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework, *High Performance Computing: Paradigm and Infrastructure*, Laurence Yang and Minyi Guo (editors), ISBN: 0-471-65471-X, Wiley Press, New Jersey, USA, June 2005.

[108] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99)*, Juan, Puerto Rico, IEEE Computer Society, Los Alamitos, April 12, 1999.

[109] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, B. Liu, L. Johnsson, and J. Mellor-Crummey, Scheduling Strategies for Mapping Application Workflows onto the Grid, In *IEEE International Symposium on High Performance Distributed Computing* (HPDC 2005), Research Triangle Park, NC, USA, 2005.

[110] A. Mani and A. Nagarajan. Understanding Quality of Service for Web Services. http://www-106.ibm.com/developerworks/library/ws-quality.html [December 2004]

[111] D. C. Marinescu. A Grid Workflow Management Architecture. GGF White Paper, 2002.

[112] G. Mateescu. Quality of Service on the Grid via Metascheduling with Resource Co-scheduling and Co-reservation. *International Journal of High Performance Computing Applications*, 17(3):209-218, SAGE Publications Inc, London, UK, August 2003.

[113] J. Mattson and M. Simon. *The Pioneers of NMR and Magnetic Resonance in Medicine: The Story of MRI*. Jericho & New York: Bar-Ilan University Press, 1996.

[114] A. Mayer, S. McGough, N. Furmento, W. Lee, S. Newhouse, and J. Darlington. ICENI Dataflow and Workflow: Composition and Scheduling in Space and Time. *Proceedings of the UK e-Science All Hands Meeting*, Nottingham, UK, IOP Publishing Ltd, Bristol, UK, September 2003; 627-634.

[115] A. Mayer, S. McGough, N. Furmento, W. Lee, M. Gulamali, S. Newhouse, and J. Darlington. Workflow Expression: Comparison of Spatial and Temporal Approaches. *Proceedings of the Workflow in Grid Systems Workshop*, GGF-10, Berlin, March 9, 2004.

[116] S. McGough, L. Young, A. Afzal, S. Newhouse, and J. Darlington. Workflow Enactment in ICENI. *Proceedings of the UK e-Science All Hands Meeting*, Nottingham, UK, IOP Publishing Ltd, Bristol, UK, Sep. 2004; 894-900.

[117] S. McGough, L. Young, A. Afzal, S. Newhouse, and J. Darlington. Performance Architecture within ICENI. *Proceedings of the UK e-Science All Hands Meeting*, Nottingham, UK, IOP Publishing Ltd, Bristol, UK, Sep. 2004; 906-911.

[118] D. A. Menasce and E. Casalicchio, A framework for resource allocation in grid computing, *Proceedings of the IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems* (MASCOTS 2004), Volendam, Netherlands,October 5-7, 2004.

[119] Message Passing Interface Forum, http://www.mpi-forum.org/ [Feb 2005]

[120] R. A. Moreno. Job Scheduling and Resource Management Techniques in Dynamic Grid Environment. *Proceedings of the 1st European Across Grids Conference*, Spain, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Heidelberg, Germany, February 2003.

[121] T. Murata, Temporal Uncertainty and Fuzzy-Timing High-Level Petri Nets. In *Application and Theory of Petri Nets*, Lecture Notes in Computer Science (LNCS), 1091:11-28, Springer-Verlag, Heidelberg, Germany, 1996.

[122] R. Nakano and T. Yamada, Conventional Genetic Algorithm for Job Shop Problems, *Proceedings of the 4th International Conference on Genetic Algorithms*, San Diego, CA, USA, 1991; 474-479.

[123] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper, and D. V. Wilcox. PACE-A Toolset for the performance Prediction of Parallel and Distributed Systems. *International Journal of High Performance Computing Applications (JHPCA)*, Special Issues on Performance Modelling- Part I, 14(3): 228-251, SAGE Publications Inc., London, UK, 2000.

[124] Object Management Group. Unified Modeling Language (UML), http://www.uml.org/ [Feb 2005]

[125] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver and K. Glover, M.R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045-3054, Oxford University Press, London, UK, 2004.

[126] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, Taverna: Lessons in creating a workflow environment for the life sciences, *Concurrency and Computation Practice and Experience,* 2005.

[127] OMG. Unified Modeling Language Version 1.3., July 1999.

[128] C. Patel, K. Supekar, and Y. Lee. A QoS Oriented Framework for Adaptive Management of Web Service based Workflows. *Lecture Notes in Computer Science*, 2736:826-835, Springer-Verlag, Heidelberg, Germany, 2003.

[129] C. A. Petri. Kommunikation mit Automaten. PhD Thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[130] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler. PISA A Platform and Programming Language Independent Interface for Search Algorithms, *Proceedings of the Conference on Evolutionary Multi-Criterion Optimization* (EMO 2003), Faro, Protugal, 2003.

[131] S. Pllana, T. Fahringer, J. Testori, S. Benkner, and I. Brandic, Towards an UML Based Graphical Representation of Grid Workflow Applications. In *Proceedings of the 2nd European AcrossGrids Conference (AxGrids 2004)*, Nicosia, Cyprus, LNCS, Springer-Verlag, Heidelberg, Germany, January 28-30, 2004.

[132] R. Prodan and T. Fahringer. Dynamic Scheduling of Scientific Workflow Applications on the Grid: A Case Study. *Proceedings of the 20th Annual ACM Symposium on Applied Computing (SAC 2005),* New Mexico USA, ACM Press, New York, NY, USA, March 2005.

[133] R. Prodan and T. Fahringer. *Grid Computing: Experiment Management, Tool Integration, and Scientific Workflows*. Lecture Notes in Computer Science 4340, Springer-Verlag, 2007.

[134] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Black-

burn, D. Meyers, and M. Samidi. Scheduling Data-intensive Workflows onto Storage-Constrained Distributed Resources, *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*, Rio de Janeiro, Brazil, 2007.

[135] R. L. Ribler, H. Simitci, and D. A. Reed. The Autopilot Performance-directed Adaptive Control System. *Future Generation Computer Systems*, 18(1): 175-187, Elsevier Inc, MA, USA, 2001.

[136] M. Romberg, The UNICORE Architecture Seamless Access to Distributed Resources, *Proceedings of the 8th IEEE International Symposium on High Performance Computing*, Redondo Beach, CA, USA, 1999, pp. 287-293.

[137] H. G. Rotithor, Taxonomy of Dynamic Task Scheduling Schemes in Distributed Computing Systems, *Proceedings of the IEEE Proceedings of Computers and Digital Techniques*, 141(1):1-10, London, UK, January 1994.

[138] A. Roy and S. K. Das, Optimizing QoS-based Multicast Routing in Wireless Networks: a Multi-Objective Genetic Algorithms Approach, *Proceedings of the Second IFIP-TC6 Networking Conference (Networking 2002)*, Pisa, Italy, 2002.

[139] S. Fitzgerald, I. Foster, C. Kesselman, G. Von Laszewski, W. Smith and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, Portland, OR, IEEE CS Press, Los Alamitos, August 1997; 365-375.

[140] R. Sakellariou and H. Zhao. A Low-Cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems. *Scientific Programming*, 12(4):253-262, IOS Press, Netherlands, December 2004.

[141] R. Sakellariou, H. Zhao, E. Tsiakkouri amd M. D. Dikaiakos. Scheduling Workflows with Budget Constraints. In S.Gorlatch, M.Danelutto (Eds.), *Integrated Research in Grid Computing*, CoreGrid series, Springer-Verlag

[142] M. Senger, P. Rice, and T. Oinn. Soaplab-a Unified Sesame Door to Analysis Tools. *Proceedings of the UK e-Science All Hands Meeting*, September 2003; 509-513.

[143] M. Siddiqui, A. Villazon, and T. Fahringer, Grid allocation and reservation - Grid capacity planning with negotiation-based advance reservation for optimized QoS. *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing*, Tampa, FL, USA, 2006.

[144] G. Singh, C. Kesselman, and E. Deelman. Optimizing Grid-Based Workflow Execution, *Journal of Grid Computing,* 3:201-219, December 2005.

[145] G. Sinh, C. Kesselman, and E. Deelman. Application-Level Resource Provisioning on the Grid, *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing,* IEEE Computer Society, 2006.

[146] G. Sinh, C. Kesselman, and E. Deelman. A Provisioning Model and its Comparison with Best-effort for Performance-cost Optimization in Grids, *Proceeding of the 16th International Symposium on High Performance Distributed Computing*, June 2007, Monterey, California, USA.

[147] A. Slominski, D. Gannon, and G. Fox. Introduction to Workflows and Use of Work-

flows in Grids and Grid Portals. *GGF 9*, Chicago, USA, 7 Oct, 2004.

[148] W. Smith, I. Foster, and V. Taylor. Predicting Application Run Times Using Historical Information. *Proceedings of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, IEEE CS Press, Los Alamitos, CA, USA, 1998.

[149] S. Smith, P. Bannister, C. Beckmann, M. Brady, S. Clare, D. Flitney, P. Hansen, M. Jenkinson, D. Leibovici, B. Ripley, M Woolrich, and Y. Zhang. FSL: New tools for functional and structural brain image analysis. *Proceedings of the 7th International Conference on Functional Mapping of the Human Brain*, June 10-14, 2001, Brighton, UK.

[150] S. S. Song and K. Hwang. Security Binding for Trusted Job Outsourcing in Open Computational Grids. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, submitted May 2004, revised Dec. 2004.

[151] S. S. Song, Y. K. Kwok, and K. Hwang. Trusted Job Scheduling in Open computational Grids: Security-Driven heuristics and A Fast Genetic Algorithm. *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS-2005)*, Denver, CO, USA., IEEE Computer Society Press, Los Alamitos, CA, USA., April 4-8, 2005.

[152] D.P. Spooner, J. Cao, J. D. Turner, H. N. Lin Chio Keung, S. A. Jarvis, and G.R. Nudd. Localized Workload Management Using Performance Prediction and QoS Contracts. *Proceedings of the 18th Annual UK Performance Engineering Workshop*, Glasgow, UK, 2002; 69-80.

[153] D.P. Spooner, J. Cao, S. A. Jarvis, L. He, and G. R. Nudd. Performance-aware Workflow Management for Grid Computing. *The Computer Journal*, Oxford University Press, London, UK, 2004.

[154] R. D. Stevens, A. J. Robinson, and C. A. Goble. myGrid:Personalized Bioinformatics on the Information Grid. *Bioinformatics*, 19(Suppl. 1):i302-i304, Oxford University Press, London, UK, 2003.

[155] A. Sulistio and R. Buyya. A Grid Simulation Infrastructure Supporting Advance Reservation, *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, MIT Cambridge, Boston, USA, ACTA Press, CA, USA, November 9-11, 2004.

[156] W. T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson, A new major SETI project based on Project Serendip data and 100,000 personal computers, *Proceedings of the Fifth International Conference on Bioastronomy*, Bologna, Italy, 1997.

[157] Sun Grid Engine. http://www.sun.com/gridware .

[158] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor - A Distributed Job Scheduler. *Beowulf Cluster Computing with Linux*, The MIT Press, MA, USA, 2002.

[159] Taverna User Manual. http://taverna.sourceforge.net/manual/docs.word.html [December 2004].

[160] I. Taylor, R. Philp, M.Shields and O.Rana, and B. Schutz. The Consumer Grid. *Global Grid Forum (2002)*. Toronto, Ontario, Canada, February 17-20, 2002.

[161] I. Taylor, M. Shields, and I. Wang. Resource Management of Triana P2P Services. *Grid Resource Management*, Kluwer, Netherlands, June 2003.

[162] I. Tayler, E. Deelman, D. Gannon, M. Shields (Editors). *Workflows for E-science: Scientific Workflows for Grids*, Springer-Verlag London Ltd, London, UK, Dec 2006.

[163] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons, NJ, USA, 2003.

[164] S. Thatte. XLANG-Web Services for Business Process Design, Technical Report, Microsoft Corporation, 2001.

[165] P. Thompson, M. S. Mega, and A. W. Toga, Sub-Population Brain Atlases, *Brain Mapping: The Methods* (2nd Edition), A. W. Toga and J. C. Mazziotta, Eds., 2002.

[166] G. Thickins, Utlity Computing: The Next New IT Model, *Darwin Magazine*, 2003.

[167] H. Topcuouglu, S. Hariri, and M. Wu, Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing, *IEEE Transaction on Parallel Distributed System,* 13:260-274, 2002.

[168] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling. Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum Draft Recommendation, June 27, 2003.

[169] UDDI Technical White Paper, September 2000, http://www.uddi.org [December 2004].

[170] J. D. Ullman, NP-complete Scheduling Problems, *Journal of Computer and System Sciences,* 10:384-393, 1975.

[171] J. Almond, D. Snelling, UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce. Future *Generation Computer Systems*, 613:1-10, 1999.

[172] S. Vadhiyar and J. Dongarra. A Performance Oriented Migration Framework for the Grid. *Proceedings of the 3rd IEEE Computing Clusters and the Grid (CCGrid 2003)*, Tokyo, Japan, IEEE CS Press, Los Alamitos, May 12-15, 2003.

[173] S. Venugopal, R. Buyya, and L. Winton. A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids, *Concurrency and Computation: Practice and Experience*, Volume 18, Issue 6, Pages: 685-699, Wiley Press, New York, USA, May 2006.

[174] S. Venugopal, K. Nadiminti, H. Gibbins, and R. Buyya, Designing a Resource Broker for Heterogeneous Grids, Software: Practice and Experience, Wiley Press, New York, USA, 2007 (in press, accepted on July 12, 2007)..

[175] H.M.W. Verbeek, A. Hirnschall, and W.M.P. van der Aalst. XRL/Flower: Supporting Inter-Organizational Workflows Using XML/Petri-nets Technology. *Proceedings of the* CAISE 2002 *Workshop on Web Services, e-Business, and the Semantic Web (WES): Foundations, Models, Architecture, Engineering and Applications*, Toronto, Ontario, Canada, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Heidelberg, Germany, May 27-28, 2002; 535-552.

[176] W3C. Extensible Markup Language (XML) 1.0 (Third Edition), http://www.w3.org/TR/REC-xml/ [Feb 2005]

[177] W3C. Web Services, 2002, http://www.w3.org/2002/ws/ [Feb 2005]

[178] W3C. XML Schema, http://www.w3.org/XML/Schema [Feb 2005]

[179] W3C. XML Pipeline Definition Language Version 1.0, http://www.w3.org/TR/2002/NOTE-xml-pipeline-20020228/ [Feb 2005]

[180] L. Wang, H. J. Siegel, V. R. Roychowdhury, A. A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, *Journal of Parallel Distributed Computing,* 47: 8-22, 1997.

[181] M. Wieczorek, R. Prodan, and T. Fahringer, Scheduling of Scientific Workflows in the ASKALON Grid Environmnet, *ACM SIGMOD Record*, 34:56-62, 2005.

[182] R. P. Woods, S. R. Cherry, J. C. Mazziotta. Rapid automated algorithm for aligning and reslicing PET images. *Journal of Computer Assisted Tomography,* 16:620-633, 1992.

[183] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15(5-6):757-768, 1999.

[184] Workflow Management Coalition. http://www.wfmc.org/ [December 2004]

[185] World Wide Web Consortium. Web Services Description Language (WSDL) Version 1.2, http://www.w3.org/TR/wsdl12 [December 2004]

[186] C. Wroe, R. Stevens, C. Goble, A. Roberts, and M. Greenwood. A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. *International Journal of Cooperative Information Systems*, 12(2):197-224, World Scientific Publishing Co., NJ, USA, 2003.

[187] X. F. Wu, V. Taylor, and R. Stevens. Design and Implementation of Prophesy Automatic Instrumentation and Data Entry System. *Proceedings of the 13th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS2001),* Philadelphia, PA, USA, August 2001.

[188] P. Wyckoff. TSpaces, *IBM Systems Journal*, 37, 1998.

[189] R. Yahyapour, P. Wieder, A. Pugliese, D. Talia, and J. Hahm. Grid Scheduling Use Cases. White Paper, Global Grid Forum, 19 July, 2004.

[190] T. Yamada and R. Nakano, Genetic Algorithms for Job-Shop Scheduling Problems, *Modern Heuristic for Decision Support*, London, UK, 1997.

[191] L. Young, S. McGough, S. Newhouse, and J. Darlington. Scheduling Architecture and Algorithms within the ICENI Grid Middleware. *Proceedings of the UK e-Science All Hands Meeting*, IOP Publishing Ltd, Bristol, UK, Nottingham, UK, Sep. 2003; 5-12.

[192] J. Yu and R. Buyya. A Novel Architecture for Realizing Grid Workflow using Tuple Spaces. *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, Pittsburgh, USA, IEEE CS Press, Los Alamitos, CA, USA, Nov. 8, 2004.

[193] J. Yu, S. Venugopal, and R. Buyya. A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Services, *The Journal of Supercomputing,* 36(1):17-31, ISSN: 0920-8542, Springer Science+Business Media, Berlin, Germany, April 2006.

[194] J. Yu, R. Buyya, and C. K. Tham. QoS-based Scheduling of Workflow Applications on Service Grids. *Proceedings of the 1st IEEE International Conference on e-Science and*

Grid Computing (e-Science 2005, IEEE CS Press, Los Alamitos, CA, USA), Dec. 5-8, 2005, Melbourne, Australia.

[195] H. Zhao and R. Sakellariou. Scheduling Multiple DAGs onto Heterogeneous Systems, *Proceedings of the 15th Heterogeneous Computing Workshop* (HCW 2006), Rhodes Island, Greece, 2006.

[196] S. Y. Zhao and V. Lo. Result Verification and Trust-based Scheduling in Open Peer-to-Peer Cycle Sharing Systems. Technical Report, University of Oregon, USA, 2005.

[197] Y. Zhao, M. Wilde, I. Foster, J. Voeckler, T. Jordan, E. Quigg, and J. Dobson. Grid Middleware Services for Virtual Data Discovery, Composition, and Integration, *Proceedings of the 2nd Workshop on Middleware for Grid Computing*, Toronto, Ontario, Canada, 2004.

[198] Y. Zhao, J. Dobson, I. Foster, L. Moreau, and M. Wilde, A Notation and System for Expressing and Executing Cleanly Typed Workflows on Messy Scientific Data, *ACM SIGMOD Record,* 34, September 2005.

[199] G. Zheng, T. Wilmarth, P. Jagadishprasad, and L. V. Kalé. Simulation-based Performance Prediction for Large Parallel Machines. *International Journal of Parallel Programming*, Kluwer Academic Publishers, The Netherlands, 2005.

[200] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach, *IEEE Transactions on Evolutionary Computation*, 3:257-271, 1999.

[201] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm, Technical Report, Swiss Federal Institute of Technology, 2001.

[202] A. Y. Zomaya, C. Ward and B. Macey. Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues, *IEEE Transactions on Parallel Distributed Systems,* 10:795-812, 1999.