Migration Management in Software-Defined Networking-enabled Edge and Cloud Computing Environments

TianZhang He

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

August 2021

ORCID: 0000-0002-5472-7681

Migration Management in Software-Defined Networking-enabled Edge and Cloud Computing Environments

TianZhang He
Principal Supervisor: Prof. Rajkumar Buyya

Abstract

Cloud and Edge Data Centers have become the backbone infrastructures of daily social and economical activities. Live migration is the cornerstone of the dynamic resource management policies for various objectives, such as application performance, networking cost, load balancing, consolidation, energy saving, user mobility, no-downtime maintenance, and disaster recovery. Live migration of VMs and containers provides a universal state-transfer standard to implement these objectives. Therefore, it is critical to manage the live migration in both computing and networking resources to guarantee the QoS, improve migration performance, and minimize migration costs and overheads. Many works have focused on the live migration mechanisms and optimization to improve the performance of individual migration. However, existing migration models in resource management neglect the resource competitions and dependencies among multiple migrations. Furthermore, performing the generated multiple live migrations in arbitrary orders can lead to service degradation. Therefore, efficient migration generation and scheduling are essential to reduce the impact of live migration overheads and improve migration performance. In addition, to prevent Quality of Service (QoS) degradations and Service Level Agreement (SLA) violations, it is necessary to respect the deadline of migration requests with various priorities and urgencies. In this thesis, we focus on network-aware multiple migration management based on Software-Defined Networking (SDN). By separating the control plane and forwarding plane, SDN provides centralized topology discovery and networking management which enables the capability of managing resource contentions in finer granularity. This thesis advances the state-of-the-art by making the following key contributions:

1. A comprehensive taxonomy and literature review on live migration management in Edge and Cloud computing environments including migration generation poli-

cies and migration planning and scheduling algorithms.

2. Empirical performance evaluation of live VM migration in SDN-enabled Clouds with respect to computing, networking, QoS, SDN traffic management, and multiple migrations.

3. A universal concurrency-aware multiple migration selector integrated with dynamic resource policy to generate scheduling-optimized migration requests.

4. SLA-aware multiple migration planning and scheduling algorithms in Cloud environments composing of a deadline-aware grouping algorithm of migrations and online scheduling to determine the migration sequence of connected VMs.

5. Efficient large-scale multiple migration algorithms in Edge environments to reduce the processing time of migration planning while maintaining multiple migration performance at scale.

6. A detailed study outlining challenges and future research directions in live migration management.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

TianZhang He, August 2021

Preface

Main Contributions

This thesis research has been carried out in the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne. The main contributions of the thesis are discussed in Chapters 2-7 and are based on the following publications:

- **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "Performance evaluation of live virtual machine migration in SDN-enabled cloud data centers", *Journal of Parallel and Distributed Computing (JPDC)*, Volume 131, Pages: 55-68, Elsevier, 2019.

- **TianZhang He**, Rajkumar Buyya, "A Taxonomy of Live Migration Management in Edge and Cloud Computing", *ACM Computing Surveys*, (submitted, July 2021).

- **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "SLA-aware multiple migration planning and scheduling in SDN-NFV-enabled clouds", *Journal of Systems and Software (JSS)*, Volume 176, Pages: 110943, Elsevier, 2021.

- **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "CAMIG: Concurrency-Aware Live Migration Management of Multiple Virtual Machines in SDN-enabled Clouds", *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, (minor revision, Aug 2021).

- **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "Efficient Large-Scale Multiple Migration Planning and Scheduling in SDN-enabled Edge Computing", *IEEE Transactions on Mobile Computing (TMC)*, (submitted, June 2021).

Supplementary Contributions

During the PhD candidature, I have also contributed to the following collaborative work (this thesis does not claim it as its contributions):

- Jungmin Son, **TianZhang He**, and Rajkumar Buyya, ”CloudSimSDN-NFV: Modeling and simulation of network function virtualization and service function chaining in edge computing environments”, *Software: Practice and Experience (SPE)*, Volume 49, Number 12, Pages: 1748–1764, Wiley, 2019.

Acknowledgements

"A journey of a thousand miles must begin with a single step." PhD is truly an enduring but rewarding journey to accomplish. I am glad that I took the step, overcame the hardship, and have a lifetime experience. It is indeed not a lonely journey.

First and foremost, my deepest gratitude goes to my principal supervisor, Professor Rajkumar Buyya, who has offered me the opportunity to pursue a PhD and provided continuous support, insightful guidance, and valuable advice throughout this journey. I also thank my co-supervisor Dr. Adel N. Toosi, for his guidance and constructive comments on my works throughout my candidature. I would also like to appreciate my PhD advisory committee's chair, Prof. Udaya Parampalli, for his comments and suggestions.

I would like to thank all the past and current CLOUDS Laboratory members at the University of Melbourne. In particular, I thank Dr. Maria Rodriguez, Dr. Yaser Mansouri, Dr. Jungmin Jay Son, Dr. Bowen Zhou, Dr. Safiollah Heidari, Dr. Minxian Xu, Dr. Xunyun Liu, Dr. Caesar Wu, Dr. Sara Kardani Moghaddam, Dr. Redowan Mahmud, Dr. Muhammad Hilman, Dr. Muhammed Tawfiqul Islam, Dr. Shashikant Ilager, Mohammad Goudarzi, Zhiheng Zhong, Rajeev Muralidhar, Samodha Pallewatta, Amanda Jayanetti, Linna Ruan, Anupama Mampage, Mohammad Reza Razian, and Jie Zhao for their support and friendship.

I acknowledge the University of Melbourne and China Scholarship Council for providing me with scholarships to pursue my doctoral studies.

I also like to thank my Master's supervisors Professor Wang Yi and Professor Nan Guan for their inspiration and guidance. My heartfelt thanks to my friends in Australia and back in China: Yunxiang Zhao, Guanli Liu, Ang Li, Yijiao Qu, Dong Ji, GaoYang Dai, Yang Wang, Yue Tang, Mingyang Zhou, Tianbo Jiang, Feiyang Li, Bolin Liu, and Yue Yin to name few and wish their success.

Finally, I am forever grateful to my mother and fiancée for their love, support, and company at all times. I am looking forward to taking a new step in my next life journey.

TianZhang He
Melbourne, Australia
August 2021

Contents

List of Figures

List of Tables

Chapter 1
Introduction

The emergence of cloud computing has facilitated the dynamic provision in computing, networking, and storage resources to deliver services on-demand basis [1, 2]. Traditionally, the application directly running on the native operating system is the foundational element to host services by utilizing the underlying hardware resources. With the development of virtualization, Virtual Machines (VM), which is one of the major technologies to host cloud services, can share computing, networking, and storage resources from the physical machines. It helps cloud computing to provide software and platforms as services to customers. In addition, containerization is the emerging virtualization technology to support more elastic service frameworks due to its flexibility and small resource footprint [3–5].

Application providers can lease virtualized instances (VMs or containers) from cloud providers with various flavors under different Service Level Agreements (SLAs). Then, the instance managers initialize the VMs and containers, and the cloud broker and orchestrator select the feasible placement based on the available resources and the allocation policy. Under highly dynamic environments, cloud providers need to prevent the violations of SLAs and guarantee the Quality of Service (QoS), such as end-to-end delay and task processing time. Therefore, there have been extensive works [6, 7] focusing on dynamic resource management in performance, accessibility, energy, and cost in order to benefit both cloud computing subscribers and providers. Live migration of process, VM, container, and storage is the key feature to support the dynamic resource management in cloud computing environments [8]. Live migration provides a generic approach without any application-specific configuration and management. It can migrate and synchronize the running states of the VM or container instance from one host

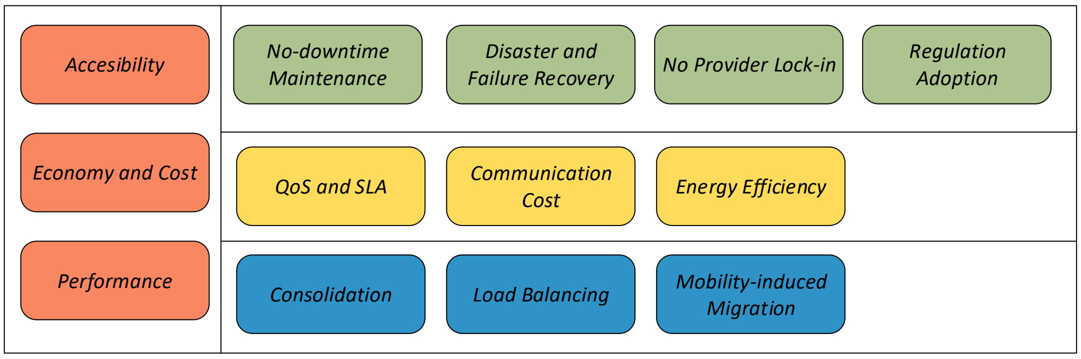


Figure 1.1: Live migration motivations

to another without service disruption.

Commercial cloud infrastructure and services providers, such as AWS, Azure, Google, IBM, etc, have been integrating live VM and container migration [9–13] for the purposes, such as higher priority task preemption, kernel and firmware software updates, hardware updates, and reallocation for performance and availability. For example, the Google cluster manager Borg controls all computing tasks and container clusters of up to tens of thousands of physical machines. In Google production fleets, a lower bound of 1,000,000 migrations monthly can be performed with 50 ms average downtime during the migration [13]. Therefore, it is critical to investigate migration management techniques in dynamic resource reallocation.

From cloud to edge computing, the processing resources and intelligence have been pushed to the edge of the network to facilitate time-critical services, which requires higher bandwidth and lower latency [14, 15]. With the combination of different paradigms, live migration can be conducted between edge servers, physical hosts in the LAN network, and data center sites through the WAN [16].

Since the state transmission and synchronization are carried out through the network, the performance of live migration heavily relies on the network resources, such as bandwidth and delay. However, with the expansion of edge and cloud computing, tens of thousands of nodes connect with each other, which makes it difficult to manage and configure networking resources at scale. To overcome the network topology complexity, Software-Defined Networking (SDN) [17–19] is introduced to provide centralized networking management by separating the data and control plane in the traditional network devices. The SDN controllers can dynamically update the knowledge of the whole network topology through the southbound interfaces based on the OpenFlow protocol. For instance, Google has presented its implementation of software-defined inter-data center WAN (B4) [20] to showcase the SDN at scale. The migration manager based on the SDN controller can manage the network resources in a fine-grained manner for the migration tasks and application services in terms of network routing and bandwidth allocation.

Many works have been focused on the different objectives of resource management through live migration in both cloud [6, 21, 22] and edge computing environments [7, 23–25], such as load balancing, over-subscription, consolidation, networking, energy, disaster recovery, and maintenance for hardware and software updates. Figure 1.1 illustrates the category of live migration motivations under three main aspects: performance, accessibility, and economy or cost. Few studies have focused on the impact of live migration overheads [26, 27] and the sequence of multiple migration scheduling [28–30]. The computing overheads of live migration on memory, I/O interfaces, and CPU can affect co-located instances on the same host. Migrations should be scheduled efficiently in order to avoid SLA violations and guarantee the performance of multiple migrations. The convergence time of live migration directly affects the convergence of the optimal instance reallocation for dynamic resource management. Moreover, we also need to manage the downtime of migrations in order to avoid SLA violations. Therefore, it is essential to minimize the overheads of migrations and improve the multiple migration performance.

However, there are gaps between current resource management algorithms and migration scheduling algorithms, which are required to successfully and efficiently perform migrations in complex edge and cloud computing environments at scale. Migration generation during resource management is focusing on management motivations, such as energy consumption and host utilization, and minimizing the total migration overheads based on the linear cost model. It disconnects two important phases of migration management: migration selection phase and migration scheduling phase. Moreover, current research neglects the network resource contentions among multiple migrations and applications, which can result in QoS degradation, migration failures, and the degraded performance of migration scheduling and resource management. In addition, with multiple migration requests generated by resource management as input, current migration planning and scheduling algorithms ignore the migration network sharing strategy and migration deadline, which may lead to QoS degradation and SLA violations. Furthermore, from cloud to edge computing, existing migration planning and scheduling frameworks and solutions can not suit the increasing number of service migrations and stochastic arrival migration, such as mobility-induced live migration.

This thesis tackles the migration management problem by considering both computing and networking resources in SDN-enabled edge and cloud data centers. We present a taxonomy and comparison of migration management in cloud computing. We propose a generic concurrency-aware migration generation algorithm along with its integration to other dynamic resource management algorithms. We also propose multiple migration planning and scheduling algorithms to improve migration performance and minimize migration overheads.

The rest of this chapter details the background of migration management in SDN-enabled cloud computing and discusses the research problems and objectives, evaluation methodology, contributions, and the organization of the thesis.

1.1 SDN-enabled Clouds

This section describes the fundamentals of cloud computing, Software-Defined Networking (SDN), and SDN-enabled cloud computing. Since live migration heavily relies on computer networks to transmit memory states and data, it is essential to study the network architecture and environment for performing migrations.

1.1.1 Cloud and Edge Computing

Cloud computing provides the on-demand availability of computing and storage resources to other users [1, 2]. The cloud services can be categorized into three categories, namely Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [1]. Infrastructure-as-a-Service is the most fundamental service which provides virtual machines and related virtual networks to cloud subscribers. The cloud infrastructure can be used to provide subscriber's own platform and software services running on servers through PaaS and SaaS, respectively. This thesis focuses on the live migration management performed by the cloud provider in the IaaS context, considering both computing and networking provisioning and overheads. The cloud provider or administrator is responsible to provision the computing, networking, and storage resources to guarantee the QoS and SLAs. The whole cloud infrastructure may be composed of several data centers in different locations. As a result, the live migration can be performed within a data center (intra-data center) or between two data centers (inter-data center).

To provide reliable services to latency-sensitive applications, cloud-like services are pushed to the edge of the networks. Edge Computing brings computing and storage resources closer to the end-user to improve the response time of the services [7]. In this case, the cloud data center can be regarded as an edge data center. It reduces the network usage between edge and core networks by allocating the tasks to the services in the adjacent edge servers. Combined with the cloud computing paradigm, the introduction of edge computing improves the performance of the emerging user-oriented applications including Vehicle to Vehicle (V2V), Vehicle to Cloud (V2C), Virtual Reality (VR), Augmented Reality (AR), Artificial Intelligent (AI), Internet of Things (IoT), etc. Edge computing introduces new challenges in live migration, mainly due to the mobility of application users. The mobility-induced migration [6, 7, 25] in edge computing is based on the user position and the coverage of each edge server and its base stations. When the position of the end-user change dramatically from one coverage of edge data center to another, the end-to-end latency will be increased. As a result, the service may need to be migrated from the previous edge servers to the adjacent one.

1.1.2 Data Center Network

In the cloud data center, the Data Center Network (DCN) are composed by interconnected physical host servers through network links and switches. A typical DCN ar-

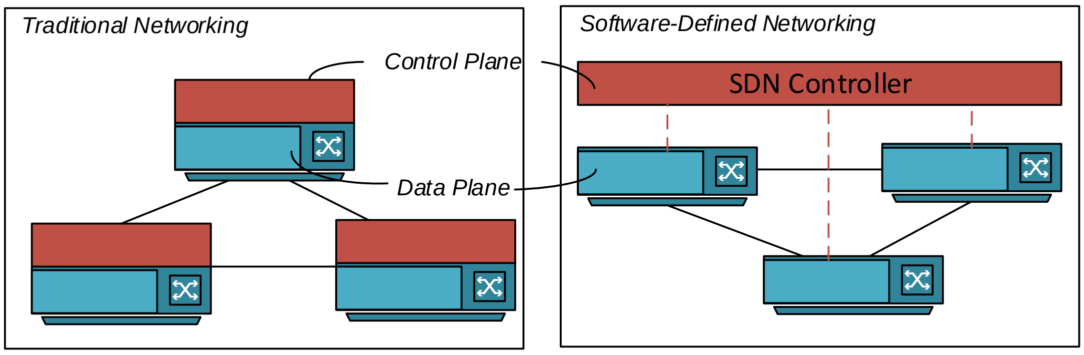


Figure 1.2: Comparisons between traditional and software-defined networking

chitecture consists of a three-layer topology, with each layer including edge, aggregation, and core switches. In the edge switch layer, host servers within the same rack are connected to one or multiple Top-of-Rack (ToR) switches. One edge switch can be connected to upper-tier aggregation switches. Each aggregation switch could be connected to one or several core switches in the top layer. The core switches are connected to the gateway of the data center. Since DCN hosts a massive number of switches and servers, researchers have been focusing on the alternative DCN topology to improve the cost, operation complexity, scalability, and performance, such as Clos and Folded-Clos/FatTree [31], DCell [32], BCube [33], and and energy consumption such as Elastic-tree [34]. For example, Google has adopted Clos topologies for its datacenters [35]. The four-post cluster architecture also applied to the commercial cloud data centers [36].

In the WAN environment, multiple data center sites are connected through the backbone network or WAN links. For the commercial massive multi-sites cloud data centers, dedicated backbone network are used for the inter-datacenter connectivity [37, 38]. For the general inter-datacenter connectivity, various WAN or Internet topologies are studied. For instance, researchers investigated an Internet topology dataset [39] of Point of Presence (PoP)-level 140 WAN topologies and over 270 network topologies with 9,500 PoP locations for data centers and hosting facilities, and 13,500 links [40].

1.1.3 Software-Defined Networking

Software-Defined Networking (SDN) [41] is the emerging paradigm in networking that manages the whole network resources with centralized software controllers with standard software interfaces. Figure 1.2 shows the difference between SDN and traditional networking. In the traditional networking, the network devices are forwarding packets to the next hop solely based on its own control logic. The traditional Layer-2 and Layer-3 models are applied to the physical switches and routers. However, SDN separates the control plane and data plane by network virtualization and centralize the control plane logically into a software controller.

At the discovery process, SDN controller sends discovery packets to the connected SDN nodes (forwarding switches) with Link Layer Discovery Protocol (LLDP) protocol to detect the entire network topology. Through the southbound interfaces of a SDN controller, OpenFlow [42], is used for the communication between SDN-enabled forwarding nodes and the controller. For each SDN-enabled node, the data plane matches flow entries to forward the traffic. Through the northbound interfaces, network management applications or routing algorithms can perform high-level networking resource management based on the monitoring network states and topology, such as limits the flow bandwidth or manages the network routing. Furthermore, the west/east interfaces of a SDN controller supports the horizontal communication between controller that facilitates the scalable multiple SDN controller framework [43].

With the separation of forwarding plane and control plane, SDN opens up more opportunities in various aspects of network virtualization to both academia and industry, such as innovative networking protocols for LAN and WAN, virtualized data plane optimization, traffic and flow management, and Software Function Chaining by Virtualized Network Functions, etc. As a result, OpenFlow becomes the de facto standard SDN communication and control protocol [44] along with the OF-CONFIG management and configuration protocol [45]. There are several open-source SDN controllers based on OpenFlow have been developed, such as NOX [46], Beacon [47], ONOS [48], OpenDayLight (ODL) [49], Ryu [50], and FloodLight [51].

Open vSwitch (OVS) [52] as the multilayer software switch running in the Hypervisor to provide the switching stack for the hardware virtualization. OVS supports the

SDN management and control (OpenFlow and OVSDB which is its own implementation of OF-CONFIG protocol), monitoring (sFlow, NetFlow, SPAN, and RSPAN), QoS (traffic shaping and queuing), and Security (VLAN and traffic filtering). To support packet processing, performance and throughput of networking virtualization, various projects and works have been conducted on the software data plane acceleration, such as DPDK [53], SR-IOV [54], VPP [55], and SoftNIC [56].

1.1.4 SDN-enabled Clouds

As the number of end-user and low-latency application services increases, the number of services in edge and cloud computing also increases accordingly. Therefore, compared with the traditional cloud data center network, the topology of edge and cloud computing including compute and network nodes has become more complicated. In traditional network management, each network device applies its own logic in a distributed manner, where the control and the data plane are in the same device. It only contains the knowledge of its adjacent nodes (next hop). Thus, it is difficult to dynamically manage network flows and monitor the status of the entire network which is composed of tens of thousands of nodes. To address this issue, SDN has been adopted in both edge and cloud computing [57], namely SDN-enabled Clouds.

Integrating with hypervisor and cloud computing management, SDN-enabled clouds bring benefits to edge and cloud computing, such as global view of DCN, dynamic and programmable control planes for network flows, network security and virtualization, and dynamic network workloads. With the centralized control plane, SDN controllers can dynamically manage and monitor the network resources in a programmable fashion. Various research has been conducted focusing on the network applications for dynamic network resource management [19, 58], such as dynamic bandwidth allocation, network slicing, network flow consolidation, and the optimization of network utilization to improve the performance and energy efficiency of data centers and QoS of applications.

In the WAN environment, SDN architecture and OpenFlow protocol are adopted for data center WAN interconnection. Since data centers are geographically located on the planet, multiple controllers are deployed for each data center site to tackle the scalability and latency issues. Global centralized Traffic Engineering (TE) servers are connected to multiple controllers through gateways [20]. Software-Defined WAN (SD-WAN) [20, 59] leverages the SDN principles combining the tenets of distributed WAN, simplifying network control policy and traffic flow management in the clouds. In the literature, SD-WAN is used as the acronym for SDN in the wide-area network. Therefore, SDN-enabled WAN and SD-WAN can be often used interchangeably.

Since live migration heavily relies on the computer network to transfer the memory states and data, SDN-enabled edge and cloud computing open up innovative opportunities for networking management to improve live migration performance and schedule each migration dynamically and efficiently, including network discovery, dynamic network routing, and bandwidth allocation.

1.2 Live Migration

Considering the scudding development of live migration virtualization and cloud computing paradigms, this section introduces the fundamentals of live migration of VM and container instance from three aspects: virtualization, migration span and migration type. We use live instance migration to generalize both live VM migration and live container migration.

1.2.1 Virtualization

Virtual Machine and container are the two standard solutions for virtualized instances for live migrations. In this section, we introduce the runtime of VM and container and memory tracing mechanism that supports the resource virtualization and isolation, which is the foundation for live migration.

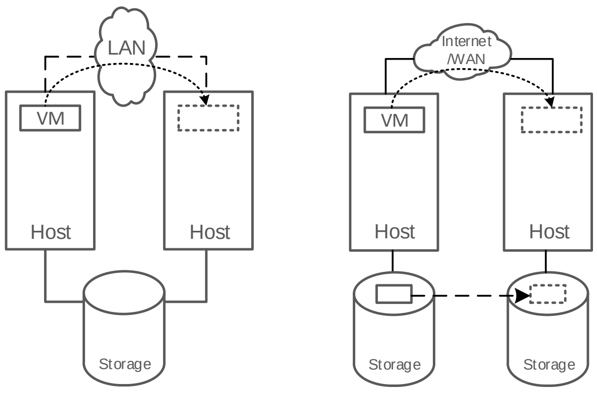
Hypervisor: Hypervisor, known as Virtual Machine Monitor (VMM), is software that creates and runs VM by virtualizing host computer resources, such as memory and processors. The hypervisor inserts shadow page tables underneath the running OS to log the dirty pages [8]. Then, the shadow tables are populated on demand by translating sections of guest page tables. By setting read-only mapping for all page-table entries (PTE) in the shadow tables, the hypervisor can trap the page fault when the guest OS tries to modify the memory page. In addition, libvirt, as an open-source toolkit for hypervisor management, is widely used in the development of cloud-based orchestration layer solutions. Integrating it with hypervisors, such as libvirt and KVM/QEMU, one can track the details of live migration through management user interface command *virsh domjobinfo* including dirty page rate, expected downtime, iteration rounds, memory bandwidth, remaining memory, total memory size, etc.

Container Runtime: Container runtime is software that creates instances and manages instances on a compute node. Except for the most popular container runtime Docker, there are other container runtimes, such as containerd, CRI-O, runc, etc. CRIU [60] is the de-facto software for live container migration. It relies on the *ptrace* to seize the processes and injects the parasite code to dump the memory pages of the process into the image file from within the address space of the process. Additional states, such as task states, register, opened files, credentials, are also gathered and stored in the dump files. CRIU creates checkpoint files for each connected child process during a process tree checkpointing. CRIU restores processes by using the information in the dump files during the checkpointing in the destination host.

1.2.2 Migration Span

Migration Span indicates the geographic environment where live migrations are performed. It is critical to analyze the migration span since various computing and networking settings and configurations directly affect migration management. In this section, we categorize live migration based on the migration span into LAN (Layer-2), such as intra-data center, and WAN (Layer-3) environment, such as inter-data center and edge-cloud migrations (Fig. 1.3).

Intra-Cloud: Live migrations are the cornerstone of cloud management, such as load balancing and consolidation. The source and destination hosts of intra-cloud migration



(a) LAN (b) WAN

Figure 1.3: Migration span in LAN and WAN environments

are in the same LAN environment. In the intra-data center environment, hosts are often shared the data via Network-Accessed Storage (NAS), excluding the need for live storage transmission (Fig. 1.3(a)). In addition, for the share-nothing data center architecture, management traffic, such as live migration flow, is separated from the tenant data network to alleviate the network overheads of migrations on other services.

Inter-Clouds & Edge-Cloud: Live migration is widely adopted for inter-data center management due to various purposes, such as managing cost, emergency recovery, energy consumption, performance and latency, data transmission reduction, and regulation adoption based on the administrative domains [61, 62]. For the migrations between edge and cloud data centers, strategies often need to consider the trade-off between processing time, network delay, energy, and economy. For example, VNF migration from edge to cloud to reduce the processing time and migrate service from cloud to edge to minimize the network delay [63].

For the migrations across WAN, such as inter-data center and edge-cloud migrations, there is no shared storage and dedicated migration network between the data center sites (Fig. 1.3(b)). Therefore, in addition to the live instance migration focusing on memory synchronization, live storage migration is necessary. It also applies to the

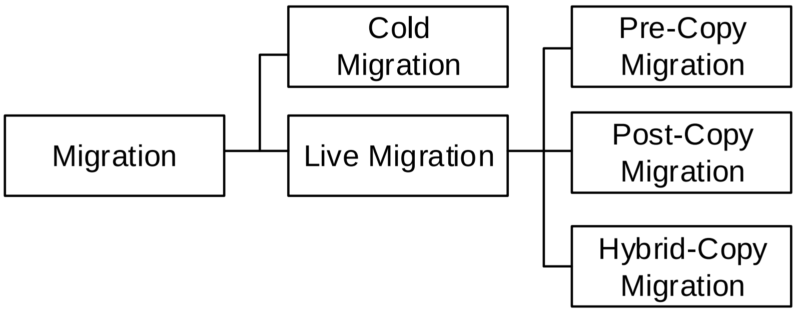


Figure 1.4: Categories of migration types

architecture without shared storage in LAN. The main challenges of migration in WAN are optimizing data transmission and handling the network continuity [6].

Edge Computing: Edge computing includes both LAN and WAN architecture. The motivations and use cases of dynamic resource management in edge computing are similar to those in cloud computing environments. On the other hand, live migration at edges is often referred to as service migration focusing on the mobility-induced migrations in Mobile Edge Computing. In the edge WAN solutions, edge data centers are connected through WAN links as the traditional inter-data center architectures. With the emerging cloud-based 5G solution [64], edge data centers can be connected through dedicated backbone links and shared the regional cloud data center and network storage.

SDN-enabled Solution: In addition, by decoupling the networking software and hardware, SDN can simplify traffic management and improve the performance and throughput of both intra-data centers (SDN-enabled data centers) and inter-data centers (SD-WAN) including migration networking management. As a result, SDN can improve traffic throughput of live migration and reduce the networking overheads of live migration on other services in both LAN and WAN environments [30].

1.2.3 Migration Types

Migration can be applied to different types of virtual resources, such as process, VM, and container, is often referred as the disk-less migration. Figure 1.4 illustrates the categories of migration types. Generally, instance and storage migration can be categorized as cold or non-live migration and live migration. The live migration can be further categorized into pre-copy, post-copy, and hybrid migration. Based on granularity, the migration can be divided into single and multiple migration.

The design and continuous optimization and improvement of live migration mechanism are striving to minimize the downtime and live migration time. The *downtime* is the time interval during the migration service is unavailable due to the need for synchronization. For a single migration, the *migration time* refers to the time interval between the start of the pre-migration phase to the finish of post-migration phases that instance is running at the destination host. On the other hand, the *total migration time* of multiple migrations is the time interval between the start of the first migration and the completion of the last migration.

For the performance trade-off analysis, memory and storage transmission can be categorized into three phases:

- *Push phase* where the instance is still running in the source host while memory pages and disk block or writing data are pushed through the network to the destination host.
- *Stop-and-Copy phase* where the instance is stopped, and the memory pages or disk data is copied to the destination across the network. At the end of the phase, the instance will resume at the destination.
- *Pull phase* where the new instance executes while pulling faulted memory pages when it is unavailable in the source from the source host across the network.

Based on the guideline of these three phases, single live migration can be categorized into three types: (1) pre-copy focusing on push phase [8], (2) post-copy using pull phase [65], and (3) hybrid migration [66] utilizing both push and pull phases. On the other hand, we can categorize multiple migration mechanisms into mainly two types,

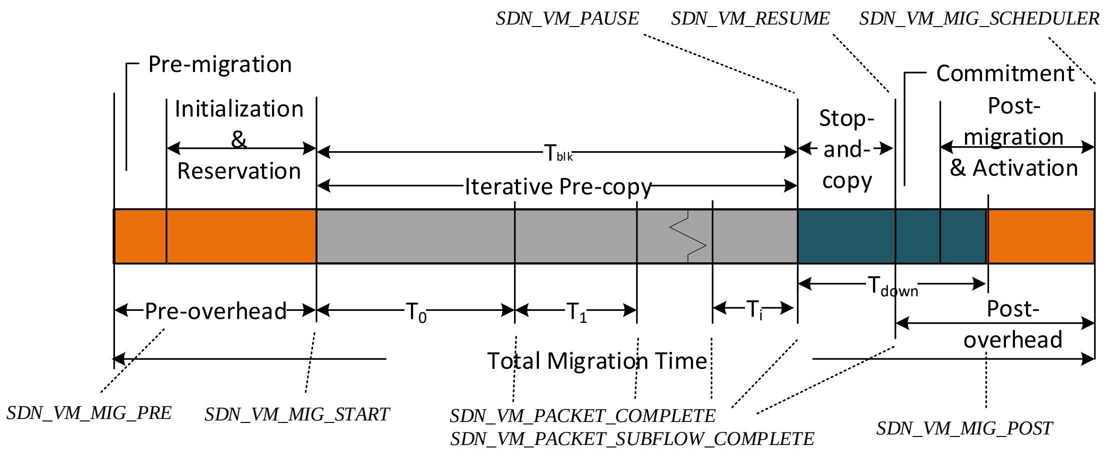


Figure 1.5: Pre-copy Live Migration

namely homogeneous and heterogeneous strategies. Furthermore, pre-migration and post-migration phases are handling the computing and network configuration. During the pre-migration phase, migration management software creates instance's virtual interfaces (VIFs) on the destination host, updates interface or ports binding, and networking management software, such as OpenStack Neutron server, configures the logical router. During the post-migration phase, migration management software updates port or interface states and rebinds the port with networking management software and the VIF driver unplugs the instance's virtual ports on the source host.

Cold Migration: Compared to the live migration, cold memory and data migrations are data transmission of only one snapshot of VM's memory and disk or the dump file of one container checkpoint from one physical host to another. In other words, pure stop-and-copy migration fits into this category. Although provides simplicity over the live migration solution, the cold migration bears the disadvantage that both migration time and downtime are proportional to the amount of physical memory allocated to the VM. It suffers the significant VM downtime and service disruptive.

Pre-copy Migration: During the pre-copy migration [8], dirty memory pages are iteratively copied from the running instance at the source host to the instance container in the destination host. Figure 1.5 illustrates the pre-copy migration phases. Generally, the pre-copy migration of instance memory can be categorized into several phases:

- *Initialization*: preselects the destination host to accelerate the future migration process.
- *Reservation*: set ups the shared file server (optional), and initializes a container of the instance for the reserved resources on the destination host.
- *Iterative pre-copy*: For pre-copy migration, send dirty pages that are modified in the previous iteration round to the destination host. The initial memory states are copied in the first round.
- *Ctop-and-Copy*: the VM is stopped in the source host in the last iteration round according to the downtime threshold.
- *Commitment*: source host gets the commitment of the successfully copied instance from the destination host
- *Activation*: reserved resources are assigned to the new instance and the old instance is deleted.

Post-copy Migration: For the post-copy live migration [65], it first suspends the instance at the source host and resumes it at the target host by migrating a minimal subset of VM execution states. Then, the source instance pro-actively pushes the remained pages to the resumed VM. A page fault may happen when the instance attempts to access the un-transferred pages solved by fetching these pages from the instance in the source. Post-copy strategy can reduce the live migration time and downtime compared with pre-copy migration. However, it is not widely adopted due to the unstable and robustness issue [67]. When the running instance crashes during the post-copy migration, the service will also be crashed as there is no running instance in the original host with full memory states and data. For certain services and applications, the instance needs to constantly pulling faulted pages from the source host while can degrades the QoS for an extensive period.

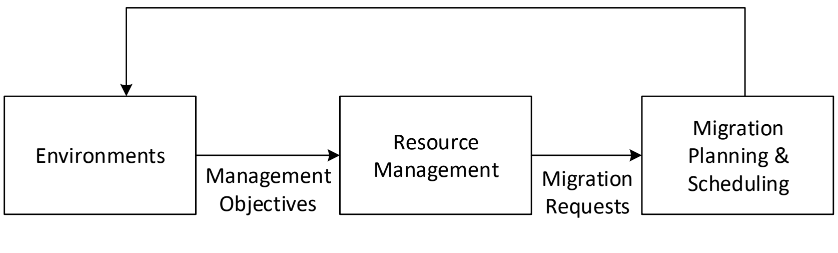


Figure 1.6: A general migration management framework

Hybrid Live Migration: Hybrid post copy [66] aims to reach the balance point by leveraging all three phases. It can also be considered as an optimization technique based on pre-copy and post-copy migrations. It starts with the pre-copy model that iteratively copies dirty pages from source to destination. The post-copy migration will be activated when the memory copy iteration does not achieve a certain percentage increase compared with the last iteration. In certain situations, it will reduce the migration time with slightly increased downtime. However, it bears the same disadvantages of post-copy migration that pulling faulted pages slow down the processing speed which may degrade the QoS, and VM reboot may occur when the network is unstable.

Multiple Migration Type: From the perspective of multiple live migrations, there is the standard homogeneous solution that each migration type is identical and heterogeneous solution [68–70] that performing pre-copy, post-copy, or hybrid copy migrations for multiple migrations at the same time. The heterogeneous strategy aims to improve network utilization and reduce the total migration time of multiple migrations while meeting the requirements of different services with various characteristics.

1.3 Research Problems and Objectives

This thesis aims to address research questions on migration management by focusing on the migration performance and overheads during resource management and migration planning and scheduling algorithms. Figure 1.6 illustrates the general migration management workflow. Based on the various objectives, the resource management algorithms find the optimal placement by generating multiple live migrations. With the generated multiple migration requests, the migration planning and scheduling algorithms optimize the performance of multiple migration, such as total and individual migration time and downtime, while minimizing the migration cost and overheads, such as migration influence on application QoS. On the other hand, the computing and networking resources are reallocated and affected by multiple migrations.

For migration management, it is essential to minimize migration costs, maximize migration performance, while achieving the objectives of dynamic resource management. There are three research questions that dominate migration management when performing dynamic resource management policies and strategies through live migrations:

- **How should the migrations be modeled?**

  To address this question, for the single migration model, we need to specify the types of migrations that should be performed, types of migrating resources, types of optimization, available resources, overhead and cost parameters (variables), and performance metrics. For multiple migrations, we need to identify the resources competitions and contentions among migrations and services, and investigate parameters and performance metrics.

- **How should the migration requests be generated?**

  To address this question, we need to identify the computing and networking environments, such as physical and virtual topologies (micro-services, SFC, and Virtual DCs), as well as the characteristics and objectives of the resource management policies, for example, the generation entities, arrival pattern of migration requests, etc. Considering the objectives of policies, the migration generation manager should determine the migration requests in a way to minimize migration costs and overheads and maximize the performance of both single and multiple migrations.

- **How should the migrations be scheduled?**

  To solve this question, the migration manager needs to efficiently determine the start time and sequence of migrations in sequential and concurrent ways. For single migrations, we should identify the consequence of each migration based on available bandwidth and computing resources before and after each migration, as well as overheads on other resources and services. For multiple migrations, the migration scheduler needs to manage the resources competitions among migrations to minimize the total migration time and determine the migration ordering for the resource deadlock problem. Moreover, the scheduler should schedule migrations on time to meet the urgency, priority, and deadline (scheduling window).

1.4 Evaluation Methodology

The proposed approaches in this thesis have been evaluated in two methodologies, namely discrete event-driven simulation and empirical system. Due to limited accessibility and management costs, simulation is a common evaluation methodology to evaluate the proposed algorithms in a complex and large-scale system. Discrete event-driven simulator supports a more realistic results compared to the mathematical modeling. In this thesis, we developed and extended CloudSimSDN [71] to model the pre-copy live migration, including total/individual migration time, downtime, transferred data, dynamic network routing, bandwidth allocation, and network monitoring. The simulation platform enables the reproducible experiments for large-scale migrations with ease of parameter reconfigurations. We utilize real application and location trace for a realistic results in the simulation. For example, base stations and taxi trace data in Shanghai, request trace data of Wikipedia pages, and network trace data of multi-tier web applications.

We also conducted performance evaluation of live migration requests in a small-scale empirical testbed to pinpoint the parameters and overheads of live migration in SDN-enabled data centers composed by OpenStack cloud management platform and OpenDayLight (ODL) SDN controller.

1.5 Thesis Contributions

To address the research problems mentioned above, this thesis makes the following **key contributions**:

1. A taxonomy on migration management and literature review of the existing migration generation in dynamic resource management algorithms and migration planning and scheduling algorithms.

2. Performance evaluation of live migration in SDN-enabled cloud computing:

   - Comprehensive evaluation of block live migration in SDN-enabled data centers regarding both computing and networking parameters.
   - Evaluation of migration downtime adjustment algorithm.
   - Modeling the trade-off between sequential and parallel migration.
   - Evaluation of the effect of flow scheduling update rate on TCP/IP.
   - Response time pattern of a multi-tier application under various migration strategies.

3. Formalized model and framework for live migration in SDN-enabled cloud computing:

   - A performance model of pre-copy migration and live storage migration.
   - A discrete event-driven simulation framework for each pre-copy migration phase and multiple migration scheduling.
   - A networking model of live migration in SDN-enabled network capable of measuring migration time, iteration rounds, downtime, transferred data, QoS, and energy consumption.
   - A computing model of live migration capable of simulating the pre and post-migration overheads.

4. Concurrency-Aware live migration management algorithm for multiple migration generation:

   - Modeling the potential resource contentions of multiple migration requests according to various source and destination candidates.
   - Integration of single and multiple migration overheads in virtual machine migration.

- A generic migration management for dynamic resource strategies maximizing the multiple migration performance while achieving the objectives of resource management.
- Experimental validation and evaluation of the proposed migration generation algorithm with various dynamic resource management focusing on load balancing, consolidation, and energy consumption.

5. SLA-Aware multiple migration planning and scheduling algorithm:

   - A Mixed-Integer Linear Programming (MILP) model of multiple migration scheduling with various deadlines that minimizes the total migration time and Service Level Objectives (SLOs) violations during scheduling.
   - A resource dependency graph model for multiple migration concurrency and resource dependency.
   - A bandwidth allocation policy to alleviate the migration overheads on other services and improve the migration performance.
   - A deadline-aware concurrent multiple migration planning capable of optimizing total and individual migration performance, QoS, and energy consumption.
   - An SDN-based online migration scheduler to manage the start of migration, bandwidth allocation, and traffic routing.

6. Efficient scheduling algorithm for multiple migration at scale:

   - A revised dependency graph model based on the migration source and destination pairs to reduce the problem complexity.
   - A novel framework of migration scheduling in edge computing for stochastic arrival migration requests.
   - An iterative Maximum Independent Set model for multiple migration scheduling to maximize the performance of migrations with the highest priorities.
   - A novel iterative Maximal Independent Set-based scheduling algorithm to reduce the processing time and efficiently schedule multiple migration requests at large scale.

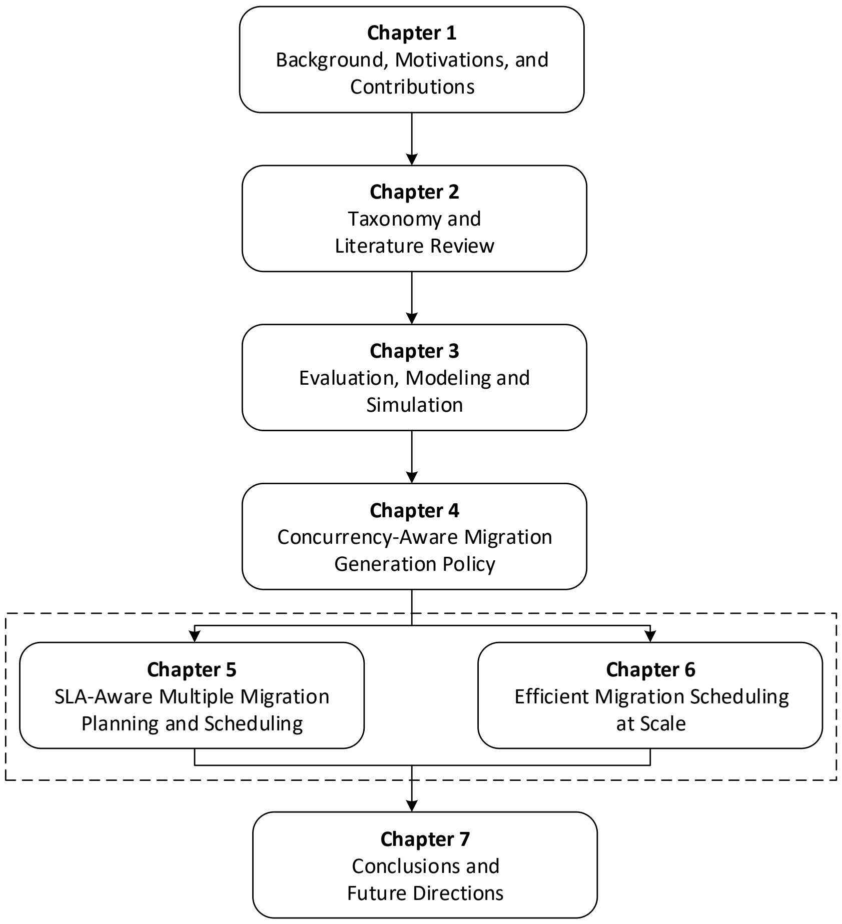

Chapter 1
Background, Motivations, and Contributions
Chapter 2
Taxonomy and Literature Review
Chapter 3
Evaluation, Modeling and Simulation
Chapter 4
Concurrency-Aware Migration Generation Policy
Chapter 5
SLA-Aware Multiple Migration Planning and Scheduling
Chapter 6
Efficient Migration Scheduling at Scale
Chapter 7
Conclusions and Future Directions

Figure 1.7: The thesis structure

- Edge data center placement and network topology based on real base station positions and taxi GPS traces.

1.6 Thesis Organization

The structure of this thesis and content of each chapter are shown in Figure 1.7. The rest of the thesis is organized as follows:

- Chapter 2 presents a taxonomy and literature review of migration management in edge and cloud computing. This chapter is derived from:

- **TianZhang He**, Rajkumar Buyya, "A Taxonomy of Live Migration Management in Edge and Cloud Computing", *ACM Computing Surveys* (submitted, July 2021).

- Chapter 3 presents modeling, evaluation and simulation of live migration in SDN-enabled Cloud data centers. This chapter is derived from:

  - **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "Performance evaluation of live virtual machine migration in SDN-enabled cloud data centers", *Journal of Parallel and Distributed Computing (JPDC)*, Volume 131, Pages: 55-68, Elsevier, 2019.

  - **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "SLA-aware multiple migration planning and scheduling in SDN-NFV-enabled clouds", *Journal of Systems and Software (JSS)*, Volume 176, Pages: 110943, Elsevier, 2021.

- Chapter 4 proposes a concurrency-aware live migration management algorithm of multiple virtual machines that deals with mitigating the migration overheads and achieving the objectives of dynamic resource policies. This chapter is derived from:

  - **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "CAMIG: Concurrency-Aware Live Migration Management of Multiple Virtual Machines in SDN-enabled Clouds", *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, (minor revision, August 2021).

- Chapter 5 proposes the deadline-aware multiple migration grouping algorithm and online migration scheduler to determine the sequence of VM/VNF migrations. This chapter is derived from:

  - **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "SLA-aware multiple migration planning and scheduling in SDN-NFV-enabled clouds", *Journal of Systems and Software (JSS)*, Volume 176, Pages: 110943, Elsevier, 2021.

- Chapter 6 presents a multiple migration planning and scheduling algorithm to deal with stochastic migration requests at scale. This chapter is derived from:

  - **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "Efficient Large-Scale Multiple Migration Planning and Scheduling in SDN-enabled Edge Computing", *IEEE*

Transactions on Mobile Computing (TMC), (submitted, June 2021).

- Chapter 7 concludes the thesis and summarizes the key findings and identifies future research directions. This chapter is derived from:

  - **TianZhang He**, Rajkumar Buyya, "A Taxonomy of Live Migration Management in Edge and Cloud Computing", *ACM Computing Surveys* (submitted, July 2021).

Chapter 2
A Taxonomy and Review on Migration Management

This chapter proposes a taxonomy of migration management in edge and cloud computing and explains each aspect in details. We present a conceptual system architecture for migration management. A detailed survey and analysis of existing approaches is conducted, including migration generation and migration planning and scheduling. We also present various simulation, emulation, and empirical evaluation methods with corresponding data that have been developed for live migration.

2.1 Introduction

To widely and efficiently apply dynamic resource management algorithms, it is necessary to investigate migration management to minimize the migration overheads, maximize the performance of migration scheduling, and optimize the objectives of resource management.

Although many surveys [6, 7, 25, 72–78] of live migration have been presented in the contexts of performance, mechanism, optimization of single live migration and general migration-based dynamic resource management, they only focus on the specific migration aspects and neglect the aspects of migration management including migration generation in dynamic resource management and migration planning and scheduling algorithms. Therefore, in this chapter, we identify the following five aspects of migra-

This chapter is derived from:

- **TianZhang He**, Rajkumar Buyya, "Live Migration Management in Edge and Cloud Computing Environments: A Taxonomy, Review and Future Directions", *ACM Computing Surveys*(submitted, July 2021).

tion management in both edge and cloud computing environments:

- migration performance and cost model
- migration generation in resource management policies
- migration planning and scheduling
- migration lifecycle management and orchestration
- evaluation methods and platforms

Since dynamic resource management algorithms require multiple instances migration to achieve the objectives, we also emphasize migration management in the context of multiple migrations solutions and challenges. Based on the proposed taxonomy, we review related state-of-art works in each category and identify the gaps.

The rest of this chapter is organized as follows. Section 2.2 summarizes the related surveys, following the system architecture overview in Section 2.3. Section 2.4 presents the proposed taxonomy of migration management. Section 2.5 describes the details of essential aspects of migration planning and scheduling algorithms. We review and analyze the related works of migration management including migration generation and migration planning and scheduling in Section 2.6. Evaluation methods and tools including simulation, emulation and empirical platforms are introduced in Section 2.7. Finally, we conclude the chapter in Section 2.8.

2.2 Related Surveys

In this section, we introduce the details of related surveys in the context of live migration. Several surveys are conducted to investigate and summarize the works on various aspects of live migration, including migration elements, migration types, migration overheads, optimization mechanisms, motivations and objectives, robustness and security, networking continuity, etc. We summarize the related surveys and illustrate the level of details covered on the respective issue in Table 2.1, where the cost aspect includes migration performance and overhead cost, the mechanism aspect includes migration type and optimization of performance or overhead, the application aspect includes

Table 2.1: Summary of related surveys in live migration

reference	Migration Issues				Resource Type				Environment		Granularity	
	cost	mechanism	application	management	VM	container	network	storage	cloud	edge	single	multiple
[72]	✓	✓	-	-	✓	-	-	-	✓	-	✓	
[73]		✓	-		✓		✓				✓	
[74]	✓	✓			✓				✓		✓	
[75]		✓	∂		✓	∂	∂	∂	∂		✓	
[76]		✓			✓	✓					✓	
[77]		✓			✓		∂	∂			✓	
[6]	✓	✓	✓		✓		✓	✓	✓	-	✓	∂
[7]		∂	✓	∂	✓	✓	∂			✓	✓	
[78]		✓			✓				✓		✓	∂
[25]	✓		✓		∂	∂	✓			✓		
this	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓ denotes broad discussion or the main scope on the respective issue.
∂ denotes partial discussion or the secondary scope on the respective issue.

migration motivations and use cases for resource management, and the migration management aspects include migration generations with different resource policies, policy-level migration networking management, single and multiple migration planning and scheduling. The migration granularity consists of single and multiple migration for heterogeneous and homogeneous migration types, co-located VMs in the same source and destination, and VMs from and to arbitrary hosts. With the development of live migration in all aspects, the main topics widely discussed in related surveys may only cover the limited scope of existing works at the time of publication.

Strunk [72] investigated and reviewed the works on the single VM live migration parameters in terms of physical machine's CPU and net utilization, VM's CPU and net utilization, memory size, and dirty page rate, a taxonomy of migration cost parameters, performance prediction modeling, migration overheads such as service performance loss and migration energy consumption. Similarly, Xu et al. [74] also reviewed and summarized the performance overheads of live VM migration in the Infrastructure-as-a-Service (IaaS) cloud. The survey investigated various causes and scenarios for VM performance overheads during migrations and the performance and overhead modeling methods and compared the complexity and effectiveness of various overhead mitigation techniques. Medina et al. [75] reviewed the works focusing on the mechanism of VM migration and process replication for the purpose of high availability. They also reviewed the mechanism and implementation of the hypervisor (Xen, VirtualBox, KVM, and VMWare) that supports live VM migration, live VM and storage migration across the WAN, and live migration use cases of load balancing, overloaded host management, and energy efficiency. They also partially covered few works on trace/replay technique and container migration by OpenVZ.

Yamada et al. [77] and Noshy et al. [78] reviewed the live migration mechanisms for pre-copy, post-copy, and hybrid migration and corresponding optimizations, such as memory compression, deduplication, checkpoint/restore or trace/replay, pipeline and multicore parallelization. Noshy et al. [78] also mentioned the research directions on live migration of multiple VMs in total migration time and impacts on co-located VMs. Zhang et al. [6] presented a comprehensive live VM migration survey mainly talking about migration motivations, migration types and optimization techniques, network layer-2 to layer-4, and SDN solutions for the network continuity issue over the WAN. They also reviewed the papers of multiple migrations on optimizing the total migration time for co-located VMs in the same host and VMs with network connections. The authors also partially reviewed the works on migration overheads on other services and had limited coverage on the difference between single and multiple migrations.

For live container migration, Milojičić et al. [79] summarized the early efforts for live process migration and indicated the challenges that process migration need to be solved. From the High Availability (HA) perspective, Li et al. [76] compared VM and container in virtualization mechanisms and implementation difference between the hypervisor and container-based platform for live migration. The authors also reviewed the works of failure detection based on CRIU and OpenVZ and pointed out the HA and optimization features missed in container-based platforms.

For the survey of mobility-induced service migration focusing on service and user mobility, Machen et al. [23] reviewed the works of live service migration in Mobile Edge Computing (MEC). They investigated the layered framework, live container migration, and the performance evaluation of various applications, including gamer server, RAM simulation, video streaming, and face detection (OpenCV). Wang et al. [7] reviewed the works of service migration in MEC. The service can be hosted by different types of instances such as VMs and containers. The authors mainly focus on the MEC context including the framework of the service migration flow, data transmission optimization, and strategies for service migration decisions. Specifically, the authors reviewed the strategies for service migration (dynamic service placement) in edge computing in (1) following the mobile users, (2) MDP-based service migration with one and two-dimensional optimization, and (3) time window-based service migration. In addition, Rejiba et al. [25] also focused on the mobility-induced service migration in edge-centric computing environments, including migration elements, migration and transmission cost. The proposed a taxonomy and review of dynamic resource management algorithms through mobility-induced live migrations based on different objectives, such as cost, time, and migration success rate.

For the security and robustness, Shetty et al. [73] focused on live migration security, including secure live migration, control policies (DoS, Internal, Guest VM, false resource advertise, Inter VM in the same host), transmission channel (insecure and unprotected), and migration module (stack, heap, integer overflow). Zhang et al. [6] reviewed the robustness aspect of different migration types. Kokkinos et al. [80] focused on live migration in long-distance networks (WAN) for disaster recovery. We observe that most surveys only investigated the single live migration at the OS system level and dynamic resource management through live migration. Therefore, in this chapter, we summarize and review the state-of-art works of live migration in the context of migration management in edge and cloud computing, including the performance and cost model, migration generations in resource management algorithms, migration planning and scheduling, migration lifecycle management and orchestration, and evaluation methods.

2.3 System Architecture

In this section, we explain the details of system architecture for migration management. The system architecture of SDN-enabled edge and cloud data centers consists of an orchestration layer, a controller layer, and a physical resource layer (Fig. 2.1). The policy orchestrator provides the algorithms and strategies for joint computing and network-

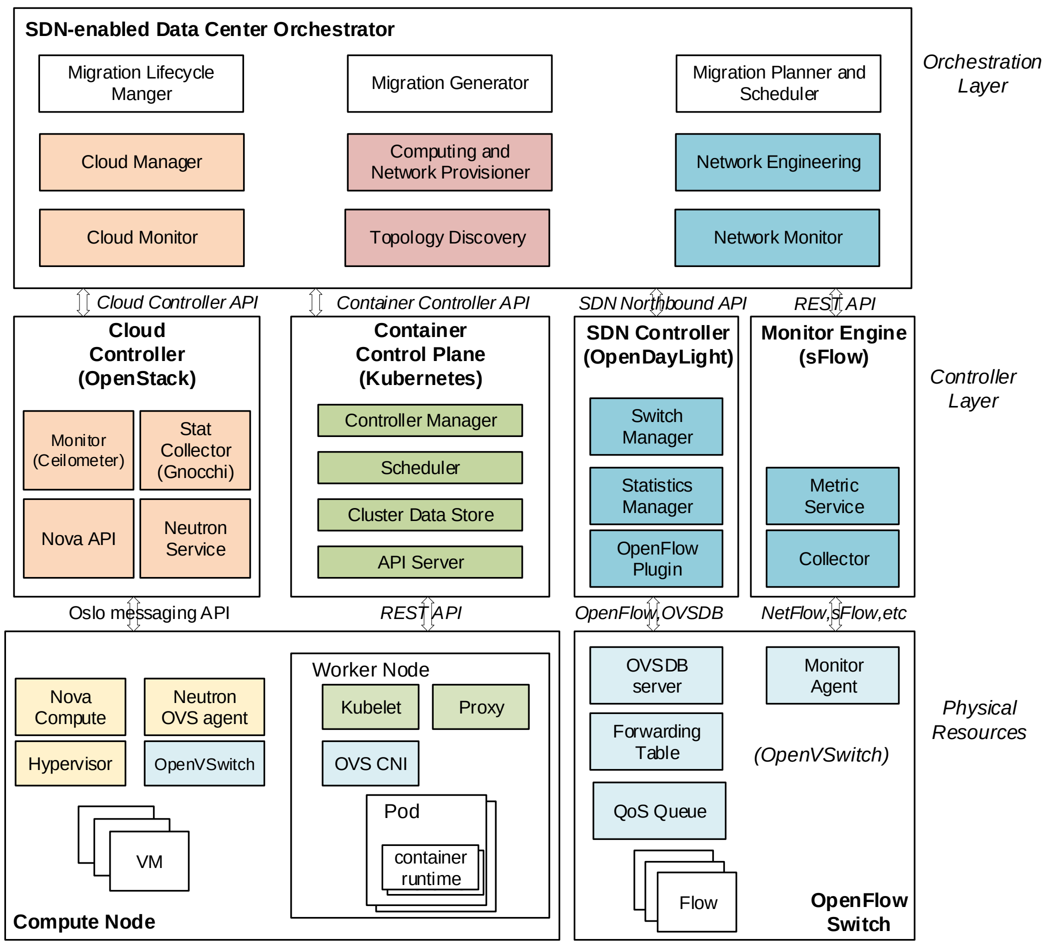


Figure 2.1: System Architecture and its software components

ing resource provisioner and network engineering server, which based on the information from network topology discovery component and the cloud and network monitors. Combined with cloud manager and network engineering server, live migration manager (lifecycle manager, migration generator, planner, and scheduler) can efficiently control the lifecycle of single migration and schedule multiple migrations in a finer granularity by jointly provisioning computing and networking resources.

In the controller layer, there are several individual controllers, including the networking manager (SDN controller), resource monitor engines, cloud controller (such as OpenStack), and container control plane (such as Kubernetes). Based on the controllers, strategies of cloud manager and container orchestrator are responsible for automating deployment, autoscaling, and management of the VMs and containerized applications. Popular container orchestrators include Kubernetes, DC/OS on the Apache Mesos, and Docker Swarm. Kubernetes is the de-facto open-source container orchestration and OpenStack is the most popular open-source cloud management platform. Therefore, we use Kubernetes and OpenStack as the example to illustrate the system components.

A Kubernetes cluster consists of container control plane components and node components. The container control plane consists of cloud controller manager, controller manager, scheduler, and API server components. It reacts and performs the decision for the clusters. The API server exports the control plane APIs. The consistent and highly available key-value store (etcd) is used as the backing store for all cluster data. Scheduler (kube-scheduler) is a broker to allocate the new pods to an available work node to run on. The controller manager runs node, job, endpoints, service account, and token controllers, which responses to node lifecycle, pod creation, endpoint objects population, account and API access token creation, respectively.

The Cloud control plane consists of the monitor (Ceilometer), statistics collector (Gnocchi), compute service (Nova), and network service (Neutron). The monitor service (OpenStack Ceilometer) collects, normalizes, and transforms data produced by other cloud services, and the statistics collector (Gnocchi) provides the time-series data storage. The compute service (OpenStack Nova) provides virtual machines provisioning. The network service (OpenStack Neutron) provides the management of virtual networking, such as start, update and bind the VM's port, as well as the communication between VMs. However, it does not control network devices (switches) but only controls networking modules in compute nodes and network nodes. The compute and network services communicate with corresponding agents (Nova agent and Neutron OVS agent) in each compute node through service APIs (Oslo messaging APIs) for the computing resource provisioning of VMs and network interfaces for interconnection.

The SDN controller manages the OpenFlow switches and flow forwarding entries and collects the device and flow statistics through southbound interfaces. Networking applications through SDN controller northbound interfaces perform topology discovery, network provisioning, and network engineering policies.

There are continuous efforts to integrate container networking in Kubernetes with OpenStack, such as Kuryr Kubernetes [81]. Open Virtual Network, an OVS-based SDN solution, is widely adopted in industrial cloud networking. With the help of Open Virtual Network (OVN) and OVS as the data plane, Kube-OVN [82] integrates the OVN-based network virtualization with Kubernetes offering some advanced overlay network features such as subnet, QoS, static IP allocation, traffic mirroring, and OpenFlow-based network policy. As shown in Fig. 2.1, the CNI driver will add the pod interface to OVS and connect it to OVN. As a result, it also provides the network interconnection with OpenStack and SDN controller, which enables the global networking topology visibility.

A Kubernetes cluster consists of a set of worker nodes that run containerized applications. Each worker node host several pods that are the components of the application workload. The node components include node agent (kubelet), network proxy (kube-proxy), and container runtime. The kubelet is the worker node agent running on each node that registers the node with the API server. The network proxy that runs on each node in the container cluster maintaining network rules on nodes.

The OpenFlow switch implemented by OpenVSwitch (OVS) consists of QoS Queue, flow forwarding table, monitor agent, and OVSDB server. The OVSDB management protocol is used to configure the OVS, such as creating, modifying, deleting the bridges, ports, and interfaces. OpenVSwitch supports flow QoS, such as dropping ingress traffic when exceeding the configured threshold and perform traffic shaping with traffic queuing. It also supports monitor agents, such as the sFlow agent, to perform real-time flow monitoring and analytics.

2.4 Taxonomy of Migration Management

In this section, we present the taxonomy of migration management in five aspects (Fig. 2.2), namely: (1) performance and cost model; (2) resource management and migration generation; (3) migration planning and scheduling; (4) migration lifecycle management and orchestration, and (5) evaluation method.

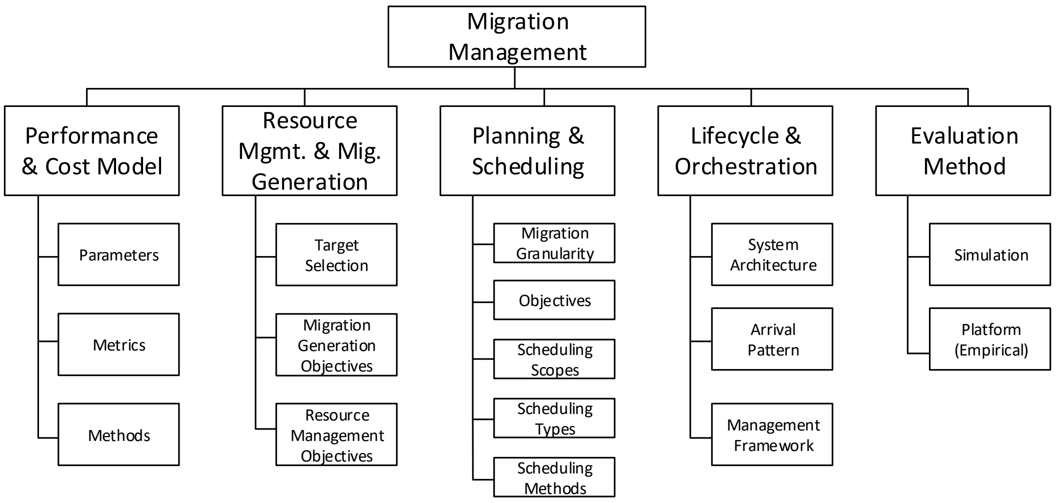


Figure 2.2: Taxonomy of migration management

2.4.1 Migration Performance and Cost Modeling

The migration performance and cost modeling is the fundamental element for migration management to evaluate and predict the total cost and scheduling performance of multiple migration requests. In this section, based on the related literature and our observation, we identify the *parameters*, *metrics*, and *modeling methods* involved in the performance and cost model of live migrations.

Parameters

Table 2.2 illustrates the parameters (variables) involved in live migration under three categories: *computing*, *networking*, and *storage* resources. Moreover, we also identify the migration parameters in the granularity aspect: *single* and *multiple* migration. The *computing resource* parameters include CPU load and utilization, memory load and utilization, memory size, dirty page rate, WSS size as frequent updating memory pages for live migration optimization, and I/O interfaces (i.e.g. cache interface, host network card interface, inter-CPU interfaces). The *networking resource* parameters include the migration routing, available bandwidth (link and routing bandwidth for single and multiple paths), migration distance (the number of network hops), the number of involved net-

Table 2.2: Parameters of the live migration model

Category	Parameters				
Computing	CPU utilization	memory utilization	I/O interface	WSS size	
Networking	bandwidth	interfaces	routing	delay/distance	layers/hops
Storage	sharing	data size	storage type	read/write speed	
Single migration	dirty page rate	iterations threshold	downtime threshold	configuration delay	priority
Multiple migration	migration impact	concurrency	migration time	routing	scheduling window

Table 2.3: Metrics of live migration performance

Category	Metric		
Time	migration time	downtime	deadline/ violation time
Data	dirty memory	storage	stop-and-copy size
QoS	response time	network delay	available resource
Energy	physcial host	network device	cooling
SLA	service availability	success ratio	policy performance

work layers, network delay (link delay and end-to-end latency). In addition, the *storage-related* parameters include writing and reading speed and storage data size.

The single live migration parameters include dirty page rate, the threshold of pre-copy iteration rounds, downtime threshold for pre-copy migration, configuration delay in pre and post-migration processes, and the priority of the migration request. On the other hand, we need to consider several parameters of multiple migration scheduling, such as the migration impact on other services and subsequent migrations, the concurrency of multiple migration scheduling with resource contentions among migrations, the single migration time in multiple migration scheduling, the migration routing considering the traffic of other services and migrations, and the scheduling window of each migration with various priorities and urgencies.

Metrics

Many works have investigated the metrics of single migration performance. However, few works are focusing on the performance metrics of multiple migrations. Therefore, we also extend the investigation of the single live migration metrics to the multiple migrations in these categories. As shown in Table 2.3, we categorize these metrics into different categories: *time*, *data*, *QoS*, *energy*, and *SLA* related. We explain each metric as follows.

Migration Time: Migration time is one of the two main metrics used to evaluate the single migration performance and overhead. A large migration time normally results in a large overhead on both computing and networking resources for the migration VMs and other services.

Downtime: Downtime is another key metric used to evaluate the single migration performance. During the downtime caused by migration, the service is not available to the users.

Iteration Number: For migration types utilizing the pre-copy strategy, the number of iteration rounds is a direct parameter affecting the migration converging hence the migration time and downtime.

Data Transmission Size: It is the key metric to judge the network overhead during the migration across the network. For pre-copy migration, it is highly positively correlated to the migration time. The total amount of data transmission is the sum of the data amount of each instance. It can be divided into two aspects: memory data and storage data.

Total Migration Time: The total migration time of multiple migrations is the time interval between the start of the first migration and the completion of the last migration. This is the key metric to evaluate the multiple migration performance and overheads.

Average Migration Time: The average migration time is the average value of individual migration time of all instances within a time interval. With the continuous arrival migration requests, the average migration time is preferable to the total migration time. The total migration time of a bunch of instances is only a suitable metric for the periodically triggered source management strategies.

Average Downtime: Similar to the average migration time, the average downtime is the mean value of individual downtime of all instance migrations within a time interval. Time unit, such as millisecond (ms) and second (s), is used for migration time and downtime.

Energy Consumption: Energy consumption consists of the electricity cost, green energy cost, cooling cost, physical host, networking devices cost. It is a critical metric of live migration overheads used for green energy algorithms and data centers. Joule (J) is used as a unit of energy and Wh or KWh is used in electrical devices.

Deadline Violation: Migration request or task is the key element for the multiple migration scheduling algorithms. Service migrations with different time requirements will have the corresponding deadline and scheduling window. As a result, the number of deadline violations for migrations with different priorities and urgency is the key metric to evaluate a deadline-aware or priority-aware migration scheduling algorithm.

Resource Availability: The remaining computing, networking, storage resources during and after the single migration and multiple migrations. It is critical for migration success rate as there should be sufficient resources for the new instance in the destination. Furthermore, resource availability during the migration affects the performance of the subsequent migrations. Resource availability after the migration is also a metric for the resource reconfiguration evaluation for various policies.

Quality of Service (QoS): QoS metrics of the migrating service and other services (co-located VMs, shared-resource VMs, connected VMs) include response time, end-to-end delay, network delay, and network bandwidth during and after the single migration or each migration during multiple migration scheduling.

Service Level Agreement (SLA): Migration may cause service unavailable due to the migration-related issues, such as downtime, network continuity, network robustness, and migration robustness. Therefore, SLAs are provided to subscribers and tenants as the availability rate for the services with and without migrations. Therefore, SLA violations is another critical metric.

Modeling Methods

This section introduces the different modeling methods for live migration performance and overheads, including *theoretical modeling* and *experimental modeling (profiling and prediction)*.

Theoretical: In theoretical modeling, the system behaviors are described mathematically using formulas and equations based on the correlation of each identified parameters [83–92]. Some works only model the migration costs and performance based on the parameters correlation. Other works follow the behaviors of live migration, such as iterative copying dirty page rate to model, the performance, and overheads in finer granularity.

Profiling: Experimental modeling methods are based on measurements with controlled parameters. Empirical running analysis, such as Monte Carlo, are relied on repeated random sampling and time-series monitoring and recording for migration performance, overheads, and energy consumption profiling [84, 93, 94]. For both overhead and performance modeling, empirical experiment profiling can also derive the coefficient parameters in the model equations [88, 91, 95, 96]. Regression algorithms are also used to model the cost and performance based on the measurement [92, 97].

Prediction: Generally, mathematic cost models can be used to estimate migration performance and overheads. Performance estimation and prediction algorithms [72, 84, 88, 92, 96, 98] are proposed to simulate migrations processes to minimize the prediction error. Furthermore, Machine Learning (ML)-based modeling [99, 100] are adopted to generalize parameters in various resources to obtain a more comprehensive cost and performance prediction model.

Cost and Overhead Modeling

The cost and overhead models of live migration are integrated with the policy objectives for modeling and optimizing the dynamic resource management problems. Similar to the migration parameters, the cost and overhead modeling can be categorized into computing, networking, storage, and energy caused by virtualization and live migration (see survey [74]), migration influence on subsequent migrations and co-located services [28, 29, 90, 101, 102], and migration networking competitions with each other in the multiple migration context [29, 30, 103].

Performance Modeling

For resource management and migration scheduling algorithms, the performance models of single and multiple migrations are used to maximize the migration performance whiling achieving the objectives of resource management. The category of performance model is similar to the performance metrics classification. In other words, performance models can be categorized as migration success rate, migration time, downtime, transferred data size, iteration number, and deadline violation [74]. With the complexity of modeling multiple migrations, it is difficult to model the performance of multiple migration directly on total migration time. Therefore, multiple migration performance and cost models are focusing on the single migration performance based on the currently available resources during multiple migration scheduling [102, 103] and the involved parameters for the resources contentions, such as shared network routing, shared links, shared CPU, memory, network interfaces, and the number of migrations in the same host [28–30, 104].

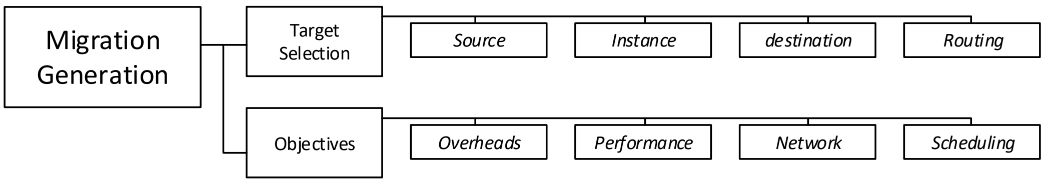


Figure 2.3: Categories of migration generation in dynamic resource management

2.4.2 Migration Generation in Resource Management

In this section, we discuss migration request generation of resource management algorithms. We investigate how the migration performance and overhead models integrated with the policy affect the optimization problems and migration generations in two aspects: *migration target selection* and *migration generation objectives*. Figure 2.3 illustrates the details of each aspect.

Migration Target Selection

The targets of one migration include the selections of source, destination, instance, and network routing. During the migration generation for resource management management, the targets of migrations can be selected simultaneously or individually. For simultaneous solutions, such as approximation algorithms, several migration instances, source or destination hosts, and network routings can be generated at the same time. For individual solutions, such as heuristic algorithms, each migration request can be generated once at a time in each algorithm loop.

Source Selection: The source host of one migration request is selected based on the objectives of resource management policies, such as failure, resource utilization, energy consumption, over-subscripted host, and under-subscripted host.

Instance Selection: During the instance selection for migration request, the migration generation algorithm needs to consider the various objectives of resource management policy, the availability of resources in potential destinations, and the overheads of live migration, such as dirty page rate and the number of allowed migrations. For use cases such as gang migration, disaster recovery, software update, and hardware maintenance, all instances within the source hosts or sites will be selected.

Destination Selection: Many works considered the selection of migration destination as a bin packing problem, where instances as items with different volumes need to be packed into a finite number of bins with fixed available volumes in a way to minimize the number of bins used. There are several online algorithms to solve the problems, such as Next-Fit (NF), First-Fit (FF), Best-Fit (BF), Worst-Fit (WF), etc. By considering both objectives of various resource management and migration overheads and performance, heuristic and approximation algorithms are proposed.

Flow Routing: The available bandwidth and network delay are critical for migration performance, such as migration time and downtime. The migration flows of pre-copy migration are elephant flows and post-copy migrations are sensitive to the network delay. Meanwhile, in the network architecture where the migration traffic and service traffic share network links, the migration traffic can significantly affect the QoS of other services. In the SDN-enabled data centers, the allocated bandwidth and the routings of both migration and services traffic can be dynamically configured to improve the performance of live migration and minimize the migration network overheads.

Migration Generation Objectives

In this section, we summarize the different objectives of migration selections in resource management policies from the perspective of live migration management, including *migration overhead*, *performance*, *network*, and *scheduling awareness*. Many works are proposed in dynamic resource management for various objectives, such as performance, networking, energy, QoS, and disaster recovery. Most of the works only consider the memory footprint (memory size) and available bandwidth. Without the proper live migration modeling, the selected migration requests for instance reallocation and corresponding scheduling will result in unacceptable scheduling performance and service degradations. Some works consider the individual migration performance and computing and networking overheads during the migration generation or instance placement selection for the dynamic management policy. The total migration cost, as a result, is only a linear model, which can not reflect the concurrency among migrations and the resource contentions between migrations and services. As a result, the solution is only optimized for sequential live migration solutions.

Overhead-aware: Many works of resource management policies are focusing on minimizing the cost or overheads of migrations and modeling the total cost as a sum of the individual live migration cost. Mann et al. [105] allocated minimal essential bandwidth for a migration flow that finished just in time to minimize the QoS violations during migrations. It may only suit the sequential scheduling with one-by-one migration as it can not model the multiple migrations contending network bandwidth in concurrent scheduling.

Performance-aware: Most of the resource management solutions through live migrations do not consider and guarantee migration performance. Some works are focusing on the multiple-objective optimization integrating with the objectives of resource management, such as the cost and performance of migrations [90, 104, 106–108] .

Network-aware: Live migration traffic and application traffic are sharing network resources. Therefore, we need to consider the network contentions and instance connectivity to optimize the networking throughput and communication cost during or after the migrations [27, 109–113].

Scheduling-aware: Current works do not consider the migration scheduling performance with the linear model of migration cost and interfaces. In the migration generation phase of resource management policies, we can optimize the performance of single or multiple migration scheduling and guarantee the optimal or the near-optimal performance of the objectives of the original resource management policy. Compared to policy-aware migration management, it is more adaptive without the need for specific modeling and design. It has less impact on the enormous amounts of existing dynamic resource management policies.

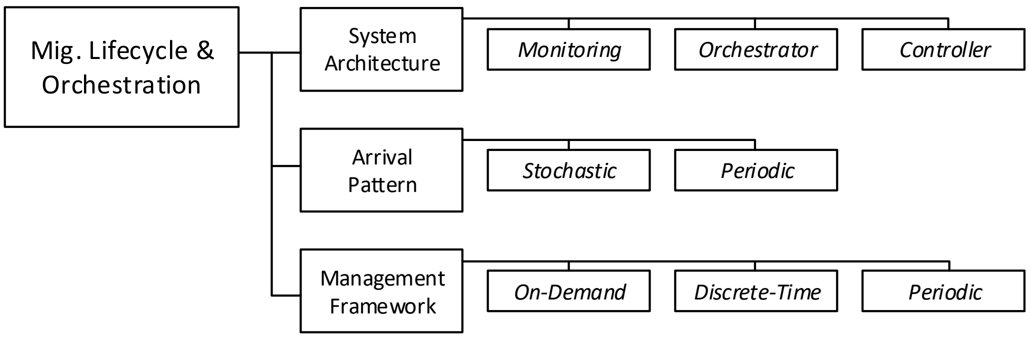


Figure 2.4: Categories of migration lifecycle management and orchestration

2.4.3 Migration Lifecycle and Orchestration

Based on various scenarios of dynamic resource management through live migrations, it is critical to investigate the migration lifecycle and orchestration layer including migration arrival patterns and corresponding management framework. Therefore, this section summarizes the migration lifecycle management and orchestration in the several aspects: *arrival pattern*, *orchestration*, *monitoring*, and *management framework*. Figure 2.4 illustrates the details of each aspect.

Arrival Patterns

Arrival patterns of migration requests based on various paradigms and objectives in fog, edge, and cloud computing environments can be categorized as *periodic* and *stochastic* pattern. For existing dynamic resource management algorithms, the migration generation and arrival patterns are periodic due to the overhead of live migrations. The dynamic resource management, such as regular maintenance and updates, load balancing, and energy-aware consolidation algorithms, triggered the reallocation periodically. On the other hand, arrival patterns of event-driven migrations are often stochastic due to the nature of the service. For example, the mobility-induced migration in edge computing is based on the user behaviors and movement.

Orchestration

Several orchestration systems have been proposed to control computing and network resources in fog, edge, and cloud computing environments [114–119]. Generally, the system architecture includes the orchestration layer on top of the cloud manager (such as VM manager) and networking manager (such as SDN controller) that perform resource provisioning, allocation, monitoring, and billing. In the WAN environments with several sites across IoT, edge, and cloud domains, a global orchestration layer oversees the global networking and service provisioning on top of the individual orchestrator in each data center site [20, 119]. Based on the joint computing and networking provisioning, migration algorithms are located in the orchestration layer to perform the high-level migration optimization, planning, and scheduling.

Monitoring

Based on the monitor components of cloud manager and network manager, migration monitor is a key component for migration management applications. The states and availability of computing and networking resources are essential for migration lifecycle management and migration planning and scheduling. The monitors collect real-time information of computing and networking resources at computer nodes, virtualized instances (VMs and containers), and network devices. Based on the resource types, monitoring can be categorized as computing, networking, storage, and energy monitoring.

For computing resources, we need kernel-level monitoring for CPU, memory utilization, network interfaces, memory tracing for dirty page rate and WSS, and process tree state parsing in user space for containers, machine-level monitoring for available resources supported by VM hypervisor and container runtime, and application-level monitoring such as cloud service daemon availability and functioning, instance reachability, and processing time. For example, Ceilometer and Gnocchi components are services for measuring and collecting the usage metrics (e.g. CPU time) and event data (e.g. instance creation) for OpenStack clouds. OpenStack Aodh based on the measurements from Gnocchi is an alarm notification service, which can trigger the policy actions based on the pre-defined Service Level Objectives (SLOs) level.

The network monitoring can be categorized as packet-level and network-level monitoring. For the packet-level monitoring, network management and sampling protocols, such as SNMP [120], NetFlow/IPFIX [121], and sFlow [122], are used. Simple Network Management Protocol (SNMP) [120] is an Internet standard protocol for collecting, organizing, and modifying the information of managed network devices. It is widely used in network management for network monitoring. Cisco NetFlow and IPFIX [121] are adopted to collect the IP traffic information and monitor the network flow. For the sFlow protocol [122], it can provide real-time measurements of the network by capturing sampled packets through the sFlow agent in the switches. The sFlow engine located in the controller gathers the information from sFlow agents through RESTful APIs. By integrating traffic analysis techniques with SDN and OpenFlow protocol, network analytics are enabled for anomaly detection and mitigation [123–125], network-wide traffic visibility [126], and large flow detection and mitigation [127], which directly benefits the network management of live migration. For the network-level monitoring, SDN controller through OpenFlow protocol can collect device information and flow statistics measured at switches' data plan [128–131], which enables the network application on network topology monitoring, traffic engineering, and abnormal detection.

Storage and volume monitoring is also essential for determining the disk and space availability, space requirement, reading and writing speed, the status of storage migration, and the usage of cache space on disk storage. For example, OpenStack Swift Recon is the object storage monitoring middleware collecting the general machine statistics and storage-specific meters. Energy consumption monitoring for both physical hosts and network devices is critical for migration overheads and performance in the energy-aware data centers [132, 133]. For example, smart meter energy monitors and Power Distribution Units (PDU) offer the data center managers fully intelligent data center power distribution units and monitoring ability.

Management Framework

Based on characteristics of resource management policies in various scenarios and use cases, the triggered pattern of migration planning can be categorized into periodic, discrete-time, and on-demand types.

On-Demand: In the on-demand framework, the migration will be planned and scheduled whenever the request arrives. The on-demand framework can be applied to the scenarios that individual users and events are the subjects to trigger the migration from one host to another, such as mobility-induced migration and migration requests by public cloud subscribers and tenants.

Discrete-Time: In the discrete-time framework, arrival migration requests are put into plan waiting queues, which regulates the migration arrival speed and processing speed of planning algorithm. The migration planner will read migration requests from the plan waiting queues and unscheduled migrations from previous rounds periodically at the start of each configured time interval, such as one second. For each input, the planner will calculate the migration plan based on the migration priority and states of computing and networking resources. The planned migration requests will be waited in the queue to be started by the migration scheduler according to the plan. Since the time interval of each input can be very small, some unscheduled requests in the previous migration plan may affect the decision of the current planning round. The discrete-time framework suits the scenarios where migration can arrive stochastically and arrive much more frequently than the traditional dynamic resource management strategies, such as mobility-induced migration in edge computing.

Periodic: In the periodic planning framework, migration plans are calculated based on periodically arrived migration requests from the dynamic resource management algorithm. The time interval between each migration planning can be configured as the value of the resource management interval. With a large time interval, multiple migration requests for the instance reallocation will not be affected by the migrations from the previous round. Within each time interval, all instance migrations can be completed before the start of next planning round.

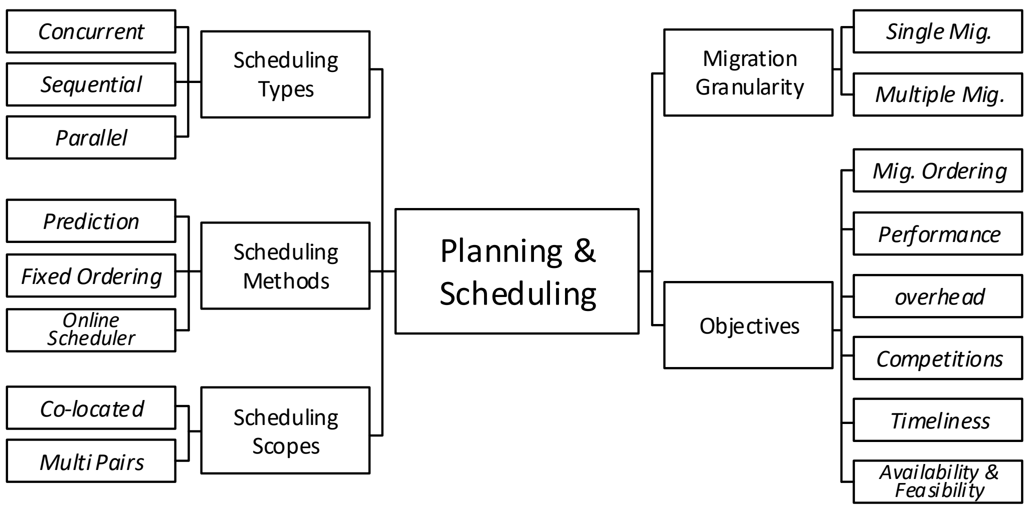


Figure 2.5: Categories of migration planning and scheduling algorithms

2.5 Migration Planning and Scheduling

In this section, we introduce the taxonomy for migration planning and scheduling, including *migration granularity*, *scheduling objectives*, *scopes*, *types*, and *methods*. Figure 2.5 illustrates the details of each category. Compared to real-time task scheduling, there is enough time to perform more complex migration scheduling, which further improves multiple migration performance and alleviates migration overheads. When it comes to multiple migrations, based on the objectives of live migration, the migration planning algorithm needs to calculate and optimize the sequence of each migration. In other words, the planning algorithm needs to consider the following issues, including availability, concurrency, correlation, and objective.

2.5.1 Migration Granularity

The migration granularity in the context of migration planning and scheduling can be categorized into *single migration* and *multiple migration*.

Single Migration: In single migration, only one instance is migrated at the same time. The research scope of single migration focusing on the performance and overhead of individual migration, including migration mechanisms, optimization techniques in both computing and networking aspects. The key metrics of single migration performance are migration time and downtime. The overheads of single migration include network interfaces, CPU, and memory stress, migration process overheads in the source and destination hosts (i.g. dirty memory tracing overheads), service-level parameters (i.g. response time), and available bandwidth for the migration service and other services in the data center.

Multiple Migrations: In multiple migration, multiple instances are considered to be migrated simultaneously. Multiple migration can be divided into various aspects based on the instance and service locations and connectivity, including co-located instances (gang migration) and cluster migrations (same source and destination pair), and related instances (connected services and applications, VNFs in SFC, entire virtual network, VMs in the same virtual data center). The overheads and performance of multiple migrations need to be evaluated and modeled in the migration generation, planning, and scheduling phases.

The overheads of multiple migrations can be categorized into service-level and system-level overheads. The service-level overheads include multiple migration influences on the migrating service, subsequent migrations performance, and other services in the data centers, and the system-level overheads include the multiple migration influence on the entire system, such as availability of networking and computing resources. On the other hand, the performance of multiple migration can be divided into global and individual performance. Total migration time, downtime, and transferred data size are the key metrics for the global migration performance of multiple migration. Performance metrics, such as average migration time, average downtime, and deadline violation, are used for the individual performance in multiple migration.

2.5.2 Objectives and Scopes

In this section, we summarize the objectives of migration planning algorithms and scheduling strategies. It can be categorized into *migration ordering*, *migration competitions*, *overhead and cost*, *migration performance*, *migration timeliness*, and *migration availability and feasibility*. We also categorized the scheduling scope in two aspects: co-located scheduling and multiple migration scheduling with multiple source-destination pairs (multi pairs).

Migration Ordering: The works of migration ordering problems focus on the feasibility of multiple migrations [28, 104] and migration ordering of co-located instances [101, 102] in the one-by-one scheduling solutions. In other words, given a group of migration requests, the feasibility problem is solving the problem that whether given migrations can be scheduled and the scheduling ordering of these requests due to the resource deadlock. The performance problem in the migration ordering context is finding an optimized order to migrate the co-located instance in order to minimize the overheads of multiple migrations and maximize the performance of multiple migration in the one-by-one scheduling manner.

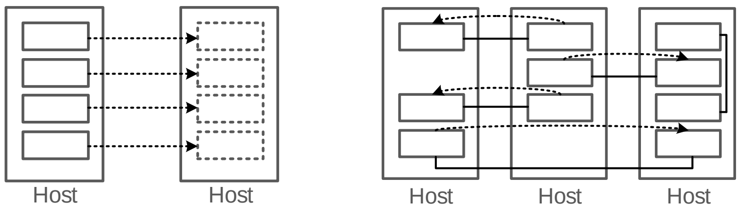
Migration Competitions: Resource competition problems include the competitions among migrations and between migrations and other services during the simultaneous migration scheduling [29, 30, 103]. Since migrations and services are sharing computing and networking resources, it is essential to determine the start sequence of migrations in both sequential and concurrent scheduling manner to minimize the resource throttling and maximize the resource utilization with respect to both QoS and migration performance. The resource dependencies and competitions among migrations and services need to be considered in both migration generation phase and the planning and scheduling phase in order to improve the performance of multiple migration and minimizing the overheads of multiple migration.

Overhead and Cost: It is critical to minimize migration overheads and costs to alleviate the QoS degradations and guarantee the SLA during live migration scheduling. The computing overheads on CPU, memory, and I/O interfaces affect the co-located instances negatively. The migrations also share the same network links with other services. It may lead to QoS degradations due to the lower bandwidth allocation. As a result, a longer migration process leads to larger computing (CPU and memory) and networking overheads (network interfaces and available bandwidth). Network management policies and routing algorithms are adopted to dynamically allocate the bandwidth and network routing to migrations. Furthermore, the migration downtime also need to be managed to avoid the unacceptable application response time and SLA violations.

Migration Performance: The performance of multiple migration scheduling is one of the major objectives of migration planning and scheduling algorithms. The total migration time is highly relative to the final management performance. In other words, a smaller migration time leads to a quicker optimization convergence. Furthermore, in green data center solutions, the energy consumptions induced by live migration need to be modeled properly.

Timeliness: The timeliness of the migration schedule is also critical to resource management performance [103, 134]. For the migration with various priorities and urgencies, inefficient migration planning or scheduling algorithms may result in migration deadline violations, which leads to QoS degradations and SLA violations. For example, some VNF needs to be migrated as soon as possible to maintain low end-to-end latency requirements. On the other hand, some migration requests of web services with high latency tolerance and robustness are configured with a much larger scheduling window.

Availability and Feasibility: Migration availability and feasibility problems are also considered in the planning and scheduling algorithms [28, 104]. There should be reserved resources in the destination hosts and sites in order to host the new instance. In the context of multiple migration, resource deadlock and network inconsistency may also affect the migration success ratio. Intermediate migration host and efficient migration ordering algorithms are proposed to solve the migration feasibility issue.



(a) Co-located (b) Multi pairs

Figure 2.6: Scheduling Scopes of multiple migration scheduling: co-located migrations and multiple migration with various source and destination pairs

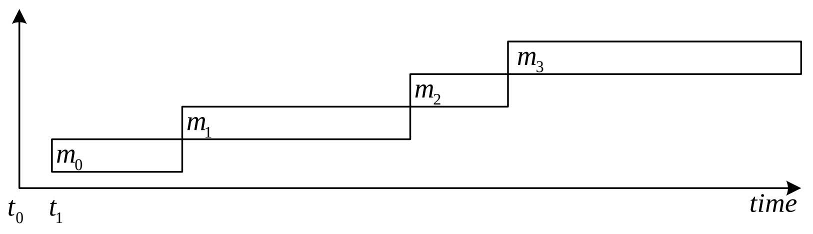
Scheduling Scopes

The multiple migration scheduling and planning algorithms can be divided into co-located and migrations with multiple source-destination pairs (Fig. 2.6). In co-located instance migrations, such as gang migration, the solution only focus on one source and destination pair. On the other hand, multiple instances migration involves various source and destination hosts or sites. For the migrations across dedicated network links, networking contentions between migration and services are omitted. In the data center network without dedicated migration networks, some works consider the virtual network connectivity among applications during the migration schedule.

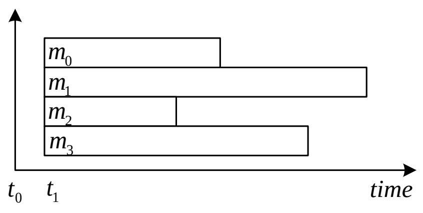
2.5.3 Scheduling Types

The migration scheduling types of multiple migration can be categorized as *sequential* multiple migration, *parallel* multiple migration, and *concurrent* multiple migration.

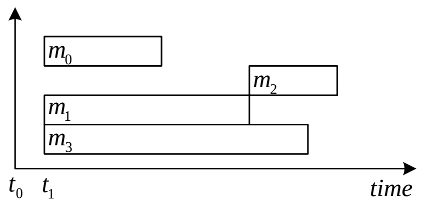
Sequential: In the sequential multiple migration solution depicted in Fig. 2.7(a), migration requests are scheduled in the one-by-one manner. In most scenarios for live VM migration, the network bandwidth is not sufficient for live migration to share network links with other migrations. Therefore, sequential migration scheduling for networking resource-dependent migrations is the optimal solution. It is used to the migration scheduling in co-located multiple migration scheduling and planning.



(a) Sequential



(b) Parallel



(c) Concurrent

Figure 2.7: Scheduling types of multiple migration scheduling

Parallel: In the parallel or synchronous multiple migration solution (Fig. 2.7(b)), migration requests start simultaneously. For the migrations with network link sharing, parallel migration scheduling is preferred only when the networking overheads induced by dirty pages and the memory footprint of migrating instances are smaller than the migration computing overheads. In other words, in most scenarios, the parallel migration may result in longer total and individual migration time and downtime. On the other hand, for the migrations without network links sharing across dedicated migration networks, the minimum total and individual migration time can be achieved. Some solutions are mixing the sequential and parallel migration solution that groups of migrations are started at the same time as in parallel solution and each group of migrations is scheduled sequentially.

Concurrent: Furthermore, concurrent or asynchronous migration planning and scheduling algorithms are proposed to schedule multiple migration requests efficiently by calculating and scheduling the start time of each migration independently (Fig. 2.7(c)). Migrations contend network resources with other migrations and services. Furthermore, the service traffic relocation induced by migration completion may affect subsequent migrations. As a result, migrations without resource contentions can be scheduled in the parallel manner and migrations with resource dependency need to be scheduled sequentially. Therefore, it is essential to manage the dependency and concurrency among migrations during multiple migration scheduling to optimize the multiple migration performance.

2.5.4 Scheduling Methods

After the phase of multiple migration planning, the calculated migration plan is going to be scheduled in the migration scheduling phase. We categorize scheduling methods into three types, namely *prediction*, *fixed ordering*, and *online scheduler*. In order to schedule single and multiple migrations, the migration manager and scheduler need to know when a migration has finished or needs to be started.

Prediction: Based on the prediction model and current available computing and networking resources, the start time of each migration is configured during the planning phase. Furthermore, the bandwidth allocated to each migration is also configured based on the available bandwidth at the time of migration planning.

Fixed ordering: Multiple migration tasks are scheduled based on the order calculated by migration planning algorithms. In other words, one migration is started as soon as possible when one or several specific migrations are finished. The fixed ordering model of multiple migration requests is similar to the dependent tasks, which can be modeled as a Directed Acyclic Graph (DAG).

Online Scheduler: The states of the networking environment, such as network topology, available links and interfaces, available bandwidth, and network delay, are constantly changed. The computing resources, such as memory, vCPU, storage, destination hosts, and sites, may also become unavailable during the multiple migration scheduling. Integrating with dynamic computing and networking management, online migration scheduler can dynamically start the migrations based on current states of computing and networking. The resource may not be available based on the predicted scheduling time or orders. The online scheduler can dynamically adjust the migration plan to efficiently schedule multiple migrations Furthermore, by balancing the allocated bandwidth to migration and application traffic, the online scheduler can guarantee both QoS and migration performance.

2.6 Current Research on Migration Management

In this section, we review the current research on migration management by focusing on the migration generation during dynamic resource management and the migration planning and scheduling algorithms. Each work can involve several categories of live migration management. Therefore, we choose the major category to organize and present the reviews in a more straight forwarding way.

2.6.1 Migration Generation

In this section, we review and summarize representative works on migration generation based on the resource management objectives, such as load balancing, energy-saving, network communication optimization, and migration-aware solutions.

Many works have studied the load balancing problem in dynamic resource management through live migrations [27, 135–138]. Singh et al. [135] propose a multi-layer virtualization system HARMONY. It uses VMs and data migration to mitigate hotspots on servers, network devices, and storage nodes. The load balancing algorithm is a variant of multi-dimensional knapsack problem based on the evenness indicator, i.e. Extended Vector Product (EVP). It considers the single live migration impact on application performance based on CPU congestion and network overheads. Verma et al. [136] estimate the migration cost based on the deduction of application throughput. The proposed algorithm (pMapper) selects the smallest memory size VMs from the over-utilized hosts and assigns them to under-utilized hosts in the First Fit Decreasing (FFD) order. Wood et al. [137] propose a load balancing algorithm Sandpiper that selects the smallest memory

Table 2.4: Characteristics of Migration Generation in Dynamic Resource Management

Reference	Mig. Com.			Mig. Net.				Mig. Obj.			Res. Obj.			
	mem	dirty rate	cpu	bw	route	hop	layer	perf.	cost	num	comp.	net.	energy	QoS
Singh et al. [135]			✓	✓					✓		✓			✓
Verma et al. [136]	✓			✓					✓		✓			✓
Wood et al. [137]	✓								✓		✓			
Mann et al. [27]	✓	✓		✓					✓			✓		
Forsman et al. [138]	✓	✓		✓					✓		✓			
Xiao et al. [139]	✓								✓		✓		✓	✓
Beloglazov et al. [140]										✓	✓		✓	
Beloglazov et al. [141]	✓			✓					✓		✓		✓	
Witanto et al. [142]									✓		✓			
Li et al. [143]														
Piao et al. [144]												✓		✓
Tso et al. [110]	✓						✓		✓			✓		
Cao et al. [145]	✓					✓			✓			✓	✓	
Cui et al. [106, 107]	✓	✓		✓					✓			✓		
Xu et al. [90]	✓	✓	✓	✓				✓	✓	✓				✓
Cui et al. [112]	✓				✓	✓			✓			✓		
Flores et al. [108]	✓			✓	✓	✓		✓	✓			✓		✓

Mig. Com.: Migration Computing parameters - mem (memory size), cpu (CPU load and utilization); **Mig. Net.**: Migration Networking parameters - bw (bandwidth), route (migration traffic routing), hop (migration distance), layer (involved network layers); **Mig. Obj.**: Migration Objectives - perf. (migration performance), cost (migration cost and overheads), num (migration number); **Res. Obj.**: Resource management Objectives - comp. (computing), net. (networking), energy (device and cooling energy cost), QoS (response time and SLA violations).

size VM from one of the most overloaded hosts to minimize the migration overheads. Mann et al. [27] (Remedy) focus on the VM and destination selection for the load balance of application network flows by considering the single migration cost model based on the dirty page rate, memory size, and available bandwidth. Forsman et al. [138] propose a pre-copy live VM migration selection algorithm for automated load balancing strategy based on the migration cost [88], expected load distribution, and utilization after migration.

Dynamic VM consolidation algorithm is one of the techniques to reduce energy consumption through VM migrations. Khan et al. [146] review the related works on dynamic virtual machine consolidation algorithms for energy-efficient cloud resource management. Xiao et al. [139] investigate dynamic resource allocation through live migration. The proposed algorithm avoids the over-subscription while satisfying resource needs of all VMs based on exponentially weighted moving average (EWMA) to predict the future loads. It also minimizes the energy consumption of physical machines by hot spot mitigation. The cost of live migration is modeled as the memory size. The authors argue that live migration with self-ballooning causes no noticeable performance degradation.

Beloglazov et al. [140] propose Minimization of Migrations (MM) algorithm to select the minimum migration number needed to fulfill the objective of overloaded host mitigation and underloaded host consolidation. Best Fit Decreasing algorithms are used. VMs are sorted in decreasing order based on the CPU utilization and network consumption of expected allocation. Furthermore, Beloglazov et al. [141] (LR-MMT) focus on energy saving with local regression (LR) based on history utilization to avoid over-subscription. The proposed algorithm chooses the VM with the least memory size from the over-utilized host and the migration destination with the largest energy saving. The minimum migration time is modeled based on VM memory size and available bandwidth.

Witanto et al. [142] propose a machine-learning selector with neural network. High-level consolidation through VM migrations may reduce energy consumption, but will result in higher SLA violations. Therefore, the proposed algorithm dynamically chooses existing consolidation algorithms and strategies to manage the trade-off between energy and SLA violation (due to VM migration downtime) based on the priority availability in the system.

For the works on networking provisioning through live migration, Piao et al. [144] propose a virtual machine placement and migration approach to minimize the data transfer cost. The proposed migration approach is triggered when the execution time crosses the threshold that specified in the SLA. Tso et al. [110] propose a distributed network-aware live VM migration scheme to dynamically reallocate VMs to minimize the overall communication footprint of active traffic flows in multi-tier data centers. The authors investigate the performance degradation issue during migration, which is caused by the congestion at the core layers of the network where bandwidth is heavily oversubscribed. The proposed distributed solution generates migration requests iteratively based on the local VM information (one-by-one migration scheduling) to localize VM traffic to the low-tier network links. The migrations are performed only when the benefits of traffic reallocation outweigh the migration cost based on VM memory and downtime. However, the work lacks a realistic comparison between migration overheads and migration benefits for communication management. With the help of SDN, the assumption of requirements that centralized approaches obtaining knowledge of global traffic dynamics is prohibitively expensive may be untenable.

Cao et al. [145] investigate the VM consolidation problem considering the network optimization and migration cost. Migration overheads are modeled as the host power consumption (the product of power increase and migration time) and traffic cost (the produce of VM memory size and distance in hop number between source and destination). Cziva et al. [115] propose an SDN-based solution to minimize network communication cost through live VM migration in a multi-tire data center network. The authors model the communication cost as the product of the average traffic load per time unit and weight of layer link. They argue that the migration cost can be introduced into the solution by modeling the migration network traffic cost. Similarly, Cui et al. [106, 107] study the joint dynamic MiddleBox/VNF network chaining policy reconfiguration and VM migration problem in the SDN-enabled environment to find the optimal placement minimizing the total communication cost. The authors consider both VM migration time and the traffic data induced by the MiddleBox/VNF migration. However, the paper lacks the information for how the communication cost is modeled. The migration cost is considered by comparing the network rate with data per time unit with the total transferred data size of live migration. Furthermore, only modeling the migration cost as transferred data size without considering the networking bandwidth and routing may result in poor migration scheduling performance and QoS degradations.

With the input of candidate VMs and destinations provided by existing resource management algorithms, Xu et al. [90] propose a migration selector (iAware) to minimize the single migration cost in terms of single migration execution time and host co-location interference. It considers dirty page rate, memory size, and available bandwidth for the single migration time. They argue that co-location interference from a single live migration on other VMs in the host in terms of performance degradation is linear to the number of VMs hosted by a physical machine in Xen. However, it only considers one-by-one migration scheduling.

Cui et al. [112] propose a new paradigm for VM migration by dynamically constructing adaptive network topologies based on the VM demands to reduce VM migration costs and increase the communication throughput among VMs. The migration cost is modeled as the product of the number of network hops and VM's memory size. The authors argue that the general VM migration cost value can be replaced by specific cost metrics, such as migration time and downtime based on allocated bandwidth and measured dirty page rate of the VMs.

Li et al. [143] propose a greedy-based VM scheduling algorithm to minimize the total energy consumption, including the cooling and server power consumption models. For the selection of migration requests, the proposed algorithm selects all physical machines with the temperature above the threshold proportion as source hosts. Then, it selects the VM with the minimum utilization from all selected source hosts and selects the server with the minimum power increase as the migration destination for each VM. The authors focus on energy consumption through live migrations without considering the migration cost. However, the proposed algorithm outputs multiple migration requests without the actual migration planning and scheduling algorithms.

Flores et al. [108] propose a placement solution that integrates migration selection with data centers policies to minimize the communication cost by considering network topology, VM virtual connections, communication cost, and network hops for live migration cost. Considering the routing cost, the proposed migration scheduling algorithm migrates VMs in order to minimize the total cost of migration and VM communication. However, the cost model of migration is still linear without considering the concurrent scheduling performance of multiple migrations.

We summarize the characteristics of migration generation in Table 2.4 based on four

Table 2.5: Comparisons of Solutions on Migration Planning and Scheduling

Reference	Schedule			Net			Scope		Heter	Mig Obj				
	Seq.	Parl.	Cnrc.	Mig. net.	Net. mgmt.	Connect.	Co-loc.	Multi src-dst		QoS aware	Mig. order	Mig. Feas.	Mig. perf.	Mig. cost
Deshpande et al. [147]		✓					✓							✓
Deshpande et al. [148, 149]		✓					✓							✓
Rybina et al. [101]	✓						✓				✓		✓	✓
Fernando et al. [102]	✓				✓		✓				✓		✓	✓
Ghorbani et al. [28]	✓				✓			✓				✓		
Sun et al. [150]	✓	✓					✓		✓				✓	
Deshpande et al. [26]		✓					✓		✓	✓			✓	✓
Zheng et al. [151]	✓	✓			✓	✓				✓			✓	✓
Liu et al. [152]		✓			✓	✓				✓			✓	✓
Lu et al. [153]	✓			✓	✓	✓				✓			✓	✓
Lu et al. [154]	✓	✓			✓	✓				✓			✓	✓
Kang et al. [155]	✓	✓			✓								✓	✓
Ye et al. [156]	✓	✓											✓	✓
Sarker et al. [104]			✓		✓	✓		✓				✓	✓	✓
Bari et al. [29]			✓			✓		✓		✓			✓	✓
Wang et al. [30]			✓	✓	✓			✓					✓	

Schedule Type: Seq. (Sequential), Parl. (Parallel), Cnrc. (Concurrent), **Net**: Networking related - Mig. net. (Dedicated migration network), Net. mgmt. (Networking management), Connect. (instance Connectivity), **Scope**: Co-loc. (Co-located instances), **Heter**: Heterogeneous solutions (mixing various migration types), and **Mig Obj**: migration management objectives - Mig. order (migration ordering), Mig. feas. (migration feasibility), Mig. perf (migration performance), Mig. cost (Migration cost and overheads).

categories: migration computing parameters, migration network parameters, objectives of migration optimization in the solution, and objectives of resource management.

2.6.2 Migration Planning and Scheduling

In this section, we review the state-of-the-art works on migration planning and scheduling algorithms. Studies of migration planning and scheduling focus on various aspects, such as migration feasibility, migration success or failure ratio, migration effects, scheduling deadline, application QoS, scheduling orders, migration scheduler, migration routing, and migration performance in total migration time, average migration time, downtime, and transferred data size. As shown in Table 2.5, we summarize the characteristics of reviewed solutions of migration planning and scheduling in various categories: scheduling type, migration networking awareness, migration scopes, heterogeneous migration types, and migration scheduling objectives.

Co-located Multiple Migrations

Deshpande et al. [147] consider the migration of multiple co-located VMs in the same host as the live gang migration problem. They optimize the performance of multiple live migrations of co-located VMs based on the memory deduplication and delta compression algorithm to eliminate the duplicated memory copying from the source to the destination host. Since co-located VMs share a large amount of identical memory, only identical memory pages need to be transferred during the iterative memory copying in pre-copy migration. They also employ delta compression between copied dirty pages to reduce the migration network traffic. Moreover, Deshpande et al. [148, 149] further investigate the same problem using cluster-wide global deduplication by improving the technique from the co-located VMs in the same host to the ones in the same server rack.

Rybina et al. [101] investigate the co-located resource contentions on the same physical host. The authors evaluate all possible migration orders in a sequential manner in terms of total migration time. They find that it is better to migrate the memory-intensive VM in order to alleviate the resource contentions. Fernando et al. [102] proposed a solution for the ordering of multiple VMs in the same physical host (gang migration) to reduce the resource contentions between the live migration process and the migrating VMs. The objectives of the solution are minimizing the migration performance impact on applications and the total migration time. The migration scheduler decides the order of migrations based on different workload characteristics (CPU, Memory, network-intensive) and resource usage to minimize the total migration time and downtime. Furthermore, an SDN-enabled network bandwidth reservation strategy is proposed that reserves bandwidth on the source and destination hosts based on the migration urgency. When the available bandwidth in the destination can not satisfy the migration requirement, network middle-boxes [157] are used as network intermediaries to temporarily buffer the dirty memory.

Migration Feasibility

For the migration scheduling feasibility, Ghorbani et al. [28] proposed a heuristic one-by-one migration sequence planning for a group of migration requests to solve the problem of transient loop and network availability during the migration. The authors consider the environments that the requirement of the virtual network must be satisfied with bandwidth over-subscription. With the bandwidth requirement of virtual links between instances, random migration sequence will lead to the failure of migrations of most of the instances. With the flow install time of current SDN controller implementation, the orders of network updates due to migration within the forwarding table may cause the transient loop issue. The authors did not consider the concurrent VM migration scheduling with various network routings.

Heterogeneous and Homogeneous Solutions

Multiple migration can be divided into heterogeneous and homogeneous solutions. For the heterogeneous solution, different types of live migration (pre-copy and post-copy migrations) are used simultaneously. In an environment that all migrations are sharing the same network, Sun et al. [150] considers the sequential and parallel migration performance of multiple VMs. The authors proposed an improved one-by-one migration scheduling based on the assumption that the downtime of live migration is large enough. When the first VM is stopped during the downtime of pre-copy migration, the algorithm stops and performs the post-copy on the remaining connected VMs within the same service. Furthermore, the authors proposed an m-mixed migration algorithm for parallel multiple migration started at the same time. The algorithm chooses the first m VMs to perform pre-copy migration, while the rest are performed with post-copy migration. They validate the efficiency of the proposed solutions, such as the blocking ratio and average waiting time of each migration request, based on the proposed M/M/C/C queuing models.

The networking contentions between migrations and application traffic can increase the migration time and degrades the application QoS. To reduce the network contentions between migrations and applications, solutions for co-located multiple VM migrations with both pre-copy and post-copy migrations are adopted. The intuition of these solutions is utilizing both inbound and outbound network interfaces. When the co-located instances are migrated using pre-copy or post-copy, the traffic between co-located instances contends with the migration traffic. Therefore, the migrating instance with post-copy in the destination host communicates to another migration instance with pre-copy in the source host, which alleviates the network contentions between the application traffic and pre-copy migration traffic. Deshpande et al. [26] proposed a traffic-sensitive live migration technique by utilizing pre-copy or post-copy migration based on the application traffic direction to reduce the migration network contention.

Instance Correlation and Connectivity

The instances in the data center are often connected with each other through network virtual links under various application architecture, such as multi-tier web applications and Virtual Network Functions (VNFs) in Service Function Chaining (SFC). Several studies focus on optimizing the multiple migration of multi-tier applications and network-related VMs. Research [158, 159] evaluates the impact of live migration on multi-tier web applications, such as response time. Zheng et al. [151] investigate the multi-tier application migration problem and propose a communication-impact-driven coordinated approach for a sequential and parallel scheduling solution. Liu et al. [152] work on the correlated VM migration problem and propose an adaptive network bandwidth allocation algorithm to minimize migration cost in terms of migration time, downtime, and network traffic. In the context of multi-tier applications, the authors proposed a synchronization technique to coordinate correlated VM migrations entering the stop-and-copy phases at the same time, which reduces the network traffic between correlated applications across the inter-data center network. Lu et al. [153] also focus on the correlated VMs migration scheduling of multi-tier web applications with dedicated network for migrations. The authors investigate the sequential and parallel migration strategies for multiple migrations which start at the same time. The proposed heuristic algorithm groups the related VMs based on the monitoring of communication traffic and sorts the sequential migration order based on migration time and resource utilization. Expending the concept from multi-tier application connectivity, Lu et al. [154] proposed a separation strategy by partitioning a large group of VMs into traffic-related subgroups for inter-cloud live migration. Partitioned by a mini-cut algorithm, subgroups of pre-copy migrations are scheduled sequentially and VMs in the same subgroup are scheduled in parallel to minimize the network traffic between applications across inter-data center networks.

Parallel and Concurrent Scheduling

Studies [150, 155] investigate solutions for mixed sequential and parallel migrations. Kang et al. [155] proposed a feedback-based algorithm to optimize the performance of multiple migration in both total and single migration time considering the sequential and parallel migration cost, and available network resources. It adaptively changes the migration number based on the TCP congestion control algorithm. Ye et al. [156] investigate the multiple migration performance in sequential and parallel migrations. The authors conclude that sequential migration is the optimal solution when the available network bandwidth is insufficient.

For the multiple migration planning and scheduling algorithms of concurrent migrations, the migration planning algorithm and migration scheduler determine when and how one migration of the multiple migration requests should be performed within the time interval of the total migration time. Sarker et al. [104] proposed a naive heuristic algorithm of multiple migration to minimize the migration time and downtime. The proposed scheduling algorithm starts all available migrations with minimum migration cost until there is no migration request left. The deadlock problem is solved by temporarily migrating VM to an intermediate host. Bari et al. [29] proposed a grouping-based multiple migration planning algorithm in an intra-data center environment where migrations and applications share the network links. The authors model the multiple VM migration planning based on a discrete-time model as a MIP problem. The available bandwidth may be changed after each migration due to the reconfiguration of virtual network links. The subsequent migrations are affected by the previous migration result. Considering the influence of each migration group during the scheduling, the proposed algorithm sets the group start time based on the prediction model. Migration in each group can be scheduled simultaneously if the resources occupied by the previous group are released. However, the authors neglect the influence of individual migration in their solution, which can lead to performance degradation of the total migration time. Without considering the connectivity among applications in a WAN environment, Wang et al. [30] simplify the problem by maximizing the network transmission rate but directly minimizing the total migration time. With the help of SDN, the authors introduce the multipath transmission for multiple migrations. A fully polynomial-time approximation FPTAS algorithm is proposed to determine the start time of each migration.

2.6.3 Summary and Comparison

All studies covered in the survey are summarized in Table 2.4 and Table 2.5 based on our taxonomy in Figure 2.3 and 2.5, respectively. For migration generation in dynamic resource management algorithms, many studies optimize at least two of the resource objectives regarding the computing resources, networking resources, energy consumption, and application QoS. Most researchers do not consider the migration scheduling performance in the proposal which has no tick in the table. These studies consider and model migration computing costs linearly or individually based on memory size, available bandwidth, or the total migration number. A number of works consider the actual single pre-copy live migration model based on memory size, dirty page rate, and available bandwidth. For the migration network cost modeling and management, most of the works only consider the available bandwidth or the transferred data volume, while few works consider the network routing or the migration distance on hops and layers. Integrating with existing resource management algorithms, few works focusing on migration interferences and migration scheduling performance. However, the linear models of these works are only suitable for sequential migration scheduling optimization.

For migration planning and scheduling algorithms, researchers are actively studying the migration scheduling performance and migration cost, some considering the application QoS during migrations, while others focusing on migration availability and scheduling feasibility. For heterogeneous and homogeneous solutions, most works are focusing on the homogeneous solution with one migration type such as pre-copy migration, while others consider both pre-copy and post-copy migrations. However, there is no concurrent planning and scheduling algorithm considering scenarios where multiple migration with mixed types. The scheduling scope is varied based on the proposed method, early studies focusing on co-located migrations with one source and destination pair, while recent proposals considering the multiple migration scheduling with various source and destination pairs.

For networking management and efficiency, some researchers consider the migration scheduling in dedicated migration networks, or do not consider the network overheads on other services and applications. Others consider the connectivity of correlation instances with virtual network communication during migrations. Without considering the network over-subscription, the bandwidth requirements of virtual links between instances are guaranteed during migrations. To improve migration performance and reduce migration traffic impacts, networking management algorithms are adopted to optimize the network routing for migration traffic and application traffic.

The scheduling types are varied based on the proposed solution and scheduling scopes, most of the works consider sequential migration scheduling, while others focusing on the parallel migrations or applying both types. Recently, several researchers focus on concurrent migration planning and scheduling for efficient, optimal, and generic solutions. Few works focus on the timeliness of migration with optimizations and prediction models, such as migration finishes before a given deadline without general concurrent migration scheduling. Furthermore, energy consumption is a critical objective in dynamic resource management of data centers. Therefore, migration energy cost modeling also needs to be investigated. Mathematical models, simulation platforms, and empirical methods are used for evaluation. Most of the studies only use one of the evaluation methods, while several studies use more than one method. We introduce the details of available evaluation methods and technologies in the following section.

2.7 Evaluation Methods and Technologies

To accelerate the research and development of dynamic resource management in cloud computing, data traces, software and tools are required for testing the migration performance and overheads in the edge and cloud data centers. Furthermore, evaluation platforms are needed to test and evaluate the networking management algorithms based on OpenFlow protocol and SDN architecture. Therefore, the testbed needs the capability to measure the energy consumption, downtime impacts, response time, and processing time to properly evaluate the proposed resource management policies and migration scheduling algorithms. In this section, we introduce the related emulators, simulation tools, and empirical solutions.

2.7.1 Simulation and Emulation

Simulators are essential evaluation platforms that accelerate the innovation and implementation of proposed solutions and algorithms by providing controllable and reproducible experiment environments with ease of configuration and modification. Emulation provides an environment for the end-system to mimic the entire system, such as a network for physical hosts and application programs, and the end-system operates as if it were in a real environment. The simulation in cloud computing can be divided into computing and networking simulation.

For the networking emulation, Mininet is an open-source network emulator for the rapid prototyping of the Software-Defined Networking, which emulates the entire network of hosts, switches, and links. As it utilizes network virtualization provided by the Linux kernel, Mininet can produce more accurate results in network delays and congestion at the Operating System level. It also natively supports the innovations and implements of SDN controllers and OpenFlow protocol.

Abstracting network traffic descriptions, discrete event network simulators, such as NS-3 , OMNet++ , and NetSim , provide the extensible, modular, and component-based

Mininet. `https://github.com/mininet/mininet`
NS-3. `https://www.nsnam.org/`
OMNet++. `https://omnetpp.org/`
NetSim. `https://www.tetcos.com/`

simulation library and framework to support network simulation. Researchers could build the SDN module extension to support the OpenFlow and SDN controller simulation. For example, Chaves et al. [160] proposed OFSWITCH13 as an SDN module to support OpenFlow in NS-3. Klein et al. [161]implement the OpenFlow model by utilizing the INET framework of OMNet++. NetSim does not support SDN controller and OpenFlow, but one can utilize the discrete-event mechanism to add a new OpenFlow switch module to support forwarding tables and OpenFlow events and use the real-time tunneling interaction supported by NetSim to create communication between the real SDN controller and evaluate the dynamic management algorithms. Some works [162] implement the migration module to generate the network traffic to simulate and evaluate the network overheads of live migration. The migration network traffic generation could be improved by profiling the live migration with various resources and applications workloads. However, it can not evaluate the computing cost and overheads induced by the live migration processes.

For the simulation platform for cloud computing, CloudSim is a popular discrete event-driven cloud simulator implemented in Java. Various data center management algorithms and solutions can be evaluated in CloudSim, including brokering policy, VM instance placement, and dynamic resource reallocation. It also supports workload processing in VMs. iFogSim [163] based on CloudSim discrete event-based architecture extending the cloud data center components to fog computing components to simulate the corresponding events in the IoT context. However, CloudSim and iFogSim do not support network events in detail. The instance reallocation through live migration is only modeled as a delayed event according to the memory size.

Based on iFogSim, Myifogsim [164] and its extension MobFogSim [165] are proposed for the simulation of user mobility on the map with radio base stations (access points). It supports the evaluation of migration policies for migration request generation. The proposed extension only simulates cold migration and post-copy migration without the support of network topology, network flow, and bandwidth emulation. However, these proposed simulation platforms and extensions lack the capability of network simulation. There is no widely-used pre-copy migration simulation model to minimize the downtime (downtime too high for lazy post copy), and no multiply migration simulation, and lacking capabilities for QoS simulation, such as response time of services during the migrations.

Integrating with CloudSim, CloudSimSDN [71] is developed and implemented to support packet-level network simulation in order to evaluate the network transmission and links delays in the data center network architecture including host, switches, and links. Based on CloudSimSDN, CloudSimSDN-NFV [166] provides the NFV supports including automatic scaling and load balancing in SFC, and expends the network architecture from intra-data center to the inter-cloud and edge computing network. By leveraging simulation capabilities for both computing and networking, we can extend the corresponding components based on the CloudSimSDN to simulate each phase of pre-copy live migration.

2.7.2 Empirical Platforms and Prototypes

The dynamic resource management and live migration management algorithms are located in the orchestration layer. However, current public and research cloud platforms only support user management at the software level. There is a lack of experimental infrastructure for cloud management through live migration since the live migration management needs to be performed at the administrator level.

OpenStackEmu [167] combines the OpenStack and SDN controller with network emulation. It enables the connection between the large-scale network and the OpenStack infrastructure hosting the actual VMs. OpenStackEmu also provides network traffic generation in the framework. SDCon [168] is a orchestration framework integrating OpenStack, OpenDayLight (ODL) SDN controller, OpenVSwitch and sFlow. By enabling the NetVirt feature in ODL and setting ODL as the default SDN controller in OpenStack Neutron, an OpenStack-based private data center can be used as a testbed for evaluating various dynamic resource management policies and VM migration management algorithms.

As containers can be hosted in VMs, live container migration can be performed in the public cloud platform with virtual networking support across VMs. However, it lacks the ability to manage the networking resources as the underlying network and

VMs' location in the hosts can not be guaranteed. The cross-layer operations can bring uncertainty for the migration overhead evaluation. CloudHopper [169] is a proof-of-concept live migration service for containers to hop around between Amazon Web Services, Google Cloud Platform, and Microsoft Azure. Live container migration in CloudHopper is automated. It also supports pre-copy optimization, connection holding, traffic redirection, and multiple interdependent container migration.

2.7.3 Cloud Trace Data

Application workloads traces and cloud cluster traces support the realistic, repeatable and comparable evaluation results for various solutions and algorithms. There are several popular traces used by the evaluation of dynamic resource management in data centers, including PlanetLab, Wikipedia , Yahoo!, , Google, Alibaba Cluster Trace .

PlanetLab data provides CPU utilization traces from PlanetLab VMs. Wikipedia data provides the workloads for multi-tier web applications which is a typical application architecture in the data center. Yahoo! cloud serving benchmark (YCSB) is a set of workloads that defines a basic benchmark for cloud data centers, including update heavy, read mostly, read-only, real least, short ranges, and real-modify-write workloads. Google Borg cluster workload trace provides traces of workloads running on Google compute cells managed by the cluster management software (Borg). Alibaba Cluster Trace Program provides trace data of a whole cluster running both online services and batch jobs. The trace data includes both parts of machines data and the workload of the whole cluster.

Planet Lab Trace Data. `https://github.com/beloglazov/planetlab-workload-traces`
Wikipedia Data. `https://wikitech.wikimedia.org/wiki/Analytics/Archive/Data/Pagecounts-raw`
Yahoo Benchmark. `https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads`
Google Cluster Trace. `https://github.com/google/cluster-data`
Alibaba Cluster Trace. `https://github.com/alibaba/clusterdata`

2.8 Summary

This chapter presents a taxonomy of live migration and migration management and the survey of the state-of-art works of migration management in edge and cloud computing. We categorize aspects of existing works in migration management, which include performance and cost model, migration generation in resource management policies, planning and scheduling algorithms, management lifecycle and orchestration, and evaluation methods. Each aspect in the taxonomy is explained in detail and corresponding papers are presented. We describe and review representative works of dynamic resource management focusing on the migration generation in migration computing parameters, networking parameters, and migration objectives. We also categorize and review the state-of-the-art on migration planning and scheduling in migration scheduling types, migration network awareness, scheduling scope, heterogeneous and homogeneous solutions, and scheduling objectives. Various objectives of multiple migration scheduling are explained, following the simulators, emulators, and empirical platforms.

Chapter 3

Performance Evaluation of Live VM Migration in SDN Clouds

In Software-Defined Networking (SDN) enabled cloud data centers, live VM migration is a key technology to facilitate the resource management and fault tolerance. Despite many research focus on the network-aware live migration of VMs in cloud computing, some parameters that affect live migration performance are neglected to a large extent. Furthermore, while SDN provides more traffic routing flexibility, the latencies within the SDN directly affect the live migration performance. In this chapter, we pinpoint the parameters from both system and network aspects affecting the performance of live migration in the environment with OpenStack platform, such as the static adjustment algorithm of live migration, the performance comparison between the parallel and the sequential migration, and the impact of SDN dynamic flow scheduling update rate on TCP/IP protocol. From the QoS view, we evaluate the pattern of client and server response time during the pre-copy, hybrid post-copy, and auto-convergence based migration. In the end, we present the extended event-driven simulation platform for live migration in SDN-enabled cloud and edge computing environments.

3.1 Introduction

There are continuous efforts to improve live VM migration, such as improving the performance of live migration algorithm [68, 170], modeling for better prediction of the cost [84, 98], network-aware live migration to alleviate the influence of migration on SLA and application QoS [26, 90, 138], optimizing the multiple live VM migration plan-

This chapter is derived from:

- **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "Performance evaluation of live virtual machine migration in SDN-enabled cloud data centers", *Journal of Parallel and Distributed Computing (JPDC)*, Volume 131, Pages: 55-68, Elsevier, 2019.

ning [28–30, 150], and benchmarking the live migration effects on applications [158, 159]. Nonetheless, many parameters, such as downtime adjustment and non-network overheads, that affect the live migration time and downtime are neglected to a large extent. During a live VM migration, the downtime threshold for the last memory-copy iteration could be changed as time elapses. This will affect the memory-copy iteration rounds, which leads to different migration time and downtime. Computing overheads of live VM migration can also constitute a large portion of total migration time, which will affect the performance of multiple VM migrations.

On the other hand, some work focus on the live VM migration in Software-Defined Networking (SDN) scenarios [27, 28, 30]. By virtualizing the network resources, we could use SDN to dynamically allocate bandwidth to services and control the route of network flows. Due to the centralized controller, SDN can provide a global view of the network topology, states of switches, and statistics on the links (bandwidth and latency). Based on the information, orchestrator can calculate the 'best' path for each flow and call SDN controller Northbound APIs to push the forwarding rules to each switch in the path. However, the latencies of the flow entry installation on the switch and the communication between SDN controller and switches could impact the traffic engineering performance in the SDN-enabled cloud data centers. Thus, the scheduling update rate of choosing the 'best' path will affect the live migration traffic.

Moreover, although some work [158, 159] focus on the impacts of live migration on the cloud services, such as multi-tier web application, the worst-case response time pattern as well as the technologies, such as hybrid post-copy and auto-convergence, for a successful live migration need to be investigated further. Hybrid post-copy (H-PC) [65] is the strategy that combines pre-copy and post-copy migration. The post-copy mode will be activated after the certain pre-copy phase where most of the memory has been transferred. Based on the CPU throttling, Auto-convergence (AC) [171] will decrease the workload where the memory write speed is relative to the CPU executing speed.

We evaluate the live migration time, downtime, and total transferred data using OpenStack [172] as the cloud computing platform. OpenStack uses the pre-copy live migration with the default driver Libvirt (virtualization API) [173]. Our study is fundamentally useful to resource scheduling, such as energy-saving strategy, load balancing, and fault tolerant, driven by SLA. The **contributions** are fourfold, and are summarized as follows:

- Evaluation of the performance of block live migration in OpenStack with different configuration of static downtime adjustment algorithm. Experimental results can be used as reference to dynamically configure optimal migration time and downtime.
- Modeling and identification of the trade-off between sequential and parallel migration when the host evacuation happens in the same network path.
- Evaluation of the effect of flow scheduling update rate on the migration performance as well as TCP/IP protocol in SDN-enabled clouds. Experimental results can guide to optimize the update rate and select the best path of SDN forwarding scheduler in order to achieve better migration performance.
- Evaluation of the response time of a multi-tier web application under pre-copy, hybrid post-copy and auto-convergence based live migration. Specifically, experimental results demonstrate the worst-case response time and the situation when the pre-copy migration could not finish in a reasonable time.

The rest of the chapter is organized as follows. Section 2 introduces the related work and motivations. In Section 3, we present the system overview of SDN-enabled data centers and details of the live migration in OpenStack. The mathematical models of block live migration, sequential and parallel migrations are presented in Section 4. In Section 5, we describe the objectives, testbed specifications, metrics, and the experimental setup of the evaluated parameters. We quantitatively show how these parameters can dramatically affect the migration time and service performance. Finally, we conclude our work in Section 6.

3.2 Related Work

Clark *et al.* [8] firstly proposed the live VM migration comparing to the naive stop-and-copy method. During the iterative memory copy phase of live migration implemented in

Xen virtualization platform, rapid dirtying pages which updated extremely frequently, called Writable Working Set (WWS), was introduced. These pages will not be transmitted to the destination host in the iteration round in order to reduce the total migration time and transferred data. In addition, the authors elaborated the implementation issues and features with regard to the managed migration (migration daemons of Xen in host and destination hosts), self migration (implementation the mechanism within the OS), dynamic rate-limiting for each iteration round, rapid page dirtying, and paravirtualized optimizations (stunning rogue process, i.e. limit write faults of each process, and freeing page cache pages, i.e. reclaiming back cold buffer cache pages). Although pre-copy migration is widely used in various virtualization platforms, such as Xen, QEMU/KVM, VMWare, it is worth noting that migration algorithms and performance of different hypervisors are different in terms of dirty pages detection and transmission and stop-and-copy threshold [94]. For instance, the page skip (WWS) mechanism does not be implemented in KVM.

In order to alleviate the overheads caused by live VM migrations, the prediction model is required to estimate the live migration performance in advance. Akoush *et al.* [84] proposed a model to estimate the migration time and downtime of live VM migration based on the two main functions of migration, i.e. peek and clean. The peek function returns the dirty bitmap and the clean function returns the dirty pages and resets them to clean state. They used both average dirty page rate (AVG) and history based page dirty rate (HIST) in their prediction algorithms. The HIST model could capture the variability of live migration and help to decide the moment at which migration begins to minimize the migration cost. Moreover, Liu *et al.* [88] introduced the rapid page dirtying in its migration performance prediction model. In order to obtain a more accurate prediction model, the authors refined the previous prediction model of migration performance by estimating the size of WWS. Based on the observation, it is an approximated proportional size of the total dirty pages in each iterative memory copy round with regard to previous iteration time and dirty pages rate. The authors also proposed an energy consumption model of live migration based on the linear regression between total transferred data and measured energy consumption. The synthesized cost for migration decision is based on the estimated values of downtime, migration time, transferred data, and energy cost. Furthermore, based on prediction model of migration cost, different migration strategies for load balancing, fault toleration, and server consolidation are proposed [98]. The algorithms choose the proper migration candidates in order to minimize the total migration cost while satisfying the requirements of rescheduling algorithms. Contrary to their work we focus on the mechanism and performance of proposed parameters and corresponding models.

Prediction models of live migration which assume a static downtime threshold [84, 88, 98] or constant dirty page rate [30] cannot reflect the real migration time and downtime in OpenStack. The downtime threshold in OpenStack uses a static adjustment algorithm. It is increased monotonically with a certain time interval and steps during the migration in order to reduce the migration time. A misconfigured downtime configuration will lead to a poor performance of live migration, such as unstable downtime which results in the SLA violation and a long-time migration that degrades the network performance. Therefore, in order to dynamically set optimal configurations, we need to have a better understanding of the relationship between **downtime adjustment configurations** and migration performance in OpenStack.

Planning of the sequential and parallel migration in intra and inter-data centers to optimize the server evacuation time and minimize the influence of live migration has attracted interest recently [28–30]. However, they only focus on the network aspect of multiple live migration planning to decide the sequence of sequential and concurrent live migration in order to minimize the migration duration. As mentioned in [84], the total migration time includes pre-migration, pre-copy phase, stop-and-copy phase and post-migration overheads. The most proportion of migration time could be the operation overheads. Therefore, in order to have a better algorithm of the multiple VM evacuation planning, we need to pinpoint the impacts of non-network overheads on the **parallel and sequential migration** in the same path.

Moreover, Software-Defined Networking (SDN) [57] as a powerful feature in Cloud computing provides a centralized view of topology and bandwidth on every path. We could flexibly implement network scheduling algorithm and set bandwidth limit for live migration and other application traffics. In a highly dynamic network environment, the 'best' path decided by scheduling algorithm based on an update rate could change frequently. Therefore, not only the bandwidth but traffic pattern, SDN control plan [174] and flow table latency [175] could also affect the live migration performance. Understanding the **SDN latency** in the flow scheduling is very important for achieving better live migration performance.

With different application context, the impacts of live migration on application performance could change dramatically. For instance, the workload of a multi-tier web application with specific write and communication pattern [158, 159] is different with the workload in scientific computing applications. The response time should be soft real-time to satisfy the QoS of application. Therefore, network service suffers more from the disruption due to the downtime and the performance degradation due to the live VM migration. As there are few works on this topic, evaluating the live migration effects on the **response time** of different types of network-sensitive applications is desirable. However, current work did not consider the worst-case response time and the situation when the pre-copy migration could not finish in a reasonable time. Thus, we need to evaluate the response time distribution of the web application during the migration, and the impacts of strategies, hybrid post-copy, and auto-convergence, on application response time which perform a successful live migration.

3.3 System Overview

In SDN-enabled data centers, the computing resources are under control of cloud management platform, such as OpenStack, while the networking resources are managed by SDN controller. The management module (orchestrator) coordinates the SDN controller and the OpenStack services by using northbound RESTful APIs to perform VM migration planning in resource scheduling algorithm such as the SLA-aware energy-saving strategy as shown in Fig. 3.1. In OpenStack, Nova service runs on top of Linux servers as daemons to provides the ability to provision the compute servers. Meanwhile, Neutron component provides 'connectivity as a service' between network interfaces managed by other services like Nova.

More specifically, the cloud controller of Infrastructure as a Service (IaaS) platform, OpenStack, is in charge of configuring and assigning all computing and storage re-

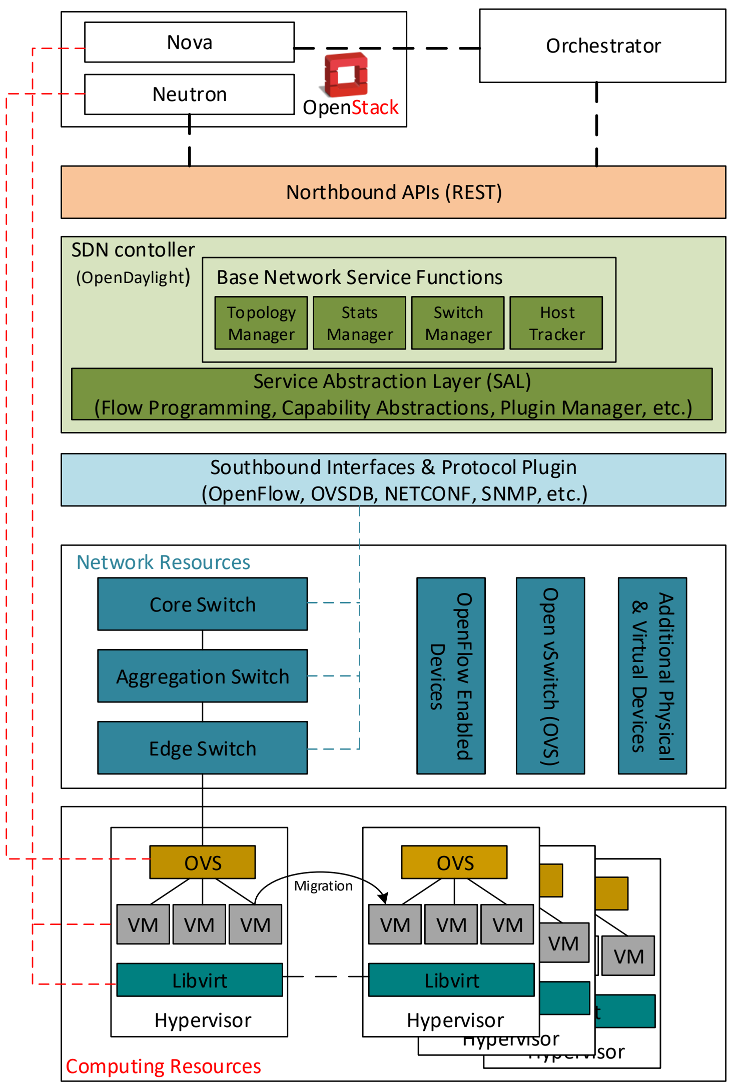


Figure 3.1: System Overview

sources, such as allocating flavor (vCPU, memory, storage) to VMs, placing the VMs on physical hosts using Nova component. It keeps all the information about physical hosts and virtual machines, such as residual storage and available computing resources. At the same time, all computer nodes update the states of hosted VMs to OpenStack Nova service. Furthermore, Neutron, the OpenStack network component, provides the management of virtual networking, such as start, update and bind the VM's port, as well as the communication between VMs. However, the OpenStack Neutron does not control network devices (switches). It only controls networking modules in compute nodes and network nodes.

Therefore, the SDN controller uses OpenFlow [42] protocol through southbound interfaces to manage the forwarding planes on network devices (switches). The open-source virtual switch, Open vSwitch (OVS) [176], provides the virtualization switching stack supporting OpenFlow and other standard protocols. Therefore, without expensive dedicated switches, we could install OVS in the white box as the OpenFlow switch in SDN-enabled data centers. Based on the link information between OpenFlow devices, the SDN controller calculates the forwarding tables for all network traffics. The OpenFlow switches forward the traffic flow according to the received forwarding tables from SDN controller. It also measures the received and transmitted data size as well as the bandwidth and latency between each other.

3.3.1 Live Migration in OpenStack

In this section, we present the details of block live migration in OpenStack. Providing a comprehensive solution to control the computing and network resources in the datacenter, OpenStack uses Libvirt [173] to manage hosts in order to support different kinds of virtualization. Nova live migration interacts with Neutron to perform the pre- and post-live-migration operations, and uses Libvirt to handle the actual live migration operations. The pre-copy live VM migration is used by default driven by libvirt.

Since libvirt 1.0.3, the QEMU's Network Block Device (NBD) server and "drive-mirror" primitive [177] are used to perform live storage migration (without shared storage setup). Similarly, since VMWare ESX 5.0, it uses VMKernel data mover (DM) and IO mirroring to perform live storage migration [178]. It separates the storage streaming data flows from the instance's RAM and hypervisor's internal state data flows. The disk transmission will perform concurrently with IO mirroring and VM migration. The write operation can be categorized into three types: 1) Into the block has been migrated, the writes will be mirrored to the target. 2) Into the block being migrated, the writes will be sent to the target first and wait in the queue until the region migration finished. 3)

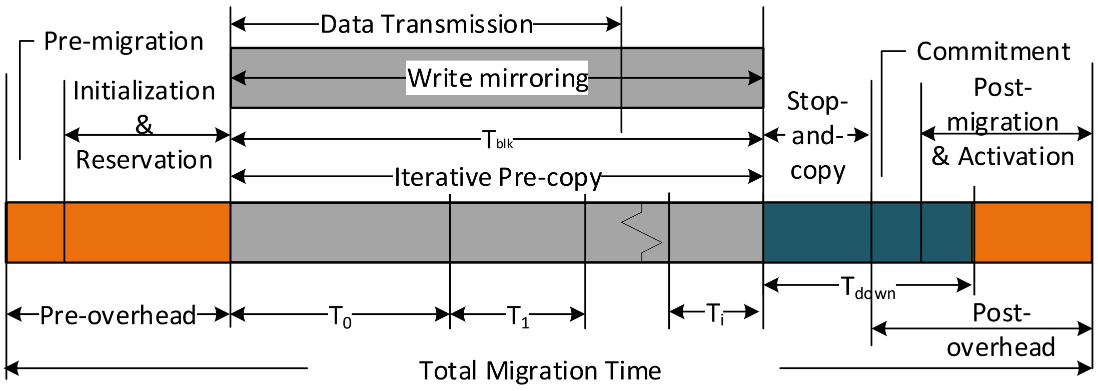


Figure 3.2: OpenStack block live migration

Into the block which will be migrated, the writes are issued to the source disk without mirroring. By caching the backing file or instance image when it boot in the Nova compute host, the mirror action could just apply to the top active overlay in the image chain. Thus, the actual disk transmission will be reduced.

Similar to the pre-copy migration described in [8], the block live migration in OpenStack includes 9 steps (Fig. 3.2):

1. **Pre-live migration** (PreMig): Creates VM's port (VIF) on the target host, updates ports binding, and sets up the logical router with Neutron server.
2. **Initialization** (Init): Preselects the target host to speed the future migration.
3. **Reservation** (Reserv): Target host sets up the temporary share file server; and initializes a container for the reserved resource on the target host.
4. **Disk Transmission**: For live storage migration, starts to perform storage migration and synchronizes the disk through IO mirroring.
5. **Iterative pre-copy**: For pre-copy VM migration, sends dirty pages that are modified in the previous iteration round to the target host. The entire RAM is sent in the first round.
6. **Stop-and-copy**: the VM is paused during the last iteration round according to the downtime threshold (remained amount is less than the required).

7. **Commitment** (Commit): Source host gets the commitment of a successfully received instance copy from the target host.
8. **Activation** (Act): Reassigns computing resource to the new VM and delete the old VM on the source host.
9. **Post-live migration** (PostMig): On the target host, updates port state and rebinds the port with Neutron. VIF driver unplugs the VM's port on the source host.

Where copying overheads are due to the pre-copy iteration and downtime is caused by the stop-and-copy, commitment and parts of the activation and post-migration operations. Although the network-related phases (disk transmission, pre-copy iteration, and stop-and-copy) usually dominate the total migration time, the pre- and post-live-migration, initialization, reservation, commitment, and activation could add a significant overhead to the migration performance in certain scenarios (large available network bandwidth, small disk size or low dirty page rate). The pre-live-migration, initialization, reservation could be classified as **pre-migration overheads** while the commitment, activation and post-live-migration as **post-migration overheads**.

Downtime Adjustment Algorithm: Unlike the stop conditions that are used in QEMU or Xen migration algorithm, the downtime threshold in OpenStack live migration increases monotonically in order to minimize the downtime for lower dirty page rate VM while increasing the availability of high dirty page rate VM migration with a reasonable downtime. The downtime adjustment algorithm used in Libvirt is basically based on three static configuration values (*max_downtime, steps, delay*):

- *live_migration_downtime*: The maximum threshold of permitted downtime;
- *live_migration_downtime_steps*: The total number of adjustment steps until the maximum threshold is reached;
- *live_migration_downtime_delay*: Multiplies the total data size with the factor equals to the time interval between two adjustment steps in seconds.

For example, the setting tuple (400, 10, 30) means that there will be 10 steps to increase the downtime threshold with 30 seconds delay for each step up to the 400ms maximum. With the total 3 GB RAM and Disk data size, the downtime threshold at time t, as $T_{d-thd}(t)$, will be increased at every 90 seconds starting from 40ms, i.e. $T_{d-thd}(0) = 40ms, T_{d-thd}(90) = 76ms, ..., T_{d-thd}(900) = 400ms, ..., T_{d-thd}(t > 900) = 400ms$. The mathematical model of downtime adjustment algorithm is shown in Equation (3.9). Although OpenStack only support static downtime adjustment in configuration files, we could use the *virsh* command to interact with the on-going migration based on the elapsed time.

3.4 Mathematical Model

We present the mathematical model of block live migration as well as the sequential and parallel migrations in the same network path.

3.4.1 Block Live Migration

The mathematical model of block live migration is presented in this section. According to the OpenStack live migration process, the components of pre and post-migration overheads can be represented as:

$$\begin{aligned} T_{pre} &= \text{PreMig} + \text{Init} + \text{Reserv} \\ T_{post} &= \text{Commit} + \text{Act} + \text{PostMig} \end{aligned} \tag{3.1}$$

We use D and M to represent the **system disk size** and the **VM memory size**, and let ρ denotes the **average compression rate** used in memory compression algorithm [179]. Let $\rho^{'}$ and $R^{'}$ denotes the **average disk compression rate** and **mirrored disk write rate**. Let R_i and L_i denote the **average dirty page rate** need to be copied and **bandwidth** in iteration round i. In total n round iterative pre-copy and stop-and-copy stages, T_i denotes the time interval of i_{th} round iteration shown in Fig. 3.2. Therefore, the **transferred**

volume V_i in round i can be calculated as:

$$V_i = \begin{cases} \rho \cdot M & \text{if} i = 0 \\ \rho \cdot T_{i-1} \cdot R_{i-1} & \text{otherwise} \end{cases} \tag{3.2}$$

As shown in Fig. 3.2, the time interval of the i_{th} iteration can be calculated as:

$$T_i = V_i / L_i = \rho^{i+1} \cdot \prod_{j=0}^{i-1} R_j \cdot M \Big/ \prod_{j=0}^{i} L_j \tag{3.3}$$

In [30], they assume that, when R_i, L_i are constant, the average dirty page rate is not larger than the network bandwidth in every iteration. Let ratio $\sigma = \rho \cdot R/L$. Therefore, $T_i = \rho \cdot M \cdot \sigma^i / L$. The total time of iterative memory pre-copy T_{mem} can be calculated as:

$$T_{mem} = \frac{\rho \cdot M}{L} \cdot \frac{1 - \sigma^{n+1}}{1 - \sigma} \tag{3.4}$$

Then, the transmission time of live storage migration T_{blk} can be represented as:

$$T_{blk} \leq \rho' \cdot \left(D + R' \cdot T_{blk}\right) \Big/ L \tag{3.5}$$

Thus, the upper bound transmission time of the live storage migration is:

$$T_{blk} \leq \frac{\rho' \cdot D}{L - \rho' \cdot R'} \tag{3.6}$$

For a more accurate T_{blk}, one need to simulate the write behavior based on the actual workload. The network part of block live migration is the maximum value of Equation (3.4) and (3.6):

$$T_{copy} = Max \left\{T_{blk}, T_{mem}\right\} \tag{3.7}$$

The total migration time of block live migration T_{mig} can be represented as:

$$T_{mig} = T_{pre} + T_{copy} + T_{post} \tag{3.8}$$

Let (θ, s, d) denotes the setting tuple (*max_downtime, steps, delay*) of the downtime adjustment algorithm. Therefore, the live migration downtime threshold at time t can be represented as:

$$T_{d-thd}(t) = \lfloor t/(d \cdot (D+M)) \rfloor \cdot (\theta s - \theta)/s^2 + \theta/s \tag{3.9}$$

The downtime threshold of remained dirty pages accordingly will be

$$V_{d-thd}(t) = T_{d-thd}(t) \cdot L_{n-1} \tag{3.10}$$

where L_{n-1} is the n-1 round bandwidth estimated by the live migration algorithm and $L_{n-1} = L$ when transmission bandwidth is a constant.

The live migration changes to the stop-and-copy phase when remained dirty pages is less than the current threshold, as $V_n \leq V_{d-thd}(t)$. Using the Equation (3.2) in the inequality, the total round of memory iteration can be represented as:

$$n = \left\lceil \log_{\sigma} \frac{V_{d-thd}(t)}{M} \right\rceil \tag{3.11}$$

Therefore, the upper bound of actual migration downtime is $T_{down} = T_d + T^{'}_{post} \leq T_{d-thd}(t) + T^{'}_{post}$, where T_d is the time that transferring the remained dirty pages and storage and $T^{'}_{post}$ is the time spent on the part of post-migration overheads to resume the VM.

3.4.2 Sequential and Parallel Migrations

When applying energy-saving policy, hardware maintenance, load balancing or encountering devastating incidents, we need to evacuate part of or all VMs from several physical hosts to others through live VM migrations as soon as possible. In this section, we establish the mathematical model of sequential and parallel live VM migrations which share the same network traffic path. For example, there are 4 same live migrations sharing the same network path as well as source and destination hosts. In Figure 3.3, lower graph shows the sequential live migration. Because each migration fully uses the path bandwidth, the network transmission part is much smaller than the part of parallel mi-

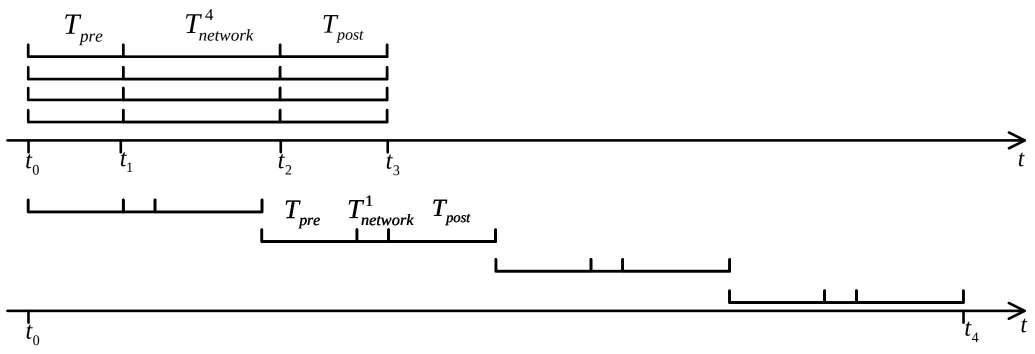


Figure 3.3: An example of sequential and parallel migrations

gration shown in the upper graph at which 4 migrations share the bandwidth evenly. However, in this example, the total network bandwidth is extremely large comparing to the dirty rate and the memory size of each VM is relatively small. Therefore, the pre and post migration overheads contribute substantially to the total migration time. As the result, even though sharing the same network path could extend the memory iteration, parallel migration running the pre and post migration on multicore in this situation actually outperforms the sequential algorithm.

Because the pre-live-migration process of next migration is executed after the completion of current migration, there is a bandwidth gap between every sequential live migration because of the non-network overheads. Therefore, the total evacuation time of N VM sequential migrations could be calculated as the sum of every migration's overhead processing time and network transmission time:

$$T_{seq} = \sum_1^N T_{mig} = \sum T_{overhead} + \sum T_{network} \tag{3.12}$$

The **response time** of VM migration task refers to the time interval from the point that migration task is released and the point it is finished. The **migration time** indicates the real execution time of the migration task which excludes the waiting time which is the time interval between the migration task release point and the actual start point. The **evacuation duration** refers to the time interval from the beginning of the first released migration task to the end of the last finished task of all VM migrations.

Pre- and post-migration overheads refer to the operations that are not part of the direct network transmission process. These non-network operations could add a significant overhead to the total migration time and downtime. For more concise explanation, we assume that every VM in parallel migration has same dirty page rate and flavor. Let m denotes the **allowed parallel number**, p denotes the **processing speed** of one core. We assume that the largest allowed parallel migration is smaller than the minimum core number on the hosts, $m \leq Num(cores), m \leq N$. When $m > N$, $m = N$ in the corresponding equations. As every migration sharing the network bandwidth equally, L/m is the transmission rate for each migration. Therefore, using the previous equations, the network transmission time of parallel m migrations can be represented as:

$$T^{m}_{network} = Max\left\{ m \cdot T_{blk}, \frac{m \cdot \rho \cdot M}{L} \cdot \frac{1-(m\sigma)^{n+1}}{1-m\sigma} \right\} \tag{3.13}$$

It is clear that $T^{m}_{network} \geq \Sigma^{m} T^{1}_{network}$.

Let W_{pre} and W_{post} denote the workload of pre and post-migration overheads. As the overheads are significant when the network bandwidth L allocated to the path is more than sufficient or the dirty page rate R is small, we assume that:

$$\Sigma^{m} W_{pre}/m \cdot p \geq T^{m}_{network} \tag{3.14}$$

Let $X = \lfloor N/m \rfloor$ denote total X busy rounds of m cores. Therefore, the maximum evacuation time of parallel migration $T_{par} = Max(T^{'}_{par}, T^{''}_{par})$ can be represented as:

$$\begin{aligned}
T^{'}_{par} &= \frac{\Sigma_1^{Xm} W_{pre}}{m \cdot p} + \frac{\Sigma_{Xm+1}^{N} W_{pre}}{N-Xm+2} + T^{N-X}_{network} + \frac{\Sigma_{Xm+1}^{N} W_{post}}{N-Xm+2} \\
&= \frac{(\lfloor N/m \rfloor + 1) \cdot W_{pre} + W_{post}}{p} + T^{N-X}_{network} \\
T^{''}_{par} &= \frac{\Sigma_1^{m} W_{pre}}{m \cdot p} + T^{m}_{network} + \frac{\Sigma_1^{Xm} W_{post}}{m \cdot p} + \frac{\Sigma_{Xm+1}^{N} W_{post}}{N-Xm+2} \\
&= \frac{(\lfloor N/m \rfloor + 1) \cdot W_{post} + W_{pre}}{p} + T^{m}_{network}
\end{aligned} \tag{3.15}$$

As $0 \leq \sigma < 1$, we could get the upper bound of parallel network transmission time:

$$T^{m}_{network} \leq Max \left\{ m \cdot T_{blk}, \frac{m \cdot \rho \cdot M}{L \cdot (1 - m\sigma)} \right\} \tag{3.16}$$

Moreover, the average response time of N sequential and parallel migrations can be represented as:

$$T^{seq}_{response} = (N+1)/2 \cdot \left(W_{overhead}/p + T^{1}_{network} \right) \tag{3.17}$$

$$T^{par}_{response} = W_{overhead}/p + T^{m}_{network} \tag{3.18}$$

Furthermore, the lost time of network transmission and the saved time of overhead processing for m concurrent live migration can be calculated as:

$$\Delta_{network} = T^{m}_{network} - \Sigma^{m} T^{1}_{network} \tag{3.19}$$

$$\Delta_{workload} = \Sigma^{m} T_{overhead} - \Sigma^{m} T_{overhead}/m \cdot p \tag{3.20}$$

Therefore, when $\Delta_{network} < \Delta_{workload}$, the evacuation time of parallel migration is smaller than the sequential migration.

All proposed models and results of single migration and sequential and parallel migrations for block live migration also apply to the general live VM migration with disk sharing by deleting the live disk transmission parts, T_{blk} and D, in the models.

3.5 Performance Evaluation

There are several parameters which can influence the live VM migration performance in SDN-enabled data centers from **system view**, such as the flavor, CPU, memory, and static downtime adjustment, **network view**, such as parallel and sequential migrations, available bandwidth, and dynamic flow scheduling update rate, and **application view**, such as response time under different migration strategies. In this section, we explore the impacts of these parameters on migration performance. The migration time, downtime, and transferred data shown in the results are the average values. In OpenStack, we can use the *nova migration-list* to measure the duration of live migration. The down-

Table 3.1: Specifications of physical hosts in CLOUDS-Pi

Machine	CPU	Cores	Memory	Storage	Nova
3 × IBM X3500 M4	Xeon(R) E5-2620 @ 2.00GHz	12	64GB (4 × 16GB DDR3 1333MHz)	2.9TB	compute1-3
4 × IBM X3200 M3	Xeon(R) X3460 @ 2.80GHz	4	16GB (4 × 16GB DDR3 1333MHz)	199GB	compute4-7
2 × Dell OptiPlex 990	Core(TM) i7-2600 @ 3.40GHz	4	8GB (4 × 16GB DDR3 1333MHz)	399GB	compute8-9

time of live migration could be calculated by the time stamp difference of VM lifecycle event (VM Paused and VM Resumed) in both Nova log files. Each configured migration experiment is performed 6 times.

3.5.1 Testbed and its Specification

As current production system will not allow users to access or modify the low-level infrastructure elements, such as resource management interfaces and SDN controllers and switches, needed for experiments, we created our own testbed. CLOUDS-Pi [180], a low-cost testbed environment for SDN-enabled cloud computing, is used as the research platform to test virtual machine block live migration. We use OpenStack combined with OpenDayLight [181] (ODL) SDN controller to manage the SDN-enabled Data Centers, which contains 9 heterogeneous physical machines connected through Raspberry Pis as OpenFlow switches whose specifications are shown in Table 3.1. The Raspberry Pis are integrated with Open vSwitch (OVS) as 4-port switches with 100 Mbps Ethernet Interfaces. The network physical topology is shown in Fig. 3.4. The OpenStack version we used is Ocata and the Nova version is 15.0.4 and the Libvirt version is 3.2.0. The Ubuntu tool *stress-ng* [182] is used as the micro-benchmark to stress memory and CPU to pinpoint the impacts of parameters on migration performance.

It will allow researchers to test any SDN-related technology in the real environment. Allowed network speed in the testbed is scaled together with the size of computing cluster. Although the testbed's scale is small regarding the number of computer nodes and the network, it can represent the key elements in the large-scale systems. The evaluation results produced by the testbed will be more serious in a large scale environment. Furthermore, as we do not focus on the IO stress on the migrating storage, the evalua-

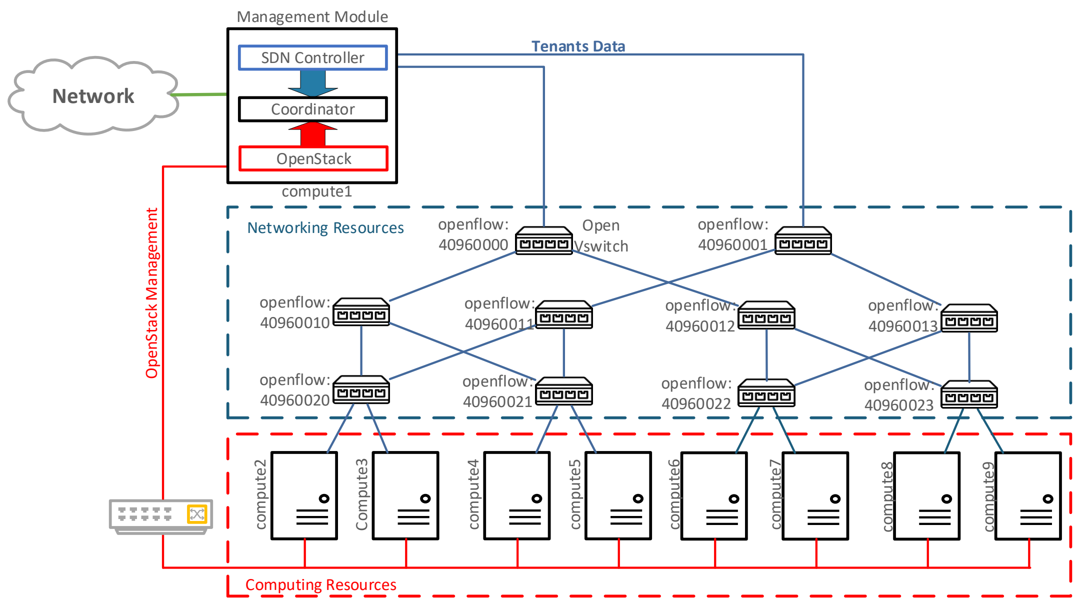
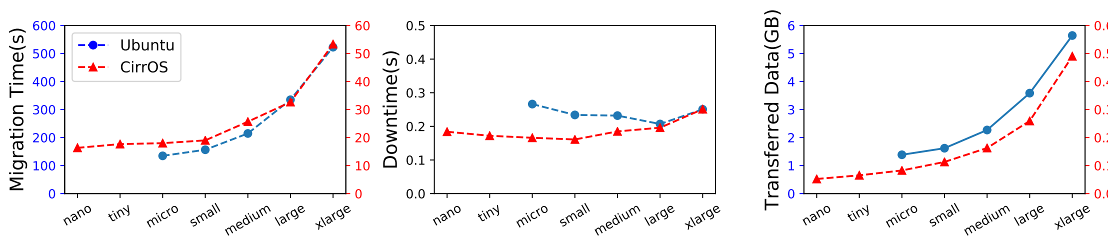


Figure 3.4: SDN-enabled Data Center Platform

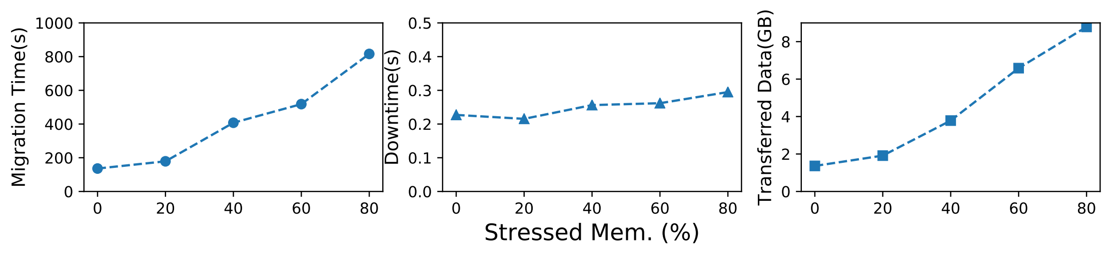
tion results could also benefit the live migration with shared storage, as well as the live container migration.

3.5.2 Primary Parameters

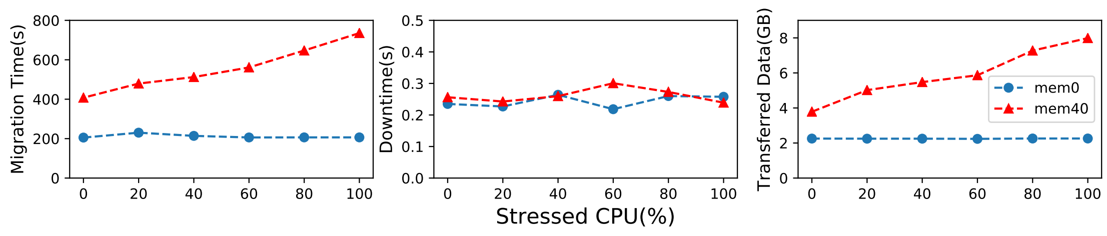
First, we evaluate the fundamental parameters, such as flavor, memory and CPU loads, which affect the migration time, downtime and total transferred data of block live VM migration in OpenStack. As we measured, the amount of data from destination to source can be omitted because it only accounts for around 1.8 percent of total transferred data. The transferred data is measured by the SDN controller through OpenFlow protocol. We set 7 flavors in OpenStack, which are nano, tiny, micro, small, medium, large, xlarge (Table 3.2). Not only the RAM size but the ephemeral disk size can affect the migration time as well as the total transferred data (Equation 3.8). We evaluate these primary parameters by migrating instances from *compute2* to *compute3*. In the flavor experiment, we use two Linux images, CirrOS and Ubuntu-16.04, and the smallest flavor suitable for the Ubuntu image is micro. The image size of CirrOS is 12.65 MB, and Ubuntu



(a) Ubuntu and CirrOS VM with different flavors



(b) Live migration based on stressed memory



(c) Live migration based on CPU loads

Figure 3.5: Primary VM parameters

is 248.38 MB. In memory stress experiment, we evaluate the migration performance of different memory-stressed Ubuntu-16.04 instance with micro flavor from 0 to 80 percent. In CPU stress memory experiment, we compare the migration performance with 0 to 100 stressed CPU between Ubuntu instance with 0 memory stress (mem0) and 40 percent memory-stressed (mem40) VMs.

Flavor: Figure 3.5(a) illustrates the migration performance (migration time, downtime, and total transferred data) of idle VMs with different flavors. Larger RAM and disk sizes lead to longer migration time and total transferred data. The VM block live migration cost with the same flavor could be a huge difference due to the system disk size and the required RAM of different OS instance. According to the downtime ad-

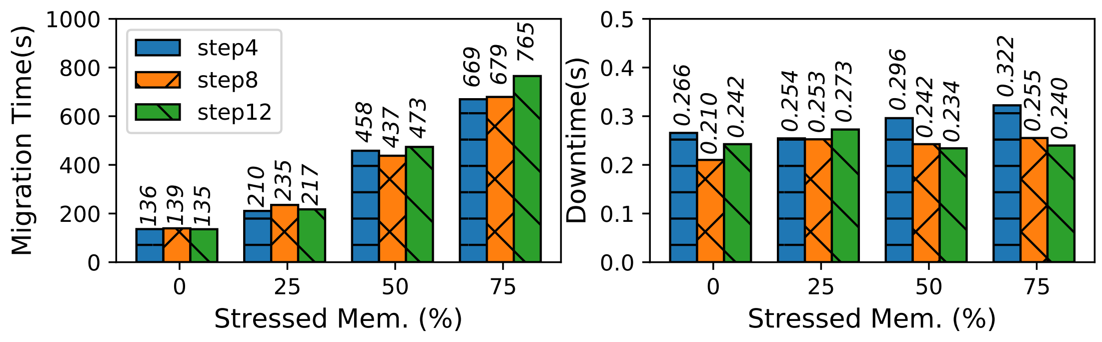
Table 3.2: Specifications of VM flavors in OpenStack

No.	Name	vCPUs	RAM	Disk	No.	Name	vCPU	RAM	Disk
1	nano	1	64MB	1GB	5	medium	2	3.5GB	40GB
2	tiny	1	512MB	1GB	6	large	4	7GB	80GB
3	micro	1	1GB	10GB	7	xlarge	8	15.49GB	160GB
4	small	1	2GB	20GB					

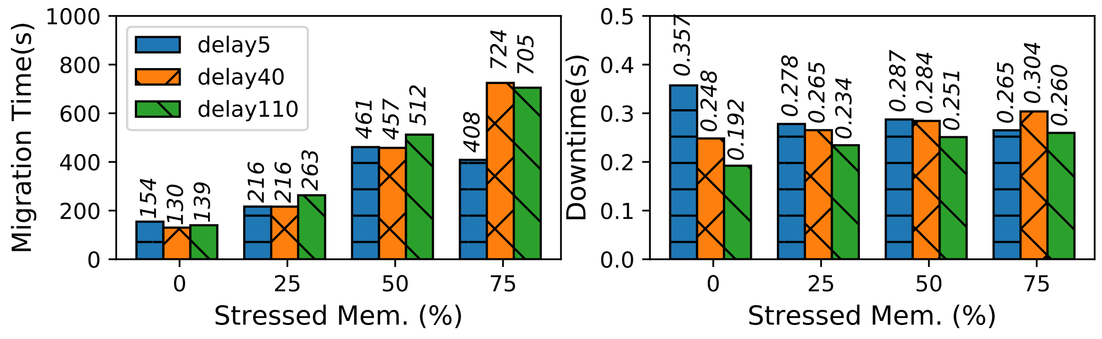
justment algorithm, a longer migration time can lead to a larger downtime. However, the difference of downtime is small compared to the significant difference of migration time. From flavor micro to xlarge, the transferred data is increased linearly. Furthermore, the transferred date vs. flavor figure illustrates that there is a constant data size difference between CirrOS and Ubuntu with the same flavor. With the same flavor, VM with a larger and more complex OS installed has a longer migration time and larger transferred data as the data size difference of the OS base image and dirty rate caused by OS processes.

Memory: The dirty page rate (and dirty block rate) directly affects the number of pages that are transferred in each pre-copy iteration. Fig. 3.5(b) shows that the performance of different memory-stressed Ubuntu instances from 0 to 80 percent on the migration time, downtime, total data transferred from source. As shown in Equation (3.8) and (3.9), the relationship between the dirty page rate and live migration performance is not linear due to the downtime adjustment algorithm. The downtimes of migrations may be constant with different dirty page rates because of the delay of every downtime adjustment, such as 0 and 20 percent memory-stressed VMs. With the downtime adjustment algorithm, the downtimes of live migrations with drastically different dirty page rate remain at a stable range.

CPU: Higher CPU workloads can lead to a migration performance degradation because of the page copying operation overhead during the pre-copy iterations. Meanwhile, the high CPU workloads can also cause interference among memory-intensive tasks which leads to a large migration time. We examine the block live migrations based



(a) *live_migration_downtime_steps*



(b) *live_migration_downtime_delay*

Figure 3.6: Live migrations based on different step and delay settings

on various CPU loads from 0 to 100 percent. Figure 3.5(c) shows that, without stressed memory, the CPU loads inside VMs are irrelevant to the downtime and duration of live migration with the minor copying overhead due to the pre-copy iterations. However, as the CPU usage of a 40-percent-stressed memory task is 100 percent, an extra CPU workload can lead to a larger amount of total transferred data and migration time. For idle VMs, the migration time and transferred data are constant under various range of CPU workload. For more busy VMs, extra CPU workload leads to a linear increase in migration time and transferred data size.

3.5.3 Downtime Configuration Effectiveness

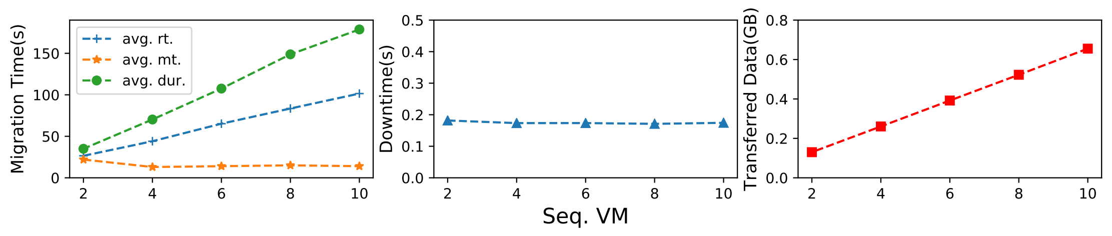
In OpenStack, the live VM migration time could shift dramatically based on different configuration tuples (*max_downtime, steps, delay*). Although only implemented in Open-

Stack, the downtime adjustment algorithm can also apply to other cloud computing platforms. In this experiment, the Ubuntu-16.04 instance with micro flavor is migrated between NOVA compute node *compute2* and *compute3*. We perform migrations based on the different step or delay settings and other two default values, i.e., (500, 4, 75) and (500, 10, 5), with 0 to 75 percent stressed memory VM.

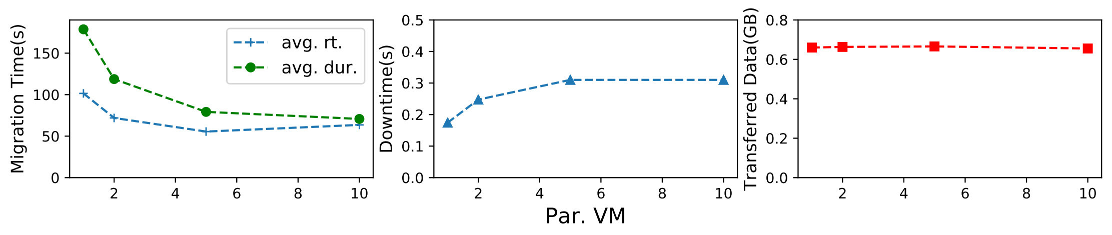
Figure 3.6 indicates that for less memory stressed VMs (low dirty page rate), the static algorithm based on short delay could lead to a higher downtime with a slightly different migration time. However, for heavy memory stressed VMs (high dirty page rate), the adjustment of large delay setting, such as delay40, delay110, leads to an extremely long migration duration. The larger adjustment step setting leads to a larger migration time with a smaller downtime. However, step8 (500, 8, 75) leads to a better result in migration time compared to step12 and in downtime compared to step4 when VM memory is 75 percent stressed. We also notice that the setting (500, 10, 5) is a better choice when VM has high dirty page rate and (500, 4, 75) is better when the rate gets lower. When the dirty page rate is high, the migration time gets benefits from quickly raised downtime threshold while the downtime remains at a stable range. When it is low, the downtime gets benefits from smaller downtime threshold with slow adjustment. We should dynamically configure the optimal downtime setting tuple to improve both migration time and downtime based on the migration model for every live migration task.

3.5.4 Live VM Migration in Parallel

The default value *max_concurrent_live_migrations=1* of NOVA configuration of max allowed parallel migration is one, which means only one live migration could be performed at the same time. In this experiment, we evaluate the migration duration of one compute host that needs to evacuate all VMs to another. First, we need to change the default max allowed parallel migration to perform maximum *m* live migrations in parallel. The CirrOS instances with tiny flavor are migrated between node *compute2* and *compute3*. All migration operations are released at the same time with different maximum parallel migration. We measure the response time of each migration task and the



(a)



(b)

Figure 3.7: (a) Sequential migrations with different number of VMs; and (b) Multiple live migrations of 10 VMs where x-axis indicates the max allowed concurrent migration.

total evacuation time of 10 idle CirrOS VMs. We also exam the sequential live migration with several VMs from 2 to 10.

Figure 3.7(a) indicates that the response time (rt), migration time (mt) and evacuation duration (dur) of sequential live migrations increase linearly with the number of VMs. Figure 3.7(b) only demonstrates the rt and dur, as the mt equals to the rt in this parallel migration experiments. However, the parallel migrations could significantly reduce the total evacuation time and each migration time of 10 idle VMs. With the max allowed concurrent migration increasing from 1 to 10, the total live migration evacuation time decreases by 59.6%. Meanwhile, the migration time of each VM decreases up to 50%.

As shown in Equation (3.19), (3.20), when $\Delta_{network} < \Delta_{workload}$, the pre- and post-migration overheads constitute a large portion of the total migration time, e.g., parallel migration of the tiny flavor CirrOS VMs with 100Mbps bandwidth (Fig. 3.7(b)). Therefore, several pre- and post-live-migration processes concurrently running on both hosts can reduce the total evacuation time (3.15) and average response time (3.18) compared to the sequential live migrations (3.12), (3.17). Therefore, when the multiple VM evacuation happens in the same network path, we need to decide the sequential and parallel

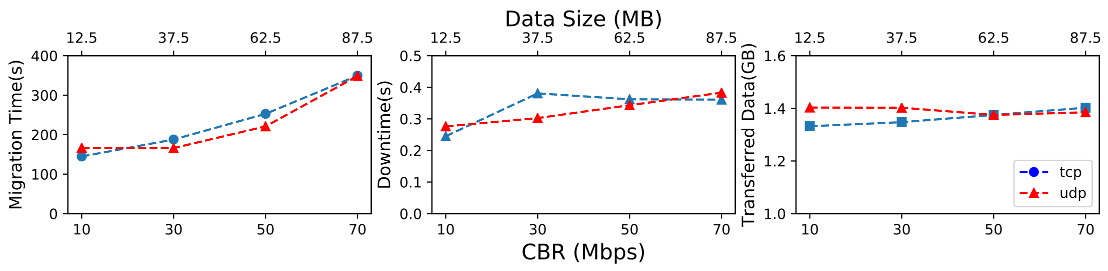


Figure 3.8: Block live migrations with TCP and UDP background traffic

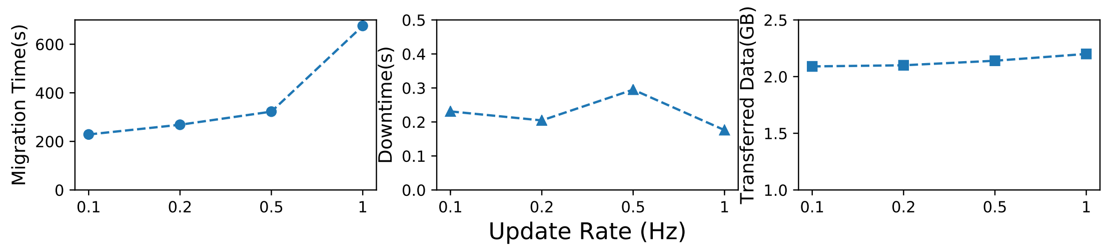
live migration based on both network and computing aspects to achieve a better total migration time (duration).

3.5.5 Network-Aware Live Migration

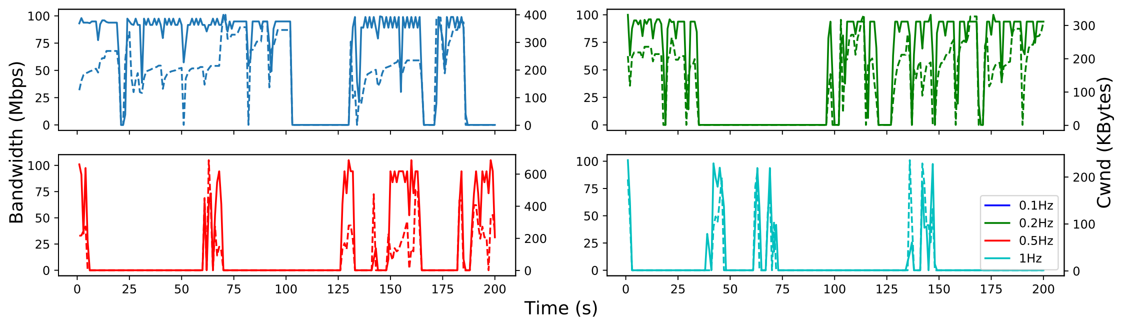
As the networking resources are limited, we pinpoint the essential network aspects that influence the efficiency of block live migration in SDN-enabled cloud computing, such as, the available network bandwidth, network patterns, SDN flow scheduling algorithms.

TCP and UDP Traffic: Block live Migration is highly relative to the network bandwidth as well as the background traffic on the links. The total migration time and downtime are negatively correlated with the network bandwidth. Therefore, we measure the migration performance under the default downtime configuration with various network traffic scenarios with different constant bandwidth rate (CBR) in TCP and burst transmission in UDP. UDP datagrams are sent in the same data size in every 10 seconds. The *iperf3* [183] is used to generate background traffic between live migration source and destination hosts through the same path in SDN-enabled data center network. The image of VM is Ubuntu-16.04 with micro flavor under no stressed memory. Figure 3.8 indicates that, when the dirty page rate is 0, the transferred data is not linearly increased with the migration time. The migration time is increased linearly with the bandwidth decreasing.

Dynamic SDN Flow Scheduling: In this experiment, we pinpoint the impact of the flow scheduling algorithm update rate on block live migration in SDN-enabled cloud computing. When SDN controller is proactively scheduling the flows, latencies exist



(a) Comparison of migrations

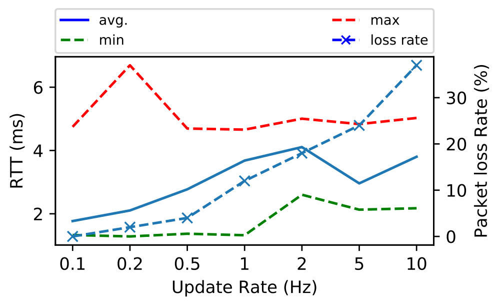


(b) Live migrations based on different SDN scheduling update rate

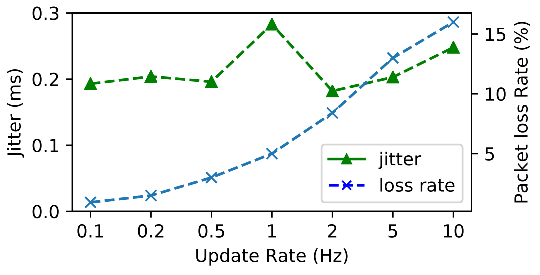
Figure 3.9: Live migrations based on different SDN scheduling update rate

between controller and switches (PacketOut message send to the switches and PacketIn to the controller). Moreover, in the flow tables, latencies occur when installing, deleting flow entities. The scheduler based on SDN controller (OpenDayLight) REST APIs proactively pushes the end-to-end flow in a certain time period to dynamically set the best path. The idle Ubuntu-16.04 instance with micro flavor is migrated from *compute3* to *compute9*. As shown in Fig. 3.4, there are two shortest paths between *compute3* and *compute9* that each one contains 5 OpenFlow nodes (OpenFlow-enabled switches). A round-robin scheduler rescheduling the traffic of live migration periodically based on these paths. We also use *iperf3* to generate TCP and UDP traffic to evaluate the latency, TCP window size, and packet loss rate.

Figure 3.9(a) shows that the migration time is positively correlated with the update rate while the transferred data is just slightly increased. As the dynamic scheduling update rate increases, the link bandwidth rapidly decreases which leads to a large migration time. Meanwhile, Figure 3.9(b) indicates that the TCP throughput goes down more frequently with high flow update rate. The TCP congestion window size decreases to



(a) TCP latency and packet loss rate



(b) UDP jitter and packet loss rate

Figure 3.10: Network performance with different SDN scheduling update rate

1.41 KBytes when the bandwidth is 0 bits/sec. Figure 3.10 shows the TCP and UDP protocol performance with different update rates from 0.1Hz to 10Hz. The packet loss rate increase linearly with the update rate and the average maximum TCP latency (Round-Trip Time) is 2 times larger at 2Hz than the minimum value at 0.1Hz. When the TCP traffic suffers the bandwidth degradation, the UDP transmission rate is always around 90 Mbps regardless of the scheduling update rate.

With the high flow entries updating in OpenFlow-enabled switches, the latencies between SDN controller and switches, and inside the switch flow tables have a significant influence on traffic forwarding performance. The network congestion leads to the high packet loss rate. The period of no traffic interval is caused by the TCP congestion avoidance algorithm. It decreases the data transfer rate when encounters packet loss based on the assumption that the loss due to the high latency and network congestion. Furthermore, the flow update rate could also impact the TCP window size that causes the bandwidth jitters due to the TCP slow start. In a highly dynamic network, the available bandwidth and delays in the routing paths can change frequently. Therefore, it is essential that optimize the update rate and best path selection of SDN forwarding scheduler based on the trade-off between OpenFlow-enabled switches performance (bandwidth degradation due to delays inside switches and between controller and switches) and the available network bandwidths and delays.

Table 3.3: Request response time without VM migration

Exp.	Duration(s)	RT(ms)	HTTP0	HTTP200	Total
Initial	1200	74.21	35	42310	51634
Initial	500	75.90	22	17435	21438
Initial	400	75.77	22	13893	17088
scheduled	400	65.380	17	14175	17357

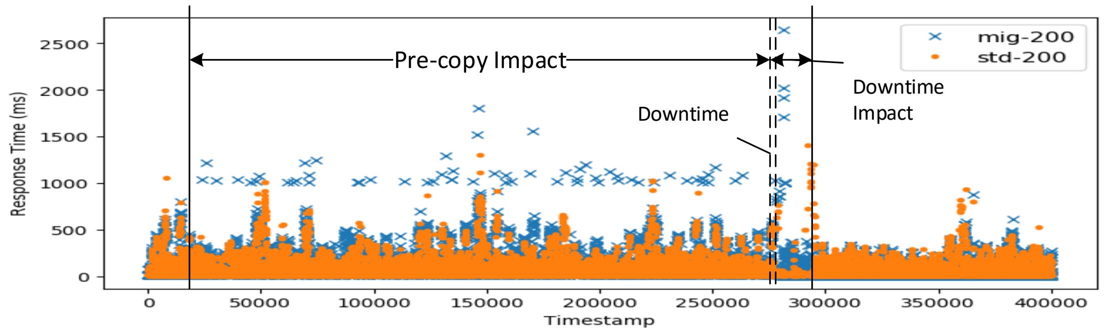
Table 3.4: Application performance in 400 seconds

Exp.	MT(s)	RT(ms)	HTTP0	HTTP200	Total
c-93	248.34	84.201	20	14166	17348
s-39	N/A	192.273	109	13410	16558

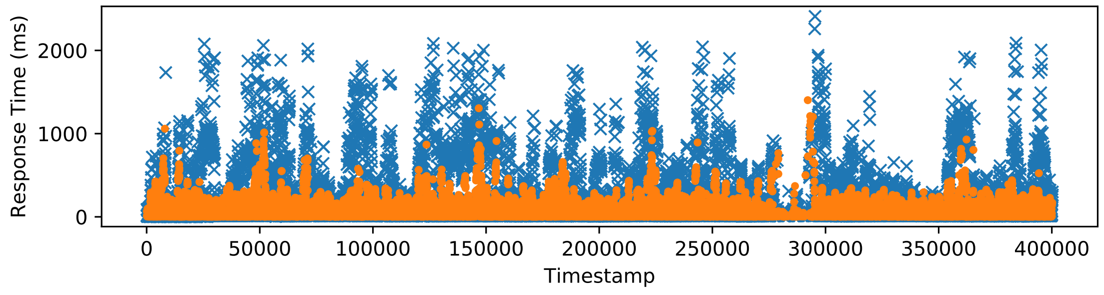
3.5.6 Impacts on Multi-tier Application Response Time

In this experiment, we evaluate the impact of VM live migration on the real web application, such as *MediaWiki*, using *WikiBench* [184]. It uses *MediaWiki* in the application server and real database dumps in the database server. In client VM, the *wikijector* as traffic injector controls the simulated client to reply the traces of real *Wikipedia* traffic. Regarding the scale of the testbed, we use 10 percent of *Wikipedia* trace to simulate the real traffic. The database and *MediaWiki* Apache servers are allocated in *compute3*, and one *WikiBench* injector as the client VM located in *compute9*. The client and server VMs are the Ubuntu instances with micro flavor and database server is with large flavor. The first scenario (c-93) is migrating the client VM to *compute3* to simulate the consolidation (scheduled) to reduce the latency. The second one (s-39) is migrating the application server to *compute9* in order to evaluate the effect of live migration on application response time.

In the scenario c-93, the major application traffic is outbound traffic from the destination host. Therefore, the live migration traffic would just slightly affect the QoS of web service. Table 3.3 indicates that the application response time (RT) is improved after the



(a) Client VM migration



(b) Service VM migration segment

Figure 3.11: Response time of Wikipedia in 400s

VM consolidation (scheduled). Figure 3.11a shows the initial response time (std-200) of the success requests (HTTP 200) and the response time of success requests during the client VM migration (mig-200). It indicates that the **response time** is increased during the migration and the **worst-case response time** occurs after the downtime of client's live VM migration because the application server needs to process extra requests and migration downtime postpones the response time of the requests which are sent before and during the downtime. On the other hand, if the injector and application server are located in the same host when the migration is performing, due to all requests happened inside the host, the live migration traffic will not affect the application response time.

However, in scenario s-39, i.e., the application traffic is sent to client VM (*compute9*), the pre-copy live migration traffic flow will contend for the shared bandwidth due to the same traffic direction. Therefore, the **worst case response time** may occur not only after downtime but during the migration time as shown in Fig. 3.11b. Meanwhile, Ta-

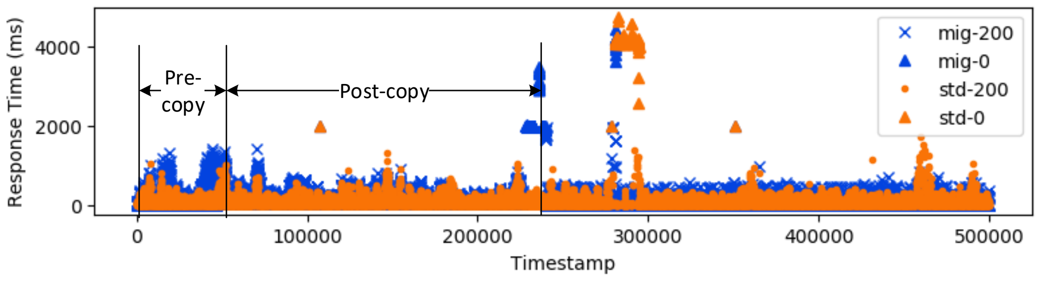
Table 3.5: Server migration under different strategies

Exp.	MT(s)	Duration(s)	RT(ms)	HTTP0	HTTP200	Total
s-39	N/A	400	192.27	109	13410	16558
AC	908	1200	245.33	6722	18461	29915
H-PC	237	500	156.73	190	16906	20912

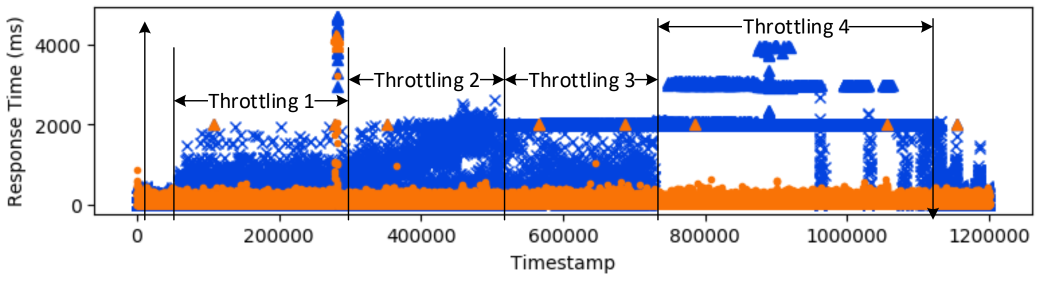
ble 3.4 shows that the average response time of requests is dramatically larger than the migration of client VM. The request timeout (HTTP 0) happens much often due to the server migration.

We notice that the server migration from *compute3* to *compute9* cannot finish in 20 minutes. For memory-intensive instances, like the Wikipedia server, there are two optional strategies to perform a successful live migration: **Hybrid post-copy (H-PC)** and **Auto-convergence (AC)**. Thus, we evaluate the migration performance and impacts on the response time of the hybrid post-copy and auto-convergence strategies for Wikipedia server in the scenario s-39. Table 3.3 shows the initial response time of 1200 seconds, 500 seconds and 400 seconds time intervals without any migration as well as the average migration time (Duration), response time (RT), and the number of success (HTTP200), timeout (HTTP0), and total requests.

Hybrid post-copy: With the start of pre-copy mode, the post-copy migration will be activated if the memory copy iteration does not make at least 10 percent increase over the last iteration. It will suspend the VM and process state on the source host. The VM will resume on the target host and fetch all missing pages as needed. However, the post-copy page fetching will slow down the VM which degrades the service performance and the VM will reboot if the network is unstable. The average **response time** of hybrid post-copy is better than the pre-copy migration as shown in Table 3.5. The **timeout requests** are slightly increased during the post-copy migration. Furthermore, Figure 3.12(a) indicates response time of success and timeout requests without migration (std-200, std-0) and during the hybrid post-copy (mig-200, mig-0). It illustrates that under a stable network environment, the impacts of missing page fetching on application response time is less than pre-copy iteration traffic.



(a) Hybrid post-copy



(b) Auto-convergence

Figure 3.12: Response time of successful server migrations

Auto-convergence: By throttling down the VM's virtual CPU, auto-convergence will only influence the workloads where the memory write speed is dependent on the CPU execution speed. As migration time flows it will continually increase the amount of CPU throttling until the dirty page rate is low enough for migration to finish. Figure 3.12(b) indicates that the task of Wikipedia request has a worse **response time** under a larger throttling amount. The request tasks are highly related to the CPU execution speed. Therefore, the throttling down leads to a successful migration of the Wikipedia server. However, as the timeout threshold of a request is 2 seconds, the performance of the server is devastated under the last throttling down, i.e., most requests are timed out (mig-0). A larger timeout threshold for requests should be set according to the amount of throttling down. Although it can successfully perform the live server migration, the average response time is even larger than the pre-copy migration requests' (Table 3.5).

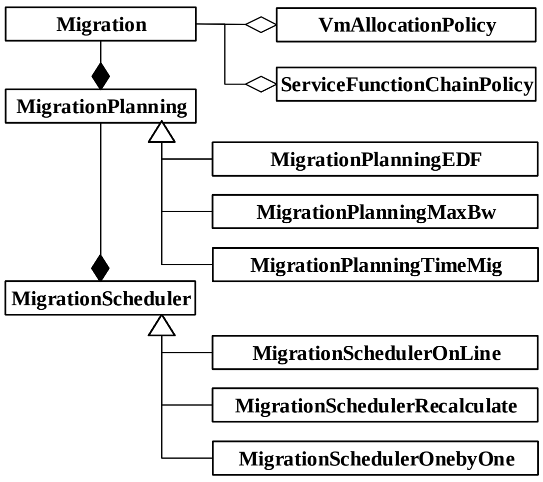


Figure 3.13: MigrationScheduler to simulate single live migration follows the sequence of MigrationPlanning

Moreover, compare to the hybrid post-copy strategy, the auto-convergence leads to a much larger migration time.

For memory-intensive VMs, H-PC is a better strategy in a stable network environment. Otherwise, AC is the option for applications that dirty page rate is highly related to the CPU speed. Due to the throttling down, service time out should be increased accordingly.

3.6 Simulation Platform

In this section, we first introduce the details of our event-driven simulation platform that used in this thesis large-scale experiments of multiple live migrations in SDN and NFV-enabled cloud data centers.

To evaluate the performance of large-scale multiple live migrations, we extended the CloudSimSDN-NFV [166] by implementing the phases of live VM migration and corresponding parameters (Table 3.6 and Fig. 1.5). It is an event-driven simulation en-

This section is derived from:

- **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, ”SLA-aware multiple migration planning and scheduling in SDN-NFV-enabled clouds”, *Journal of Systems and Software (JSS)*, Volume 176, Pages: 110943, Elsevier, 2021.

Table 3.6: Simulation events in *MigrationScheduler*, *SDNDataCenter*, and *NetworkOperatingSystem*

Num.	Event and Operation	Function
0	*SDN_VM_MIG_PRE*	check available network and set up the migration routing
1	*SDN_VM_MIG_START*	start the pre-copy phases
2	*SDN_PACKET_COMPLETE*	check the application and migration flows, estimate the downtime and send the remaining dirty page
3	*SDN_PACKET_SUBFLOW_COMPLETE*	check the completion of multiple migration flows
4	*SDN_VM_PAUSE*	pause the VM/VNF based on the downtime and iteration threshold
5	*SDN_VM_RESUME*	resume the VM/VNF on the dest host after the completion of the stop-and-copy flow
6	*SDN_VM_MIG_POST*	shut and delete the original instance and rerouting the flows to the new VM/VNF.
7	*SDN_VM_MIG_SCHEDULER*	process the migration scheduling in the current time.

Table 3.7: Parameters supported in event-driven simulator

Type	Parameters						
computing	CPU	Memory	Disk	Workloads	Task Scheduling	Task Priority	Overbooking Ratio
networking	Bandwidth	Topology	Switch Buffer	Ports	Channel	Control Plane	Data Plane
monitoring	Statistic	Energy Consumption	Utilization	Response time	Network Delay	Fault Handling	
live migration	dirty page rate	mig. time	downtime	transferred data	deadline	available bw	flow path

vironment supporting SDN-enabled cloud computing. It also provides the mechanism of auto-scaling of VNF and automatic load balancing through different SFCs. Table 3.7 illustrates some parameters supported by the extended version.

Fig. 3.13 illustrates the implemented components regarding live VM migration: *Migration* Class contains all the information regarding one migration task, such as the migrating VM/VNF (RAM size, dirty page rate, data compression ratio, remaining dirty pages), source and destination hosts, the scheduling window, assigned routings, the current phase of live migration, etc. The *MigrationPlanner* takes the current migration tasks in the waiting queue as input and calculates the sequence of multiple migrations and sends the result to the *MigrationScheduler*. If there are additional migration tasks arrive, it will calculate the sequence again based on the on-going and waiting to schedule migration tasks. *MigrationScheduler* takes charge of starting the migration task based on the output of the *MigrationPlanner*. When a migration complete, the *SDNDataCenter* will send the *event 7* (Table 3.6) to trigger the scheduler to start new migrations according to the remaining scheduling planning. With the events of live migration, the Class *SDNDataCenter* emulate the live migration in every phase as shown in Fig. 1.5: (1) checking the availability of network and computing resources; (2) sending the memory and dirty pages to the destination hosts iteratively; (3) checking the current downtime and iterative rounds with the thresholds; (4) pausing the workload processing and refusing the new packets arrive at the instance; (5) resuming the workload processing and rerouting the network packets to the new location. (6) noticing the on-line scheduler about the completion; (7) if selected, storing the statistic for every migration step. The *NetworkOperatingSystem* calculates the routings and allocated bandwidth to the migration flows based on the selected network routing policy and bandwidth sharing scheme. It simulates the network packet transmission based on the bandwidth and delay along the path, packs and unpacks the contents from and to the compute nodes.

3.7 Summary

We established the mathematical model of block live migration to have a better understanding of the static downtime adjustment algorithm in OpenStack, as well as the parallel and sequential migration cost in the same network path. For the downtime adjustment algorithm, we should dynamically set the downtime configuration (maximum downtime, adjustment steps, and delays) to achieve the optimal migration performance. When non-network overheads, such as pre- and post-migration workloads, constitute a large portion of total migration time, parallel migration should be chosen to reduce the response time, downtime, and the total evacuation time of multiple migrations in the same path. We also evaluated the impacts of SDN scheduling update rate on live migration performance. The result suggests that a high update rate leads to a large TCP/UDP packet loss which will affect the migration performance.

From the QoS perspective, we investigated the response time pattern of client and server live migrations with pre-copy, hybrid post-copy, and auto-convergence strategies. For memory-intensive VM, as the pre-copy migration cannot finish in a reasonable time, we should choose hybrid post-copy to perform a successful migration if the network environment is stable. Otherwise, we could perform the auto-convergence feature during the pre-copy migration. However, the auto-convergence dramatically influences the application response time, i.e., requests are timed out because of the CPU slowdown.

Moreover, for the pre-copy migration of server VM, as the migration and application traffic flows contend with each other, the worst-case response time will not just occur after the downtime but during the migration. Moreover, the models and parameters in the chapter are compatible with other optimization technologies for single live VM migration [8, 68, 170, 185] and algorithms of multiple migrations [26, 28–30, 90, 150] because these work focus on different optimization factors. Therefore, the results in the chapter still stand and can benefit other optimization methods and algorithms.

Chapter 4
Concurrency-Aware Live Migration Management

By neglecting the resource dependency among potential migration requests, the existing solutions of dynamic resource management can result in the Quality of Service (QoS) degradation and Service Level Agreement (SLA) violations during the migration schedule. Therefore, it is essential to integrate both single and multiple migration overheads into VM reallocation planning. In this chapter, we propose a concurrency-aware multiple migration selector that operates based on the maximal cliques and independent sets of the resource dependency graph of multiple migration requests. Our proposed method can be integrated with existing dynamic resource management policies. The experimental results demonstrate that our solution efficiently minimizes migration interference and shortens the convergence time of reallocation by maximizing the multiple migration performance while achieving the objective of dynamic resource management.

4.1 Introduction

Dynamic resource management such as load balancing and energy-saving policies can request multiple migrations when the algorithms are triggered periodically. There exist notable research efforts in dynamic resource management that alleviate single migration overheads, such as single migration time and co-location interference while selecting the potential VMs and migration destinations. As a resource-intensive operation, live migration consumes both computing and networking resources when transmitting

This chapter is derived from:

- **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "CAMIG: Concurrency-Aware Live Migration Management of Multiple Virtual Machines in SDN-enabled Clouds", *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, (minor revision, Aug 2021).

the memory dirty pages from the source to the destination host. It puts stress on both the migrating services and other services in the cloud data centers. Thus, it is crucial to minimize migration interference during dynamic resource management. There are continuous efforts to take migration overheads into consideration during the dynamic resource management [90, 136, 137, 141].

Currently, most migration cost models consider overheads of single migration [84, 99], such as migration time (single execution time), downtime, transferred data with respect to the size of memory, dirty page rate, data compression rate and available bandwidth while allowing multiple migrations in dynamic resource management. For the migration selection, the existing resource management algorithms utilize the cost model of single migration to minimize the overheads. Then, with the migration requests generated as the input, multiple migration planning and scheduling algorithms [28–30] decide the sequence of these migration requests to achieve the maximal migration performance.

There are obvious gaps regarding the multiple migration performance between the existing dynamic resource management policies, the migration cost model and the multiple migration scheduling. The total migration time, the time interval between the start of the first migration and the end of the last migration, is the convergence time for the resource management solution. Overall, the real-time demands for live migration should be met by improving the performance in total migration time. For example, with the nature of highly variable workloads, SLA violations will occur as the resource demand surpasses the provisioned amount. In this case, a faster live migration convergence equals to less SLA violations.

Resource dependency between two migrations, such as sharing source and destination hosts or network paths, can largely affect the performance of multiple migration scheduling. With the network as a bottleneck, two resource-dependent migrations can only be scheduled sequentially, while independent ones scheduled concurrently [29, 30]. If large amount of resource dependencies among migrations are generated by dynamic resource management, the performance of multiple migration scheduling will suffer a significant degradation. Since single migration overheads are only related to one migration, it is critical to consider multiple migration overheads in order to generate migration requests with less resource dependencies.

Therefore, this chapter incorporates the resource dependency of multiple migrations into the cost model to bridge the gaps. Based on the maximal cliques and independent sets of the dependency graph of potential migrations, we propose a concurrency-aware migration (CAMIG) selection strategy for migrating VMs and destination hosts of the dynamic resource management. The **contributions** of this chapter are summarized as follows:

- We propose and model the multiple migration selection problem to minimize interference due to resource dependency among multiple migrations while achieving the objective of dynamic resource management.
- We introduce the resource dependency graph to model migration concurrency.
- We propose a flexible concurrency-aware migration selection strategy for dynamic resource management.
- We conduct extensive experiments in an event-driven simulation to show the performance improvement in terms of total migration time in correspondence with resource management objective.

The rest of the chapter is organized as follows. The existing work in migration cost management and multiple migration scheduling are reviewed in Section 4.2. The system framework and the migration overheads are discussed in Section 4.3. The problem model is described in Section 4.4. In Section 4.5, we propose the concurrency-aware migration selection algorithm. In Section 4.6, we compare our proposed algorithm with other dynamic resource management algorithms in both load-balancing and energy-saving scenarios. Finally, we summarize the chapter in Section 4.7.

4.2 Related Work

Many dynamic resource management solutions utilize live migration as a tool to achieve objectives, such as load-balancing [27, 135–137], energy efficiency [186, 187], delay maintenance [188, 189], communication cost [190]. Among these solutions, some resource management algorithms consider a linear model of the total migration overheads as the sum of individual migration overhead [27, 90, 135–137, 141, 187]. However, existing research only considers the objectives of resource management while neglecting the multiple migration overheads and migration scheduling performance. Generally, during dynamic resource management, there are three steps to generate migration requests: (1) source host selection; (2) VM selection; and (3) destination host selection. The overhead or interference model of single migration [84, 99] is considered during the VM and destination selections.

For the VM and destination host selection, many dynamic resource management policies consider single migration overheads in terms of the memory size of migrating VM, single migration time, and the impact of one migration on other VMs located in the source or destination host, such as CPU, bandwidth of host network interface, and application bandwidth. In the load balancing scenario, Verma et al. [136] estimated the migration cost based on the deduction of application throughput. It selects the smallest memory size VMs from the over-utilized hosts and assigns them to the under-utilized hosts in the First Fit Decreasing (FFD) order. Singh et al. [135] proposed a multi-layer virtualization system HARMONY. It migrates VMs and data from hotspots on servers, network devices, and storage nodes. The load balancing algorithm is a variant of Toyoda multi-dimensional knapsack problem based on the evenness indicator Extended Vector Product (EVP). It considers the single live migration impact on application performance based on CPU congestion and network overheads. Wood et al. [137] proposed the load balancing algorithm Sandpiper that selects the smallest memory size VM from one of the most overloaded hosts to minimize the migration overheads. Mann et al. [27] focused on the VM and destination selection for the load balance of application network flows by considering the single migration cost model based on the dirty page rate, memory size, and available bandwidth. In the energy-saving scenario, Xiao et al. [139] investigated dynamic resource allocation through live migration. The proposed algorithm avoids the over-subscription while satisfying the resource needs of all VMs based on exponentially weighted moving average to predict the future loads. It also minimizes the physical machines regarding the energy consumption. Similarly, LR-MMT [141] focused on energy saving with local regression based on history utilization to avoid over-subscription. It chooses the least memory size VM from the over-utilized host and the most-energy saving destination. Wu et al. [187] also studied the same problem of maximizing the power saving through VM consolidation by limiting individual migration costs. With the input of candidate VMs and destinations provided by other resource management algorithms, iAware [90] is a migration selector minimizing the single migration cost in terms of single migration execution time and host co-location interference. It considers dirty page rate, memory size, and available bandwidth for the single migration time. They argue that co-location interference from a single live migration on other VMs in the destination host in terms of performance degradation is linear to the number of VMs hosted by a physical machine in Xen. However, it only considers one-by-one migration scheduling.

Taking the list of migration tasks generated by dynamic resource policies as input, the migration scheduling algorithms focus on minimizing the migration time by efficiently scheduling them. To find a possible sequence of migration tasks, one-by-one scheduling [28] focused on avoiding the deadlock on the available resource of physical hosts. The multiple migration planning and scheduling algorithms [29, 30] focused on the migration performance in terms of minimizing the total migration time by scheduling given migration tasks concurrently when necessary.

However, in existing studies, dynamic resource management and multiple migration scheduling have been considered separately. Current works only minimize the sum of single migration overheads. With the requirement of several live migrations to achieve the objective of dynamic resource management, the concurrency or resource-dependency in networking and computing resources among potential VM migrations in the selection process of VMs and destination hosts has been neglected.

4.3 Live Migration in Dynamic Resource Management

4.3.1 System Overview

By integrating Software-Defined Networks (SDN) [42], the SDN-enabled cloud data centers have a centralized solution for the monitoring, planning, and scheduling of virtualized computing and networking resources [57]. As shown in Fig. 4.1, the dynamic resource manager is integrated with migration selector and multiple migration scheduler

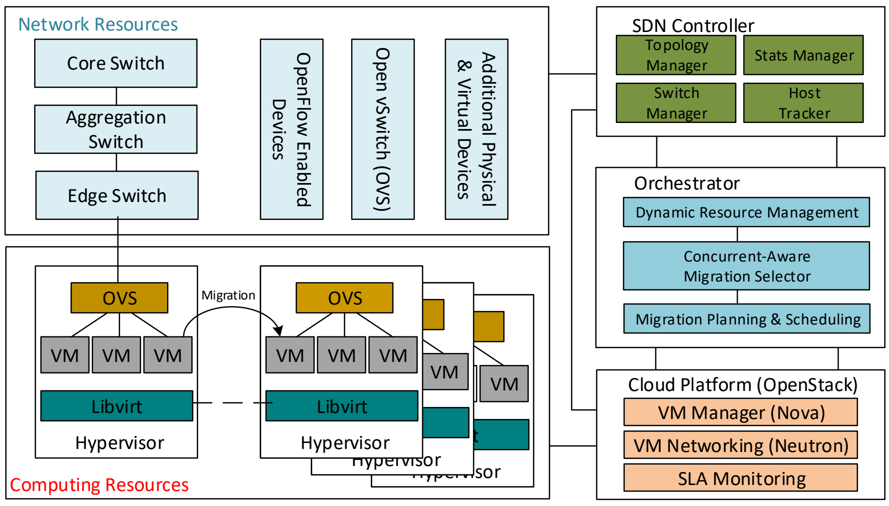


Figure 4.1: System Overview

based on both monitoring computing resource and network resources. VMs are hosted on physical machines to provide various cloud services. The computing resources are controlled by VM Manager (VMM), such as OpenStack Nova, while the networking resources (such as available bandwidth and routing) are managed by the SDN controller and VM Networking Service, such as OpenStack Neutron, in a centralized way. The SDN controller can dynamically manage the routing for elephant flows (such as flows of live migrations) to avoid the congestion and alleviate the impact on cloud services. We can predict the cost of live migration by the available bandwidth between the source and destination hosts.

Driven by the QoS and SLA or other parameters, such as energy and communication cost, the dynamic resource manager migrates VMs from one host to another through live migration based on the current and historical data. After the dynamic resource management [186–188, 190] generates a list of VM migrations for the new placement, the multiple migration scheduler [28–30] plan and schedule the migration tasks. It will schedule migrations to be conducted concurrently when they are resource independent and sequentially when dependent.

4.3.2 Single Migration Cost Model

Compared to the live block migration, we simplifies the live migration model by considering the iterative memory copying of pre-copy migration with storage migration. To better understand the impact of multiple migrations on performance in dynamic resource management settings, we introduce the mathematical model of a single live migration. Live migration can be categorized into two types: post-copy and pre-copy migration. Since the pre-copy migration is the most widely used approach in hypervisors (KVM, VMWare, Xen, etc), we consider it as the base model. During the pre-copy live migration for VMs or Containers, the hypervisor or the Checkpoint/Restore agent in the userspace (CRIU) [60] iteratively copies the dirty memory pages in the previous transmission interval from the source host to the destination host.

The most important aspect of single migration overheads is the migration time or the single migration execution time. According to the live migration process [8], the pre-copy live migration consists of eight phases (see Fig. 1.5): pre-migration, initialization, reservation, iterative memory copy, stop-and-copy, commitment, and post-migration. Thus, live migration consumes both computing resources (pre-/post-migration overheads) and networking resources (memory copy and dirty page transmission) as shown in Chapter 3. The total single migration time T_{mig} can be categorized into three parts: pre-migration computing overheads, memory-copy networking overheads, and post-migration computing overheads:

$$T_{mig} = T_{pre} + T_{mem} + T_{post} \tag{4.1}$$

Based on the iterative pre-copy illustrated in Fig. 1.5, the migration performance in terms of memory-copy can be represented as:

$$T_{mem} = \frac{\rho \cdot Mem}{L} \cdot \frac{1 - \sigma^{i+1}}{1 - \sigma} \tag{4.2}$$

$$i = \min\left(\left\lceil \log_{\sigma} \frac{V_{thd}}{M} \right\rceil, \Theta\right) \tag{4.3}$$

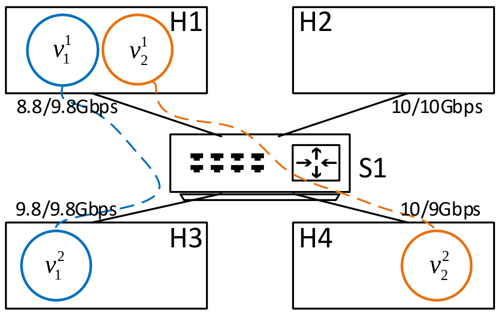
where the ratio $\sigma = \rho \cdot R/L$, ρ is the compression rate of dirty memory, *Mem* is memory size, L is available bandwidth, R is dirty page rate, i is the total migration round, Θ denotes the maximum allowed number of iteration rounds, $V_{thd} = T_{dthd} \cdot L_{i-1}$ is the remaining dirty pages need to be transferred in the stop-and-copy phase, and T_{dthd} is the configured downtime threshold.

4.3.3 Resource Dependency

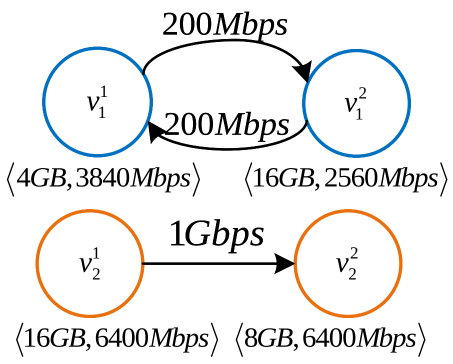
Not only the overheads of the single migration but also resource dependencies among multiple migrations can heavily affect the performance of dynamic resource management.

For dynamic resource management policies, there are mainly three selection steps: (1) selection of source physical hosts that need to be adjusted based on the management objective; (2) selection of VM(s) which need to be migrated from the selected host(s); and (3) selection of destination hosts of live VM migrations among potential candidates. With the input of candidate VMs and available destination hosts, different combinations of source and destination can achieve the same objective of dynamic resource management. However, there is a huge difference between these combinations in the scheduling performance of multiple migrations due to the resource dependencies among migrations. If sharing the same source or destination hosts, or part of the network routing, two live migrations are resource-dependent.

Two resource-dependent migrations can not be scheduled at the same time [29]. Because, according to equation (4.2), larger bandwidth allocation means a smaller migration execution time and downtime. Thus, the networking resources are the bottlenecks which need to be optimized during the multiple migrations. For example, we have a number of migrations partially or entirely sharing network paths. Based on equation (4.2), if scheduled at the same time, experimental results in Chapter 3 show that the total migration time will be more than the sum of single execution time. Thus, sequential scheduling of dependent migrations is the most efficient way to optimize the migration performance as shown in Chapter 3. Meanwhile, migrations which are resource-independent can be scheduled concurrently to reduce the total migration time. Therefore, it is essential to exclusively allocate one network path to only one migration until



(a) Initial Placement



(b) Virtual Connections between VMs with Memory Size and Dirty Page Rate

Figure 4.2: Scenario of Resource Dependencies during Migration Selections

it is finished to achieve the optimal total migration time, average execution time, and downtime.

4.3.4 Illustrative Example

Fig. 4.2(a) shows the initial VM placement of the illustrative example along with the resource dependency among possible migration selections. Fig. 4.2(b) illustrates the virtual connections between VMs and the memory size (GB) and dirty page rate (Mbps) for each. Moreover, the threshold of iteration rounds is 30 and downtime threshold is 0.5 seconds. The objective of the management policy is to reduce the communication cost by VM consolidation. There are several potential migration combinations which can fulfill the objective: M1: $v_1^1 : H1 \rightarrow H3$ and $v_2^1 : H1 \rightarrow H4$; M2: $v_1^1 : H1 \rightarrow H3$ and $v_2^2 : H4 \rightarrow H1$; M3: $v_1^2 : H3 \rightarrow H1$ and $v_2^1 : H1 \rightarrow H4$; and M4: $v_1^2 : H3 \rightarrow H1$ and $v_2^2 : H4 \rightarrow H1$. We can schedule two resource-independent migrations concurrently (M2 and M3). On the other hand, one migration can only be scheduled in sequence after the completion of another dependent migration (M1 and M4).

We used Mininet [191] to emulate the iterative network transmission of the live migration. The execution time for each potential migration of v_1^1, v_1^2, v_2^1, and v_2^2 based on the available bandwidth is 6.2791, 15.0889, 29.1980, and 12.5143 seconds, respectively. The total migration time of combination M1-M4 is 34.8858, 12.4334, 28.4711, and 27.6032 seconds. Moreover, when the service network and migration (control) network are running separately [103], the available bandwidth for each live migration is the same (10 Gbps). Then, the total migration time of four different combinations M1-M4 is 28.1936, 12.1227, 22.6056, and 26.8893 seconds, respectively. Comparing M2 with M1 and M4, since there is no resource-dependent migration in M2, the total migration time is significantly shorter. Comparing M2 with M3, although there is no network resource sharing in both combinations, the single live migration overheads of M2 is smaller due to the memory size, dirty page rate, and the available bandwidth. In summary, although all the potential combinations can achieve the desired objective, the scheduling performance of multiple migrations varies considerably. Therefore, it is essential to minimize both resource dependencies among migration requests and single live migration overheads during dynamic resource management.

4.4 Problem Modeling

In this section, we model the problem of multiple migration selection to minimize the migration dependency while achieving the objective of dynamic resource management as a Mixed Integer Programming (MIP) problem.

In the model, H is the set of all candidate destination physical hosts $h \in H$ while N denotes the set of candidate VMs $i \in N$ for the migration. H_i is the set of candidate hosts for VM i. Let binary variable $y_{(i,h)} \in \{1, 0\}$ indicate both initial and final placement of VM i in host h. When the VM i is in the initial host p_i, $y_{(i,p_i)} = 1$. When VM i is in the host h in the final placement, $y_{(i,h)} = 1$. Otherwise, $y_{(i,h)} = 0$. Let the binary variable $x_{(i,h)} \in \{1, 0\}$ indicate whether VM i is in the host i in the final placement. In other words, if VM i is migrated to host h, then $x_{(i,h)} = 1$ and $h! = p_i$. If VM i is not migrated, then $x_{(i,h)} = 1$ and $h = p_i$. Otherwise, $x_{(i,h)} = 0$ which indicates that VM i is not in host h in the final placement determined by the dynamic resource management policy.

To generalize the problem, we can omit the VM index i for $h \in H_i$ by adding extra constraints to $x_{(i,h)}$ when some destination hosts are not available for the specific VM i:

$$x_{(i,\overline{h}_i)} = 0 \quad \forall \overline{h}_i \in \overline{H}_i = H \backslash H_i \tag{4.4}$$

where $\overline{h}_i$ indicates the unavailable host for VM i.

The migration execution time t_i^h of $x_{(i,h)} = 1, h! = p_i$ can be calculated according to equations (4.1)-(4.3). Furthermore, we normalize the migration execution time based on the largest and smallest execution time among the different source and destination pairs for every VMs.

As there can be only one destination and the VM must be allocated in one and only one host at the same time, we add the following constraints to the binary variable $x_{(i,h)}$:

$$\sum_{h \in H} x_{(i,h)} = 1 \quad \forall i \in N \tag{4.5}$$

The VM i can only be migrated from source host of the initial placement $h_s = p_i$ where $y_{(i,p_i)} = 1$ to the destination host of the final placement h_d that $y_{(i,h_d)} = 1$, $x_{(i,h_d)} = 1$ and $x_{(i,p_i)} = 0$ or not be migrated at all $x_{(i,h_d)} = 1, h_d = p_i$. Thus, we have the constraints expression as follows:

$$x_{(i,h)} - y_{(i,h)} \leq 0 \quad \forall i, h \in N \times H \tag{4.6}$$

Constraints of the placement binary variable $y_{(i,h)}$ are:

$$1 \leq \sum_{h \in H} y_{(i,h)} \leq 2 \quad \forall i \in N \tag{4.7}$$

where $\sum_{h \in H} y_{(i,h)} = 2$, when VM i is migrated to other host in the final placement. $\sum_{h \in H} y_{(i,h)} = 1$, when VM i is still in host p_i in the final placement.

Let $z_{(i,j,h_1,h_2)}$ denote the binary variable indicating whether VM i and j are migrated to destination h_1 and h_2:

$$z_{(i,j,h_1,h_2)} \in \{1,0\} \quad \forall i,j \in N, h_1, h_2 \in H \tag{4.8}$$

where $z_{(i,j,h1,h2)} = 1$, if $y_{(i,h1)} = 1$, $y_{(j,h2)} = 1$ and $p_i! = h1$, $p_j! = h2$. Otherwise, $z_{(i,j,h1,h2)} = 0$.

There is a resource dependency graph G_{dep} for all possible migrations. Let $v_{s,d}$ denote a migration with source host s and destination host d. If node $v_{p_i,h1}$ and $v_{p_j,h2}$ are connected in graph G_{dep}, then edge $e_{(i,j,h1,h2)} = 1$. This indicates that potential migrations of VM i from host p_i to $h1$ and VM j from host p_j to $h2$ are resource-dependent which can only be scheduled in a sequential manner. Thus, the resource dependency between two potential migrations can be represented as:

$$e_{(i,j,h1,h2)} \cdot z_{(i,j,h1,h2)} \tag{4.9}$$

Let O_{init} and O_{tar} denote the initial score and target score of dynamic resource management and ε represent the tolerant value for accepted range. Let $O(x_{(i,h)})$ denote the objective score achieved after all migrations based on $x_{(i,h)}$ indicator. Thus, the constraints of final placement for dynamic resource management can be represented as:

$$\left| O\left(x_{(i,h)}\right) - O_{tar} \right| \leq \varepsilon \quad \forall (i,h) \in N \times H \tag{4.10}$$

In practice, we can replace (4.10) for a specific placement score function. For example, in load balancing policies, let w_i and w_j denote the load of VM i and j. We can represent the constraints of dynamic resource target for the final placement as:

$$\left| \sum_{i \in N} x_{(i,h_1)} \cdot w_i - \sum_{j \in N} x_{(i,h_2)} \cdot w_j \right| \leq \varepsilon^{'} \tag{4.11}$$

where $\forall (h_1, h_2) \in H \times H : h_1 \neq h_2$ and $\varepsilon^{'}$ is the tolerant value among the physical hosts.

In addition, let $C^h_{(Mem,Core,Disk,Work)} = (1,1,1,1)$ denote the normalized computing resource capacity of physical host h for memory *Mem*, CPU *Core*, storage disk *Disk*, and total workload *Work*. Therefore, the constraints of computing resources, such as workload, can be represented by:

$$\sum_{i \in N} x_{(i,h)} \cdot w_i \leq C^h_{(Work)} \quad \forall h \in H \tag{4.12}$$

The single and multiple migration overheads, $Inter_{single}$ and $Inter_{multi}$, are calculated as:

$$Inter_{single} = \sum_{i \in N} \left(y_{(i,p_i)} - x_{(i,p_i)} \right) \cdot t_i^h \tag{4.13}$$

$$Inter_{multi} = \sum_{i,j \in N, h1, h2 \in H} \left(t_i^{h1} + t_j^{h2} \right) \cdot e \cdot z \tag{4.14}$$

where e and z omit the subscripts for a concise equation.

Therefore, the objective of the problem in terms of minimizing single migration overheads and resource dependencies among multiple migration requests can be formulated as:

$$\min(Inter_{single} + Inter_{multi}) \tag{4.15}$$

subject to constraints (4.4) - (4.12).

The objective function contains two parts: the first objective is for the sum of single migration overhead, where t_i^h indicates single migration time of VM i from source host p_i to destination host h. Note that, although we only model the migration time in our model, it can be extended to other interference, such as CPU congestions, bandwidth overheads on other applications, and the number of co-located VMs in the destination host. The second part is the multiple migration overheads during multiple migration scheduling. In other words, it indicates how much overheads due to resource dependencies happened. The fewer dependencies in migration requests with less individual overheads, the greater the possibility of larger concurrent migration groups during scheduling resulting in a shorter total migration time.

4.5 Concurrency-Aware Selection

Solving the MIP model in Equation (4.15) is NP-hard, it is not practical to use MIP solver to get the solution. In this section, we introduce the Concurrency-Aware Migration (CAMIG) selection algorithm for minimizing the resource dependencies and overheads among VM migrations during dynamic resource management. Based on the three selection steps of resource management policy, CAMIG has the flexibility to integrate with existing policies. Provided that the VMs are selected by the policy, CAMIG can select migration destinations to minimize the resource dependency. Furthermore, if only the management objective and source host selection criteria are given, CAMIG selects both VMs and migration destinations.

The rationale behind CAMIG is to select the migration with the least resource dependency and single migration overheads in each round with the currently selected migrations and minimize the dependency for the future one based on the maximal cliques and independent sets of the resource dependency graph. The graph theory concepts, such as maximal cliques and independent sets, are explained in Section 4.5.2. There are mainly three steps in the algorithm: (1) build the migration dependency graph; (2) get all maximal cliques and independent sets of a migration from the dependency graph; and (3) calculate the single migration interference and migration concurrency metric (MIGC) of all candidate migrations.

4.5.1 Migration Dependency Graph Build

We first explain how to generate the resource dependency graph G_{dep} based on the potential migrating VMs and destinations. For the undirected graph $G_{dep} = (V, E)$, let v ($v \in V$) be the source-destination pair (src-dst) node or vertex representing one potential migration. Migrations with same src-dst node are categorized in list $M(v_{sd})$. Let $e(v, u) \in E$ be the dependency between two migrations with src-dst node v and u. As shown in Algorithm 1, with the input of potential migrating VMs and corresponding destination candidates H_i, we first add src-dst nodes and classify potential migrations into the corresponding node in $M(v_{sd})$. Then, we add edges into G_{dep} based on the source and destination of each node. Fig. 4.3 demonstrates an illustrative example of

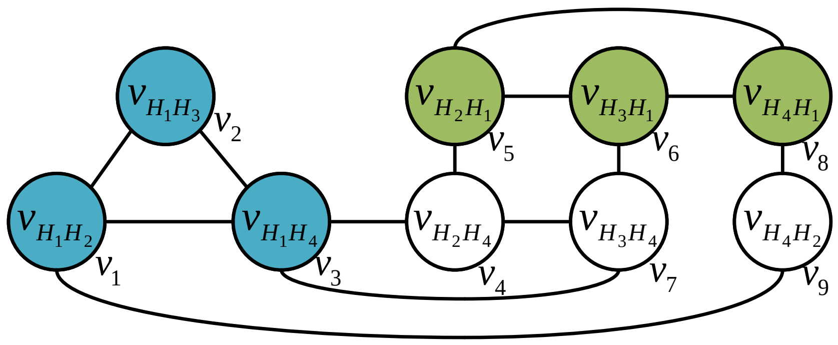


Figure 4.3: A Resource Dependency Graph with two of its Maximal Cliques Marked by color

resource dependency graph based on a given list of potential migrations (v_1 to v_9) in a specific dynamic resource management which involves 9 src-dst pairs in the same physical network topology shown in Fig. 4.2(a) (four hosts connected through one switch). Each vertex $v_{H_sH_d}$ indicates the pair of source and destination host for a group of potential migrations. For the sake of conciseness, we use v_1 to v_9 to represent node $v_{H_1H_2}$ to $v_{H_4H_2}$.

Algorithm 1: Create G_{dep} and v_{sd} queues

```
Input: potential VM $i \in N$, Destinations $\{H_i\}$
Result: migration depGraph $G_{dep}$, $\{M(v_{sd})\}$
1  foreach $i \in N$ do
2      $s \leftarrow p_i$;
3      foreach $d \in H_i$ do
4          AddNode ($G_{dep}, v_{sd}$);
5          $M(v_{sd}) \leftarrow M(v_{sd}) \cup i$;
6  foreach $v \in V(G_{dep})$ do
7      foreach $u \in V(G_{dep})$ do
8          if $v != u$ then
9              if IsDependent $(u,v)$ then
10                 AddEdge ($G_{dep}, (u, v)$);
11 return $G_{dep}$ , $\{M(v_{sd})\}$
```

Regardless of the number of potential migrations, the scale of G_{dep} only depends on the source and destination hosts involved. Given a list of migrations $M = \{m_0, m_1, ..., m_n\}$, the dependency graph $G(M)$ of M can be constructed as $G(M) = (V, E)$. As migrations with the same source and destination are always resource-dependent, we categorize migrations into different lists of src-dst pair v. Then, all migrations can be represented as $\{M(v_{sd})\} = \{M(v_0), ..., M(v_{|V|})\}$. The size of node $|V|$ in the migration dependency graph will be the total combination of source and destination hosts. Through this preprocessing, the total nodes of G_{dep} can be reduced from as many as the potential migrations $|M|$ to the migration pair participated $|V|$. Therefore, the upper-bound of total nodes in graph $G_{dep}(M)$ is $|H_{src}| \cdot |H_{dst}|$. H_{src} and H_{dst} are the number of potential source and destination hosts, respectively.

Note that the dependency graph supports the multiple routing transmission and dynamic migration routing based on the current network status. In certain data center networks, multi-path transmission and multiple network interfaces of physical hosts are supported. Thus, the vertex v_{sd}^{P} in G_{dep} can be extended to indicate the network paths P_{sd} for migrations from the specified network interfaces set s of source host to interfaces set d of destination host. Let $u(P)$ indicate the available bandwidth of network paths P. Given two pairs of src-dst interfaces set (s_j, d_j) and (s_k, d_k) and corresponding network paths P_j and P_k, two vertices v_j and v_k are resource-independent, when statement (4.16) are true and $s_k \cap s_j = \varnothing$ and $d_k \cap d_j = \varnothing$:

$$\begin{aligned} u\left(P_j\right) - u\left(P_j \cap P_k\right) \geq \min\left(u\left(P_j\right), NC_s^j, NC_d^j\right) \wedge \\ u\left(P_k\right) - u\left(P_k \cap P_j\right) \geq \min\left(u\left(P_k\right), NC_s^k, NC_d^k\right) \end{aligned} \tag{4.16}$$

where (NC_s^j, NC_d^j) and (NC_s^k, NC_d^k) indicate the network capacity of interface set and $u\left(P_j\right)$ and $u\left(P_k\right)$ indicate the available bandwidth of network paths. Otherwise, the two vertices are resource-dependent. The upper bound of total nodes in G_{dep} is the total number of P_{sd}.

$$\{C\}:\{\{v_3,v_1,v_2\},\{v_3,v_4,v_7\},\{v_5,v_8,v_6\},\{v_5,v_4\},\{v_6,v_7\},\{v_8,v_9\},\{v_9,v_1\}\}$$

$$\begin{aligned}\{I\}:\{&\{v_1,v_4,v_8\},\{v_1,v_4,v_6\},\{v_1,v_7,v_5\},\{v_1,v_7,v_8\},\\ &\{v_2,v_4,v_6,v_9\},\{v_2,v_4,v_8\},\{v_2,v_7,v_5,v_9\},\{v_2,v_7,v_8\},\\ &\{v_3,v_5,v_9\},\{v_3,v_6,v_9\},\{v_3,v_8\}\}\end{aligned}$$

Figure 4.4: All Maximal Cliques and MISs of G_{dep} in Fig. 4.3

4.5.2 Maximal Cliques and Independent Sets

Before discussing how to get maximal cliques and maximal independent sets (MISs) which include a certain node v, we first review some basic concepts, such as clique, independent set, and degeneracy. A clique is a subset of vertices of an undirected graph G such that every two distinct vertices in the subset are adjacent [192]. The maximal clique is a clique that cannot be extended by including one more adjacent vertex. On the other hand, an independent set of a graph G is the opposite of a clique that no two nodes in the set are adjacent. Fig. 4.4 shows all maximal cliques and MISs of the G_{dep} (Fig. 4.3). For example, $\{v_3, v_1, v_2\}$ is one of its maximal cliques and $\{v_2, v_7, v_5, v_9\}$ is one of its maximal independent sets. The problems of finding all maximal independent sets and cliques are complementary and NP-hard [192, 193]. Finding all maximal independent sets of a graph is equal to finding all maximal cliques of its complement graph [194]. As a robust metric to indicate graph density or spareness, degeneracy of a graph G is the smallest value d such that every nonempty subgraph of G contains a vertex of degree at most d [195].

A clique of G_{dep} is a set of src-dst nodes, where migrations with these nodes can not be scheduled at the same time. In contrast, the migrations from the src-dst nodes within an independent set can be scheduled concurrently. To check and evaluate the resource dependency or concurrency of each migration with src-dst pair node v, we need to generate all maximal cliques $\{C^v\}$ and MISs $\{I^v\}$ of G_{dep} including node v . Let

Algorithm 2: Get All Maximal Independent Sets of Node v in G_{dep}

```
Input: $G, \bar{G}, v$, node neighbors in $\bar{G}$ $\{\bar{G}_N(n)\}, n \in V$
Result: All MISs $\{I^v\}$ of node $v$, $v \in V(G_{dep})$
1  Function CLIQUES ($\bar{G}$, cand):
2      $n \leftarrow$ GETNodeMaxDegree ($\bar{G}$);
3      foreach $m \in cand - \bar{G}_N(n)$ do
4          $del(m, cand)$;
5          $I \leftarrow I \cup \{m\}$;
6          if $\bar{G} \cap \bar{G}_N(m) = \emptyset$ then
7              $\{I^v\} \leftarrow \{I^v\} \cup I$;
8          else
9              if $cand \cap adj[m]! = \emptyset$ then
10                 CLIQUES ($\bar{G} \cap \bar{G}_N(m), cand \cap \bar{G}_N(m)$);
11         $del(m, I)$;
12 End Function
13 $I^v \leftarrow \emptyset$; $I \leftarrow \{v\}$;
14 $cand \leftarrow cand - G_N(v) - \{v\}$;
15 $V(G) \leftarrow V(G) - G_N(v) - \{v\}$;
16 return CLIQUES ($\bar{G}$, cand);
```

$\{C\}$ and $\{I\}$ be all maximal cliques and all maximal independent sets of G_{dep}, where $C \in \{C\}$ and $I \in \{I\}$ is one of the maximal cliques and MISs. Let $C^v \in \{C\}$ and $I^v \in \{I\}$ denote one of the maximal cliques and independent sets including node v. Then, $\{C^v\} \subseteq \{C\}$ and $\{I^v\} \subseteq \{I\}$.

We propose an algorithm for listing $\{C^v\}$ and $\{I^v\}$ based on $\{C\}$ of dependency graph. For getting all maximal cliques $\{C\}$ of a graph, the general-purpose algorithms for listing all maximal cliques [194, 196] based on Bron-Kerbosch algorithm [192] take exponential time due to the maximum possible number of cliques. These general-purpose algorithms are not sensitive to the density of a graph. Therefore, parametrized by degeneracy, we use a variant algorithm Bron-Kerbosch Degeneracy [197] to generate all maximal cliques of the original resource-dependency graph without duplication. All maximal cliques are generated in the tree-like structure by employing the pruning methods with pivoting to allow quick backtrack during the search. Based on the Bron-Kerbosch algorithm with pivoting, the Bron-Kerbosch Degeneracy uses a degeneracy ordering to order the sequence of recursive calls without pivoting at the outer level of the original

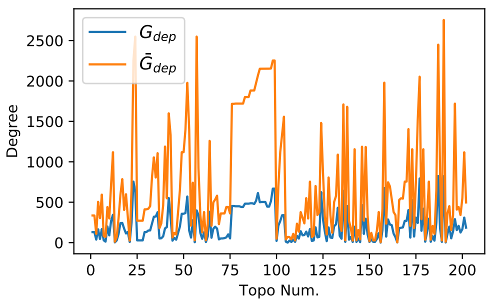
Bron-Kerbosch algorithm [197]. Applied to a n-vertex graph with d degeneracy, it lists all maximal cliques in time $O\left(dn3^{d/3}\right)$.

4.5.3 Dependency Graph Properties

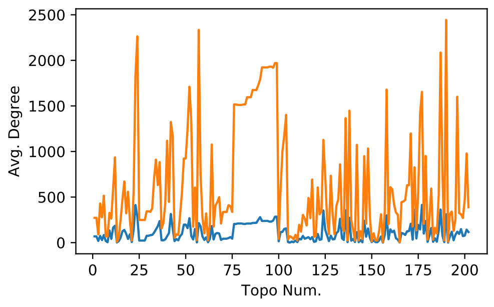
In this section, we analyze the properties of dependency graphs in different data center networks in terms of degree and density (degeneracy). The evaluation results demonstrate the rationale of using Bron-Kerbosch Degeneracy and cliques-based algorithm on resource dependency graphs. Besides the FatTree topology used in the experimental evaluation, we evaluated the resource dependency graph of network topologies in the WAN environment for inter-datacenter network. We investigated total of 202 network topologies in the Internet Zoo topology [39]. To maximize the complexity, we generate the dependency graph based on all source and destination combinations. The graph maximum degree, average degree, and degeneracy of the original dependency graph G_{dep} and its complement graph $\bar{G}_{dep}$ are studied. The maximum degree of graph and average node degree of G_{dep} and $\bar{G}_{dep}$ for each topology are shown in Fig. 4.5(a) and 4.5(b). Fig. 4.5(c) demonstrates the degeneracy against the total src-dst node number. As the number of total node in the graph grows, the density of the complement graph grows much faster than the original dependency graph. For all network topologies, the maximum degree of graph, average node degree, and degeneracy of graph from $\bar{G}_{dep}$ are 2.69, 5.01, and 4.34 times of G_{dep}.

It is known that all maximal cliques can be calculated in a total time proportional to the maximum number of cliques in an n-vertex graph [194]. In other words, each clique is listed in polynomial time for all maximal cliques [193]. CLIQUES algorithms [194, 196] based on Bron-Kerbosch are optimal by considering only vertex. However, with an exponential growth of the maximum possible number of cliques, the running time of these algorithms for all maximal cliques is also exponential [197]. The worst-case running time of CLIQUES is $O\left(3^{n/3}\right)$ [196]. The upper bound of all maximal cliques/independent sets of a Graph G is $3^{n/3}$.

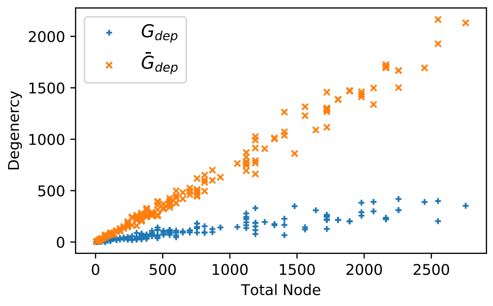
For an n-vertex graph with degeneracy d, by introducing the sequence ordering based on degeneracy, Bron-Kerbosch Degeneracy algorithm [197] can list all maximal



(a) Maximum Degree



(b) Avg. Degree



(c) Degeneracy

Figure 4.5: Dependency Graph Properties of WAN

cliques in time $O\left(dn3^{d/3}\right)$. $(n-d)\,3^{d/3}$, $n \geq d+3$ is the upper bound of all maximal cliques number. Therefore, compared to other general purpose algorithms, it can list all maximal cliques in spare graphs (G_{dep}) in near-optimal time (Fig. 4.5(c)). If one set of vertices I is maximal independent set in one graph, it is a maximal clique in the complement of graph. Since getting all maximal independent sets of a graph is equal to getting all cliques of its complement graph. As a dense graph ($\bar{G}_{dep}$), it is impractical to get maximal cliques of the complement graph. Therefore, we cannot efficiently calculate all maximal independent sets from the dependency graph directly based on the Bron-Kerbosch algorithm.

As shown in the dependency graph property analysis (Section 4.5.3) and the time analysis in the performance evaluation (Section 4.6.2), it is not practical to generate all maximal independent sets $\{I\}$ due to the density of the complement of G_{dep}. Thus, we propose a clique-based maximal independent set algorithm to calculate $\{I^v\}$. As shown in Algorithm 2, it fist excludes all adjacent nodes of v in the resource dependency graph

G. Then, it chooses node with maximum degree from each connected candidates of the remaining complement graph $\hat{G}$ recursively in a branch-and-bound method until there is no vertex left. Algorithm 2 can achieve the worst-case optimal time complexity of finding all MISs of a node v as $O\left(3^{m/3}\right)$ [196], where $m = |V(G) - G_N(v)| - 1$.

4.5.4 Concurrency for Migration Candidates

In this section, we introduce the migration concurrency metric (MIGC) to indicate the resource dependency level of a potential migration. It is based on the maximal cliques and independent sets of an src-dst pair node. Let M^x_{mig} be the list of migrations have been selected currently. Let M^x be the list of src-dst pair nodes v_j of each migration $m_j \in M^x_{mig}$. For the first round $x = 0$, when the list of selected VM migration is empty, MIGC can be calculated as:

$$MIGC_v = \kappa \cdot \max\left(|C^v|\right) \big/ \max\left(|I^v|\right) \tag{4.17}$$

where $I^v \in \{I^v\}$ and $C^v \in \{C^v\}$, κ is the coefficient for the value normalization. When $x > 0$, the MIGC of migration with src-dst pair node v in G_{dep} can be represented as:

$$MIGC_v^{M^x} = MIGCliq_v^{M^x} + 1 \Big/ MIGInd_v^{M^x} \tag{4.18}$$

The migration independent score of the testing node v regarding to the selected migration list can be calculated as:

$$MIGInd_v^{M^x} = \frac{\sum\limits_{v_j \in M^x} \sum\limits_{I^v \in \{I^v\}} \left|v_j \cap I^v\right|}{|\{I^v\}| \cdot |M^x|} \tag{4.19}$$

where $\sum\limits_{v_j \in M^x} \sum\limits_{I^v \in \{I^v\}} \left|v_j \cap I^v\right|$ indicates how many times src-dst nodes v_j of migration from the currently selected list $v_j \in M^x$ is shown in all MISs of the testing node v. $|\{I^v\}| \cdot |M^x|$ is the product of the total number of I^v and the number of selected migrations.

Similarly, the migration clique score for src-dst pair node v according to the node list of currently selected migrations M^x is represented as:

$$MIGCliq_v^{M^x} = \frac{\sum_{v_j \in M^x} \sum_{I^v \in \{C^v\}} \left| v_j \cap C^v \right|}{|\{C^v\}| \cdot |M^x|} \tag{4.20}$$

where the numerator part indicates how many times the src-dst pair nodes of currently selected migrations is included in the maximal cliques of the node v.

The range of the migration clique score and independent set score is $MIGCliq \in [0,1]$ and $MIGInd \in (0,1]$. The largest $MIGCliq$ is 1 when all src-dst pair nodes of selected migrations in M shown in every maximal clique of the testing node. $MIGCliq$ is 0 when there is no pair node included. If there is no src-dst pair from the existing migration list included in the MISs of node v, we set the second part of $MIGC$ as max $\left(1/MIGInd\right) + \Psi$ with minimum $MIGInd$ value and Ψ as the extra cost. Thus, the smaller $MIGC$ of a potential migration, the fewer migration dependencies for the selected migration lists and future selections. Note that we do not need to check $MIGC$ of two migrations with the same node, as the result will be the same.

4.5.5 Concurrency-Aware Migration Selector

In this section, we explain the details of the proposed concurrency-aware migration selector (CAMIG) in Algorithm 3. It minimizes resource dependency and migration overheads while achieving the objective of resource management. Given the input of the objective of the dynamic resource management, the objective function, available VMs, candidates source and destination hosts, the networking information monitored by the SDN controller, and the VM and host information, CAMIG will generate the live migration list which consists of the selected VMs and the corresponding destinations.

In **Step 1**, G_{dep} and $M\ (v_{sd})$ are generated according to Algorithm 1. In **line 3**, we find all maximal cliques $\{C\}$ of G_{dep}. From **line 5-18**, at each round x, we select the optimal migration from src-dst node $\hat{v}_{sd}^{j}$ based on both $MIGC$ and single migration overhead $Inter_{single}$. As a result, it gets the overall minimal dependencies and single overheads of the total migrations to satisfy the objective of the dynamic resource management. For **Step 2**, in each optimal round, it first updates the single migration interference of

Algorithm 3: CAMIG

```
   Input: Performance Objective $Score^*$, potential VMs $i$, source $H_s$, dst $H_d$
   Result: Selected Migration List $M_{mig}$
1  Step 1. get node clique and indep matrix
2  $G_{dep}$, $\{M(v_{sd})\} \leftarrow$ CREATEdepGraph $(H_s, H_d, k)$;
3  $\{C\} \leftarrow$ ALLCliques $(G_{dep})$;
4  $x \leftarrow 0;\ M^x \leftarrow \emptyset;\ M^x_{mig} \leftarrow \emptyset$;
5  do
6      Step 2. get candidate VMs
7      UPDATEMigInterference $(\mathrm{VM}_i, H^i_d, L^i_{sd})$;
8      $\hat{Score}^{x+1}, \{v^j_{sd}\}, \{m^j_{sd}\} \leftarrow$ GETMigCandidates $(p_{current}, \{w_i\}, \{H^i_d\}, Score^x,$
         $M^x_{mig})$;
9      Step 3. select the optimal migration
10     $\hat{v}^j_{sd} \leftarrow v^0_{sd};\ \hat{m}^j \leftarrow m^0_{sd}$;
11     if $|\{v^j_{sd}\}| > 1$ then
12         foreach $v \in \{v_{sd}\}$ do
13             $C^v$ = ALLCliques $(\{C\}, v)$;
14             $I^v$ = ALLIndepSet $(G_{dep}, \{C\}, v)$;
15             if $Inter^{j,v} < Inter_{\min}$ then
16                 $Inter_{\min} \leftarrow Inter^{j,v}$;
17                 $\hat{v}^j_{sd} \leftarrow v^j_{sd};\ \hat{m}^j \leftarrow m^j_{sd}$;
18     $M^{x+1} \leftarrow M^x \cup \hat{v}^j_{sd};\ M^{x+1}_{mig} \leftarrow M^x_{mig} \cup \hat{m}^j_{sd}$;
19     UPDATEdepGraph $(G_{dep}, \{C\}, \hat{m}^j_{sd}, \hat{v}^j_{sd})$
20 while $|\hat{Score}^{x+1} - Score^*| > \delta$ and $\hat{Score}^{x+1} > \hat{Score}^x$ and $x + 1 < |\{m\}|$;
21 return $M_{mig}$
```

each candidate VM for its potential destinations. According to the selected migrations of previous rounds M^x_{mig} and current placement, it gets the newest VM to Host mapping. Then, it obtains the candidate migrations $\{m^j_{sd}\}$ and corresponding pairs v^j_{sd} in this round with the same objective score $\hat{score}^{x+1}$. It can generate more potential migrations by enlarging the score tolerance of the optimal objective in each round. For **Step 3**, the optimal migration with the minimum total migration interference $Inter_{\min}$ is selected. It first calculates $\{C^v\}$ based on all maximal cliques $\{C\}$ generated based on Bron-Kerbosch Degeneracy algorithm and $\{I^v\}$ according to Algorithm 2. Then, based on the pair list of already selected migrations M^x, the migration overhead of migration

m_i with src-dst pair v can be calculated as:

$$Inter^{i,v} = \kappa_{mig} \cdot Inter^{i,v}_{single} + \kappa_{mig} \cdot Inter^{i,v}_{single} \cdot MIGC^{M^x}_{v} \quad (4.21)$$

where κ_{mig} is the coefficient for the value normalization of single migration overheads. Then, the single migration overhead $Inter^{i,v}_{single}$ and $MIGC^{M^x}_{v}$ can be calculated based on Equation (4.1)-(4.3) and (4.17)-(4.19), respectively. In **line 17**, it adds the optimal migration of this round $\hat{m}^{j}_{sd}$ and its pair node $\hat{v}^{j}_{sd}$ to the currently selected migration list M^{x}_{mig} and corresponding node list M^x.

In **line 19**, algorithm **UpdatedepGraph** updates the dependency graph and all maximal cliques according to the selected migration. Certain potential migrations related to the selected optimal migration are deleted from the the pair list. For example, in Section 4.3.4, if we choose migration $v^1_1 : H1 \rightarrow H3$, then $v^2_1 : H3 \rightarrow H1$ is excluded for future selection. Note that we do not need to use Bron-Kerbosch Degeneracy to recalculate $\{C\}$ based on the new subgraph (Theorem 1). If the pair list is empty after update $M_{sd} = \emptyset$, the corresponding node v_{sd} will be removed from G_{dep} and $\{C\}$. If the updated clique size is 1 and the only one vertex left has connected edge, remove such clique. Duplicated cliques are also removed.

The stop conditions of CAMIG are: (1) at the round x, the currently selected VM migrations achieve the objective of dynamic resource management; (2) the objective is not improved in the last round; (3) round number equals to the total number of potential VMs.

Theorem 1 (Correctness of UpdatedepGraph). *Given a graph $G = (V, E)$, $V \neq \emptyset$, its all maximal cliques $\{C\}$ and its subgraph $G' = G[V \backslash \{v'\}]$ with removing vertices $\{v'\}$, results of UpdatedepGraph algorithm $\{C''\}$ and listing all maximal cliques $\{C'\}$ of G' are the same.*

Proof. Bron-Kerbosch Degeneracy generates all and only maximal cliques $\{C\}$ of G [197]. (1) For $\forall C'$, $\forall C''$, $|C'| = 1$ and $|C''| = 1$. Because the $V(G) \backslash \{v'\} = V(G')$. Thus, $\{C'\} = \{C''\}$. (2) For $\forall C'$, $\forall C''$, $|C'| > 1$ and $|C''| > 1$. For the sake of prove, we assume that $\exists C', C' \notin \{C''\}$. Then, $\exists C_e, \exists C'_e$, $C_e = C'_e \cup \{v_e\} \cup \{v'_e\}$, where $C_e \in \{C\}$, $C'_e \in \{C'\}$, part of remaining vertices $\{v_e\} \subseteq V(G) \backslash \{v'\}$, part of removing vertices $\{v'_e\} \subseteq \{v'\}$. Then, we have $C'_e \cup \{v_e\} \in \{C''\}$. If $\{v_e\} \neq \emptyset$, because $\forall C', \exists C, C' \subseteq C$, then $C_e^{'} \cup \{v_e\} \in \{C^{'}\}$. We have a contradiction, as $C_e^{'}$ is a maximal clique of $G^{'}$. If $\{v_e\} = \emptyset$ or $C_e = C_e^{'}$, as the UpdatedepGraph removes all $v^{'} \in \{v^{'}\}$, we have a contradiction $C_e^{'} \in \{C^{''}\}$. Thus, $\forall C^{'} \in \{C^{''}\}$. Similarly, we can prove $\forall C^{''} \in \{C^{'}\}$. Therefore, $\{C^{'}\} = \{C^{''}\}$. □

The worst-case running time of Bron-Kerbosch Degeneracy is $O\left(dn3^{d/3}\right)$ [197] with total n vertices and degeneracy d. The upper bound of all maximal cliques/independent sets of a Graph G is $(n-d)\,3^{d/3}$. Thus, given c maximal cliques, the time complexity of the algorithm for calculating MIGC is $O(cn)$. Then, the worst-case running time of CAMIG is $O\left((n-d)\,n^2 3^{d/3}\right)$. We perform extensive computational evaluation on time complexity in Section 4.6.2. It demonstrate that algorithm CAMIG is very fast in practice.

4.6 Performance Evaluation

In this section, we evaluate the performance of our proposed concurrency-aware migration selection (CAMIG) algorithm for dynamic resource management with several parameters, such as total migration time, total migration number, and corresponding dynamic resource management performance in load balancing and energy-saving scenarios. We used both real-world workload trace from PlanetLab [198] and synthetic workloads for the evaluation. We also performed extensive computational experiments for time analysis. The results show that the proposed algorithm can significantly improve the multiple migration performance while achieving the target of resource management.

The scalability of Mininet is limited due to the limitation of its resource usage and the operating systems, which prevents the cloud-scale simulations. Furthermore, it can not simulate the computing resource for the dynamic resource management and multiple migration scheduling. Thus, we have implemented components for the multiple migration scheduling simulations based on the CloudSimSDN [166]. The accuracy of network processing of CloudSimSDN compared to Mininet is validated in [71]. Based on the event shown in Fig. 1.5, the event-driven simulator can evaluate the performance of multiple migrations in terms of the total migration time, migration execution time, total transferred data, and downtime. Based on the system architecture, we also implemented the corresponding components to support the network resource monitoring, the multiple migration planning algorithm, and the on-line migration scheduler based on the resource-dependency graph of the selected migration list.

4.6.1 Load Balancing Scenario

In this section, we evaluate the influence of migration concurrency during the dynamic resource management in the load balancing scenario. The target of the resource management policy in this experiment is to keep the total CPU utilization of each physical host to 50%. For other solutions besides the optimal, we set the target range of the total CPU utilization from 45% to 55%. We compare our algorithm CAMIG with the result of the optimal and other load-balancing algorithms: Sandpiper [137], FFD [136], and iAware [90]. We first evaluate the proposed algorithms on small-scale experiments with 8 physical hosts in a Fat Tree topology. Then, we extend the scale of experiments for more complex scenarios. In extensive experiments, by integrating the proposed concurrency-aware algorithm with existing dynamic resource management algorithms, we directly evaluate and illustrate the scheduling performance improvement in multiple migration planning and scheduling algorithm.

Experimental Setup

In order to focus on the performance of multiple migrations for different migration requests generated by various resource management algorithms, we controlled variables of single migration overheads, such as dirty page rate, that other comparison algorithms ignore. In the load-balancing scenario, we use the same source selection as Sandpiper to choose over-utilized source hosts for potential migration.

The actual location of physical hosts in Fat Tree topology with different resource utilization is generated randomly, which causes different source and destination selections in each random setup. Without specific explanation, the result is the average value of 10 experiments. Causing utilization difference among hosts, the initial placement of VMs in each machine with different CPU utilization and memory size is shown in Fig. 4.6.

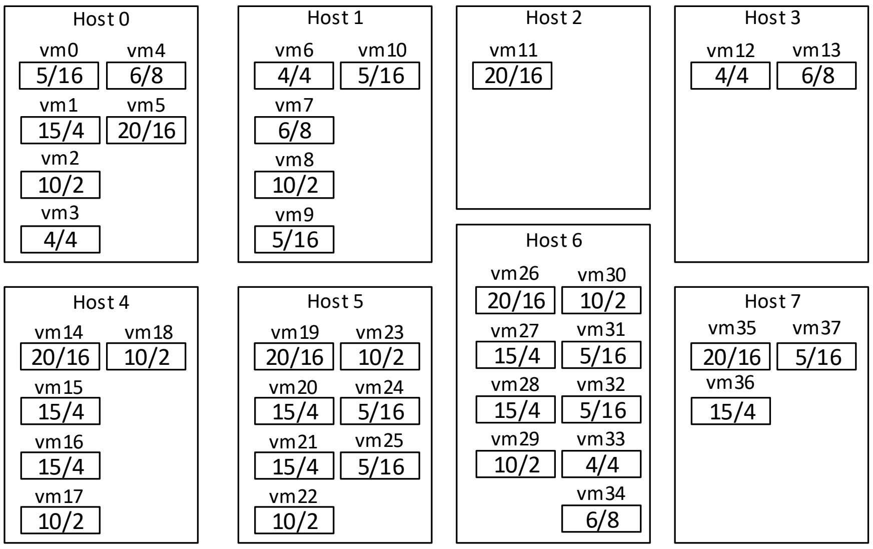


Figure 4.6: One of random data center setups with initial mapping for 8 different physical hosts with CPU utilization(%) and Requested Memory(GB)

To differentiate the migration value in management objective and migration schedule, we create VMs with different combinations of high, medium, and low value of resource utilization and memory size. The CPU utilization of each VM is from 4% to 20% of the total host CPU resource. As a result, the CPU utilization of each host is from 10% to 90%. The Memory size of each VM is from 2 GB to 16 GB, which can result in various migration overheads. Other parameters of pre-copy migration are set as the same for each VM. The dirty page rate factor is 0.001 per second. For example, with a 0.001 per second dirty page rate factor, the dirty page rate of a VM with 16 GB memory is 128 Mbps. The data compression ratio is 0.8. The iteration and downtime threshold is 30 and 0.5 seconds, respectively. For the physical topology, we create a k-8 FatTree Data Center Network (128 hosts) with 1 Gbps bandwidth between switches. Each physical host has 16 CPUs with 10000 MIPS each, 10GB RAM, 1 TB storage, and 1 Gbps network interface.

Dual simplex (Gurobi optimizer 9.0 and Python-MIP 1.6.7) were used to get the optimal solution of the MIP model. We also proposed a baseline algorithm called HostHits (hht). As shown in CAMIG selections, several potential destinations can achieve the

Gurobi solver, `https://www.gurobi.com/`
Python-MIP. `https://github.com/coin-or/python-mip`

Table 4.1: Total migration time and sum of migration execution time comparison in the extending mapping scenarios

approach	multi1	multi2	multi3	multi4
optimal	71.5313 / 172.9520	71.5313 / 345.9040	71.5313 / 518.8560	71.5313 / 691.8080
camig	86.5060 / 189.5725	86.5060 / 379.1451	86.5060 / 568.7177	86.5060 / 758.2903
sandpiper	86.5060 / 189.5725	86.5060 / 379.1451	99.4928 / 594.7547	99.4860 / 784.4188
optimal+sandpiper	86.5329 / 189.6183	86.5329 / 379.2367	86.5094 / 568.8412	86.5329 / 758.4734
ffd	73.2070 / 133.0450	88.1817 / 266.1101	73.2203 / 399.2128	88.1949 / 532.3334
iaware	86.5158 / 174.6271	578.5142 / 969.6401	374.0354 / 1448.9137	419.1750 / 1941.2873

Table 4.2: Comparisons of dependent migrations, multiple migration interference, and standard deviation of CPU utilization

approach	multi1	multi2	multi3	multi4
optimal	5/ 3.1648/ 0	10/8.9682/ 0	15/ 10.2091/ 0	20/ 14.3697/ 0
camig	10/ 6.2048/ 7.4286	20/13.0928/ 6.9333	30/ 31.2534/ 6.7826	40/ 36.4625/ 6.7097
sandpiper	10/ 6.2048/ 7.1428	34/ 22.9404/ 6.6667	55/ 58.0650/ 6.6087	76/ 70.0414/ 6.5161
optimal+sandpiper	10/ 6.8879/ 14.2857	20/ 13.9321/ 10	30/ 21.4943/ 8.7826	40/ 32.6992/ 9.7419
ffd	11/ 6.3697/ 84.5714	21/ 19.2937/ 78.9333	33/ 23.1770/ 77.2173	54/ 45.3416/ 76.3870
iaware	15/ 9.0528/ 35.7142	53/ 49.4754/ 210.8	48/ 38.6271/ 235.9130	79/ 68.3587/ 248.25801

same objective of dynamic resource management. It chooses the least selected/hit host as the destination of VM migration in each migration selection iteration.

For original Sandpiper, FFD and iAware without multiple migration scheduling, the sum of migration execution time is the actual total migration time of these algorithms, because they only consider one-by-one migration scheduling. However, given the multiple migration requests, we apply the multiple migration planning and scheduling algorithm proposed in Chapter 5 to all resource management algorithms in experiments and evaluate and show the results of corresponding performance in multiple migration scheduling.

Results Analysis

In one setup of scenario multi1 (Fig. 4.6), the optimal result of multiple migrations for load balancing of the CPU utilization is $\langle 2,0,7\rangle$, $\langle 17,4,1\rangle$, $\langle 18,4,1\rangle$, $\langle 20,5,2\rangle$, $\langle 21,5,2\rangle$, $\langle 27,6,3\rangle$, $\langle 28,6,3\rangle$, $\langle 30,6,3\rangle$, where the three-tuple $\langle vmnumber, sourcehost, destinationhost\rangle$

indicates a live VM migration. The standard deviation of the CPU loads in the optimal results is 0. The total migration number is 8, the dependency number is 5, and the maximum clique size is 3.

The total 10 migration requests of Sandpiper is $\langle 29,6,3\rangle$, $\langle 30,6,3\rangle$, $\langle 22,5,2\rangle$, $\langle 27,6,3\rangle$, $\langle 23,5,2\rangle$, $\langle 17,4,1\rangle$, $\langle 20,5,2\rangle$, $\langle 2,0,7\rangle$, $\langle 28,6,3\rangle$, $\langle 18,4,1\rangle$. In the load-balancing scenario, we use the same source selection as Sandpiper to choose over-utilized source hosts for potential migration. The result of CAMIG is $\langle 29,6,3\rangle$, $\langle 30,6,3\rangle$, $\langle 22,5,2\rangle$, $\langle 27,6,3\rangle$, $\langle 23,5,2\rangle$, $\langle 17,4,1\rangle$, $\langle 20,5,2\rangle$, $\langle 2,0,7\rangle$, $\langle 33,6,3\rangle$, $\langle 18,4,1\rangle$. The 9 migrations of FFD is: $\langle 17,4,1\rangle$, $\langle 18,4,2\rangle$, $\langle 22,5,7\rangle$, $\langle 23,5,1\rangle$, $\langle 21,5,3\rangle$, $\langle 27,6,2\rangle$, $\langle 28,6,3\rangle$, $\langle 29,6,3\rangle$, $\langle 30,6,2\rangle$. As shown in Table 4.2, the standard deviation of CPU utilization of FFD is 84.57 which is the worst among other load-balancing algorithms. The total 9 migration list of iAware is $\langle 17,4,1\rangle$, $\langle 18,4,2\rangle$, $\langle 22,5,7\rangle$, $\langle 23,5,1\rangle$, $\langle 21,5,3\rangle$, $\langle 27,6,2\rangle$, $\langle 28,6,3\rangle$, $\langle 29,6,3\rangle$, $\langle 30,6,2\rangle$.

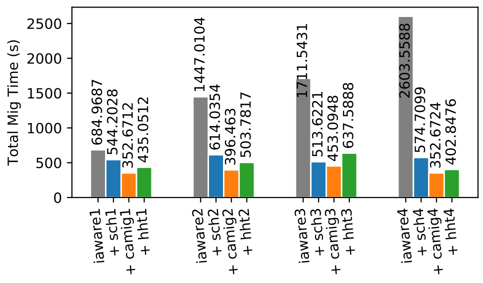
The rationale is that Sandpiper chooses the largest volume/memory VM from one of the most overloaded physical host to minimize live migration overheads. The volume as the multi-dimensional loads indicator is defined as: $Volume = \frac{1}{(1-cpu)(1-net)(1-mem)}$ [137], where *cpu*, *net*, and *memory* are normalized utilizations of corresponding resources. FFD (First-Fit Decreasing) algorithm selects the smallest size VMs from over-utilized hosts and assigns them in the FFD ordering of the spare resources to under-utilized hosts. iAware considers both co-location VM interference and the single live migration overheads. The co-location VM interference is linear to the number of VMs one physical machine hosts in Xen. The migration selection in iAware is sequentially decided in each round of the greedy algorithm. The migration tasks are also scheduled one by one.

Scalability Evaluation

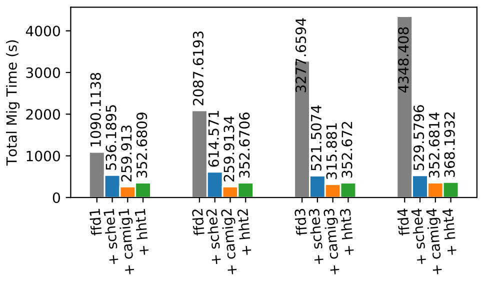
We extend the scale of experiments (multi2, multi3, and multi4) by multiplying the same mapping 2, 3, and 4 times with random physical host locations. Each scenario has 16, 24, 32 candidate destination hosts with a total 76, 114, and 152 potential migration VMs, respectively. For example, the physical Host 16, Host 8 and Host 0 have the same VM initial allocation. However, for each scenario, the placement of each physical host in the FatTree is generated randomly. As the resource management algorithms do not have the prior knowledge of the initial placement, the combination of source, destination, and instances during migration selection is increased exponentially. As a result, with the experiment scale increasing, more random source and destination combinations of potential migrations are generated for each experiment. We conducted 10 experiments in each scenario and show the average results. In Table 4.1 and 4.2, we show the results of the optimal solution, CAMIG and the optimal solution with Sandpiper VM selection, Sandpiper, FFD, and iAware in total migration time with multiple migration schedule, total migration execution time (one-by-one schedule), the number of dependent migration tasks, multiple migration interference value, and the load-balancing performance (standard deviation of CPU utilization). The multiple migration interference value is the sum of normalized single overheads from dependent migrations. Although all physical hosts are arranged randomly, the optimal result should be the same as in scenario multi1.

Analysis: Table 4.1 and 4.2 show that the MIP model achieves the optimal in all scenarios. With the source host selection from Sandpiper, comparing CAMIG with the optimal solution, as the problem scale increases, CAMIG can maintain the optimal performance in multiple migration scheduling as well as the number of resource-dependent migrations. In multi3 and multi4, CAMIG over-satisfies the requirement of load-balancing by losing the value of multiple migration interference. For the Sandpiper and iAware, as the the scale of the problem increases, the number of dependent migrations and the value of multiple migration interference increase dramatically, which leads to a larger total migration time in both multiple and one-by-one scheduling. FFD can not satisfy the requirement of load-balancing in the system.

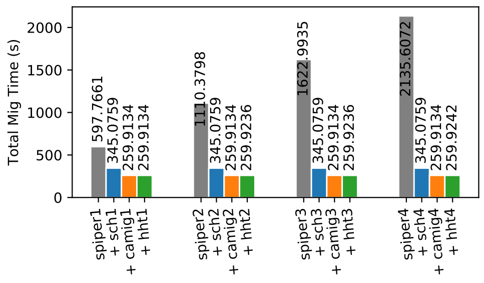
The total migration time of Sandpiper is increased by 15.01% in multi3 and multi4. In Table 4.2, although FFD has the lowest total migration time and migration execution time, it cannot achieve the ideal load-balancing performance. The standard deviation of FFD is the largest among other algorithms. Moreover, the largest total migration is increased by 21.33% compared to the lowest. For iAware, the actual total migration time equals to the total migration execution time by only allowing one-by-one scheduling. With multiple migration scheduling, iAware has the worst performance in total migration time and load-balancing due to the trade-off between migration execution time and



(a) iAware



(b) FFD



(c) Sandpiper

Figure 4.7: Performance comparison with one-by-one, multiple scheduling, CAMIG, and HostHits

co-location interference. The total migration time varies largely in different scenarios, increasing at most 568.68%.

Extensive Evaluation

As every load-balancing policy has its own logic for VM selection, it is difficult to evaluate the improvement of multiple migration directly. Thus, in this section, we extended the experiments by integrating the HostHits and CAMIG algorithm with the existing policies: iAware, FFD, and Sandpiper. With the benefit of flexibility, CAMIG can be adapted to other existing dynamic resource management algorithms. We randomly generated VM Memory Size from 8 to 14 GB with the same scenarios (Fig. 4.6). Each result is the average value of 10 experiments in each scenario. Fig. 4.7 illustrates the multiple migration performance in total migration time based on the migration requests of these policies with one-by-one scheduling and multiple migration scheduling (+sch), and multiple migration scheduling performance based on the migration requests of CAMIG (+camig) and HostHits (+hht) in 4 different scenarios.

Analysis: Fig. 4.7(a) indicates that iAware with CAMIG can achieve the best performance with multiple migration scheduler in all 4 scenarios. The performance is increased by 20.55%, 57.57%, 70.02%, and 77.93% when migration requests scheduled by the multiple migration scheduler, respectively. However, with CAMIG the performance is increased by 48.54%, 72.63%, 73.52%, and 86.48% compared to the original iAware and increased by 35.29%, 35.50%, 11.89%, and 38.68% compared to the performance of iAware with only multiple migration scheduler. Moreover, although iAware with HostHits generally has a better performance compared to iAware+scheduler, as shown in scenario multi3, it results in a worse total migration time due to creating a larger clique of the dependency graph. For FFD, CAMIG can increase the performance up to 91.90%, 57.82%, and 26.42% compared to FFD with one-by-one scheduler, multiple migration scheduler, and HostHits (Fig. 4.7(b)). Moreover, Fig. 4.7(c) shows that the performance of Sandpiper with CAMIG in total migration time is increased by up to 87.87% and 24.68% than Sandpiper with one-by-one scheduler and multiple migration scheduler, respectively.

Summary

In summary, CAMIG can efficiently improve the multiple migration performance while achieving the target of load-balancing resource management. The performance of comparing load-balancing policies can be increased by up to 91.90%, 57.82%, and 28.89% as compared to the one-by-one scheduler, the multiple migration scheduler, and HostHits, respectively. CAMIG always outperforms the original policy and the HostHits. The round-robin algorithm HostHits cannot guarantee the multiple migration performance though it generally can decrease the total migration time.

4.6.2 Processing Time Analysis

In this section, we analyze the time complexity of the proposed CAMIG algorithm. The experiments were run in the computer with i7-7500U CPU with 2.70 GHz, and 15.9 GB

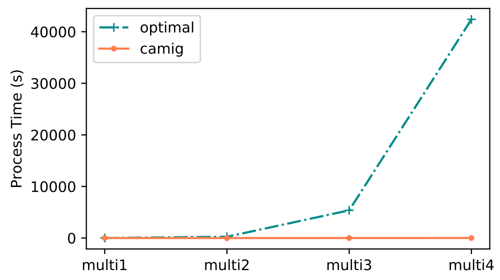


Figure 4.8: Runtime comparison between optimal and CAMIG

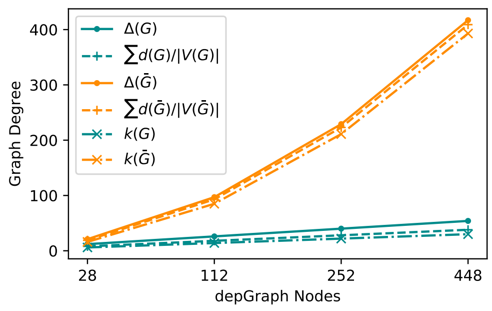


Figure 4.9: Average and maximum degree and degeneracy

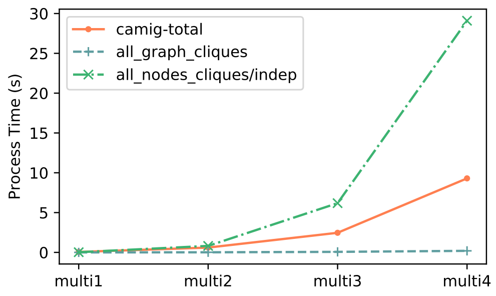


Figure 4.10: Runtime of CAMIG, all maximal cliques, all maximal cliques and independent set of nodes

RAM in Windows 10 64-bit Operating System. Fig. 4.8 illustrates that the runtime of the optimal solution solved by MIP solver is increased exponentially against the linear growth of the problem size. The runtime of the optimal solution on average is 3.07s, 251.51s, 5373.35s, and 42388.0s in 4 scenarios, respectively. Thus, it is impractical to generate the optimal result when facing the problem in real life.

Fig. 4.9 illustrates the connectivity properties of dependency graph in terms of average degree $\sum d(G)/|V(G)|$, maximum degree $\Delta(G)$, and degeneracy of the dependency $k(G)$ and its complement $\bar{G}$. The number of maximal cliques is 12, 28, 42, 56 with the degeneracy (a measure of graph spareness) of the dependency graph as 6, 14, 22, 30. Therefore, it is much easier to generate all maximal cliques with a small degeneracy. However, the degeneracy of the complement dependency graph increased dramatically as 16, 85, 211, 393. Thus, it is impractical to generate all maximal cliques of the complement graph as the problem size becomes significantly large. In other words,

Table 4.3: Performance Comparison between LR-MMT, HostHits, CAMIG in energy-saving scenario

algorithm	mig. num	∑ total mig. time	∑ dt. (s)	workload num		serve time incl. and excl. timeout (s)			energy cost (Wh)		
				total	timeout	total excl.	avg. excl.	avg. incl.	total	host	switch
NoMig	-	-	-	1506464	0	11214923.24	7.44	-	1733432.22	1733432.22	0
LR-MMT	3741	28038.66	355.079	1399857	106497	8700783.51	6.21	1105.63	470492.05	465412.23	5079.82
HostHits	3680	25872.79	359.032	1416806	89550	9028858.54	6.37	447.61	487254.15	481810.21	5443.94
CAMIG	2534	7453.37	178.071	1458906	47522	9945354.17	6.82	80.76	450966.81	447817.74	3149.07

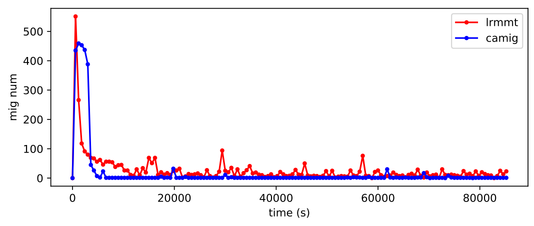


Figure 4.11: Migration number within each interval

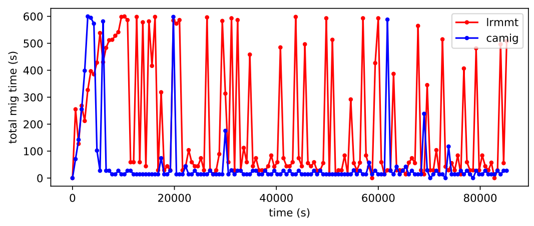


Figure 4.12: Total migration time within each interval

Bron-Kerbosch Degeneracy algorithm can reach the worst-case runtime when the graph becomes considerably dense. As a result, it can only generate all 661 maximal independent sets in the smallest scale scenario (multi1). Fig. 4.10 shows the runtime comparison of CAMIG in total processing time, finding all maximal cliques, and generating all maximal cliques and independent sets for every node. As shown in Algorithm 2, we do not need to calculate all maximal cliques and independent sets of every node in the graph. The all_nodes_cliques/indep illustrates the upper-bound of runtime. The processing time of CAMIG is increased linearly against the total src-dst node in resource dependency and the average degree or the degeneracy of the complement of the dependency graph as shown in Fig. 4.9.

4.6.3 Long-term Energy Saving Scenario

To evaluate the proposed algorithm with the real-world long-term workloads [198], we integrated and compared CAMIG with LR-MMT [141] in the energy-saving scenario in terms of total migration time, migration numbers, downtime, total/average CPU serve time with and without the timeout workloads, and energy (power) cost of both hosts and switches.

Evaluation Configuration

For the long-term experiments, we created a k-16 FatTree topology (1024 hosts) with 1 Gbps physical links between switches to simulate the environment with limited network resources for live migrations. Each physical host has 8 CPUs with 4000 MIPS, 1024 GB Memory size, 1000 GB Storage, and 1 Gbps network interface. The real-world workload trace of CPU utilization from Planetlab [198] was used for the experiments running in 24 hours. There are 1052 CPU utilization files mapping to the same amount of VMs. We generated the workloads based on the MIPS requirement and the CPU utilization varied along the time. In order to illustrate the influence of multiple migration performance, there is no application traffic between different VMs other than the migration flows. There are 4 flavors of VM: 2 vCPUs, [2500, 2000, 1000, 1000] MIPS, [2, 4, 4, 2] GB RAM, 100 Mbps virtual bandwidth, and 4 GB Disk Size. The initial placement of VMs are allocated based on the optimization criteria defined by LR-MMT [141].

The LR-MMT algorithm utilizes the Local Regression (LR) method to predict the overloading hosts in the upcoming monitor interval. Minimum Migration Time (MMT) policy is used for VM selection to minimize the migration overheads. During each monitoring interval of dynamic resource management, CAMIG, as a flexible algorithm, utilizes the same local regression to detect over/under-utilized hosts. In LR-MMT, though there are many equivalent optimal destinations, it only chooses the first fit. In this experiment, for the sake of fair comparison, the destination candidates used in CAMIG are provided by the same energy-saving policy in LR-MMT.

Evaluation Results

As shown in Table 4.3, CAMIG algorithm outperforms both LR-MMT and HostHits. The total energy consumption under no dynamic resource management is 1733432.22 Wh. The LR-MMT algorithm saves 72.86% energy consumption. Comparing CAMIG with LR-MMT, the host and switch energy consumptions are 3.78% and 38.01% less, respectively. The total migration number is 32.26% less, the sum of total migration time of each monitoring interval is 73.42% less, the total downtime is 49.85% less than the LR-MMT algorithm. The performance improvements in total migration time also result in fewer workload timeouts and CPU resource shortages. For VM processing, the average CPU server time is 92.70% less when there is no timeout mechanism. With a timeout mechanism, CAMIG also reduces the workload timeout by 14.30% compared to the LR-MMT.

As the sum of total migration time and total migration time of each monitoring interval shown in Table 4.3 and Fig. 4.12, within the 24 hours experiment, the performance of CAMIG in multiple migration scheduling is largely better than the LR-MMT. A shorter total migration time during each monitoring interval means a quicker state convergence for minimizing the over-utilization period and maximizing the energy-saving through VM consolidation for under-utilizing hosts. In other words, minimizing the dependencies among multiple migrations is not only critical for the migration scheduling, but also for the dynamic resource management that provides the migration list.

During the experiments, we find out that there are relatively large equivalent destination candidates in terms of energy saving. Therefore, by exploring the concurrency score among these candidates, we can minimize the resource dependencies among the migrations. As shown in Fig. 4.11, there are more migrations in CAMIG from 1200s to 3600s than LR-MMT. It is because in LR-MMT once the candidate is used it will be excluded from the remaining destinations. However, by choosing equivalent hosts during the destination selection, CAMIG algorithm enables more available destinations for VMs which need to be migrated from both under and over-utilized hosts. Thus, CAMIG algorithm actually produces fewer migrations in the remaining monitor intervals. It also illustrates that in some cases even the total migration number of CAMIG is larger, the total migration time is much smaller due to the minimum dependency among the migrations. Fig. 4.12 shows that, under certain circumstances (the peak migration time at 20000 second), even if there is a small number of migration tasks, the total migration time is still very large. Due to the nature of the consolidation algorithm, there are many migration tasks sharing the same destination or source hosts. Therefore, in traditional architectures, such as FatTree or even the dedicated migration network, it is inevitable that the convergence of multiple migrations is slower. As a result, the performance of multiple migration scheduling may be limited by this nature of resource competition among the consolidating VM migrations.

To summarize, the evaluation demonstrates that, our proposed Concurrency-Aware migration (CAMIG) algorithm can efficiently minimize the resource dependency among the multiple migration tasks and achieve the objective of dynamic resource management in the long run. Thus, it also improves the performance of dynamic resource policies in terms of both QoS and energy consumption.

4.7 Summary

In this chapter, we formally established a MIP model for the problem and proposed two algorithms: (1) HostHits and (2) CAMIG. We conducted experiments to compare our proposed algorithms with existing dynamic resource management policies in load balancing and energy-saving scenarios by using both random synthetic setup and real trace data. Without changing the framework of existing policies, the results indicate that CAMIG can largely improve the performance of multiple migrations by up to 91.90% while achieving the target of dynamic resource management efficiently with near-linear computation growth in practice. In the long-term experiments, it can also reduce the total migration number, service downtime and management target in the host and switch energy consumptions.

Chapter 5
SLA-Aware Multiple Migration Planning and Scheduling

In this chapter, we propose SLAMIG, a set of algorithms that composes deadline-aware multiple migration grouping algorithm and on-line migration scheduling to determine the sequence of VM/VNF migrations. The experimental results show that our approach with reasonable algorithm runtime can efficiently reduce the number of deadline misses and has a good migration performance compared with the one-by-one scheduling and two state-of-the-art algorithms in terms of total migration time, average execution time, downtime, and transferred data. We also evaluate and analyze the impact of multiple migrations on QoS and energy consumption.

5.1 Introduction

With the rapid adoption of cloud computing, the requirement of providing Quality of Service (QoS) guarantees is critical for cloud services, such as Web, big data, virtual reality, and scientific computing. For the benefit of cloud administrators, it is essential to prevent violations of Service Level Agreements (SLAs) and maintain QoS in heterogeneous environments. Therefore, there has been a notable focus on the quality, efficiency, accessibility, and robustness of cloud services. For instance, the latency of service function chaining (SFC) [199] should be optimized to benefit both network service providers and end users.

This chapter is derived from:

- **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "SLA-aware multiple migration planning and scheduling in SDN-NFV-enabled clouds", *Journal of Systems and Software (JSS)*, Volume 176, Pages: 110943, Elsevier, 2021.

Although researchers have been trying to achieve the objectives of dynamic resource rescheduling through live migration [63, 200], few studies have focused on the impact of live migration overheads [26, 27] and the sequencing of multiple migrations [28–30]. Due to the end of life of some VMs and the variance of workloads in cloud computing, dynamic resource management constantly generates many migration requests in optimization rounds. As a result, multiple migration requests need to be scheduled. For example, dynamic resource management policies for performance efficiency and energy-saving [141] can generated up to 12500 migrations within 10 days. Moreover, commercial cloud platforms provides live migration to keep VM instances running during the host event, such as hardware or software update. For example, in Google Cloud Compute Engine, live migration occurs to one VM at least once every two weeks due to the software or hardware update. In 2020, to make the compute infrastructure cost effective, reliable and performant, Google Cloud Compute Engine also introduced dynamic resource management for E2 VMs through performance-aware live migration. Therefore, it is important to determine the order (sequence) of the migration tasks to optimize the total migration time [28–30], which is the interval between the start of the first migration and the completion of the last migration.

In cloud data centers, Software-Defined Networking (SDN) can enable the centralized control of network resources in terms of network topology, connectivity, flow routing, and allocated bandwidth. The Virtual Network Functions (VNFs) hosted in cloud data centers can also be linked as a Service Function Chaining (SFC) [199] by SDN controller. Migration planning for VNFs in the chain is not trivial since SFC requires traffic to traverse through a certain sequence of VNFs. In addition, because migrations share the network resources with other services, it is necessary to efficiently plan and schedule migration tasks to reduce the overhead impact on the QoS of other applications. The migration planner and scheduler based on the SDN controller can manage the network resources in a fine-grained manner for the migration tasks and application services in terms of network routing and bandwidth allocation.

Google Cloud Compute Engine. `https://cloud.google.com/compute/docs/instances/setting-instance-scheduling-options`

Dynamic resource management in E2 VMs. `https://cloud.google.com/blog/products/compute/understanding-dynamic-resource-management-in-e2-vms`

Connectivity and Sequence: Compared with services such as scientific computing, the connectivity and corresponding network requirement of links in SFC between source and destination are dynamically changing. This will also cause the remaining bandwidth of the migration to change. Furthermore, as the available bandwidth of the physical link changes according to the connectivity of the SFC, we also need to carefully consider the impact of the new placement of the VNF. As a result, the new placement will affect the rest of the migration requests that use the same path. In addition, two migrations could be performed concurrently if there are no shared paths between them. However, performing multiple live migrations in arbitrary order will result in service quality degradation. Therefore, efficient planning of the migration sequence is crucial to reduce the impact of live migration overheads.

In addition to the optimization of total migration, several other parameters that affect migration performance are largely neglected:

Scheduling window (deadline): For migration such as scheduled maintenance, disaster evacuation, load balancing policy, and other dynamic allocation algorithms [63, 103], it is usually associated with a time window (defined deadline) that requires the VM or VNF to be evacuated from the source and run on the destination host. For instance, the deadlines for SLA-related migration tasks are based on the violation speed and the cumulative violations threshold of Service Level Objective (SLO), such as response time and end-to-end delay. The scheduling window refers to the time interval between the arrival of migration task request and the deadline for the new placement. Failure to meet the deadline will result in QoS degradation and SLA violation.

Allocated bandwidth: During the live VM migration, the applications running inside VM constantly modify the local stack and variables in the memory. The memory modified during the last round of dirty memory transmission needs to be transferred again. The goal of live migration is to reduce the memory difference between the source and destination hosts in order to stop the VM and copy the remaining dirty memory pages to the destination. A smaller memory difference in the stop-and-copy round means that the service has much shorter downtime. Therefore, live migration is highly dependent on the network for dirty memory transmission. We can consider live migration as a non-preemptive task. If the available bandwidth is lower than the rate of memory dirtying during the iterative transmission, then the data transferred previously used for migration convergence will be in vain. Furthermore, although the average bandwidth for the entire process of a migration might be the same, the insufficient bandwidth at the beginning may severely extend the migration time. Therefore, we should carefully allocate available paths to multiple migration requests.

To help cloud providers guarantee QoS and SLAs during the multiple live migrations, we investigate the problem of optimizing the total migration time, transferred data, downtime, and average execution time of multiple VM/VNF migrations within the scheduling window in software-defined cloud data centers. We propose SLAMIG (SLA-aware Migration), which is a set of algorithms that includes the deadline-aware concurrent migration grouping for multiple VM/VNF migrations and an on-line migration scheduler to minimize the total migration time by considering the migration deadlines.

The main **contributions** of this chapter are summarized as follows:

- We modeled the multiple migration planning problem to optimize total migration time and deadline missing in the context of VMs/VNFs connectivity.
- We are the first to introduce the scheduling window for multiple migration scheduling.
- We investigated the impact of allocated bandwidth on the beginning of one migration.
- By maximizing the concurrent migration groups with minimal weight, we proposed a heuristic algorithm that achieves good performance for multiple migrations by considering the migration deadline.
- We designed an on-line migration scheduler to dynamically schedule migrations from different migration groups.
- We not only analyzed the multiple migration scheduling in total migration time and downtime, but the average execution time, total transferred data, deadline violations, QoS, and energy consumption.

Table 5.1: Configurations of different flavors

No.	Flavor	Mem (GB)	CPU (cores)	Disk (GB)	No.	Flavor	Mem (GB)	CPU (cores)	Disk (GB)
1	xlarge	64	12	120	5	tiny	2	1	2
2	large	16	8	60	6	micro	1	1	1
3	medium	8	4	20	7	lb/ids/fw	8	10/12/16	8
4	small	4	2	10	8	web/app/db	256	8/4/12	1000

The rest of this chapter is organized as follows. In Section 5.2, we present the motivation example, the impact of migration bandwidth, the model of sequential and parallel migrations, deadline of the migration, and the problem formulation of multiple migration planning. Section 5.3 explains the observations, rationales of algorithm design, and the details of proposed algorithms. Section 5.4 describes the experiment design and analyzes results. Finally, we review the related work in Section 5.5 and summarize the chapter in Section 5.6.

5.2 Motivation and Problem Formulation

5.2.1 Motivation Example

In this section, we discuss the problem and our motivation using an example of optimizing the total migration time to show the impact of migration orders on the total migration time, VNF/VM downtime, and SFC/VDC migration time and downtime.

Migration processes produce elephant flows which take a disproportionate part of network resources for a long time. At the end of each migration, the network flows within the data center network are redistributed accordingly due to the relocation of VNFs or VMs and their connectivity. With the change of available bandwidth in Data Center Network (DCN), the result of one migration will affect subsequent migrations that share the links with the completed one. The objective of migration planning is to

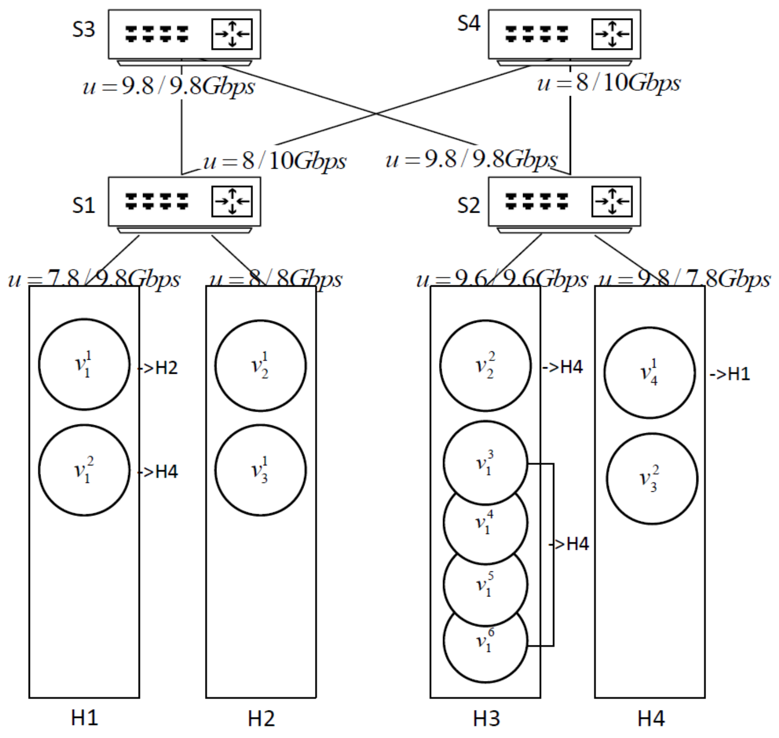


Figure 5.1: Initial mapping for VM/VNF v_i^j of VDC/SFC G^j, migration requests (s_j, d_j), and available bandwidth u of upload/download interfaces

find the orders of migrations to optimize the total live migration time of all requested migrations with certain constraints, such as the priority, required bandwidth, residual bandwidth on the links, and capacity of CPU, memory, and storage resources.

In the network of tree topology shown in Fig. 5.1, there are 4 switches which include 2 top-of-rack (S1 and S2), 2 aggregation switches (S3 and S4), and 4 hosts (H1 to H4). All the hosts and switches are connected through 10Gbps links. One SFC G^1, one VDC G^2 and four other VMs (v_1^3 to v_1^6) are hosted in different hosts accordingly. Fig. 5.2 shows the connectivity and reserved bandwidth of virtual links among instance with different flavors (Table 5.1) of G^1 and G^2, as well as the dirty page rate and CPU, memory, and storage requirements. SFC G^1 contains 4 VNFs where v_2^1 and v_3^1 are the same type of VNF. The migration time is composed of the processing time of pre-migration and post-migration overheads on computing resources and the network transmission time of the dirty memory. We assume the processing time of pre-migration and post-migration overheads on the single core are 0.8 seconds and 1.2 seconds, respectively.

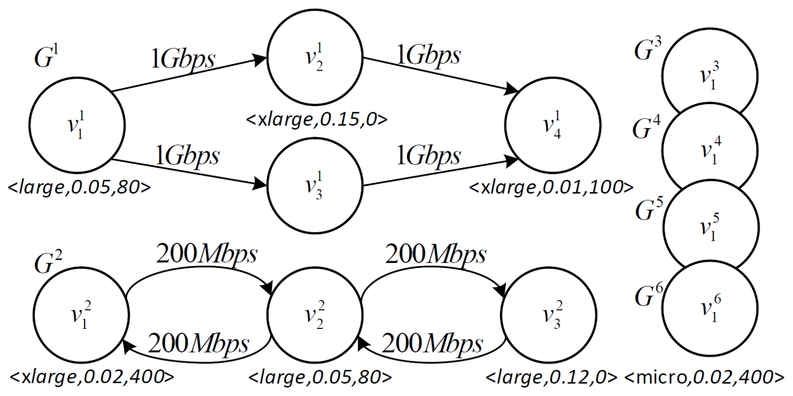
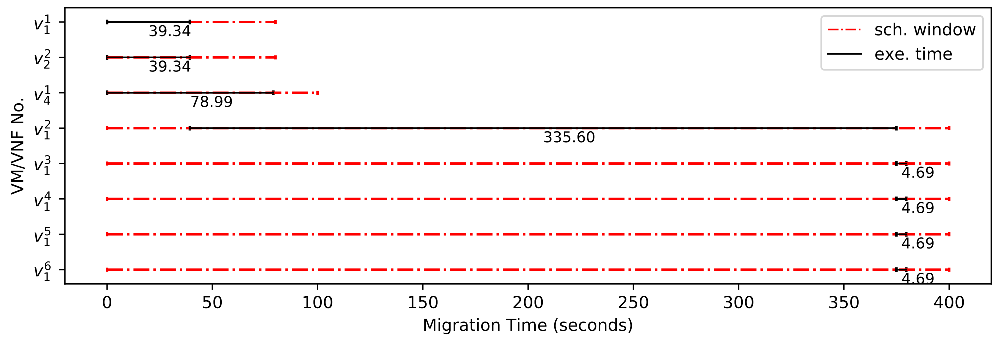


Figure 5.2: SFC and VDC configurations <flavor, dirty page rate, migration deadline>

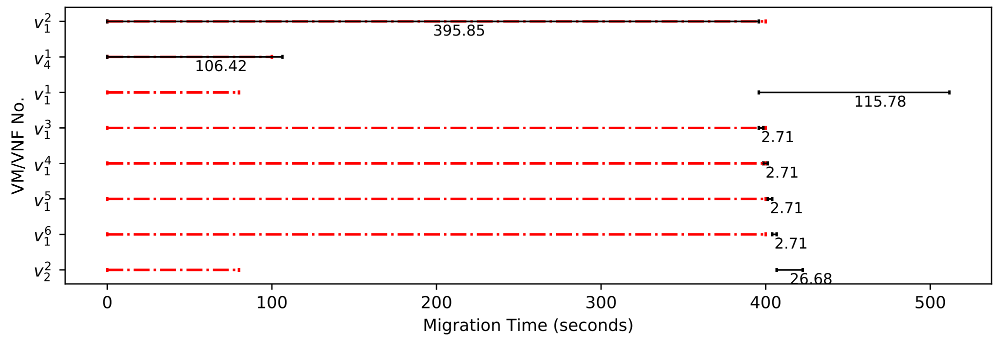
Fig. 5.1 illustrates the initial mapping of these VNFs/VMs and migration requests for another possible mapping in the physical topology. Let u denote the residual bandwidth on the links. According to the reserved requirements of virtual links, we calculate the initial available inbound and outbound bandwidth of each network interface.

At the time $t_0 = 0$, the coordinator receives several migration requests at the same time based on the configured optimal reallocation interval as shown in Fig. 5.1. Other VDCs and SFCs which are unrelated to the migration in the host are not shown. The maximum memory copy round is 30 and the downtime threshold is 0.5 second. There are two of possible orders: $S_1 = (v_1^1, v_2^2, v_4^1), (v_1^2), (v_1^3, v_1^4, v_1^5, v_1^6)$ and $S_2 = (v_1^2, v_4^1), (v_1^1, v_1^3)$, $(v_1^4), (v_1^5), (v_1^6), (v_2^2)$, shown in Fig. 5.3. Migration tasks within the same group could perform concurrently. For subsequent migrations from different concurrent migration group, the scheduler will start a migration as soon as a sharing-resource migration in the other group is finished. For example, migration of v_1^2 will start after the migration of instance v_1^1 is finished (Fig. 5.3(a)). After each migration, all virtual links connected to the migrated instance will be rerouted to the destination host. Therefore, the available bandwidth of the remaining migrations will be updated according to the new instance placement at the end of each migration.

By leveraging simulation capabilities for both computing and networking, we implemented and extended the corresponding components based on the CloudSimSDN [166] to simulate each phase of pre-copy live migration. As shown in Fig. 5.3, the total migration time T_{total} of two migration scheduling orders is 377.645 and 511.625 seconds, respectively. The average downtime $\Sigma T_d/n$ is 0.317 and 0.353 with maximum 0.807 and



(a) migration order S_1



(b) migration order S_2

Figure 5.3: Results of different scheduling orders

1.538 seconds for v_1^2 instance. Furthermore, for the migrations of v_1^3, v_1^4, v_1^5, and v_1^6 instance, as the processing time of computing overheads is 2.0 seconds, the total migration time from the start of migration v_1^3 is 4.691 and 10.593 seconds by using parallel and sequential method, respectively. The average value of the remaining scheduling window $\Sigma\xi/n$ of two orders is 26.104 and -99.708, respectively. Although these orders both perform concurrent migrations that do not share the same resources, the first scheduling order leads to a better performance in terms of total migration time, average downtime, SFC/VFC migration time, and remaining scheduling window (i.e. less SLO violations).

5.2.2 Impact of Bandwidth and Dirty Rate

First, we argue that the bandwidth allocated to the early iterative transmission rounds can highly affect the performance of a migration. Based on the mathematical model

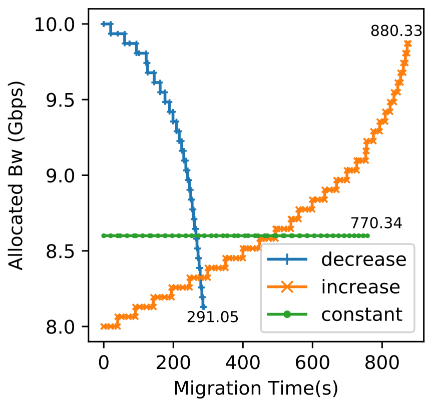


Figure 5.4: Migrations with xlarge flavor under various bandwidth functions

shown in Section 4.3.2, Figure 5.4 illustrates the migration performance under three different bandwidth functions, where: 1) begins with low bandwidth then increases the bandwidth for each iteration round; 2) has a constant bandwidth; 3) starts with high bandwidth then decreases the bandwidth for each iteration round. It indicates that even with the same average bandwidth during the migration, insufficient bandwidth in the early steps will cause a huge amount of dirty pages remained to transfer for the subsequent transmission rounds (Equation (3.3)). This causes a much slower convergence process to reach the point that remaining dirty pages satisfies the downtime threshold. Furthermore, according to the migration threshold and round constraints (Equation (4.3)), starting the migration with insufficient bandwidth results in a large accumulation of the remaining dirty pages in the previous rounds. In other words, in order to complete the migration within a reasonable time, an unacceptable service downtime will occur regardless of the downtime threshold due to the migration round constraint.

5.2.3 Deadline-related Migration

In this section, we discuss deadline-related migration. As one of the reasons for SLA violation, the Service Level Objective (SLO) violation speed and total violation threshold are the main monitoring parameters. If the cumulative violations of SLO exceed the threshold, one SLA violation happens. In this situation, the migration request is generated due to the SLO violations under the current placement of VMs/VNFs with the

Table 5.2: List of commonly used notations

Notation	Description
M	Memory size of VM/VNF
R	Dirty page rate
L	Bandwidth assigned to Migration
T_{pre}	Pre-migration processing time
T_{post}	Post-migration processing time
$T_{network}$	memory copy network transmission time
V_i	the transferred data of i_{th} round of memory copy
T_i	the time interval of i_{th} round of memory copy
T_{dthd}	the downtime threshold
ρ	the memory compression rate
σ	the ratio of R to L multiple ρ
$\lambda(p)$	the maximum allowed parallel number in path p
r	the processing speed of one compute core
$T^n_{network}$	parallel $T_{network}$ of n migrations in the same path
m_j	the memory size of migration j;
s_j, d_j	the ordered pair of source and destination
N	the set of physical network nodes
E	the set of physical network links
N^i	nodes set of VDC/SFC G^i
E^i	links set of VDC/SFC G^i
P	set of all paths in the network
P_j	set of all paths between (s_j, d_j)
$c(e)$	capacity of link e
$u(e)$	residual bandwidth in link e
$u(p)$	available bandwidth of path p
D_j	deadline of migration j
$D(G^j)$	deadline of all migrations that $n_j \in G^i$
θ	maximum tolerant number of SLA violations
ω	cumulative SLA violation rate
ξ	remaining migration scheduling window
Y_t	cumulated violations at time stamp t

workload bursting, end-user mobility, and resource over-subscription [63, 185, 200]. The SLO thresholds are configured by the cloud infrastructure management to alert on the significant events. A migration request can be issued due to the SLO violations under the current situation. For instance, VNF need to be relocated to the cloud before exceeding the threshold of cumulative violation due to current SLO violation rate in terms of response time and latency [63], and VMs need to be migrated from the overbooked hosts due to the service workload burst [185]. The burst of workload and location changing of end users can cause serious SLA violations and QoS degradation. Thus, VM/VNF migrations need to be finished before a certain deadline to prevent the cumulative SLO violations exceeding the threshold.

Thus, the deadline for the VM migration can be estimated based on the threshold of total allowed SLO violations and the current SLO violation speed. Based on the new optimal allocation, the placement algorithm will request corresponding VM migrations to prevent the accumulated SLO violations from exceeding the threshold. Among these migration tasks, different services and VNFs have several levels of urgency in terms of the current average SLO violation cumulative rate ω based on the monitoring, the current number of cumulated violations Y_t, and the threshold of total violations θ. Therefore, when the dynamic resource policy triggered at time t by the configured period, the deadline of migration task can be calculated as:

$$D = (\theta - Y_t)/\omega \tag{5.1}$$

For the migration tasks which specify the scheduling window (e.g., scheduled maintenance), the deadlines can be directly used as the input for migration scheduling.

Furthermore, there are time-critical migration requests with specific deadline $D(G^i)$ for the whole SFC or VDC G^i. In other words, all related VMs/VNFs inside the SFC/VDC need to be migrated and run in the destination hosts before the required deadline. A simple solution is to directly assign the group deadline to each migration task. For better performance, the deadline for each task can be calculated by subtracting the sum of the worst execution time of other migration tasks from the group deadline:

$$D_k = D\left(G^i\right) - \sum_{n_j \in G^i / n_k} T^j_{exe} \tag{5.2}$$

In practice, live VM migrations can be scheduled in low activity periods [103]. VMs or VNFs interact with different groups of end users with geographical variances or applications with different access features. For instance, VMs of Web applications allocated in the same physical host serve different areas, such as China, Japan, Australia, and Europe, may experience hours or minutes low-activity scheduling window. As a large amount of VMs/VNFs with various features allocated in relative limited physical hosts, the low activity window for migration scheduling can be extremely limited.

5.2.4 Problem Formulation

In this section, we formally define the problem of live SFC/VDC migration planning as a Mixed Integer Linear Programming (MILP) problem. In the model, the physical data center is represented by a graph $G = (N, E)$, where N denotes the set of nodes including physical hosts, switches, and routers, and E denotes the set of directed links between nodes. The remaining CPU, memory, and disk in the destination node N should be larger than the resources required by the migrating VM.

Let τ denote the instant of time when a migration starts or finishes. From the beginning of the first migration to the end of the last migration, at least one migration is is in progress in the data centers. Let T^i_{mig} denote the response time (single execution time of the migration i plus the waiting time to start). Then, for more concise expression, we use τ_0 as the begin time instant and τ_K as the end time instant for a total of K migration tasks.

$$\tau_i \in \left[0, T^1_{mig}, ..., T^K_{mig}\right] = [\tau_0, \tau_1, ..., \tau_K] \tag{5.3}$$

where the total K migrations are sorted by the completion time and $\tau_i \geq \tau_{i-1}, i = 0, 1, ..., K$. It converted the original problem to total K discrete state.

Let $X^{\tau_i}_k \in \{0, 1\}$ denote the binary variable that indicates whether the migration

$k \in \mathbb{R}^+$ occurs at time interval $\tau \in [\tau_i, \tau_{i+1})$. Therefore, the response time of migration k can be calculated as:

$$\tau_k = T_{mig}^k = \sum_{i=1}^{k} X_k^{\tau_i} \cdot (\tau_i - \tau_{i-1}), 1 \leq k \leq K \tag{5.4}$$

As mentioned in pre-copy migration model, the migration task cannot be preempted (stopped) after it is started. For memory state synchronization, the transferred memory data (dirty pages) in previous iterative rounds will become infeasible and cause an unacceptable overhead on the DCNs. Thus, we need to add the following constraint to the binary variable:

$$X_k^{\tau_i} \geq X_k^{\tau_{i-1}}, 0 \leq \tau_i < \tau_k \tag{5.5}$$

Furthermore, let $Z_{j,k}$ denote the binary variable indicating whether two migrations j and k can be performed concurrently:

$$Z_{j,k} = X_j \cdot X_k = \begin{cases} 1, & indep. \\ 0, & sharing. \end{cases} \tag{5.6}$$

If migration j and k share the same pair of source and destination or network paths, thereby affecting the available bandwidth allocated to either migration, the two migrations are resource dependent (sharing). Otherwise, two independent migrations can be performed concurrently.

Let P_k denote the set of paths p available for the migration k. The relation between allocated bandwidth for migration k and available bandwidth along the path p can be represented as:

$$l_k = \textstyle\sum_{p \in P_k} x(p) \tag{5.7}$$

According to the SFC/VDC G^j and physical DCN G, the total flows including migration transmission p and reserved service virtual links p' can not exceed the capacity $u(e)$ of link e. For $\forall \tau_i, i = 0, 1, ..., K$, we calculate the available bandwidth for migration $l_k^{\tau_i}$ under the new input because the migration i is finished at time instant τ_i and the virtual links need to be rerouted due to the new placement. The constraint during time interval $\tau = [\tau_i, \tau_{i+1})$ can be represented as follows:

$$\sum\nolimits_{p \in P_e} x(p) + \sum\nolimits_{p' \in P_e} x(p') \leq u(e), \forall e \in E \tag{5.8}$$

Moreover, the allocated bandwidth for migration k cannot exceed the interface capacity of source and destination hosts. There is no allocated bandwidth before the migration begins and after it is completed. Thus, we have the constraints expression as follows:

$$l_k \leq \min\left\{C_s^k, C_d^k\right\} \tag{5.9}$$

$$l_k^\tau \leq X_k^\tau \cdot \Psi \tag{5.10}$$

where C_s^k, C_d^k denote the interface capacity of source and destination host. $\Psi \in \mathbb{R}^+$ is a constant larger than the maximum bandwidth of paths in the network that could be allocated to the migration.

In addition, as shown in Section 5.2.2, if the allocated bandwidth for the first few transmission rounds is smaller than the dirty page rate, there will be a huge performance degradation. Thus, we add the extra constraint to $l_k^{\tau_0}$ for the migration start:

$$l_k^{\tau_0} > r_k \tag{5.11}$$

The problem of minimizing the total migration time and SLO violations during the scheduling can be formulated as:

$$\min\left(\sum_{i=1}^{K} X_K^{\tau_i} \cdot (\tau_i - \tau_{i-1}) + \sum_{k=1}^{K} (\tau_k - D_k)\right) \tag{5.12}$$

subject to constraints (5.3)-(5.11). The commonly used notations in the chapter are shown in Table 5.2.

The problem is NP-hard to solve because it generalizes the data migration problem [29] without the extra constraints of resources and migration deadline. The model in [30] also represents the same problem, but it didn't consider the impact of flow relocation on the performance of remaining migrations. They are all proved to be NP-hard. Solving the MILP problem in a reasonable time is not feasible, because the general algorithms supported in MILP solver will lead to extremely high time complexity.

5.3 Algorithm Design

In this section, we describe the details of our algorithm. The proposed deadline-aware multiple migration planning algorithm has two main components: the migration group planning and the on-line scheduler. Observations and algorithm rationales are as follows:

- Since live migration is highly dependent on available network bandwidth, migrations with different network paths, source and destination hosts can be performed concurrently. The scheduling algorithm should maximize the number of resource-independent tasks migrating at the same time. In addition, for a single migration, multi-path transmission can improve performance.

- Due to the computational overhead, migrations with low dirty page rate and small VM memory size can be migrated in parallel through the same paths by treating them as one migration as discussed in Chapter 3. On the other hand, for migrations with large memory and dirty page rate, the sequential schedule for resource-dependent migrations can optimize the total migration time.

- One physical host interface can only receive one and send one migration at the same time, i.e, one pair of ordered source and destination hosts (s_j, d_j) can only be assigned to one migration at the same time.

- After each migration completion, the network resources used by both migrations and cloud services will change. For migration plans such as consolidation, migrations with small execution time quickly free up more bandwidth for subsequent migrations, thereby reducing the total migration time. On the other hand, migrations that negatively affect network bandwidth will increase the execution time of other migrations.

- If the available bandwidth is smaller than the dirty page rate, the migration should not be started as the accumulated dirty pages will become bigger after each round of memory copy, resulting in unacceptable migration execution time and downtime.

5.3.1 Multiple Migration Planning

The proposed heuristic algorithm for concurrent migration group is shown in Algorithm 4. Given the input of migration requests in terms of flavor, dirty rate, compression ratios, the scheduling windows of migration tasks (deadlines), and the pair of source and destination host, the algorithm will return the ordered list of concurrent migration groups where each group is the maximal resource-independent migration group.

First, we need to assign the deadline to each SLO-related migration task based on the Equation (5.1) and (5.2). Secondly, considering the computing overheads, the migration tasks need to preprocess the integrated network-sharing migrations that suit the parallel method. In other words, if the migration time not exceeds the deadline and the total migration time is reduced, the scheduler will perform the parallel method for such migration.

From **line 2-5** in Algorithm 4, the dependency graph G_{dep} is created for all feasible migration tasks. If two tasks share migration resources (dependent), the edge (n_j, n_k) will be added into G_{dep}. As we allow multi-path transmission for memory copying, not only the ordered pair of source and destination (s, d) but also intersected network paths of migrations with different (s, d) will be shared. Therefore, whether two migrations can be concurrently performed is described in Algorithm 6, where P_k denotes the set of paths that can be allocated to migration k, $u(P_k)$ denotes the total available bandwidth, and C_s^k, C_d^k denote the interface capacity of source and destination hosts. In addition, $X_j * X_k = 0$ denotes that migration j and k share resources (dependent). Otherwise, the two migrations with $X_j * X_k = 1$ can be performed concurrently.

From **line 7-13** in Algorithm 4, we divide the dependency graph G_{dep} into the largest complete dependency subgraph of the remaining graphs $\{N_{dep}\}$, where each migration is dependent on others. One migration j exist and can only exist in one complete sub-

Algorithm 4: Heuristic graph-based algorithm of concurrent migration grouping

```
Input: $\{n : s_n \rightarrow d_n\}$
Result: migGroups $\{G_{mig}^{S_q}\}$
$\{$Creating Dependency Graph $G_{dep}$ of Mig Tasks$\}$
$G_{dep} \leftarrow null;$
foreach $n_j$ in FeasibleMigs do
    for $n_{j+1}$ in FeasibleMigs do
        if IsIndependent($m_k, m_j$)==0 then
            addEdge($G_{dep}$,($m_k, m_j$));
$\{$Creating resource-dependent complete subgraphs$\}$
$\{N_{dep}\} \leftarrow \emptyset$;
foreach $n_j \in G_{dep}$ do
    if IsVisited($n_j$)==False then
        $N_{dep}^{j} \leftarrow \{n_j\}$;   //complete graph contains mig $j$;
        SetVisited($n_j$) $\leftarrow$ Ture;
        CREATECompleteDepGroup($n_j, G_{dep}, N_{dep}^{j}$);
        $\{N_{dep}\} \leftarrow \{N_{dep}\} \cup N_{dep}^{j}$;
$\{$Scoring and Sorting each node$\}$
foreach $N_{dep}^{i} \in \{N_{dep}\}$ do
    foreach $n_j$ in $N_{dep}^{i}$ do
        $cost\,(n_j) \leftarrow \alpha \cdot T_{mig}^{j} + \beta \cdot \left(T_{mig}^{j} - D_j\right) + \gamma \cdot I_j$;
    $N_{dep}^{i} \leftarrow$ sorting($N_{dep}^{i}$,{cost($n_j$)});
$\{$Get migration groups from node-weighted subgraphs$\}$
return $\{G_{mig}^{S_q}\} \leftarrow$ GETConcurrentGroup($\{N_{dep}\}$);
```

Algorithm 5: Creating concurrent migration groups

```
GetConcurrentGroup($\{N_{dep}\}$):
$S_q \leftarrow 0$;   //scheduling priority for migration groups;
$G_{mig}^{S_q} \leftarrow \emptyset$;
foreach $n_j \in \{N_{dep}\}$ do
    $new \leftarrow Ture$;
    for $s = 0$ to $S_q$ do
        $flag \leftarrow True$;
        foreach $n_k \in G_{mig}^{s}$ do
            if $getEdge(n_j, n_k, G_{dep})$ then
                $flag \leftarrow False$;
        if $flag == True$ then
            $G_{mig}^{s} \leftarrow G_{mig}^{s} \cup \{n_j\}$
            $new \leftarrow False$;
            break;
    if $new == True$ then
        $S_q \leftarrow S_q + 1$;
        $G_{mig}^{S_q} \leftarrow \{n_j\}$;
    delete($G_{dep}, \{N_{dep}\}, n_j$);
sorting($\{G_{mig}^{S_q}\}, \sum cost(n_j \in G_{mig}^{s})$);
return $\{G_{mig}^{S_q}\}$
```

graph $n_j \in N_{dep}^{i}$ as the complete dependency subgraph $|N_{dep}|$ is the largest. Between complete subgraphs, there are links remained according to the original dependency graph. The corresponding recursive algorithm is described in Algorithm 7.

From **line 15-18**, in each complete subgraph, we calculate the score of each migration (line 17) and sort them from the smallest to the largest based on the score. For the function of migration cost $cost(n_j)$, it is the weighted sum of the migration time (Equation (4.1), (3.4), and (4.3)), minus slack time, and the impact of migration j on other migrations, where α, β, γ are coefficients. In our algorithm, as the cost of each individual migration is evaluated separately, we categorize the benefit of single migration into direct and potential impact $I_j = a \cdot I_j^{direct} + b \cdot I_j^{potent}$, where $a + b = 1$. The direct impact of

Algorithm 6: Check independence of two migrations with multi-paths and interface constraints

```
Input: $P_k, P_j, (s_k, d_k), (s_j, d_j)$
Result: $X_j * X_k = 1$ or 0
Function IsIndependent($m_k, m_j$):
if $s_k == s_j$ and $d_k == d_j$ then
    return $X_j * X_k = 0$;
else
    if $s_k \neq s_j$ and $d_k \neq d_j$ then
        if $u(P_j) - u(P_j \cap P_k) \geq \min\left(u(P_j), C_s^j, C_d^j\right)$ and
         $u(P_k) - u(P_k \cap P_j) \geq \min\left(u(P_k), C_s^k, C_d^k\right)$ then
            return $X_j * X_k = 1$;
        end
    else
        return $X_j * X_k = 0$;
    end
end
```

migration j can be represented as:

$$\begin{aligned} I_j^{direct} = & \left(\sum_{n_k \in \{N_{dep}\} - n_j} {T_{exe}^k}' - \sum_{n_k \in \{N_{dep}\}} T_{exe}^k \right) \\ & + \left(\sum_{n_k \in \{N_{dep}\} - n_j} \left({T_{mig}^k}' - D_k \right) - \sum_{n_k \in \{N_{dep}\}} \left(T_{mig}^k - D_k \right) \right) \end{aligned} \tag{5.13}$$

where $\{N_{dep}\}$ is the set of all complete dependency subgraphs. ${T_{exe}^k}'$ and ${T_{mig}^k}'$ denotes the execution time and the migration time after the migration n_j is completed. If the migration n_k and n_j are resource dependent, ${T_{mig}^k}'$ will be the sum of ${T_{exe}^k}'$ and T_{exe}^j.

The potential impact considers the possibility of decreased migration time when the bandwidth of some parts of the migration paths increases. Then, it can be represented as:

$$I_j^{potent} = \sum_{n_k \in \{N_{dep}\} - n_j} \sum_{p \in P_k} \frac{|\{\hat{e}\}|}{|p|} \cdot \left(T_{mig}^{k,u(\bar{e})} - T_{mig}^k \right) \tag{5.14}$$

where $|\{\hat{e}\}|$ is the number of links with increased bandwidth and $\hat{e} \in p$, $p \in P_k$. The migration time $T_{mig}^{k,u(\bar{e})}$ is based on the minimal increased bandwidth among the links

Algorithm 7: Create Complete dependency Subgraph

```
CREATECompleteDepGroup($n_j, G_{dep}, N^j_{dep}$):
    $N^j_{adj} \leftarrow$ adjacency($G_{dep}, n_j$);
    SetVisited($n_j$) $\leftarrow$ *Ture*;
    **for** $n_k \in N^j_{adj}$ **do**
        **if** *IsVisited($n_k$)==False* ***and*** *IsCompleteGraph($N^j_{dep}, n_j$)* **then**
            $N^j_{dep} \leftarrow N^j_{dep} \cup \{n_k\}$;
            CREATECompleteDepGroup($n_k, G_{dep}, N^j_{dep}$);
    **if** $|N^j_{dep}|$ *larger than previous* **then**
        **return** $N^j_{dep}$
```

$u(\bar{e}) = \min(u(\hat{e}))$.

In the final step (line 18), the cost-driven algorithm creates concurrent migration groups (Algorithm 5), where the selected migrations are resource independent. As shown in Algorithm 5, according to the sorted $N^j_{dep} \in \{N_{dep}\}$, it will always first select a migration n_j with the lowest score from each complete dependency subgraph $N^j_{dep} \in \{N_{dep}\}$ (line 3). If there is no migration group feasible for n_j ($new == true$), it will create a new concurrent migration group G^s_{dep}. After adding the migration to one migration group G^s_{mig}, it will be deleted from the dependency graph G_{dep} and the corresponding subgraph $N^j_{dep} \in N_{dep}$ (line 17). In line 18, migration groups are added to the sorted list from minimum to maximum score in seconds.

When additional migration tasks arrive after the initial processing, the on-line migration scheduler will first remove the node from the migration dependency graph after completing one migration. If additional migration tasks arrive, our proposed algorithm will add the new tasks to the existing migration dependency graph. The planning algorithm will also remove the ongoing migrations from the dependency graph. Then, it recalculates the plan based on the current system status (available network and computing resources).

Algorithm 8: Updating and scheduling feasible migrations

```
Data: migGroups, currentGroupNum
Result: Start feasible migrations and groups
foreach G in migGroups do
    groupNum = getGroupNum(G);
    if groupNum¡= currentGroupNum then
        foreach mig in G do
            if isMigFeasible(mig) then
                processMigrationStart(mig);
{Scheduling migration in subsequent group};
if hasNext($G_{current}$) then
    $G_{next}$ = getNextGroup($G_{current}$);
    flag = $False$; for mig in $G_{next}$ do
        if isMigFeasible(mig) then
            preocessMigrationStart(mig);
            flag = true;
    if flag then
        $G_{current}$ = $G_{next}$;
else
    if size(inMigrationList) == 0 and size(migPlan)==0 then
        setTotalMigTime(migPlan);
```

5.3.2 Time Complexity Analysis

Let N denote the total migration tasks number. Then, the process for creating dependency graph requires $O(N)$. For the breadth-first research in dependence graph to create complete subgraphs, it requires $O\left(N + N(N-1)/2\right)$. Let E denote the total number of physical links. Then, the time complexity of cost function (Line 16) is $O(NE)$. Thus, The worst case time complexity of scoring and sorting is $O\left(N^2E + N\log(N)\right)$. The worst case for creating concurrent migration group is $O\left(N^2\right)$. Therefore, the time complexity of worst case of Algorithm 4 is $O\left(N^2E\right)$.

5.3.3 On-line Migration Scheduler

For the real data center environment, the network workloads vary greatly over time. Therefore, it is impracticable to set the start time of each migration just based on the prediction model and the available bandwidth at the current time $\tau = 0$. The proposed on-line migration scheduler can dynamically schedule the subsequent migrations at the end of each migration.

The algorithm used by the SDN-enabled migration scheduler is shown in Algorithm 8. It includes two steps: 1) check feasible migrations in the previous and current migration groups; 2) start all feasible migrations in the next group. By only considering to start the next migration group in the ordered list at each time, it prevents the occurrence of priority inversion. The priority inversion refers to the migration group with a higher score (lower priority) may start to migrate before the group with a smaller score.

5.4 Performance Evaluation

In this section, we first introduce the various scenarios and parameters to be evaluated in both inter and intra-datacenter environments. Then, we analyze the results and conclude the experiments. We compare the performance of SLAMIG with the one-by-one scheduling and other two state-of-art algorithms [29, 30]. The results indicate that our proposed algorithms achieve good migration performance in terms of the total migration time, total transferred data, average migration time, and average downtime, meanwhile can efficiently reduce the deadline violations during the multiple live migrations. Furthermore, we evaluate and analyze the impact of multiple migration planning and scheduling on the energy consumptions and the QoS of applications.

5.4.1 Evaluation Scenarios and Configurations

In this section, we list the details of various evaluation scenarios and corresponding setups regarding the physical datacenter topologies, virtual topologies (applications), and workloads.

For the physical data center topology, we evaluated the performance of multiple migrations planning algorithms in both (1) WAN environment for Inter-Data Centers Network [39] and (2) Intra-Data Center Network (FatTree). The three-tier 8-pod FatTree [31] intra-data center network consists of 128 physical hosts with the configuration of

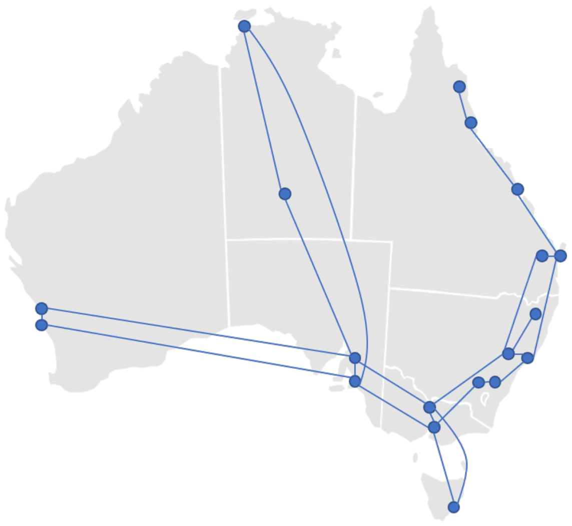
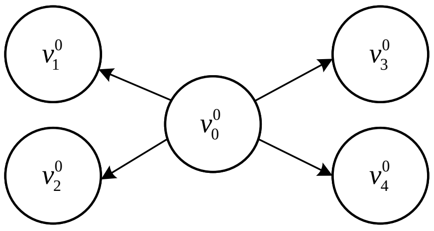


Figure 5.5: AARNET as the inter-datacenter WAN

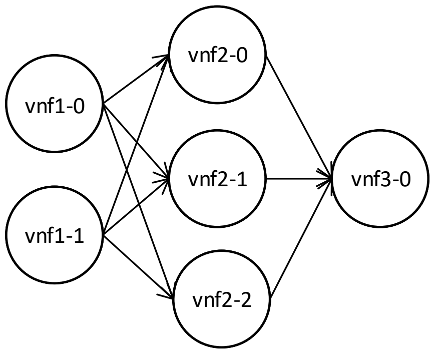
24 cores, 10000 MIPS each, 10240 GB RAM, 10PB storage, and 10 Gbps for all physical links. The inter-data center network used in the experiment is shown in Fig. 5.5. Each link between routers has 10 Gbps bandwidth. Each router as the gateway connects to the local data center cluster through the 40 Gbps link. Each local data center includes 2048 hosts with the same configuration of the one in FatTree which designed to be sufficient for all instances during the experiments.

Regarding the types of virtual topology (application), we selected them by different flavors and connectivity. Table 5.1 illustrates the flavors we used for different applications, such as multi-tier web applications and SFCs. In general, we generated 4 different types of virtual topologies: (1) *single*; (2) *star-to-slave*; (3) *sfc*; and (4) *wiki* (multi-tier web application with SFCs).

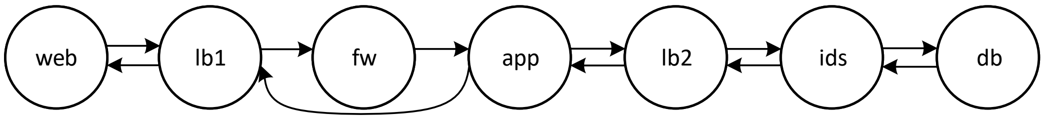
There is no connection or network communication between VMs in the *single* topology. For every group of *star-to-slave*, there is one master instance that connects to other slave instances in a star fashion. The network requests and workloads are only sent from the master to the slave instance. Figure 5.6(a) indicates a *star-to-slave* virtual topology where v_0^0 is the master instance and v_1^0 to v_4^0 are the slave instances. The *sfc* consists of VNFs chained together where each tier can have multiple identical VNFs with the same function. The workloads are sent evenly to the VNFs with same function as shown in Fig. 5.6(b).



(a) *star-to-slave*



(b) *sfc*



(c) *wiki* for 3-tier web application with SFCs

Figure 5.6: Virtual Topologies used in the experiments

Each request generated between two VMs/VNFs in *star-to-slave* and *sfc* experiments consists of three parts: computing workload in the sender VM (instruction numbers), data transmission workload (bits), and computing workload in the recipient VM. The request is first processed in the sender VM. Then, network data is generated and sent to the recipient VM. Finally, the recipient VM processes the request. The service request arrival time of *star-to-slave* and *sfc* experiments are generated in a finite time interval based on the Poisson distribution with a mean as 20 and 200 per second, respectively. Each packet size (*pSize*) is generated in the normal distribution with *pSize* as the mean value and $0.1pSize$ as the variance. The CPU processing workloads in the sender and recipient are generated based on the given workload size (*loadsize*) of request sender and recipient in the normal distribution with *loadsize* as the mean value and $0.2loadsize$ as the variance. The *pSize* of each packet is 5 Mbits. The *loadsize* for request sender and recipient is 100 and 50, respectively.

In the scenarios of *wiki*, we simulate the three-tier web applications consisting of web (web), application (app), and database (db) servers. We generate synthetic workloads based on Wikipedia trace [184] following the three-tier application model [201]. Network traffics between servers is forwarded to different types of VNFs: Load Balancer (lb), Firewall (fw), and Intrusion Detection System (ids). The configuration of different types of servers and VNFs are shown in Table 5.1 and 5.3. As shown in Figure 5.6(c), flows from the web servers are forwarded to VNF lb1 then fw before reach to the app servers. Meanwhile, flows from the app servers are forwarded to VNF lb2 and ids before reach to the db servers. For those flows coming back to the web servers from db servers, they need through VNFs ids and lb2 then app servers and the VNF lb1. In addition to those general VM specifications, VNFs have a specific field named MIPO (million instructions per operation)[166], which models the throughput of the VNF. The MIPO specifies the CPU workload length for a single network operation provided by a VNF. Thus, it can provide the throughput of the VNF along with the MIPS. For example, a VNF with 10000MIPS and 10MIPO can handle 100 operations (request) per second. We assign MIPO to Load Balancer, IDS, Firewall as 20, 200, and 800, respectively.

5.4.2 Results and Analysis

In this section, we evaluate the performance of our proposed algorithms SLAMIG through several experiments, including **migration performance**, **QoS awareness**, **deadline awareness**, and **energy consumption**. In order to compare with other multiple migration scheduling algorithms [29, 30], we use the similar simulation settings in terms of initial placement and dynamic resource management policy. Using the settings, we highlight the benefits of our multiple migration planning and scheduling algorithm compared to other algorithms. Note that, given the multiple migration requests provided by the dynamic resource management policies, multiple migration planning and scheduling algorithms are not confined to any specific resource reallocation scenario. The initial placement of all instances are generated in the way that connected VMs and VNFs are distributed among hosts in Least Full First (LLF) manner. The dynamic resource algorithm generates migration requests to consolidate all connected VMs and VNFs into the

Table 5.3: Experiment scenarios profile of *wiki*

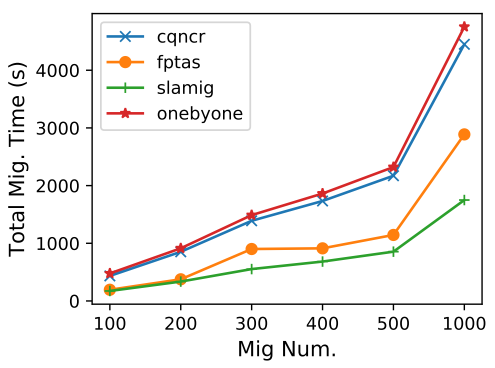
Scen-	VNFs #				VMs #			Reser. bw	Target Rate	Mig.
arios	lb1	fw	lb2	ids	web	app	db	(Mbps)	(Request/s)	#
wiki-s1	1	3	1	3	8	24	2	2	7.8402	34
wiki-s2	2	6	2	6	32	96	8	2	1.9601	118
wiki-s3	2	6	2	6	80	240	20	2	1.5680	180

same host as compactly as possible, and if not, allocate them to the most full hosts. The configuration can simulate a large amount of resource contention between the multiple migration requests for the dynamic resource management to efficiently utilize the cloud resources. We compare the performance of SLAMIG with the one-by-one migration policy as the baseline and the other two state-of-art algorithms. One algorithm (CQNCR) [29] migrates VMs by groups. The other is the approximation algorithm (FPTAS). It optimizes the total migration time by maximizing the total assigned network bandwidth to migrations [30].

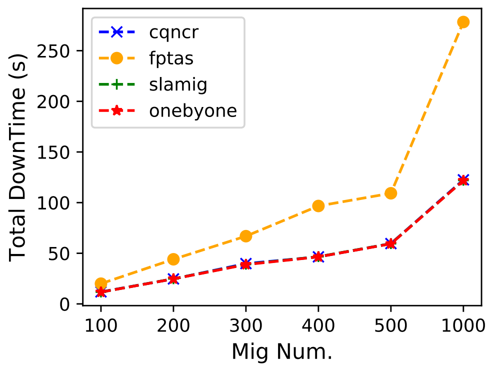
Migration Performance

In this section, we evaluated the migration performance in terms of total migration time, total downtime, average execution time, total transferred data, and processing time. In experiment *single*, we randomly generated a total of 100 to 1000 instances with flavor from micro to large in the inter-data center topology or Wide Area Network (WAN) (Fig. 5.5). We use the dirty page factor in the simulation experiments, which is the ratio of the dirty memory rate (bits per seconds) to the total memory of the VM (bits) being migrated. For the scenarios of migrating instances with low and high dirty page rate, we randomly generate the dirty page factor from 0.01 to 0.05 and from 0.01 to 0.15, perspectively. The dirty page rate (Gbps) is the product of the total memory size and the dirty page factor.

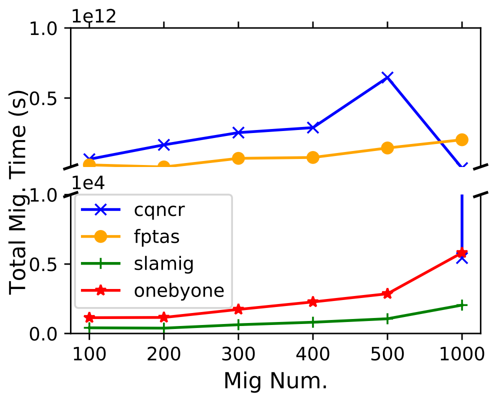
Furthermore, we evaluate the migration performance of *wiki* scenarios in FatTree. Table 5.3 illustrates the details of three scenarios in the *wiki* experiment, including the virtual topologies of SFCs and multi-tier web applications, reserved virtual bandwidth, the request arrival rate, and the number of migration tasks. The dirty page factor is set as 0.001 for all instances.



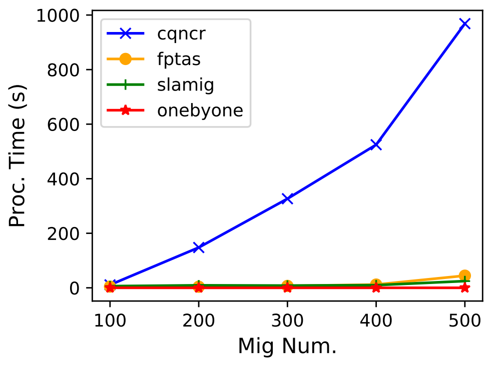
(a) total migration time with low dirty rate



(b) total downtime with low dirty rate



(c) total migraion time with high dirty rate



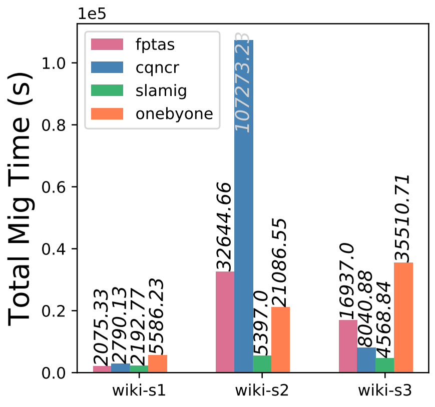
(d) processing time with low dirty rate

Figure 5.7: Live migration of non-connected VM (*single*) in AARNET

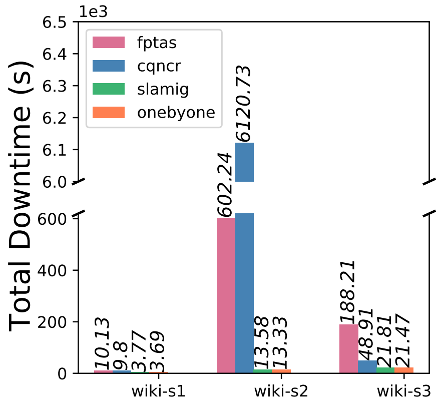
Single VM Topology in Inter-Data Centers: First, we evaluated the migration performance in a large scale manner from 100 to 1000 total migration tasks (Fig. 5.7). The results indicate that in SLAMIG can achieve the best total migration time without scarifying the downtime performance in both high dirty page rate and low dirty page rate cases. Regarding the processing time of multiple migration planning, our algorithm is less time consuming than the approximated algorithm (FPTAS) and iterative heuristic grouping (CQNCR). In the scenarios of the low-dirty-page-rate single experiment, the total migration time of SLAMIG is 62.85% to 63.69% less than the baseline and 10.50% to 39.41% less than the FPTAS. By starting the migrations group by group at the same time, the total migration time of CQNCR is only marginally smaller than the result of the baseline (maximum 9.06%). Meanwhile, as shown in Fig 5.7(b), the total downtime of SLAMIG is at least 40.27% and at most 55.87% less than the FPTAS.

For the result of algorithm running time, we observe that the SLAMIG algorithm can significantly reduce the computation time compared to solving the approximate MIP problem in FPTAS. In addition to Fig. 5.7(d), when there are 1000 migration tasks, the processing times of CQNCR, FPTAS, SLAMIG are 15471.29, 89.94, and 30.23 seconds respectively. When performing 500 migration tasks, the processing time of SLAMIG (24.64s) is 44.70% less than that of FPTAS. The runtime of sequential scheduling is less than 1 second, because in our experiments, the available sequence only needs to be calculated once as all sequential combinations are schedulable. For the algorithm CQNCR, after updating the network bandwidth and computing resources in each round, it iteratively groups the migrations in a greedy manner and selects the migration group with the most positive impact. Thus, when the number of migration tasks increases, the processing time will increase dramatically (Fig. 5.7(d)). Compared with CQNCR, our proposed algorithm can calculate all concurrent migration groups in one round. Since each migration task has been given a weight in the dependency graph, we also generate the largest possible migration group with minimal weight. Therefore, it can achieve better performance in total migration time. Note that in our algorithm, generating a migration dependency graph takes up most of the processing time in multiple routing environments. For the single routing environment such as FatTree, we only need to check the source and destination hosts, which will further reduce the processing time.

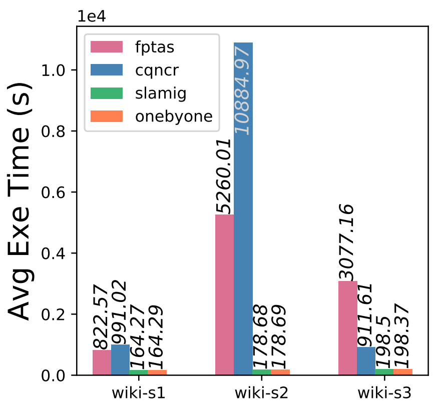
Fig. 5.7(c) shows the details of the experiment of single instances with a high dirty rate. Compared with the other two algorithms, SLAMIG can maintain the performance of the total migration time. By allowing other migration tasks to be initiated when there is a small amount of bandwidth to maximize the overall network transmission rate, FPTAS may cause significant performance degradation in both total migration time and downtime. In the worst case, the total migration time shown is even greater (10^6 times) than the result of one-by-one scheduling. Moreover, all migration start times are based



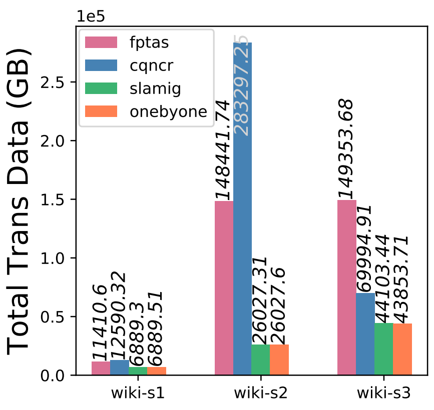
(a) total migration time



(b) total downtime



(c) average execution time



(d) total transferred data

Figure 5.8: Live migration of multi-tier applications with SFCs (*wiki*) in FatTree

on the prediction model in CQNCR. Inevitably, in the worst case, several migrations will start when their resource-dependent migration tasks haven not been completed, which will cause the allocated bandwidth to be less than the dirty page rate. In other words, the allocated bandwidth is insufficient to converge the migration in the worst case.

Web Application Topology in FatTree: In the experiment of *wiki*, we evaluated the algorithm performance regarding the total migration time, total downtime, average execution time, and total transferred data during the live migrations (Fig. 5.8). In all three scenarios, the SLAMIG achieves the optimal total migration time while maintaining other migration performance criteria at the level of sequential scheduling. Compared with the baseline, the SLAMIG reduce the total migration time by 60.74%, 74.41%, and 87.13%. The results are −5.66%, 83.47%, and 73.02% less than FPTAS and 21.41%, 94.96%, and 43.17% less than CQNCR.

In some cases, such as wiki-s1 in Fig. 5.8(a), we noticed that the total migration time of FPTAS may be slightly better than our algorithm. It is because several migration tasks can be scheduled in the same paths when a small amount of bandwidth is available to maximize the overall network transmission rate. One migration can be started even the allocated bandwidth is smaller than the dirty page rate. Although the sum of migration execution time is larger, the total migration time may be smaller due to the early start time. As mentioned in Section 5.2.2, we argue that it will increase the average execution time of each migration task (Fig. 5.8(c)), resulting in larger downtime (Fig. 5.8(b) and 5.7(b)) and total transferred data (Fig. 5.7(d)).

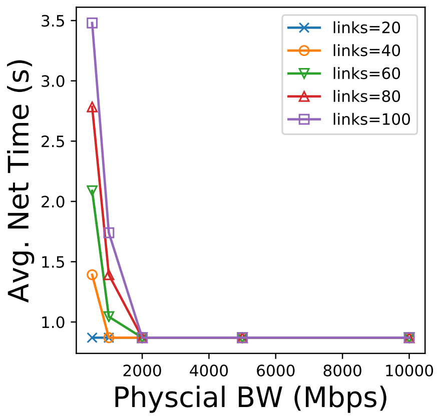
Considering total downtime, average execution time, and total transferred data, we should concurrently schedule the resource-dependent migration tasks to alleviate the impact of multiple live migrations on the system and guarantee the QoS of the migrating instances. The results indicate that there is no statistical difference between SLAMIG and the sequential scheduling in these parameters. However, the FPTAS and CQNCR drastically increase the total downtime by 1.75/1.66, 44.19/458.23, 7.77/1.28 times, the average execution time by 4.006/5.03 times, 28.44/59.92, 14.51/3.60, and the total transferred date by 0.66/0.83, 4.70/9.88, 2.41/0.60 times, respectively. Although the FPTAS and CQNCR can achieve a better performance of total migration time compared to the baseline in other scenarios, bandwidth sharing among resource-dependent instances with large memory and high dirty page rate will lead to unacceptable results (wiki-s2).

Summary: (1) SLAMIG achieves the optimal migration performance in terms of the total migration time, downtime, average execution time, and total transferred data, while the processing time is less than the CQNCR iterative grouping and FPTAS approximation algorithm. (2) The prediction model of migration is used to estimate the execution time of one migration and the total migration time of a concurrent migration group. However, it is not efficient to assign a fixed start time for a live migration only based on the prediction model. In an independent migration group, the execution time varies, which leads to multiple time gaps between the completion time and the estimated start time of the next group. Moreover, in the real environment, the real-time dirty page rate may be different from the historical statistics and the current monitoring value. In a dynamic network environment, the available network bandwidth used in the prediction model may also change over time. In short, the prediction execution time is not necessarily identical to the actual time during the scheduling, which will cause two resource-dependent migrations to run concurrently. Therefore, it is essential for the online scheduler to dynamically schedule migration tasks according to the plan. (3) By maximizing the total network transmission rate, the total migration time can be reduced to a certain extent, but the optimal migration performance cannot be achieved. If one migration starts with the allocated bandwidth below its dirty page rate, it will extremely enlarge the execution time. For migrations with large dirty page rates, allocating bandwidth that is just slightly larger than the dirty page rate will still result in an unacceptable migration performance with a large downtime and memory-copy iteration round. Therefore, we should not neglect the concurrency or resource sharing dependencies between different migration tasks. (4) Regarding the performance and impact of multiple migration scheduling, total migration time is not the only parameter that needs to be optimized. The average bandwidth for each migration can also reflect the efficiency of multiple migration scheduling. A larger allocated bandwidth means smaller single migration execution and the downtime. As shown in the Equation 4.3, it will also result in fewer iteration rounds for dirty page copying. Thus, it we should also achieve better performance in terms of average bandwidth of each migration resulting in better average execution time, total/average migration downtime, and total/average transferred data.

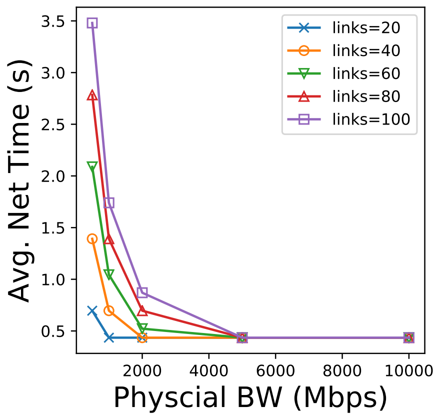
QoS-Aware

In this experiment, we evaluated the impact of multiple migration planning on QoS in terms of the network transmission time of application requests.

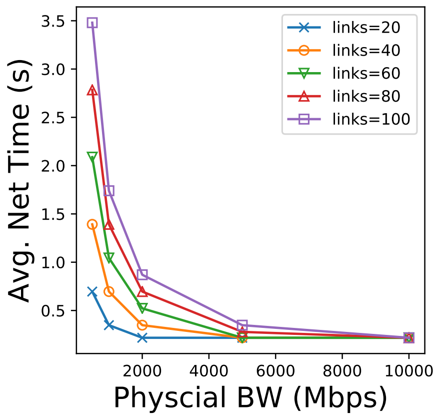
There are three network bandwidth sharing policies to manage the migration flow in Section 5.4.2: (1) *free* used by FPTAS; (2) *reserved* used by CQNCR; and (3) *ratio* used



(a) 20Mbps reserved



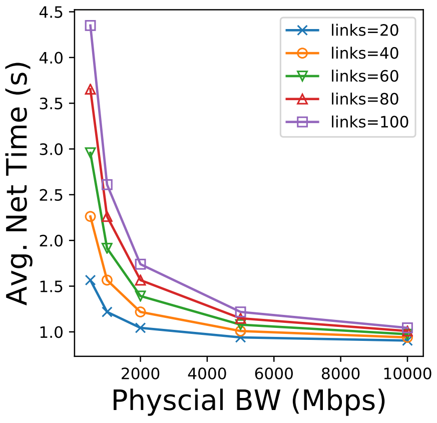
(b) 40Mbps reserved



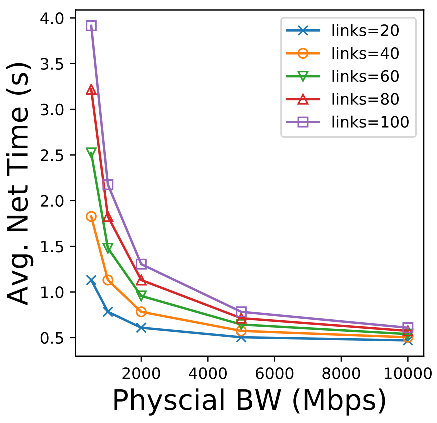
(c) 80Mbps reserved

Figure 5.9: Average network transmission time without live migration with different number of links and reserved virtual link bandwidth under *ratio* bandwidth sharing policy

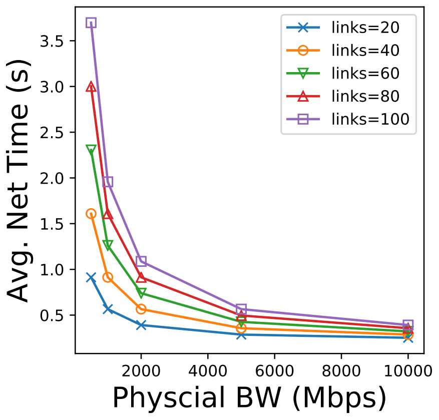
by SLAMIG and OneByOne. The bandwidth sharing solutions proposed by FPTAS and CQNCR which do not consider the bandwidth competition can only be adopted in an ideal scenario where the remaining bandwidth for the live migration is sufficient. For the *free* policy, the live migration can only utilize the available bandwidth along the network paths left by other service traffic. For the *reserved* policy, the live migration only use the remaining unreserved bandwidth left by other virtual links. The available bandwidth reserved by other services can not be allocated to the migration flow. Therefore, under the free or reserved bandwidth sharing policy, the live migration flow will not



(a) 20Mbps reserved



(b) 40Mbps reserved



(c) 80Mbps reserved

Figure 5.10: Average network transmission time during live migration with different number of links and reserved virtual link bandwidth under *ratio* bandwidth sharing policy

affect the network transmission time of other services in terms of network bandwidth competition. Note that in the separated control (migration) network [103], the migration flow will not affect the bandwidth allocation of service traffic. However, we argue that the *free* and *reserved* policies can only be adopted when the remaining bandwidth for the live migration is sufficient to converge the live migration in time. Furthermore, in some worst cases, as shown in Fig. 5.8, the massive downtime caused by the *free* or *reserved* policy will seriously affect the request response time of the migrating service.

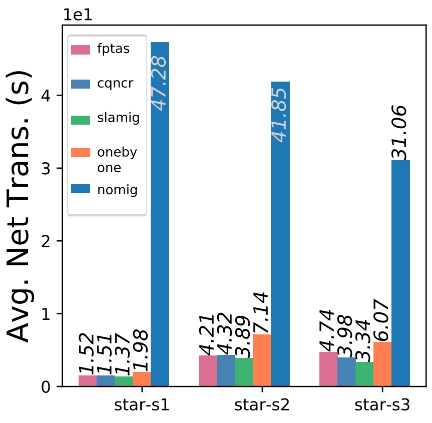
When other service traffic and migration flows compete on the network bandwidth, Chapter 3 shows the effect of single live migration on the service response time of the migrating VM. Chapter 3 also evaluates the impact on the TCP and UDP traffic and [26, 90] investigate the effect on other service traffic during the migration. For the *ratio* policy, the actual allocated bandwidth of a network flow is based on the ratio of the reserved bandwidth of the flow to the total bandwidth demand along the network path. It is practical to use *ratio* bandwidth sharing policy when the remaining bandwidth for the migration flow is insufficient to converge the migration or it is urgent to finish the migration to avoid QoS degradation and SLA violations.

With the *ratio* policy, we first explain the principle of the impact of live migration on the network traffic between VMs. In the experiment, we control the number of virtual links between VMs along the network path of one migration. The network traffic between two VMs is generated based on the *wiki* workload. Figure 5.9 illustrates the average network transmission time of network traffic between VMs, where reserved bandwidth size for each virtual link, the total number of virtual links in the evaluating network path, and the available bandwidth of the evaluating network path are controlled variables. The results indicate that when the total bandwidth of reserved virtual link is lower than the physical bandwidth, the reserved bandwidth of each virtual link can be satisfied. As the number of links increases, the actual bandwidth allocated for each virtual link decreases, which leads to the longer network transmission time. Figure 5.10 shows the average network transmission time when the service traffic is sharing the bandwidth with one live migration under the *ratio* policy. In our experiments, the reserved bandwidth for live migration is equal to the physical network bandwidth. As the physical network bandwidth increases, the impact of live migration on the network transmission time of other service traffic decreases. Furthermore, it also indicates that when the number of virtual links along the migration path or the reserved bandwidth for each virtual link increases, the live migration has less impact on the network transmission time of service traffic.

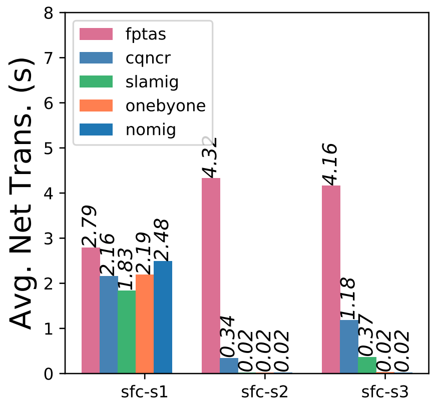
To demonstrate the performance of different migration scheduling algorithms with *ratio* bandwidth sharing policy, in addition to the *wiki* experiment configuration in the FatTree data center network, we also added the experimental results from two types of virtual topologies: (1) *start-to-slave* and (2) *sfc*. The *star-to-slave* and *sfc* experiments are

Table 5.4: Simulation configurations of *star-to-slave* and *sfc* experiments

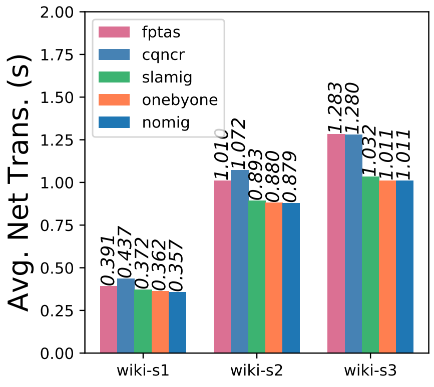
group	startoslave	vm	link bw (Mbps)	Mig #	sfc	vnf	link bw (Gbps)	Mig #
5	star-s1	25	100	19	sfc-s1	21	1.0	19
10	star-s2	50	100	37	sfc-s2	43	1.0	40
15	star-s3	75	100	55	sfc-s3	69	1.0	65



(a) *star-to-slave* in WAN



(b) *sfc* in WAN



(c) *wiki* in FatTree

Figure 5.11: Average network transmission of application requests under *ratio* policy

both evaluated in the inter-data center network. Table 5.4 describes the configuration of the group number, instance number, link reserved bandwidth, and the number of migration tasks in these two experiments. We set up the network resources in the way that network traffic within the host can take full advantage of the reserved bandwidth of the virtual link between VMs/VNFs. For the master instance with small flavor, the dirty page factor is 0.12, and for the slave instance with tiny flavor and a VNF with large flavor, the dirty page factor is 0.02.

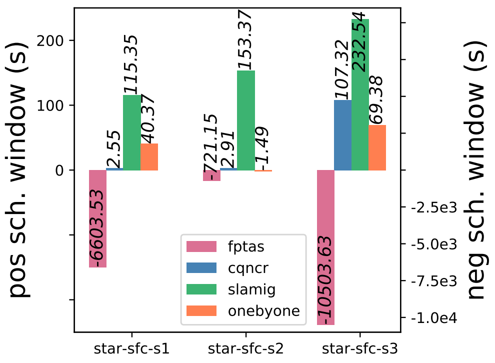
Fig. 5.11(a) and 5.11(b) demonstrate the average network transmission time of applications in the initial placement (nomig): (1) In the *star-to-slave* experiment, applications experience large delay from master to slave instances; (2) In *sfc* experiment, the network transmission time between applications is small in the initial placement. The average network transmission time is 2.48s, 0.02s, and 0.02s, respectively.

The results of *star-to-slave* indicate that the consolidating migrations can efficiently reduce the delay encountered by the application. The SLAMIG achieves the minimal average network transmission time of application requests in all three scenarios which are 0.14s, 0.32s, and 0.64s less than the second-best results. Compared to the non-migration situation, it can also reduce the network transmission time by 95.79%, 90.70%, and 89.25%. In the experiment of *sfc*, FPTAS excessively increases the network transmission time of application requests. As the FPTAS algorithm intends to maximize the network transmission rate of all migration tasks, it significantly reduces the transmission bandwidth among the application servers. In scenario sfc-s1, SLAMIG reduces the average network transmission time due to consolidation. Because less total migration time and average execution time will result in a shorter network transmission time during the multiple migrations. For the scenario sfc-s2 and sfc-s3, the initial placement is sufficient to provide enough bandwidth according to the virtual link reservation. SLAMIG does not increase network transmission time in sfc-s2, and only increases 0.35s in sfc-s3, which can guarantee the QoS during the multiple live migrations. For the experiment of *wiki*, SLAMIG can maintain the QoS at the same level of the sequential scheduling with *ratio* bandwidth sharing policy. However, the average transmission time of all service requests increases by 0.04s, 0.131s, and 0.272s in FPTAS and 0.08s, 0.193s, and 0.269s in CQNCR.

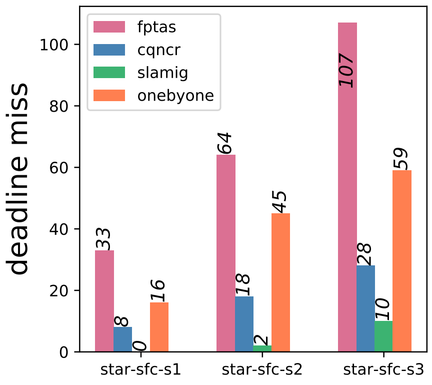
Summary: Although the migration downtime is an important parameter to evaluate the impact of migration on the migrating instances, the QoS of other services in the data center network is largely ignored. By utilizing the *free* and reserved bandwidth sharing

Table 5.5: Evaluation scenarios of deadline-related migrations

name	vm	nfv	D(star) (s)	D(sfc) (s)	total mig
star-sfc-5	25	23	100	300	46
star-sfc-10	50	40	200	500	87
star-sfc-15	75	75	300	800	138



(a) avg. remaining scheduling window



(b) total missing deadline

Figure 5.12: Deadline-related experiments in inter-datacenter

policy, the transmission time of application requests will not be affected. However, in the case where *ratio* bandwidth sharing policy is required to converge the migration, our proposed algorithms can minimize the impact of multiple migrations on the application, thereby ensuring the QoS and mitigating SLA violations.

Deadline-Aware

In this section, we evaluate and analyze the performance of different multiple migration plans under various urgency and priorities. In the experiment *star-sfc*, we evaluated the deadline awareness in the remaining scheduling window and the number of total missing deadlines. In Table 5.5, as shown in the QoS-aware experiment, instances in *star-to-slave* have large delays due to the burst workloads, so the deadlines are tight. Meanwhile, the deadline for migration VNFs in *sfc* with sufficient bandwidth is larger.

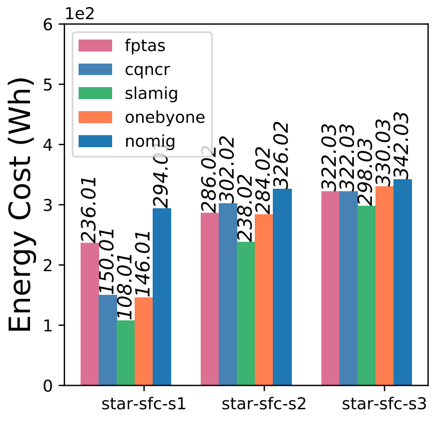
The dirty page factor is 0.02 for all instances in this experiment.

Fig. 5.12 illustrates the results of the remaining scheduling window and the total missing deadlines. By ignoring the nature of migrations with various urgency and priorities, the two algorithms (FTPAS and CQNCR) as a comparison have unacceptable performance in terms of the remaining scheduling windows and the number of migration deadline violations. The average number of remaining scheduling window of FPTAS is negative due to the large execution time by allowing insufficient migration bandwidth. In all three scenarios, SLAMIG has the most remaining scheduling window, which can reduce SLA violations and guarantee the QoS during the migration with different priorities. Compared with FPTAS, CQNCR, and the baseline, FPTAS reduces the deadline violations by 100%, 96.875%/88.89%/95.56%, and 90.65%/64.29%/83.08%.

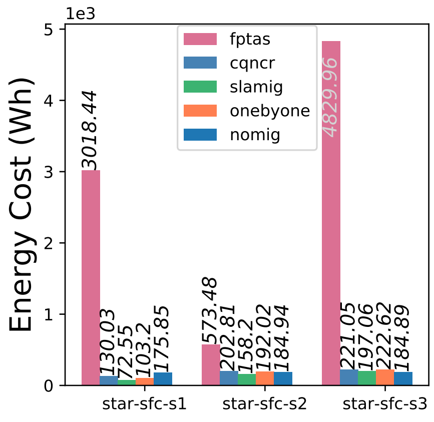
Summary: By comprehensively considering the scheduling window, execution time, and the impact of one migration, SLAMIG can efficiently reduce the deadline missing while achieving the optimal migration performance. As a result, the total number of SLO violations can be minimized. Due to the flexibility of SLAMIG, one can also change the weight function to further reduce the migration deadline violations by trading off the performance of total migration time.

Energy Consumption

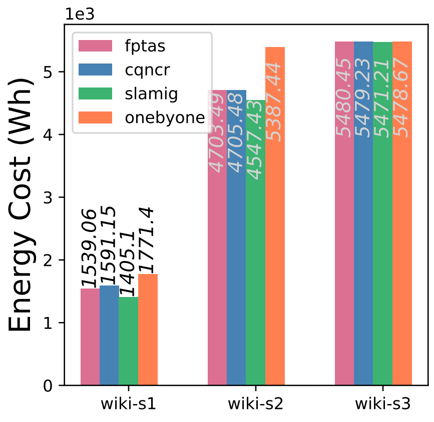
In this section, we evaluate and analyze how different multiple migration plans can affect the energy consumption of hosts and switches. Switch [202] and Host [203] power models are used to calculate the overheads of multiple live migrations in the data centers. Fig. 5.13 shows the power consumption of host and switch in experiment *star-sfc* and *wiki*. As fewer hosts are involved after the consolidation, earlier migration convergence can reduce the host power consumption. In the experiment of *star-sfc*, SLAMIG reduces the host power consumption by 63.26%, 26.99%, and 12.86% compared to the non-migration and reduces by 26.03%, 16.20%, and 7.45% compared to the second-best results. We also observed similar results of host energy consumption in wiki-s1 and wiki-s2 scenarios. In wiki-s3, due to involved hosts are consistent after migrations, there are only ignorable variances of host energy consumption among different algorithms.



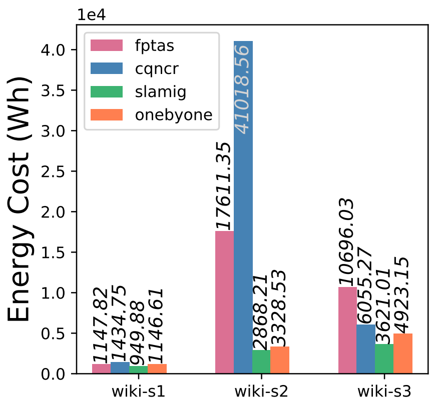
(a) host energy in star-sfc



(b) switch energy in star-sfc



(c) host energy in wiki



(d) switch energy in wiki

Figure 5.13: Energy consumption in hosts and switches

For the power consumption in networking resources (switches), the main contribution comes from the elephant flows of migrations from source to destination hosts. Another contribution comes from the application communications, where requests are sent between different physical hosts. In Fig. 5.13(b), the networking energy consumption of FPTAS is much larger than other algorithms because it allows small bandwidth allocation to maximize the global migration network transmission rate. Our proposed approach is 29.70%, 17.61%, 10.85% less than the second-best result. Fig. 5.13(d) indicates that SLAMIG also comsumes the least energy during the multiple migrations in *wiki* experiment. Compared with the sequential scheduling, SLAMIG reduces by

17.16%, 13.83%, and 26.45%. As mentioned, although the total migration time of FPTAS is smaller in wiki-s1, it costs 197.94Wh more than the SLAMIG. Therefore, the average migration execution time is also related to the migration overhead of energy consumption.

Summary: Although smaller total migration time can reduce the total energy consumption due to consolidation, maintaining the average execution time is critical to the network power consumption. Due to the heavy usage of network resources, the switches consume a lot of energy during the migration. Even though consolidation and dynamic switching off the switches and hosts can help data centers save energy, migrating high-dirty-rate instances will increase the energy consumption of switches. Therefore, multiple migration tasks must be carefully planned based on the migrating candidates, sources and destination hosts. Dynamic resource management policies also need to consider the trade-off between the optimal allocation and migration energy overheads.

5.5 Related Work

Table 5.6: Comparisons of multiple migration planning and scheduling

App-roach	deadline awareness	QoS awareness	energy consumption	concurrent mig. scheduling	online mig. scheduler	multipath routing	objectives
[204]	✓	x	x	x	x	x	reduce the dirty memory transmission
[28]	x	✓	x	x	x	x	sequence for loop-free and bandwidth constraints
[27],[90]	x	✓	x	-	-	-	select migrating VMs to minimize interference
[29]	x	✓	x	✓	x	x	total mig. time and downtime with reserved bandwidth sharing
[103]	✓	x	x	-	✓	x	converge migration tasks before deadline
[30]	x	x	x	✓	x	✓	total mig. time and downtime with free bandwidth sharing
SLAMIG	✓	✓	✓	✓	✓	✓	total mig. time, downtime, avg. exe. time, and transferred data

Akoush et al. [84] explored the important parameters, link bandwidth and page dirty rate, that affect migration performance. They conducted experiments on migration performance under various workloads and proposed two simulation models based on the assumption of average memory dirty rate and history-based dirty rate of VM to predict migration performance. There are some works on the VM migration selector to minimize the overall cost and reduce interference. Remedy [27] relied on the SDN controller to monitor the state of the data center network and predict the cost of VM migration. The VM migration controller of heuristic destination selector minimizes the migration impact on the network by considering the cost of migration, the available bandwidth for migration, and the network balance achieved after migration. iAware [90] proposed a simple and light-weight interface-aware VM live migration strategy. It jointly estimates and minimizes the overall performance overhead of both migration interference and VM co-location interference with respect to I/O, CPU, and memory resources during and after migration.

There are Few studies related to the (soft) real-time issue in live VM migration. These studies mainly focused on how to reduce the execution time of a single live migration. Tsakalozos et al. [103] studied the live VM migration with time-constraints in the sharing-nothing IaaS-Clouds, where the cloud operator can assign specific scheduling windows for each migration task. For alleviating the SLA violations, they proposed a migration broker to monitor and limit the resource consumption, that is, to reduce the dirty page rate to force certain migrations to converge on time. By investigating the computing and network resources used by single live migration, Checconi et al. [204] presented a method to delay the frequent page dirtying in order to reduce the execution time and downtime of a single live migration.

Furthermore, there are several works focus on optimizing multiple live VM migration planning. Ghorbani et al. [28] proposed a simple one-by-one heuristic VM migration planning, which did not consider parallel VM migration through different network paths. Sun et al. [150] explore the optimal planning for multiple VM migrations by mixing pre-copy and post-copy migration. Based on the fact of application network traffic direction characteristic, it maximizes the available bandwidth to improve serial and parallel migrations. Similarly, Deshpande et al. [26] improved the live migration performance by considering pre-copy or post-copy migration based on the application traffic direction. CQNCR [29] focuses on the multiple VM migration planning in one data center environment by considering the available bandwidth and network traffic cost after migration. They modeled the multiple VM migration planning based on a discrete-time model as a Mixed-Integer Programming (MIP) problem. A heuristic migration grouping algorithm by setting the group start time based on the prediction model is proposed. However, because there are different combinations of migration grouping, grouping and weighting the migration groups directly can lead to performance degradation of the total migration time. Without considering the connectivity between VMs and the change of bandwidth, FPTAS [30] simplifies the problem by maximizing the net transmission rate rather than minimizing the total migration time. In the context of SDN, the primary contribution compared to other research is the introduction of the multipath transmission when migrating VMs. As a MIP problem, they propose a fully polynomial-time approximation by further omitting certain variables. Table 5.6 summarizes the comparison of live migration planning and scheduling methods for the objectives to be migrated, and whether the deadline of different migration tasks, QoS of applications, the energy consumption of hosts and switches, concurrent migration scheduling, and enables the multiple routing of migration flows and online scheduler to manage migration tasks are considered. The dash mark indicates the parameter of the work is not relevant.

5.6 Summary

Due to the limited computing and network resources as well as migration overheads, it is essential to intelligently schedule the migration tasks in data centers to achieve optimal migration performance, while mitigating the impacts of migration on cloud services and preventing SLO violations during the migration schedule. In this chapter, we proposed a set of algorithms (SLAMIG) which includes concurrent migration grouping and the on-line migration scheduler. Instead of grouping migrations directly, SLAMIG can optimize the order of concurrent migration groups by sorting each migration based on complete dependency subgraphs. In addition to the dirty page rate, extra bandwidth constraints can significantly improve the performance. The on-line migration scheduler can guarantee the concurrency and scheduling order of different migrations in a dynamic network environment.

We argue that along with the total migration time, optimizing the average execution time, transferred data, and downtime are essential metrics to evaluate the multiple migration performance. The total migration time is more related to the time requirements (for example, migration deadlines and SLO violations), while the sum of execution time, transferred data, and service downtime are related to the actual overheads. By optimizing the total migration time, we can guarantee the SLA and dynamic performance requirements of cloud services. By optimizing the sum of execution time, transferred data, and downtime, we can guarantee the QoS of services and achieve more revenue as the cloud provider. Experimental results show that SLAMIG can efficiently reduce the number of migration deadline missing and meanwhile achieve good migration performance in total migration time, average execution time, downtime, transferred data with acceptable algorithm runtime. Furthermore, the average execution time is an essential parameter to minimize the impact of multiple migration scheduling on the QoS of applications and energy consumption.

Live container migration has been introduced to facilitate user mobility to guarantee service delays in the edge computing environment. In the next chapter, we intend to investigate the planning and scheduling algorithms for live container migration in terms of the algorithm complexity and networking management in the edge computing or cloud radio access network environment.

Chapter 6

Efficient Large-Scale Multiple Migration Planning and Scheduling

With the expansion of network topology scale and increasing migration requests, the current multiple migration planning and scheduling algorithms of cloud data centers can not suit large-scale scenarios in edge computing. The user mobility-induced live migrations in edge computing require near real-time level scheduling. Therefore, in this chapter, through the Software-Defined Networking (SDN) controller, the resource competitions among live migrations are modeled as a dynamic resource dependency graph. We propose an iterative Maximal Independent Set (MIS)-based multiple migration planning and scheduling algorithm. Using real-world mobility traces of taxis and telecom base station coordinates, the evaluation results indicate that our solution can efficiently schedule multiple live container migrations in large-scale edge computing environments. It improves the processing time by 3000 times compared with the state-of-the-art migration planning algorithm in clouds while providing guaranteed migration performance for time-critical migrations.

6.1 Introduction

The introduction of edge computing [205] brings opportunities to improve the performance of the emerging user-oriented applications by pushing computation and intelligence to end-users, including Vehicle to Cloud (V2C), Vehicle to Vehicle (V2V), Virtual Reality (VR), Augmented Reality (AR), Artificial Intelligent (AI), or Internet of Things (IoT) applications and so forth. Driven by container virtualization, microservices are

This chapter is derived from:

- **TianZhang He**, Adel N Toosi, and Rajkumar Buyya, "Efficient Large-Scale Multiple Migration Planning and Scheduling in SDN-enabled Edge Computing", *IEEE Transactions on Mobile Computing (TMC)*, (submitted, June 2021).

more suitable for dynamic deployment on edge computing [23, 206] due to smaller memory footprint and faster startup. By allocating the containerized services in the Edge Data Centers (EDCs) or Mobile Edge Clouds (MECs) [15], strict end-to-end (E2E) communication delays between end-users and services can be guaranteed.

From the centralized cloud computing framework to decentralized edge computing, surveys [7, 25] investigated the challenges faced by the infrastructure and service providers regarding dynamic resource management and user mobility. By providing non-application-specific compute and memory state management, live migration is the solution to these challenges. Live migration of VM [8] and container [207] through the open-source Checkpoint/Restore in Userspace (CRIU) software [60], which had kernel support since Linux 3.11, aims to provide little or no disruption to the running service during migrating in the edge computing. It iteratively copies unfinished computation tasks with intermediate computation states in the memory from source to destination until the memory difference between two synchronizing instances is small enough for the stop-and-copy phase. In addition, for the container image, if the image does not exist in the destination, it can be transferred from the previous EDC or remote clouds or accessed through shared storage. Thus, live migration performance highly relies on the available bandwidth of the network routing connecting source to destination.

Industrial infrastructure and service providers, such as IBM, RedHat, Google, etc, have been integrating live container migration into their productions [9, 12]. Google has adopted live VM and container migration with CRIU into Borg cluster manager [11–13] for reasons, such as, higher priority task preemption, software updates, such as kernel and firmware, or reallocating for availability or performance. It manages all compute tasks and runs on numerous container clusters each with up to tens of thousands of machines. A lower bound of 1,000,000 migrations monthly in the production fleet have been performed with 50ms median blackout [13]. Live container migration provides technical simplicity without handling state management and application-specific evictions. However, it is also identified that writing and reading to remote storage through network dominates the checkpoint/restore process and the scheduling delay is the large source of delay regarding the performance of multiple live migrations.

Recently, some works have focused on user or service mobility in mobile edge com-

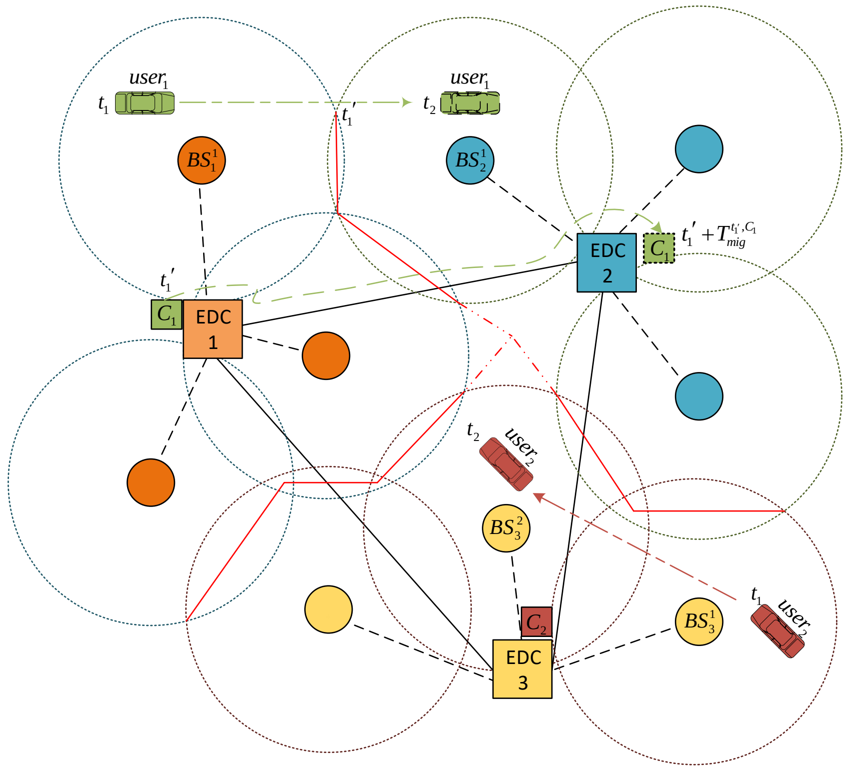


Figure 6.1: User-mobility induced live container migration in edge computing

puting through live container migration [7, 25, 189, 208–211]. With the limited coverage range of each EDC, when a user moves from one base station to another, the network latency could deteriorate after the network handover. To guarantee the Quality of Service (QoS), the service may need to be migrated from the previous EDC to the proximal one through live container migration. Figure 6.1 illustrates an example scenario where an autonomous vehicle sends workload to the corresponding stateful service in the EDC for real-time object detection. There is a total of 9 base stations assigned to 3 different EDCs. Two autonomous vehicles move to a new position from time t_1 to t_2. For vehicle user1, when leaving the range of base station BS_1^1 and entering the range of base station BS_2^1 at time t_1', there is a live container migration from $EDC1$ to $EDC2$ induced by the user1's movement. Meanwhile, as user2 crosses the range boundary of the base station BS_3^1 to BS_3^2, since the two base stations both belong to the same edge data center $EDC3$, the E2E delays of the service can be guaranteed. Therefore, there is no live migration induced by user2's movement.

In cloud computing environments, dynamic resource management policies triggers live migration requests periodically to optimize the resource usage or to maintain QoS of applications [141]. However, in the edge environment, service migration/mobility is highly relative to user mobility [25, 189, 208, 209, 211]. Migration requests of containers or VMs from services and users may share and compete for the computing and network resources, such as migration source, destination, and network routings. It brings more challenges for the multiple live migration planning and scheduling. However, most research on live service migration in edge and cloud computing neglects the actual live migration cost regarding the iterative dirty memory transmission [8] and resource competition among migrations in both computing and network resources. As a result, performing multiple live migrations in arbitrary order can lead to service degradation.

Few works focus on multiple VM migration planning and scheduling in cloud data centers [29, 30]. The framework of current migration scheduling algorithms periodically triggered by resource management policies with a long time interval is not suitable for stochastic scenarios of mobility-induced migration in edge computing. Furthermore, the network scale, the numbers of end-users and live container migration requests increase ten thousand times in edge environments. Without proper modeling, the problem complexity will increase dramatically as the number of migration requests and network scale increase. As a result, the complexity and processing time of current algorithms do not meet the real-time requirement of the live migration at scale in edge environments.

Therefore, this chapter proposes efficient large-scale live container migration planning and scheduling algorithms focusing on mobility-induced migrations in edge computing environments. It can also apply to multiple migration scheduling for the general periodical dynamic resource management at scale. The **contributions** of this chapter are summarized as follows:

- We introduce the resource dependency graph of the source-destination pair for resource competition among migration requests to reduce the problem complexity.
- We model the problem as finding the Maximum Independent Set of the Dependency Graph iteratively.

- We propose iterative-Maximal Independent Set (MIS)-based algorithms for efficient large-scale migration scheduling and prove the corresponding Thermos.
- We implement an event-driven simulator to evaluate the user mobility and live container migrations. The experiments are conducted with real-world dataset and traces.

The rest this chapter is organized as follows. We review the related work in Section 6.2. Section 6.4 analyzes and models the problem of multiple container migration scheduling. Section 6.5 proposes two main methods of large-scale migration scheduling. Section 6.6 shows the performance analysis of proposed algorithms and Section 6.7 illustrates the experimental design and evaluation of proposed algorithms with real-world dataset. Section 6.8 summarizes the chapter.

6.2 Related Work

The live VM migration realization [75] and its application in cloud data centers have been matured last few years. The research on live container migration in edge computing is an active field [7, 25]. Clark et al. [8] proposed the live VM migrations and discussed the details of pre-copy or iterative live migration. On the other hand, the research on live container migration is trending and becomes more mature in recent years. Mirkin et al. [207] represented the checkpointing and restart features for the OpenVZ container. The checkpointing function is also used for live migration. Checkpoint/Restore In Userspace (CRIU) [60] is a Linux software to migrate container's in-memory state in userspace. It is currently integrated with LXC, Docker (runC), and OpenVZ to achieve the live container migration. Nadgowda et al. also proposed [212] a CRIU-based memory migration together with the data federation capabilities of union mounts to minimize migration downtime. Similarly, Ma et al. [209, 210] utilized the layered storage feature based on AUFS storage drive and implemented a prototype system to improve the performance of docker container migration. Furthermore, several works studied the performance difference between container and VM live migration [23, 206]. Results show that the live container migration is much faster than the live VM migration

Table 6.1: Comparisons of multiple migration planning and scheduling works

research	real-time planning	large-scale	service correlations	user-mobility	deadline	SDN-enabled	Cloud DC	Edge computing
CQNCR [29]	–	–	✓	–	–	–	✓	–
FPTAS [30]	–	–	–	–	–	✓	✓	–
Our work	✓	✓	✓	✓	✓	✓	✓	✓

due to its much smaller memory footprint and fast startup features.

More research recently focuses on dynamic resource scheduling in fog and edge computing environments based on the live container migration (details in survey [7, 25]). In [189, 208], the authors modeled the sequential decision making problem of generating service migration requests using the distance-based Markov Decision Process (MDP) framework. By reducing the state space, they proposed a distance-based MDP to get the approximated results. The research [211] also investigated the same problem by using the MDP framework. The authors proposed a reinforcement learning-based online microservice coordination algorithm to learn the optimal strategy for live migration requests to maintain the QoS in end-to-end delay.

However, current algorithms can not meet the requirement of live container migrations in edge computing (Table 6.1). The framework of migration scheduling [29] which are periodically triggered by resource management policies with a long time interval is not suitable for the mobility-induced migration scenario. Furthermore, by modeling and calculating every resource competition of migration directly, the problem complexity [29, 30] increases along with the migration request number which is not suitable for large-scale situation. The running time of migration planning is also too large to schedule time-critical live container migrations. The algorithms do not consider the deadline or urgency (priority) of migration. In addition, without an on-line scheduler, the start time of a migration schedule is only based on the estimated migration time which can lead to migration performance and QoS degradation.

6.3 System Architecture

In the edge computing, there is no dedicated network for the live migration to support the user mobility compared with the traditional setups in cloud data centers [103]. By in-

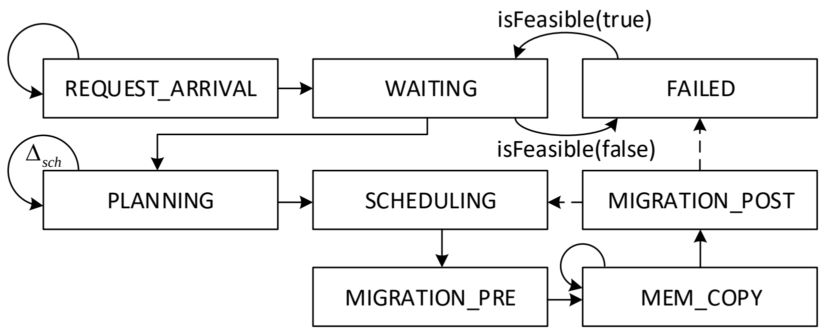


Figure 6.2: Lifecycle for live container migrations

tegrating the Software-Defined Networking (SDN) into edge computing, the centralized SDN controller can dynamically separate network resources from the service network to build a virtual WAN network for live migrations. The available bandwidth and network routing are dynamically allocated based on the reserved bandwidth of the service network. This solution alleviates the overheads of live migration on other services and guarantees the performance of multiple live migrations. To achieve a fine-grained live migration scheduling, the migration scheduling service is integrated with the SDN controller, such as OpenDayLight (ODL), Open Network Operating System (ONOS) and Ryu, and container management and orchestration module, such as Kubernetes and Docker Swarm, to control both network and computing resources during each migration lifecycle.

6.3.1 Migration Lifecycle

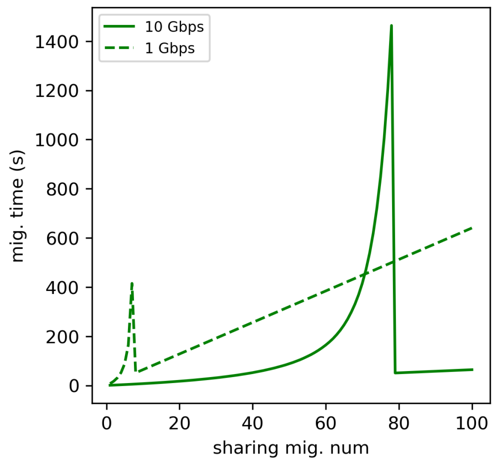
In this section, we introduce the framework of migration scheduling in edge computing. Compared with periodically arrived multiple live migrations in cloud data centers, the arrival of live container migration induced by user mobility is stochastic. Therefore, we design the scheduling framework for the planning and scheduling of live container migration in edge computing with stochastic environments. As shown in Fig. 6.2, when a migration request arrives, it enters the WAITING state if it is feasible for scheduling, which means the container is not in migration. Otherwise, it will enter into the FAILED waiting migration list of the corresponding container. The migration planning event is triggered periodically within a short interval (such as every 1 second). It will generate the migration scheduling plan according to both waiting and running migrations. Based on the migration plan, the SDN-enabled on-line scheduler starts the migration with the allocated bandwidth and routing. Then, the live container migration will start the pre-migration phase to extract the container procedure tree. This will trace the dirty memory in the userspace of the source server and create an empty container instance in the destination for state synchronization [60]. In MEM_COPY, the dirty memory is transferred iteratively to the synchronizing instance in the destination. In the post-migration phase, the network communication of the migrated service will be redirected to the new instance in the destination. Then, the migrated container will recover at the destination. It will also trigger the start of subsequent resource-dependent migrations in the plan and change the feasibility flag of the first migration request of the same container in the FAILED migration waiting list.

6.4 Motivations and Problem Formulation

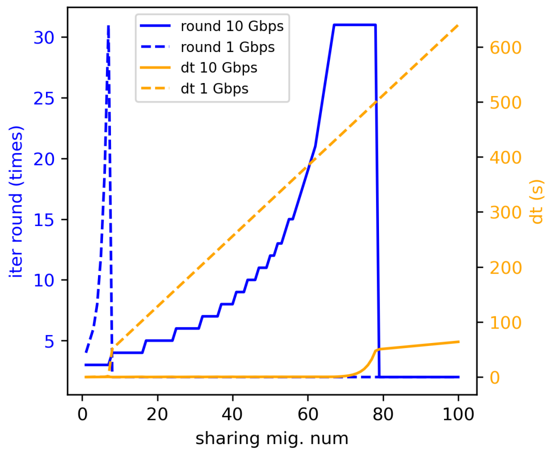
In this section, we first present the performance model of single live migration. Then, we analyze the challenges faced by multiple live container migrations scheduling in edge computing: resource competition or dependency and real-time planning and scheduling. Finally, we model the problem as iteratively generating the Maximal Independent Set (MIS) based on the resource dependency graph.

6.4.1 Resource Competition

We first explain the network sharing competition overheads in multiple migration scheduling. Two migrations may share the same source, destination, or part of network routings. Therefore, performing multiple live migrations in arbitrary order can lead to service degradation and unacceptable migration performance [29, 30]. A smaller bandwidth during the live migration means a longer migration time and more dirty pages need to transfer in order to limit the state difference between two instances for the last



(a) Average migration time



(b) Iterations and downtime(dt)

Figure 6.3: Migration performance against the number of migration sharing network bandwidth

stop-and-copy phases which contributes as the downtime. Thus, the sum of the individual migration time of several live migrations is less than the total live migration time as shown in Chapter 3. For example, based on the live migration model, Fig. 6.3(a) and 6.3(b) show the situation when several identical migrations sharing the same network path. The container's initial memory size is 1 GB with a 20 MB/s dirty page rate. The downtime and iteration threshold is configured at 0.5 seconds and 30 times, respectively. In this example, for the sake of a clear comparison between the sum of individual migration time and the total migration time, we start all migrations at the same time. In this case, the average migration time as shown also equals the total migration time.

The average execution time of live migrations scheduled sequentially with 10 Gbps and 1 Gbps is 0.8482 and 7.5241 seconds. The average downtime is 0.0082 and 0.1048 seconds. The iteration rounds are 3 and 4, respectively. However, the average migration time or total multiple migration time of 5 live migrations sharing 10 Gbps and 1 Gbps is 3.604 and 88.43 seconds. The average downtime is 0.2048 and 0.3689 seconds with 3 and 12 iterations, respectively. As the number of migrations increases (Fig. 6.3(a)), the allocated bandwidth decreases linearly. However, to achieve the required migration downtime, the average migration time will increase exponentially. At 7 and 80 migrations sharing of 1 Gbps and 10 Gbps respectively, the iteration rounds reach the

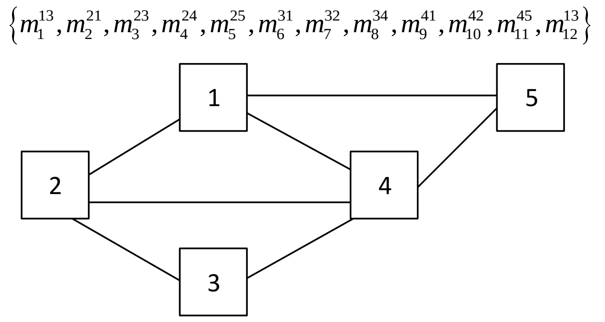

$\{m_1^{13}, m_2^{21}, m_3^{23}, m_4^{24}, m_5^{25}, m_6^{31}, m_7^{32}, m_8^{34}, m_9^{41}, m_{10}^{42}, m_{11}^{45}, m_{12}^{13}\}$
1
5
2
4
3

Figure 6.4: Example live migration requests and the network topology with 5 edge data centers

threshold as 30 (Fig. 6.3(b)). Then, with more bandwidth-sharing migrations, the dirty page rate is larger than the allocated migration bandwidth. The downtime exceeds the 0.5 seconds threshold and increases significantly from 0.78 to 64.0 seconds and from 1.85 to 640 seconds. For the time-critical live migrations, a longer migration time will increase the possibility of migration deadline violation and QoS degradation. Therefore, it is optimal to sequentially schedule the resource-dependent migrations while concurrently schedule the independent ones. If there is a set of independent migrations and no other resource-dependent migrations are running, we can start all migration in such a concurrent scheduling group. The objective of migration scheduling is to maximize the number of migrations that can be scheduled concurrently.

Figure 6.4 shows an illustrative example with twelve live migrations requests on the edge network topology of 5 total EDCs. Let m_i^{sd} denote the migration request that migrating container i from EDC s to EDC d. For the sake of a concise example, we limit the network interfaces used by the migration traffic. In other words, migration traffics share the same interfaces when the source or the destination is the same. It can be easily extended to the set of network interfaces in source $\{s\}$ and destination servers $\{d\}$ and the corresponding network paths $\{p\}$. The network routing policy considers the shortest network path with the minimal number of migration flows. For example, there are two network routes between EDC1 and EDC3. As there is one migration from EDC2 to EDC3, it chooses network path $\{EDC1, EDC4, EDC3\}$ in this case.

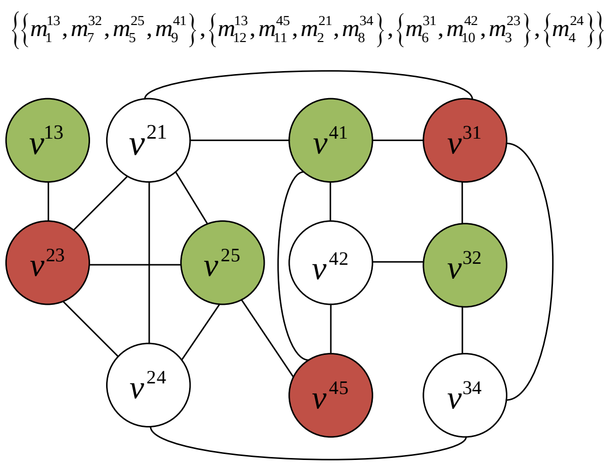

$\{\{m_1^{13}, m_7^{32}, m_5^{25}, m_9^{41}\}, \{m_{12}^{13}, m_{11}^{45}, m_2^{21}, m_8^{34}\}, \{m_6^{31}, m_{10}^{42}, m_3^{23}\}, \{m_4^{24}\}\}$
v^{13}
v^{21}
v^{41}
v^{31}
v^{23}
v^{25}
v^{42}
v^{32}
v^{24}
v^{45}
v^{34}

Figure 6.5: The resource dependency graph of example migrations, iterative maximal independent set as concurrent migration groups, and two colored possible maximal independent sets for the first iteration, and one possible concurrent migration groups

Resource-independent migrations from one concurrent scheduling group can be scheduled at the same time. The planning algorithm needs to generate a scheduling plan consists of several concurrent migration groups that each group size is as large as possible. A larger concurrent scheduling group indicates that there are more migrations could be performed at the same time. As a result, the better performance of multiple migrations in total migration time and the QoS of migrating service can be guaranteed. Note that two migrations from different migration groups are not necessarily resource-dependent. As shown in Fig. 6.5, based on the network topology provided by the SDN controller, we create an undirected graph of resource dependency among migrations based on the source, destination, and network routing of migration requests. Each node v_p^{sd} represents a list of migrations sharing the same source s, destination d, and network path p. In other words, migrations in one node list form a complete graph as all migrations are resource-dependent to all others in the list. For example, the migration list of node v^{13} is $\{m_1^{13}, m_{12}^{13}\}$. It significantly limits the problem complexity as the number of migration requests increases. The edge of the dependency graph indicates resource competition (network interfaces at source or destination, or bandwidth sharing along network routes) between migrations. A concurrent group equals to an independent set of the resource dependency graph. A maximal concurrent scheduling group is a set of resource-independent migrations that is not a subnet of any other concurrent group. In other words, there is no other migration outside the concurrent group can be added to it so that all migrations can be performed at the same time. Therefore, it equals a maximal independent set (MIS). The largest size MIS is a maximum independent set. As shown in Fig. 6.5, there are several combinations of migrations for a maximal concurrent scheduling group. In the first iteration, one of the maximum group is $\{m_1^{13}, m_7^{32}, m_5^{25}, m_9^{41}\}$ and one of the maximal group is $\{m_3^{23}, m_{11}^{45}, m_6^{31}\}$. Thus, the maximum group is a better choice. After selecting the migrations from the nodes of the maximal independent set, we delete these migrations and update the dependency graph. One node is deleted from the graph when there is no migrations left in its migration list. For example, after the first iteration, we only delete nodes v^{25}, v^{41}, v^{32}, because there is still one migration m_{12}^{13} left in node v^{13} list. Thus, it is essential to select migrations carefully to achieve the maximum size of the concurrent groups. At the end, the on-line scheduler schedules all migration in the first group. Then, when there is one migration finishes, the scheduler starts all migrations blocked by the finished migration following the order of migration groups.

Before discussing how to get the Maximum Independent Set, the largest Maximal Independent Set (MIS), of the resource dependency graph, we first review some basic graph concepts [192], such as clique C and independent set I. A clique is a subset of vertices of an undirected graph G such that every two distinct vertices in the subset are adjacent. The maximal clique is a clique that cannot be extended by including one more adjacent vertex. On the other hand, an independent set of a graph G is the opposite of a clique that no two nodes in the set are adjacent. The maximum clique or independent set is the maximal clique or independent set with the largest size. $\alpha(G)$ denotes the size of the largest MIS of graph G. Therefore, an independent set of the resource dependency graph equals a concurrent migration group. The migrations from the nodes in an independent set I can be scheduled concurrently. Meanwhile, migrations from the nodes in a clique are resource-dependent which need to be scheduled sequentially.

6.4.2 Real-Time Planning

There are few multiple migration planning and scheduling algorithms for live VM migration in cloud data centers [29, 30]. However, the processing time of the scheduling sequence of multiple live migrations based on the algorithms in cloud data centers is not suitable for the real-time requirement of mobility-induced migrations in edge computing. For example, the processing time of FPTAS [30] and CQNCR [29] for migration planning is about 5 and 10 seconds for 100 migrations. The processing time increases to 44.56 and 968.46 seconds for FPTAS and CQNCR to generate the scheduling plan of 500 migrations. For traditional dynamic resource management, the algorithm triggered every 10 minutes or 30 minutes. This leaves enough time budget for algorithms to generate the optimal scheduling sequence. However, in the edge computing environment for mobility-induce live migrations, the live migration requests arrive at any time stochastically. Most of the migration requests are also time-critical. Thus, the processing time of the planning and scheduling algorithm for mobility-induced migrations should be adapted to suit the real-time scenario.

6.4.3 Problem Modeling

The planning and scheduling algorithm is triggered periodically after every time interval Δ_{sch}. We let M^t_{arriv} denote the set of arrival migration requests at planning time t. M^t_{wait} is the set of migration requests waiting for planning at time t. M^t_{fail} is the set of infeasible migrations, such as its requested container is in migration. M^t_{plan} is the set of migrations that have been planned but not finished at time t, and M^t_{finish} is the set of finished migrations at time t.

The input of migration requests at every migration planning time t is $M^t_{input} = M^t_{plan} \cup M^t_{wait}$. For each live container migration m_j, we have source and destination edge data center and allocated network routing, (s_j, d_j, p_j), available bandwidth l_j, arrival time a_j, estimated migration time T_j, relative deadline D_j, start time b_j, and finish time f_j. Therefore, the response time can be represented as $r_j = f_j - a_j$. The slack time of migration scheduling, the remaining scheduling window that one migration will not miss its deadline, is $\tau_j = a_j + D_j - T_j - t$. The objective of live container migration planning and scheduling is to maximize the number of running resource-independent live migrations until the next planning time $t + \Delta_{sch}$.

At every planning and scheduling time t, the resource dependency graph $G = (V, E)$ denotes the acyclic undirected graph where $|G| = |V|$. Each node $u \in |V|$ represents the list of migrations $M(u)$ where migration shares the same source s, destination d, and network routing p. By sharing the same source and destination and network routing, migrations in the list of a node are all resource-dependent. Let $(u, v) \in E$ denote the edge between node u and v. It indicates the resource dependency between migrations from both nodes. $V(G)$ denotes the set of nodes of graph G.

We model the multiple migration planning problem as generating the maximal independent set of the dependency graph iteratively. In other words, in each iteration, we get the maximal independent set of the remaining graph, then update the graph by deleting corresponding migrations. Let $G_{i+1} = G_i [V(G_i) - S_i]$ represent the remained graph by directly deleting vertex from set of nodes S_i. Let I_i denote the maximal independent set of graph G_i. Then, the remained graph G_{i+1} in each iteration can be represented as:

$$G_{i+1} = G_i [\![V(G_i) - I_i]\!] = G_i [V(G_i) - S_i] \tag{6.1}$$

by deleting set of nodes $S_i = \{u | u \in I_i, M(u) = \emptyset\}$, where the migration list of the deleted node u is empty. Therefore, for each migration planning, the objective is to generate the iterative maximum independent set of dependency graph:

$$\max |I_i|, \forall I_i \in \left\{ I^i_{iter} \right\} \tag{6.2}$$

where $\{I^i_{iter}\} = \{I_1, I_2, ..., I_K\}$ is the total K iterative independent sets and there is no vertices left in the $K + 1$ remaining graph as $G_{K+1} = \emptyset$. In other words, each iterative independent set size equals the size of maximum independent set of remaining graph $|I_i| = \alpha(G_i)$.

We extend the model to generate the iterative maximum weighted independent set for migration with different priorities, such as migration deadline. The weight of an independent set is $W(I) = \sum_{u \in I} W(u)$. The largest weight of migration $\hat{m}$ in the node migration list is the weight of its corresponding node in the dependency graph $W(u) =$

$W(\hat{m})$ that

$$W(\hat{m}) \geq W(m), \forall \hat{m}, m \in M(u) \tag{6.3}$$

Then, the objective of multiple migration planning can be represented as:

$$\max W(I_i), \forall I_i \in \left\{ I^i_{iter} \right\} \tag{6.4}$$

The weight of node $W(u) = 1$ when there is no need to differentiate migrations in different nodes. Generating the maximum (weighted) independent set of an undirected acyclic graph is a well known NP-hard problem [193, 196]. Therefore, generating the iterative maximum independent set as the subset is also NP-hard.

6.4.4 Complexity Analysis

Because an independent set of G is a clique in the complement graph of G and vice versa, the independent set problem and the clique problem are complementary [192–194]. In other words, listing all maximal independent sets or finding the maximum independent set of a graph equals listing all maximal cliques or finding the maximum clique of its complement graph. Thus, in each iteration, we can equivalently find the maximum independent set by getting the maximum clique C_i of the complement graph $C_i(\bar{G}_i) = I_i(G_i)$.

It is known that all maximal cliques can be calculated in a total time proportional to the maximum number of cliques in an n-vertex graph [194]. In other words, each clique is generated in a polynomial time in all maximal cliques listing [193]. When we only consider vertex, the maximal cliques listing algorithm (CLIQUES) [194, 196] based on Bron-Kerbosch [192] is the optimal algorithm. The worst-case running time of CLIQUES is $O(3^{n/3})$. The upper bound of all maximal cliques or independent sets of a graph is $3^{n/3}$ [213]. For the problem of finding one maximum independent set, the time complexity is improved from $O(2^{n/3})$ in [214] to $O(2^{0.276n})$ [215]. Based on the work [215], the best-known time complexity is $O\left(2^{n/4}\right)$ [216]. Therefore, it is computationally impossible to solve the exact problem of listing all maximal cliques (maximum clique) of its complement graph $\bar{G}_{dep}$ or all maximal independent sets (maximum independent set) of G_{dep} for the real-time live container migration scheduling in edge computing which exhibits an exponential time complexity.

6.5 Migration Planning and Scheduling

In this section, we present the proposed planning and scheduling algorithms for large-scale live container migrations in edge computing. With the waiting live container migration requests and planned unfinished live migrations as the input, the migration planner needs to efficiently schedule arriving migrations while maintaining the QoS. Based on the problem modeling in Section 6.4.3, this problem is reduced to finding an MIS of the migration dependency graph iteratively. Therefore, we propose two major approaches to generate the iterative MISs of the dependency graph: (1) Direct iterative MIS generation and (2) Maximum Cliques (MCs)-based MIS generation.

6.5.1 Direct iterative-Maximal Independent Sets

For the direct iterative MIS generation, we follow the rationals based on the planning model as follows: (1) Create dependency graph G_{dep} based on the source, destination, and network routing of the input migrations and the network topology; (2) Generate the Maximum Independent Set (MIS) I of G; (3) Delete the nodes $u \in I$ from G if its migration list $M(u)$ is empty; and (4) Repeat the procedure 2 and 3 until there is no vertices left $G_{dep} = \emptyset$.

The Approximation

For the approximation algorithm (approx) of creating the iterative maximum independent set, the procedure is as follows: In the approximation algorithm (Algorithm 9), we use the approximating maximum independent sets algorithm by excluding subgraph [217] to generate MIS in each iteration. Note that we skip the MIS generation and remove the migrations directly if the node size of G_{dep} is unchanged in the current iteration. In other words, if we need to recalculate the MIS of the remaining graph, there is at least one node removed from the graph G_i. Given total m live container migrations, we create the corresponding dependency graph with n vertices. Therefore, regardless of the total number of migration requests, the upper bound of the complexity of planning multiple migrations scheduling is limited by the involved source, destination, and network routing. In the worst case, the planning algorithm only needs at most n iteration rounds to calculate the concurrent migration group. In each iteration, it guarantees $O(n/(\log n)^2)$ approximate maximum independent set in polynomial time [217].

Algorithm 9: Iterative approximation grouping

```
Input: $\{G_{dep}\}$
Result: migGroups $\{I_{iter}\}$
$i \leftarrow 0$; $G_i \leftarrow G_{dep}$; $\{I_{iter}\} \leftarrow \emptyset$;
while $V(G_i) \neq \emptyset$ do
    $I_i \leftarrow$ APPROX_MIS$(G_i)$;
    $G_{i+1} \leftarrow G_i [\![V(G_i) - I_i]\!]$;
    $\{I_{iter}\} \leftarrow \{I_{iter}\} \cup I_i$;
    $i \leftarrow i + 1$;
```

Based on the newly generated scheduling plan $\{I_{iter}\}$, the SDN-enabled on-line migration scheduler will start all feasible migrations in the first group I_0, considering the resource dependency with current running migrations. Then, whenever a migration finishes, the scheduler starts all remaining feasible migrations in each concurrent migration group I_i followed by the scheduling plan order.

Greedy MIS Algorithm

The greedy algorithm (iter-GWIN) generates the concurrent groups (MIS) of live migration iteratively. A greedy maximal independent set algorithm (GWIN) [218] based on the weight and the degree of a node is adapted to directly generate the MIS in each iteration. Let Δ_G denote the maximum degree and $\bar{d}_G$ is the average degree of G. The degree of node u in G is $d_G(u) = |N_G(u)|$. $N_G(u)$ is the set of neighbor nodes of vertex u and $N_G^+(u) = N_G(u) \cup \{u\}$.

As shown in Algorithm 10, from line 3-8, it selects the node with largest score re-

Algorithm 10: iter-GWIN

```
Input: $\{G_{dep}\}$
Result: migGroups $\{I_{iter}\}$
$i \leftarrow 0$; $G_i \leftarrow G_{dep}$; $\{I_{iter}\} \leftarrow \emptyset$;
while $V(G_i) \neq \emptyset$ do
    $I_i \leftarrow \emptyset$; $G_j \leftarrow G_i$; $j \leftarrow 0$;
    while $V(G_j) \neq \emptyset$ do
        select node $\hat{u}$ in $G_j$;
        $I_i \leftarrow I_i \cup \{\hat{u}\}$;
        $G_{j+1} \leftarrow G_j\left[V(G_j) - N_G^+(\hat{u})\right]$;
        $j \leftarrow j + 1$;
    $G_{i+1} \leftarrow G_i[[V(G_i) - I_i]]$;
    $\{I_{iter}\} \leftarrow \{I_{iter}\} \cup I_i$;
    $i \leftarrow i + 1$;
```

garding the minimal degree and maximal weight:

$$W(u)/(d_{G_i}(u) + 1) \tag{6.5}$$

It removes the selected node and its neighbors from the graph and repeats the procedure until there is no vertices left.

As mentioned in the problem modeling, the weighted node equals the maximum weight of migrations from its list. The migration weight could be arrival time, estimated migration time, or correlation network influence [29] after migration for non-time-critical migrations and the deadline or slack time for real-time migrations scheduling. In this chapter, we consider the weight function regarding the slack time τ as follows:

$$W(m) = \begin{cases} 10 \cdot \beta/\tau & \tau > \beta \\ 100 \cdot |\tau|/\beta & \tau < -\beta \\ 100 & other \end{cases} \tag{6.6}$$

where β is the slack time threshold. We set $\beta = 1$. The weight of node is $W(u) = \gamma \cdot W(m)$, where γ is the coefficient regulator for the urgency of the scheduling migration.

We set $\gamma = 1$. Moreover, in the situation that the priorities of all migrations are the same, we only need to consider the size of MIS. The node weight is set to 1 $W(u) = 1$. In each iteration, the lower-bound of the maximum (weighted) independent set is $\sum_{u \in V} W(u)/(d_G(u)+1)$ [218]. As iteration is n in the worst case, the time complexity of iter-GWIN is $O(n^2 \log n)$ for weighted graph and $O\left(n^2\right)$ for unweighted graph.

6.5.2 The Maximum Cliques-based Heuristics

In this section, based on the observation of the density property of migration resource dependency graph, we propose the iterative Maximum Cliques (MCs)-based algorithm. We first discuss the rationals of the proposed algorithm.

The degeneracy of a graph G is the smallest number d such that every subgraph of G contains a vertex of degree at most d. It is a measure for the graph spareness. For an n-vertex graph with degeneracy d, by introducing the sequence ordering based on degeneracy, Bron-Kerbosch Degeneracy algorithm [197] can list all maximal cliques in time $O(dn3^{d/3})$. With a spare graph that $n \geq d+3$, the upper bound of all maximal cliques number is $(n-d)\,3^{d/3}$. Figure 4.5(c) illustrates the nodes and the density (degeneracy) of the resource dependency graph of WAN network topologies [39] and its complement. It shows that the degeneracy of the complement graph $\bar{G}_{dep}$ is 4.34 times that of G_{dep}. For G_{dep} and its complement graph, the average ratio of dependency d to the total number of nodes n is 0.153 and 0.714, respectively. The resource dependency graph is considerably more sparse than its complement graph. Therefore, for G_{dep}, there are much fewer maximal cliques than the total MIS. As a result, according to the theoretical time complexity, the running time of listing all maximal cliques or maximum clique of G_{dep} is much smaller than that of listing all maximal independent sets or maximum independent set of G_{dep}. Therefore, the iterative Maximum Cliques (MCs)-based heuristics algorithm has two steps: (1) calculates the list of iterative maximum cliques and (2) generates the iterative maximal independent set based on the list. As nodes from one maximal clique can not be included into the same independent set, the iterative maximum cliques serve as a heuristic pruning decider to speed up the algorithm.

Algorithm 11: Iterative heuristic of migration grouping

```
Input: $\{G_{dep}\}$
Result: migGroups $\{I_{iter}\}$
$\{C_{iter}\} \leftarrow \emptyset;\{I_{iter}\} \leftarrow \emptyset;$
while $|G_{dep}| \,!= 0$ do
    {Iterative creating Maximum Cliques}
    $\hat{C}_i \leftarrow$ MAXIMUM_CLIQUE$(G_{dep})$;
    $G_{dep} \leftarrow G_{dep}\left[V\left(G_{dep}\right) - \hat{C}_i\right]$;
    $\{C_{iter}\} \leftarrow \{C_{iter}\} \cup \hat{C}_i$;
while $\{C_{iter}\} \neq \emptyset$ do
    $I \leftarrow \emptyset$
    foreach $\hat{C}_i$ in $\{C_{iter}\}$ do
        $m \leftarrow$ ADDINDEP$(I, \hat{C}_i)$;
        DELNODE $(\hat{C}_i, m)$;
    $\{I_{iter}\} \leftarrow \{I_{iter}\} \cup I$;
return $\{I_{iter}\}$
```

Iterative-rounds MCs algorithm

Let $\hat{C}_i$ denote the maximum clique and $\{\bar{C}_i\}$ denote the maximal cliques list of round i graph. The iterative-rounds Maximum Cliques (MCs)-based heuristic algorithm (Algorithm 11) follows two steps: (1) generating the maximum clique iteratively and (2) obtaining the MIS from the iterative maximum cliques.

As shown in Algorithm 11, we first create dependency graph G_{dep} as the input based on the source-destination of the given migrations and the network topology. From line 1-6, the algorithm calculates the iterative maximum cliques of the dependency graph until there is no vertices left. In each iteration, it generates the maximum clique (Bron-Kerbosch Degeneracy algorithm) [197] of the remaining graph. It is proved that the algorithm is highly efficient in a sparse graph, such as the resource dependency graph [197]. Then, it updates the remaining graph by deleting the nodes of the maximum clique from G_{dep}. Let $d_{G[C]}(u) = |N_{G[C]}(u)|$ denote the degree of node u to the remaining graph which excludes all nodes in the clique. The node score can be represented as:

$$W(u) \Big/ \left(d_{G[C]}(u) + 1\right) \tag{6.7}$$

In the second step (line 7-12), it generates maximal independent sets based on the iterative maximum cliques. In each round (line 9-11), it selects the feasible node with maximum score of each maximum clique $\hat{C}_i$ and adds largest-weight migration from its list into the independent set. A node is feasible when it can be included in the current independent set. If there is no migrations left in the migration list of the selected node $M(u)$, the selected node is removed from the clique. As the largest possible number of maximal cliques in an n-vertex graph with degeneracy d is $(n-d)\,3^{d/3}$. Therefore, according the iter-MCs algorithm, the upper bound of the size of the iterative maximum independent set of each iteration is also $(n-d)\,3^{d/3}$. In the worst case, the time complexity of iter-MCs is $O(dn^2 3^{d/3})$.

Theorem 2 (Correctness of MIS from Maximal Cliques)**.** *The Independent Sets generated from maximal cliques are the maximal independent sets of the graph.*

Proof. $I_q = \{q_0, q_1, q_2, ..., q_d\}$ is one of the independent sets generated from the maximal cliques of $G\,(V,E)$, where one vertex comes from only one maximal clique $q \in C_q$. Assume, for the sake of contradiction, there is at least one vertex p, $p \in C_p$ exists, that $I_q \cup \{p\}$ is also an independent set. That is, there is no edge between p and any other vertex $\forall q, q \in I_q$, $\neg\exists\,(p,q) \in E$. Based on the definition of the heuristic algorithm, we can get $\forall r \in C_p$, $r \notin I_q$, that $\exists q \in I_q$, where $(p,r) \in E$. Thus, $\exists p, q$, where $p \in C_p, q \in I_q$, that $\neg\exists\,(p,q) \in E$ and $\exists\,(p,q) \in E$, which is impossible. Since, we have a contradiction, it must be that I_q is a maximal independent set. □

Single-Round MCs Algorithm

Furthermore, we propose a single-round MCs-based algorithm (single-MCs). It generates the optimal iterative maximum cliques only based on the all maximal cliques of the initial dependency graph G_{dep}. The maximum clique size of each iteration is the same as the iter-MCs. We also prove the correctness of the proposed single-round iterative maximum cliques algorithm.

The first step of the iter-MCs algorithm is replaced by Algorithm 12. The algorithm only generates the list of all maximal cliques $\{\bar{C}\}$ once by using the Bron-Kerbosch Degeneracy algorithm. Until there is no vertices left in the clique list, it selects the maximum clique (largest maximal clique) $\hat{C}_i$ from the list and deletes the nodes of the selected

Algorithm 12: Single-round iterative maximum cliques

```
Input: $\{G_{dep}\}$
Result: migGroups $\{C_{iter}\}$
SINGLE-ITER($\{G_{dep}\}$):
$\{C_{iter}\} \leftarrow \emptyset$;
$\{\bar{C}\} \leftarrow$ FINDCLIQUES($G_{dep}$);
while $\{\bar{C}_i\} \neq \emptyset$ do
    $\hat{C}_i \leftarrow$ MAX($\{\bar{C}\}$);
    $\{C_{iter}\} \leftarrow \{C_{iter}\} \cup \hat{C}_i$;
    $\{\bar{C}\} \leftarrow$ DELNODES ($\hat{C}_i$, $\{\bar{C}\}$);
return $\{C_{iter}\}$
```

maximum clique from all maximal cliques.

Theorem 3 (Correctness of the algorithm single-MCs)**.** *Given a graph $G = (V, E)$ $V \neq \emptyset$, the single iteration algorithm SINGLE-MCs generates all and only iteration maximum cliques.*

Proof. It is proven that the Bron-Kerbosch Degeneracy algorithm generates all and only maximal cliques without duplications [197]. Then, we only need to prove the results of iterative maximum cliques are the same in iter-MCs and single-MCs, i.e., one can get all the iterative maximum cliques based on the maximal cliques of the original graph by deleting the vertices from the maximum clique in the last round.

Let $C(G) = \{C_0, C_1, ..., C_d\}$ denote all maximal cliques of the original graph G, where $|C_i| \geq |C_{i+1}|$. $\forall C_i, C_j \in C(G)$, that $C_i \neq C_j, C_i \not\subset C_j$. The next iteration graph is $G \backslash C_0 = G[V(G) - C_0]$. Then, $C(G \backslash C_0) = \{C_1^{'}, C_2^{'}, ..., C_e^{'}\}$. The output of first round of single-MCs is $C(G) \backslash C_0 = \{C_1^{''}, C_2^{''}, ..., C_e^{''}\}$.

Assume, for the sake of contradiction, there is one maximal clique $C_f = C_i^{''} \cup \{q\}$, $q \in V - C_0, q \notin C_i^{''}$, which $C_f \in \left\{C_j^{'}\right\}$ and $C_f \notin \left\{C_i^{''}\right\}$. Based on the algorithm single-MCs and definition of maximal clique, due to $\{q\} \notin C_0$, $C_f \cup C_0$ is also a maximal clique that $C_f \cup C_0 \in C(G)$. However, as C_0 is the maximum clique of G, it is impossible that $C_f \notin \emptyset$. Since, we have a contradiction, the C_f is not exist. Therefore, $C(G) \backslash C_0 = C(G \backslash C_0)$.

□

Table 6.2: Performance comparison with first, second, and third quartile of processing time, total MIS number (iteration), maximum, mean, 95th, and 99th quartile of the independent set size in each result of the total 202 WAN topologies

algorithm	proc. time (s)	total sets $\|\{I\}\|$	max($\{\|I\|\}$)	mean($\{\|I\|\}$)	95%($\{\|I\|\}$)	99%($\{\|I\|\}$)
single-MCs	8.8115 1.0047 0.1554	164.5 75.0 30.0	88.0 54.0 36.0	8.1594 6.1667 5.0	23.05 16.8 12.825	45.9 25.8 17.7
iter-MCs	49.1566 4.5807 0.4723	165.0 75.0 30.0	88.0 54.0 36.0	8.1594 6.2124 5.0	23.0 16.8 12.65	45.6 26.0 17.7
iter-GWIN	14.2786 1.4916 0.1610	159.5 76.0 31.0	88.0 54.0 36.0	8.2844 6.3448 5.1909	24.8 18.0 13.15	48.9 27.0 18.6
approx	1115.2929 57.2547 5.5257	171.5 84.0 32.0	59.0 36.0 25.0	7.7568 5.9492 4.6946	21.6 15.85 12.0	36.8 23.2 15.8

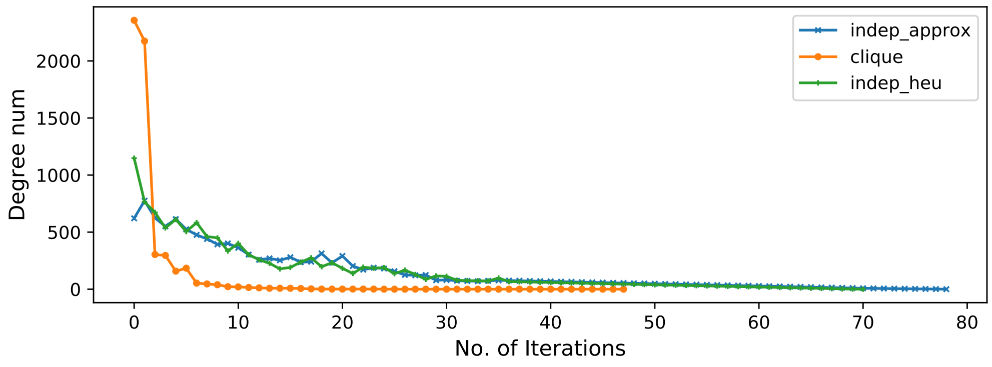


Figure 6.6: Degree of iteration cliques/independent set to the remaining nodes

6.6 Graph Algorithm Performance and Analysis

In this section, we evaluate proposed migration planning algorithms for the problem of iterative MIS generation: (1) iter-MCs (2) single-MCs; (3) approximation; and (4) iter-GWIN, in processing time, maximal independent set size, and iteration rounds. Based on more than two hundred real network WAN topologies [39], we consider a set of live migration requests with each source and destination combination. Each migration request corresponds to one combination with the network routing of the shortest path. We run the computational experiments in Python 3.6.3 and NetworkX package [219] version 2.4 as the graph library with source code.

The iterative maximum clique generation of migration dependency graph is faster than that of iterative MIS in three aspects: (1) As the analysis of dependency graph in section 6.5.2, dependency graph G_{dep} is more sparse than its complement $\bar{G}_{dep}$. Therefore, getting the maximum clique of G_{dep} in one iteration is faster than that of $\bar{G}_{dep}$; (2)

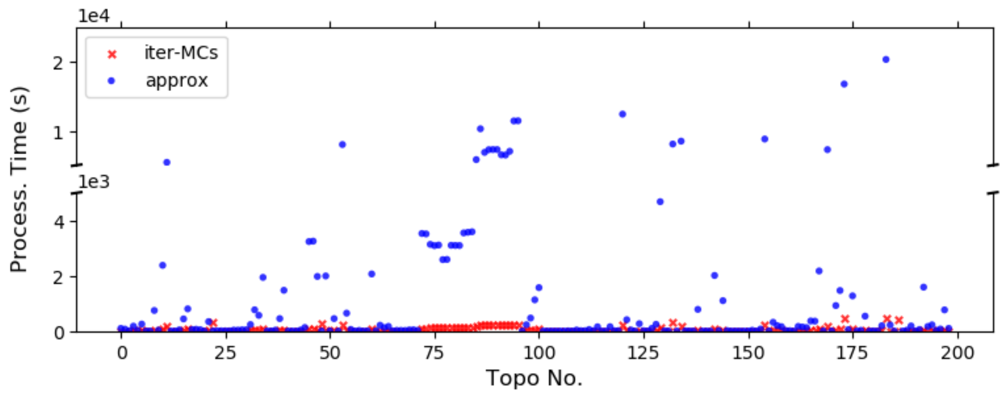


Figure 6.7: Processing time comparison between iter-MCs and approximation

The maximum clique can reduce the complexity of the graph much more efficiently in each iteration; and (3) There are fewer iterative maximum cliques of G_{dep} than the iterative independent sets. In other words, the number of iteration rounds of the iterative maximum clique is smaller.

Figure 6.6 demonstrates an illustrative result of one of the network topology (Australia's Academic and Research Network, AARNet). The dependency graph consists of a total of 342 nodes and 11754 edges. It shows the degree of the maximum clique and independent set in each iteration to the remaining nodes. In other words, it is the edges of the removed nodes in each iteration excluding the edges between nodes from the maximum clique. Note that there is no edges (degree is zero) between nodes in one independent set. With the degree in the maximum clique and the degree to the remaining graph, the complexity of G_{dep} is dropped dramatically in the first three iterations. On contrary, by removing the maximum independent set, the complexity of the graph remains at a high level and declines steadily. Furthermore, the number of total iterative maximum cliques and iterative MISs is 47 and 70, respectively. Comparing the result of the approximation (indep_approx) with iter-MCs (indep_heu), the heuristic iterative MCs-based algorithm achieves better performance in the size of the maximum clique in each iteration and the total iteration rounds.

Since the processing time varies greatly, we use two separated figures to represent the results of processing time. Figure 6.7 shows the performance comparison between the approximation (approx) and iter-MCs in processing time. Figure 6.8 shows the com-

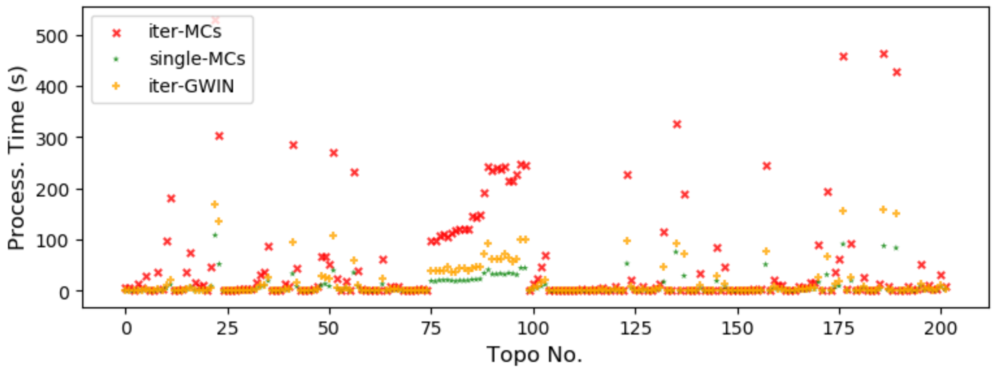


Figure 6.8: Processing time comparison for iter-GWIN, iter-MCs, and single-MCs

parison between iter-MCs, single-MCs and iter-GWIN. The results of computational experiments indicate that the approximation algorithm has the worst performance in processing time. From approx to iter-MCs (Fig. 6.7), the average processing time of all topologies decreases by 91.32%. From iter-MCs to iter-GWIN and iter-GWIN to single-MCs, the average processing time decreases by 57.40% and 20.84%. Table 6.2 also illustrates the third (Q3), second (mean), and first quartile (Q1) of the average processing time. For the dependency graph with every source and destination combinations of a relative small size network, the single-MCs and iter-GWIN can both generate the scheduling plan in around 0.15 seconds. However, the performance difference in processing time increases with the size of the network topology. For mean and the Q3 of all processing time results, the average processing time of single-MCs decreased by 32.64% and 38.29% from iter-GWIN, respectively. In summary, the single-MCs algorithm has the best performance in processing time.

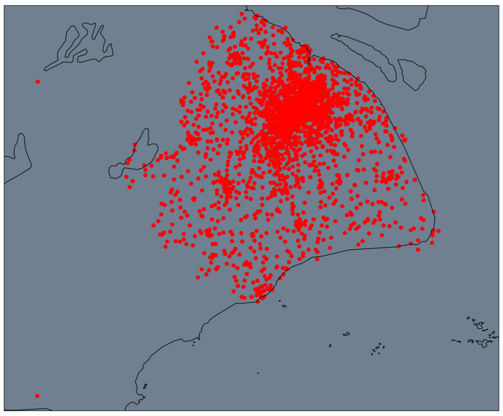
We also evaluate the size of the result list or iteration rounds $|\{I\}|$. It is the number of sets the algorithm divides into different concurrent groups for the given migrations. For the approx algorithm, from Q3 to Q1, it generates 171.5, 84.0, and 32.0 many of iterative MIS in one planning result. From approx to iter-MCs, the iteration number decreases by 3.79%, 10.71%, and 6.25%, respectively.

For the performance in iterative MISs of each graph, we examine the size of the largest iterative MIS ($\max(\{|I|\})$) and the mean size ($\mathrm{mean}(\{|I|\})$). As the first several rounds of the result are the most essential factors on scheduling performance, we also evaluate the algorithm in the 95-quartile and 99-quartile of the iterative MISs size. The algorithm iter-GWIN has the best performance in the large network topology. The total number of iterative MIS is reduced by 3.04% compared to the results of single-MCs. Although the mean results of the set size mean($\{|I|\}$) of approx algorithm is close to other three algorithms, its performance in the first several iterations is the worst. As a result, the total set of approx algorithm is significantly larger than other algorithms. For the maximum set size, single-MCs, iter-MCs and iter-GWIN has the identical performance in Q1, mean, and Q3 from all results of network topologies. For the 95th and 99th quartile iter-GWIN for directly calculate the maximum clique has a slightly better performance over the iterative MCs-based heuristic algorithms even though the processing time is higher.

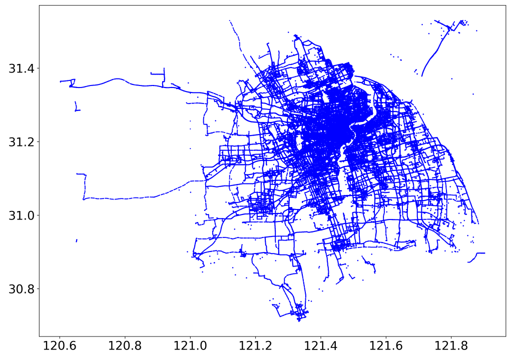
6.7 Simulation and Performance Evaluation

In this section, we evaluate proposed solutions using real-world traces on an event driven simulator. We first describe the real-world telecom base station dataset and taxi GPS traces used in the experiments. We explain the placement of edge data centers and the network topology and region coverage of each EDC. The event-driven simulator for software-defined network-enabled edge-cloud computing CloudSimSDN [166] is extended to emulate the the user movement and the live container migration in edge computing. It provides a network operating system based on the software-defined networking for dynamic service and network resource monitoring and allocation. Compared to the simulation results driven by mathematical models, this can generate more realistic results without following the strong assumption encoded in the proposed mathematical modeling.

We compare and evaluate the performance of live container migration planning and scheduling algorithm (iter-GWIN and single-MCs) against a policy with no planning scheduling and the state-of-art live VM migration cloud algorithm FPTAS [30] in processing time, migration time, downtime, transferred data, deadline violations, and network transmission time.

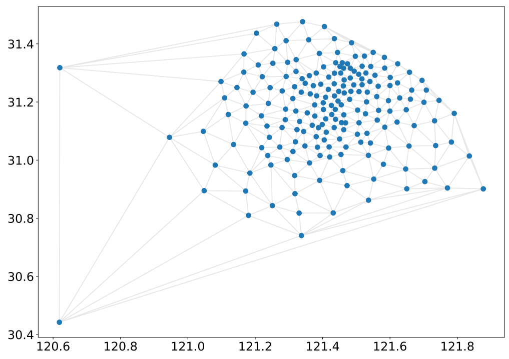


(a) Shanghai Telecom base station locations

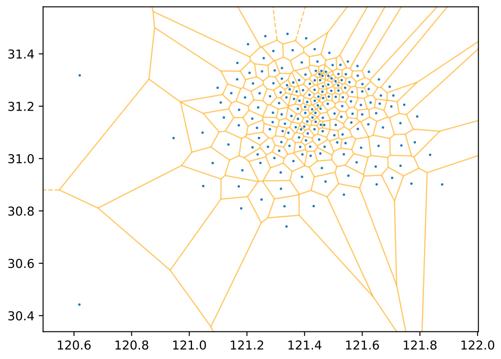


(b) The taxi GPS trace in the first hour

Figure 6.9: Experimental dataset and configurations of longitude and latitude



(a) Delaunay triangulation based Edge computing topology



(b) Edge data center coverage with Voronoi cells

Figure 6.10: Edge data center topology and coverage in longitude and latitude

6.7.1 Experimental Data

In this section, we describe the base stations coordinates provided by Shanghai Telecom dataset and Shanghai Qiangsheng taxi GPS trace dataset (April 1, 2018) used in our experiments. The given data is preprocessed by limiting the range of the longitude and latitude from 30.40° N to 31.35° N and 120.51° E to 122.12° E as there are some taxis travel to nearby cities. The Shanghai Telecom dataset contains the longitude and latitude coordinates of a total of 3233 base stations as shown in Fig. 6.9(a). We use K-means

http://sguangwang.com/TelecomDataset.html
http://soda.shdataic.org.cn/download/31

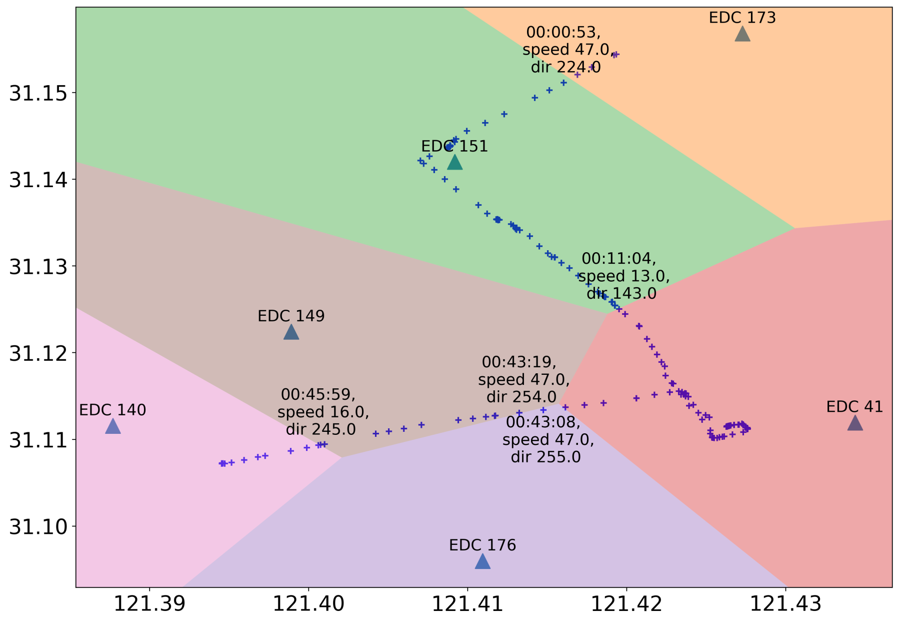


Figure 6.11: Example of live migration request triggered by user movement of longitude and latitude

algorithms [220] to generate the location of a total of 200 Edge Data Centers (EDCs) based on the longitude and latitude of the given base stations. Figure 6.9(b) illustrate the taxi GPS trace in the first hour. Besides the GPS coordinates and timestamp, each data record of the taxi dataset also includes the taxi id, service status, such as alarm, occupation, taxi light, road type, and breaking, as well as vehicle speed, direction, and the number of connected satellites.

Figure 6.10(a) illustrates base stations are clustered and connected to one of the regional Edge Data Centers. There is no information on the physical network topology and connectivity between EDCs. As shown in Fig. 6.10(a), for the geometric spanner, we choose Delaunay Triangulation [221] to generate links between the gateway of each EDC. For the network routing within the generated network topology, we consider the shortest path, which is no longer than $4\pi/3\sqrt{3}$ times the Euclidean distance between source and destination. As a result, the boundary of EDC regions (Fig. 6.10(b)) is a Voronoi diagram [221] where the Euclidean distance of any point to its corresponding EDC region is less than or equal to its distance to any other EDC.

Similar to other research regarding the generation of mobility-induced live migra-

Table 6.3: Multiple container migration scenarios

Scenario	S1	S2	S3
vehicles	1000	2000	4000
migrations	9933	19522	37822

tions in edge computing [189, 211], we combine these two datasets to simulate the scenarios where the user needs to connect to the services and maintains the low end-to-end latency through live container migration in edge computing environments. Figure 6.11 demonstrates an example that the request of live container migration is induced when a taxi moves across the boundary between two clusters of EDCs. The deadline of each live container migration is generated based on the average mobility speed of users in the last 3 GPS records, travel direction, and the signal strength of base stations.

6.7.2 Experimental Setup

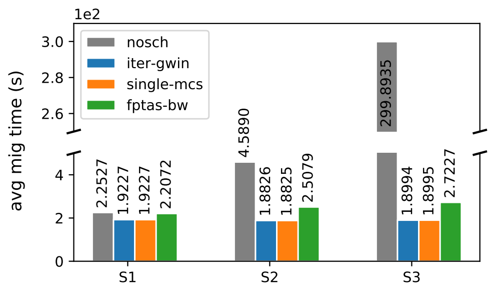
In this section, we describe details of the experiment setup. The end-to-end delay between the user and the service is the time interval from the user (taxi) sent workload to the container assigned in the EDC to the result is received by the user. To generalize computer vision use case workloads, the service task generated during the experiments follows the Poisson distribution with a mean of 24 per second (24 FPS). In each task, the network packet size sent from a user is 16384 bytes ($128 * 128$ bytes). The processing workloads in the container are randomly generated from 500 to 1000 cycles per bit [211]. The result packet sent back from the container to the user is 128 bytes. The sum CPU power frequency for each EDC with multiple CPUs is 25 GHz [222, 223]. To simulate the limited network resources for migrations in the edge, we consider that the reserved network bandwidth between a container and its user is 3 Mbps. The physical network bandwidth is 1 Gbps. The network delay between any based station to its regional EDC is 5 ms and the delay between two EDCs is randomly generated in the range 5 to 50 ms [211].

According to the evaluation results of container memory and dirty memory size during live container migrations [210], we generate the container memory from 100 MB to 400 MB. The dirty page rate for each dirty memory transmission is from 2MB/s to 8MB/s and the data compression rate is 0.8 [179]. We configure the downtime threshold and the maximum iterations for live container migration at 0.5 seconds and 30 times as shown in Chapter 5, respectively. Based on the SDN controller, the remaining network bandwidth between the source and destination EDC which is not utilized by services is allocated to the live container migration traffic. If several live migrations are sharing part of their routings, the bandwidth will be allocated evenly to each of the network flows.

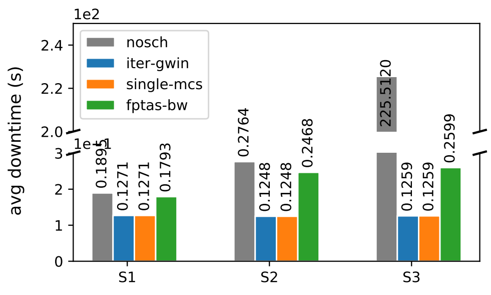
From the experimental scenario S1 to S3 (Table 6.3), 1000, 2000 and 4000 vehicles are selected randomly. We consider the GPS trace of selected vehicles within 1 hour. For the initial placement at the start of the experiment, we allocate corresponding containers for each vehicle at the same edge data center according to its GPS coordinates. The nearest edge data center first policy is considered for our experiment to generate the live container migration requests. According to the user mobility, one live container migration will be triggered when one vehicle exits the coverage area of its current edge data center. There are 9933, 19522, and 37822 migration requests induced by these vehicles' movement, respectively. During the live container migration, the dirty memory of migrating container will be copied iteratively from the source edge data center to the nearest edge data center through the shortest network path. For the evaluation sensitivity, the results of each scenario are an average of 10 individual experiments. In this experiment, we consider that the container image as a universal service is already available in all edge data centers or shared by the network storage between EDCs. CloudSimSDN-NFV [166], an event-driven simulator, is extended with corresponding components to support the live container migration and user mobility in edge computing environments.

6.7.3 Experimental Results and Analysis

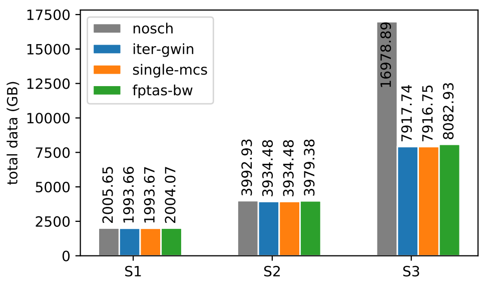
In Section 6.6, we compare the algorithm performance in terms of the size of iterative MISs and processing time. Thus, we only evaluate the two best algorithms in this section. We compare the experimental results between no migration scheduler, iter-GWIN,



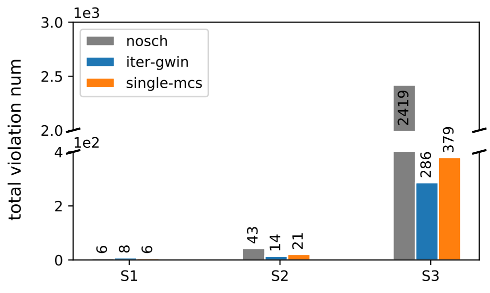
(a) average migration time



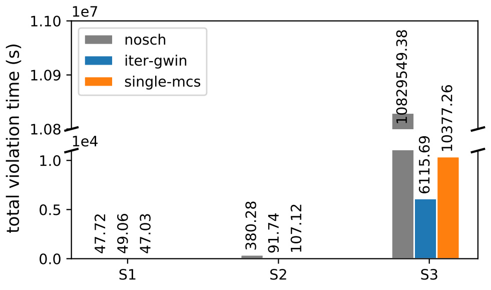
(b) average downtime



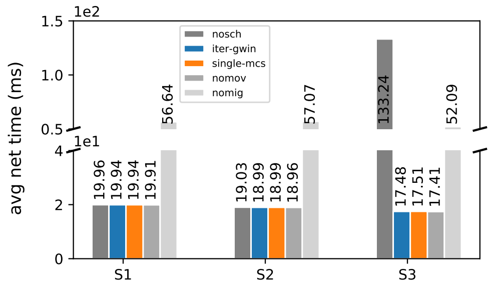
(c) total transferred data



(d) total deadline violations



(e) total violation time



(f) network transmission time

Figure 6.12: Migration performance comparison with no scheduler, iter-GWIN, and single-MCs under different scenarios.

single-MCs, and the current state-of-the-art migration planning and scheduling in WAN FPTAS [30]. In FPTAS, to maximize the total bandwidth utilized by migrations, one migration can be started even there is considerably limited bandwidth which is much lower than the dirty page rate per second. The solution can cause devastating migration performance. Thus, we improve FPTAS by adding a bandwidth threshold (FPTAS-BW) that the available bandwidth must larger than the dirty page rate as the migration start

Table 6.4: Total processing time comparison in milliseconds

algorithm	S1	S2	S3
iter-GWIN	306.9607	762.3356	3997.8309
single-MCs	332.5682	583.3951	1544.6291
FPTAS[30]	903597.39	1923036.81	4677990.57

condition. As the vehicle number increases from scenario 1 (S1) to scenario 3 (S3), the density of live migration requests in certain areas increases dramatically. The resource competition or resource dependency among live container migration requests will also increase. As a result, the complexity of the dependency graph may also increase. When the requirements of live container migration requests exceed the resource capacity provided by the edge computing, it is inevitable that some of the deadlines of some migration requests can not be satisfied.

Table 6.4 shows the total processing time of migration planning and scheduling algorithms within 1 hour in milliseconds. From S1 to S3, the average processing time of single-MCs for each migration planning is 0.1175, 0.1936, and 0.4904 milliseconds. Compared to iter-GWIN, the processing time of single-MCs decreased by 61.36% in scenario S3. The results are consistent with the algorithm evaluation in Section 6.6. Furthermore, compared to FPTAS [30], the performance of our solution in terms of processing time has been improved by more than 3000 times. In S3, the processing time of FPTAS is about 78 minutes. As a result, even with any weight modification in the algorithm, the migration deadline in seconds will be missed. Therefore, as the results of FPTAS-BW in deadline violation are off the limit of chart comparison, we only compare it in the migration performance.

From S1 to S3, without migration planning and scheduling, more live migrations compete with each other on the network routing and the available bandwidth. As a result, the average migration time increases dramatically from 2.25 and 4.59 seconds to 299.89 seconds (Fig. 6.12(a)). Particularly, in S3, the allocated bandwidth may either be smaller than the dirty page rate and cause a large downtime for some migrations. Or, it causes a much longer migration time due to a large number of memory-copying iterations. As a result, the migrating service suffers a devastating consequence. Furthermore, for FPTAS-BW, by maximizing the total migration bandwidth rather than the resource competitions, it suffers smaller average bandwidth per migration. Thus, as shown in Fig. 6.12 the performance of our purposed solution in terms of average migration time, average downtime, and total transferred data are increased by up to 30.24%, 51.56%, and 2.06%, respectively. Meanwhile, for the proposed planning and scheduling algorithms iter-GWIN and single-MCs, the performance of live migration can be guaranteed even with severe resource competitions. Results (Fig. 6.12(a), 6.12(b)) show that the average migration time and downtime are optimal at 1.9 sand 0.13 seconds as there is no bandwidth sharing between resource-dependent migrations. Furthermore, for all the migrations that arrive within the 3600 seconds time interval in S3, iter-GWIN and single-MCs can finish the scheduling of all migrations in 3603.91 and 3601.43 seconds. However, the total migration time of no scheduler is 3603.43 seconds in S2 and 48878.65 seconds in S3. A shorter average migration means less possibility of QoS degradation and less occupation time on the network resource. A smaller downtime equals fewer disruptions on the migrating services.

Another critical migration performance is the transferred data of the live migration. It is also highly related to network energy efficiency. In S1 and S2, although average migration time and downtime increase due to less allocated bandwidth, there is no surge in the transferred data for the no migration scheduling situation (Fig. 6.12(c)). Because of the container's small memory footprint, the shared bandwidth can still satisfy the downtime threshold with relatively small memory-copying iterations. However, when the bandwidth becomes the bottleneck, a large number of memory-copying iteration needed to meet the downtime threshold. Therefore, the total transferred data in S3 increase by 114.47% compared with the optimal result from single-MCs.

The deadline of a live migration request is highly related to the QoS and SLA requirement of the real-time migrating service. For iter-GWIN and single-MCs, the ratio of migration violation numbers to the total migration number is 0.071% and 0.107% in S2 and 0.756% and 1.002% in S3 (Fig. 6.12(d)). However, the ratio for no migration scheduler is 3.07 times in S2 and 8.46 times compared to the best result from iter-GWIN. The ratio of total violation time to the service time of all containers in one hour is 0.00127%

and 0.00148% in S2 and 0.0425% and 0.0720% in S3, respectively (Fig. 6.12(e)). In S3, although migration performance in terms of migration time and downtime is optimized by the migration scheduler, the network resource is insufficient to schedule all 37822 migration requests on time with the live migration competitions. It is inevitable to violate the deadline of certain migrations with lower priority to satisfy the deadline for others. As a solution, one needs to increase the network resource by providing duplicate EDCs and additional network routing or available bandwidth in the hot spot to alleviate the deadline violation of real-time migrations.

The end-to-end delay for the migrating edge service is affected by the migration downtime and the duration of deadline violation. For the network transmission time, we compare the results of no user movement and no migration, no migration requests with user movement, no scheduler, iter-GWIN and single-MCs (Fig. 6.12(f)). In the scenario that all vehicles stay at the s and do not move during the experiment time (nomov), the average network transmission time to the service or the end-user is from 17.4 to 19.9 milliseconds from S3 to S1. Without the live migration requests (nomig), the end-to-end delay can be not guaranteed due to the network delay between the EDC and the end-user. Specifically, the average network transmission time is around 56 milliseconds. The live migration planning and scheduling algorithm (iter-GWIN and single-MCs) can guarantee the average service network transmission time. In S3, without a migration scheduler, the downtime and deadline violation have a considerable impact on the service network delay. The network delay increases by 6.62 times compared to the result of iter-GWIN.

In summary, our proposed algorithms can efficiently plan and schedule large-scale mobility-induced live container migrations in edge computing. Even in the case of a migration request surge, it guarantees the performance of live container migrations and maintains the QoS of migrating services. It significantly reduces average migration time (up to 99.36%), average down time (up to 99.94%), total deadline violations (up to 88.18%) and violation time (up to 99.94%).

6.8 Summary

In this chapter, we investigated the challenges of live container migration scheduling in edge computing environments including (1) resource competition or dependency among live migrations and (2) real-time migration planning and scheduling. We modeled the relationship of resource dependency among migrations as an undirected graph. and the scheduling problem as generating the maximum independent set of the dependency graph iteratively. We proposed a framework for user-triggered or mobility-induced migration scheduling which is different from the traditional scheduling for live VM migrations in cloud data centers. The SDN is introduced to separate the computer network to minimize the impact of migration flows on other edge services. Based on the dynamic computing resources, network resources and topology provided by the container/VM orchestration engine and SDN controllers, the migration management service can plan and schedule multiple migration requests in a fine-grained manner. We proposed two methods for large-scale migration planning and scheduling algorithms based on iterative Maximal Independent Sets. Computational experiments were conducted to evaluate the algorithms' performance. Furthermore, the results of experiments based on real-world data indicate that proposed algorithms can efficiently plan and schedule large-scale mobility-induced live container migrations in a complex network environment in a timely manner, while maintaining the QoS of migrating services. It can optimize the live migration performance and minimize the deadline violation in migration scheduling.

Chapter 7
Conclusions and Future Directions

This chapter concludes the thesis and highlights the key contributions. It also discusses future research directions for live migration management in edge and cloud computing.

7.1 Conclusions and Discussion

The introduction of live migration offers flexible resource reallocation when it is applied in cloud computing environments. Resource management through migration requests enables dynamic adaption and reconfiguration in both computing and networking resources to achieve various optimizations, such as load balancing, traffic consolidation, energy efficiency, QoS awareness, and SLA guarantee. Minimizing the migration overheads and maximizing the scheduling performance of multiple migration are the key components for a successful resource reallocation. The massive demand of cloud and edge services results in complex, large-scale, and heterogeneous migration environments.

The state-of-the-art solutions neglect resource dependency, migration concurrency and timeliness in processing time and scheduling window, which have become inadequate in the current cloud and edge computing environments. Traditional data center networks can not provide virtual traffic management and topology discovery required for multiple migration management in complex and large-scale systems. With the adoption of SDN in clouds, it is feasible and efficient to integrate both computing and networking resources to manage the multiple migration generation policies and multiple migration scheduling algorithms. In this thesis, we investigate multiple migration management in both edge and cloud computing environments to optimize migration performance and satisfy various objectives of resource policies. In detail, Chapter 1 introduces the fundamental background of SDN-enabled cloud computing and live migration, and delineates the research problems and corresponding objectives.

Chapter 2 presented a taxonomy and comprehensive literature review of migration management in parameter and metrics, performance and cost model, resource management policies, migration generation, granularity, lifecycle and orchestration, and management framework. We also investigated the migration planning and scheduling algorithms in objectives, scopes, scheduling types and methods. A conceptual system architecture and various simulation and empirical evaluation methods were presented.

Chapter 3 investigated the performance and impact of live virtual machine migration in OpenStack and OpenDayLight platform under various parameters, configurations, and scenarios, such as CPU utilization, memory overheads, bandwidth allocation, iteration threshold, downtime configuration, parallel migration, network routing, flow entry latency, response time for different application types, and hybrid migration. We also presented the mathematical model for live block migration and sequential and parallel migrations.

Chapter 4 presented a generic concurrency-aware live migration generation algorithm integrating with existing dynamic resource management policies and strategies to optimize the scheduling performance of multiple migration requests and satisfy the policy objectives. The proposed graph-based algorithm evaluates the potential migration combinations of candidate source, destination, and VM and minimizes the migration overheads and the potential resource dependencies among generated migration requests.

Chapter 5 proposed an SLA-Aware multiple migration planner and scheduler that composes a deadline-aware multiple migration grouping algorithm and an online migration scheduler to determine the migration sequence of connected VMs and VNFs. The proposed solution efficiently reduces the number of deadline violations and achieves a good migration performance in total and individual migration time, downtime, transferred data. We also analyzed the impacts of multiple migrations on application QoS and energy consumption.

Chapter 6 proposed an efficient multiple migration scheduling algorithm for live container migration at scale. A novel migration framework was proposed to adapt to the stochastic migration arrival pattern in edge computing environments. The proposed iterative Maximal Independent Set-based planning and scheduling algorithm can greatly reduce the processing time to suit the real-time requirement of mobility-induced migrations in large-scale edge computing while providing the guaranteed migration performance for time-critical migration requests.

The chapters mentioned above collectively present the multiple migration management in SDN-enabled clouds, which is a timely contribution to the state-of-the-art.

7.2 Future Research Directions

From cloud computing to mobile edge computing, service requirements of low latency and high mobility bring new challenges to live migration management. This thesis provides fundamental framework and solutions to the multiple migration management in edge and cloud computing environments. The solutions of Chapter 4 and 5 can be applied to the periodic multiple migration management scenarios such as dynamic resource and energy management through live VM and container migrations in edge computing. The novel framework and algorithms of Chapter 6 are suitable for the stochastic multiple migration scheduling scenarios in mobile edge computing, such as mobility-induced service migration, with high mobility and low latency.

This thesis addressed several challenges of migration management in cloud and edge data centers. However, cloud and edge computing can be improved by addressing several key issues of live migration that require further investigation. Figure 7.1 illustrates the overview of future directions discussed in this section.

7.2.1 Flexible Networking Management by Network Slicing

The containerized services allocated in the mobile edge clouds bring up the opportunity for large-scale and real-time applications to have low latency responses. Meanwhile, live container migration is introduced to support dynamic resource management and users' mobility. The container footprint becomes much smaller compared to the VM. As

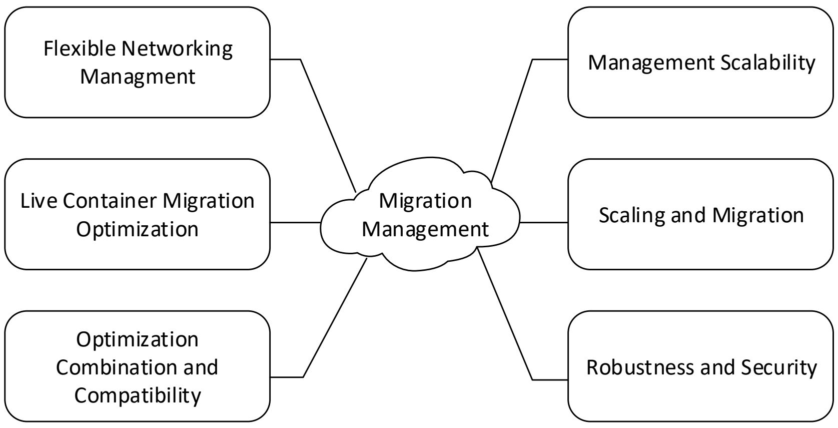

Flexible Networking Managment
Live Container Migration Optimization
Optimization Combination and Compatibility
Migration Management
Management Scalability
Scaling and Migration
Robustness and Security

Figure 7.1: Future directions

a result, the proportion of network resources limitations on migration performance is reduced. Instance-level parallelizing for multiple migration can improve the scheduling performance due to the computing cost of migration. Therefore, it is critical to investigate the networking slicing algorithms to concurrently schedule both VM and container migrations to improve multiple migration performance and alleviate the impacts on service traffic. Holistic solutions are needed to manage the networking routing and bandwidth allocation based on the various network requirements of live VM migrations, live container migrations, and applications.

7.2.2 Optimization Modeling, Compatibility and Combination

There are continuous efforts striving to improve the performance and alleviate the overheads of live migration through system-level optimization [6]. The disadvantage of the original pre-copy migration is the migration convergence. The optimization works of improving live migration performance mainly focus on reducing the live migration time and downtime, including reducing the memory data required for transmission to the destination, speeding up the migration process in the source host, and increasing the bandwidth between the source and destination hosts. However, there are gaps between current migration cost and performance modeling and the exiting migration optimization mechanisms in process parallelizing, compression, de-duplication, memory introspection, checkpoint and recovery, remote direct memory access (RDMA), and application awareness. The existing migration optimization is at the system level which needs to be modeled carefully and properly to reflect the nature and characteristics of the optimization. Although there are extensive optimizations on live VM migration, each one claims a certain performance improvement compared to the default live VM migration mechanism. However, it is unclear the compatibility of each migration optimization technology and the performance improvement of the combination of various mechanisms.

7.2.3 Live Container Migration Optimization

There is a research interest and trend of adapting or directing imitating optimization mechanisms of live VM migration to develop and improve live container migrations [23, 76, 207, 212, 224]. For example, Remote Direct Memory Access (RDMA) can be utilized as the high-speed interconnect techniques [225], which has been applied to improve the performance of live VM migration. RDMA enables the remote accessing of memory and disk data without the CPU and cache. Multipath TCP (MPTCP) can be used to allow TCP connections to use multiple paths to increase the network throughput and redundancy [226]. The container layered storage can be leveraged to alleviate synchronization overheads of a file system in the architecture without shared storage [210]. However, there are still gaps in this research direction and more evaluation and investigation of single and multiple container migrations with real applications need to be done in cloud and edge data centers to speed up this improvement for live container migration and its scheduling.

7.2.4 Management Scalability

With the expansion of edge and cloud computing networks, the complexity of the migration planning and scheduling problem becomes unavoidably large. However, the timeliness requirement of the management algorithm is also changed from cloud to edge computing. For the time-critical applications, the resource and migration management algorithms need to be scalable to suit the large-scale computing and networking environments. Therefore, it is essential to investigate the distributed management framework and algorithms to enhance scalability of the migration management algorithms. Controller placement problems need to be investigated regarding the SDN controller capacity and the network latency between controller and OpenFlow switches. The problems of edge data center placement and base station clustering need to be investigated based on the user mobility information to reduce the number of live migrations. Each edge manager and SDN controller needs to cover a certain area and data centers and cooperates with other controllers. A certain strategy needs to be developed to determine the size of the cluster area and the placement of each manager and controller based on the parameters such as network delay and processing capability.

7.2.5 Autoscaling and Live Migration

For instance-level scaling, scaling up and down is used for allocation and deallocation virtualized instances in cloud computing to elastically provision the resource in the same host. In addition, system-level scaling up [227] supports fine-grained scaling on the system resources, such as CPU, memory, and I/O, to reduce the considerable overhead and extra cost. On the other hand, scaling out is used to support application allocation in other compute nodes to increase the processing capacity of the service. The load-balancer will distribute the traffic to all running instances of the application.

Current resource management strategies of cloud providers perform live migration for both stateful and stateless services to achieve the objectives, such as energy consumption and traffic consolidation. There is no configuration and information to differentiate the instance types (stateful and stateless) across various cloud types (IaaS, PaaS, SaaS, and FaaS). The cloud or service providers can further reduce management overheads by performing autoscaling and live migration to stateful and stateless instances respectively. Therefore, it is critical to integrate the autoscaling and live migration strategies to holistically manage the resources based on the instance types to minimize the management overhead and cost. The advantages and disadvantages of instance-level scaling, resource-level scaling, and live migration need to be investigated. In addition, the specific SLA of migration and scaling is needed for various instance types.

7.2.6 Robustness and Security

For the migration security, Shetty et al. [73] focus on secure live migration, control policies (DoS, Internal, Guest VM, false resource advertise, Inter VM in the same host), transmission channel (insecure and unprotected), and migration module (stack, heap, integer overflow). SDN-based network resilience management and strategies, such as traffic shaping, anomaly detection, and traffic classification, need to be investigated to tackle the network security issue for live migration.

Furthermore, migration robustness also needs to be investigated to increase the availability and accessibility of dynamic resource management. Compared to pre-copy migration, post-copy migration starts the instance at the destination host as soon as the initial memory is copied, which makes it vulnerable to state synchronization failure. As a result, if there is a post-copy migration failure, the running processes can not be resumed and the migrating instance is shutdown. Fernando et al. [67] proposed a failure recovery mechanism for post-copy migration to alleviate the cost of post-copy migration failure. Furthermore, failure cost models need to be developed based on different migration types and mechanisms and applied to services with various SLA levels accordingly.

7.3 Final Remarks

Edge and cloud computing have become the backbone of the digital world for hosting applications in science, business, transportation, and content delivery. Resource management through live migration enables the edge and cloud data centers to reach their full potentials in performance, availability, energy efficiency and running cost. This thesis investigated how to efficiently manage live migrations during resource management in SDN-enabled Clouds, including SDN networking management, migration overheads, migration performance, application QoS, scheduling timeliness, scheduling concurrency, management framework, competitions of migration generation, and migration planning and scheduling. The proposed methods and solutions guarantee migration timeliness, reduce energy consumption, minimize migration overheads, SLA violations, and QoS degradations, maximize the performance of individual migration and multiple migration scheduling, and facilitate and improve the performance and effectiveness of dynamic resource management. Research of migration management, such as presented in this thesis, will enable cloud and edge providers to successfully and efficiently perform live migrations in complex edge and cloud computing environments at scale. Moreover, these research outcomes can drive further innovations and developments of edge and cloud computing systems.

Bibliography

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing—the business perspective," *Decision Support Systems*, vol. 51, no. 1, pp. 176–189, 2011.

[3] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[4] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[5] A. M. Joy, "Performance comparison between linux containers and virtual machines," in *Proceedings of 2015 International Conference on Advances in Computer Engineering and Applications*. IEEE, 2015, pp. 342–346.

[6] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, "A survey on virtual machine migration: Challenges, techniques, and open issues," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1206–1243, 2018.

[7] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23 511–23 528, 2018.

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd confer-*

ence on Symposium on Networked Systems Design & Implementation-Volume 2, 2005, pp. 273–286.

[9] "Container migration with Podman on RHEL," accessed 22 Jan 2020. [Online]. Available: https://www.redhat.com/en/blog/container-migration-podman-rhel

[10] "Dynamic resource management in e2 vms," accessed 29 June 2021. [Online]. Available: https://cloud.google.com/blog/products/compute/understanding-dynamic-resource-management-in-e2-vms

[11] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*, 2015, pp. 1–17.

[12] V. Marmol and A. Tucker, "Task migration at scale using criu," in *Proceedings of Linux Plumbers Conference*, 2018.

[13] A. Ruprecht, D. Jones, D. Shiraev, G. Harmon, M. Spivak, M. Krebs, M. Baker-Harvey, and T. Sanderson, "Vm live migration at scale," *ACM SIGPLAN Notices*, vol. 53, no. 3, pp. 45–56, 2018.

[14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[15] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.

[16] E. Harney, S. Goasguen, J. Martin, M. Murphy, and M. Westall, "The efficacy of live virtual machine migrations over the internet," in *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing (VTDC'07)*. IEEE, 2007, pp. 1–7.

[17] K. Kirkpatrick, "Software-defined networking," *Communications of the ACM*, vol. 56, no. 9, pp. 16–19, 2013.

[18] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[19] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[20] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[21] S. S. Manvi and G. K. Shyam, "Resource management for infrastructure as a service (iaas) in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 41, pp. 424–440, 2014.

[22] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, 2015.

[23] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 140–147, 2017.

[24] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–37, 2019.

[25] Z. Rejiba, X. Masip-Bruin, and E. Marín-Tordera, "A survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–33, 2019.

[26] U. Deshpande and K. Keahey, "Traffic-sensitive live migration of virtual machines," *Future Generation Computer Systems*, vol. 72, pp. 118–128, 2017.

[27] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer, "Remedy: Network-aware steady state vm management for data centers," in *Proceedings of International Conference on Research in Networking*. Springer, 2012, pp. 190–204.

[28] S. Ghorbani and M. Caesar, "Walk the line: consistent network updates with bandwidth guarantees," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. ACM, 2012, pp. 67–72.

[29] M. F. Bari, M. F. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "Cqncr: Optimal vm migration planning in cloud data centers," in *Proceedings of 2014 IFIP Networking Conference*. IEEE, 2014, pp. 1–9.

[30] H. Wang, Y. Li, Y. Zhang, and D. Jin, "Virtual machine migration planning in software-defined networks," *IEEE Transactions on Cloud Computing*, vol. 7, no. 4, pp. 1168–1182, 2019.

[31] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.

[32] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, 2008, pp. 75–86.

[33] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, 2009, pp. 63–74.

[34] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks." in *Proceedings of the 7th {USENIX} Conference on Networked Systems Design and Implementation ({NSDI} 10)*, vol. 10, 2010, pp. 249–264.

[35] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 183–197, 2015.

[36] N. Farrington and A. Andreyev, "Facebook's data center network architecture," in *Proceedings of 2013 Optical Interconnects Conference*. Citeseer, 2013, pp. 49–50.

[37] Azure, "Global network infrastructure," 2021. [Online]. Available: https://azure.microsoft.com/en-au/global-infrastructure/global-network/

[38] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 123–137.

[39] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[40] R. Durairajan, S. Ghosh, X. Tang, P. Barford, and B. Eriksson, "Internet atlas: a geographic database of the internet," in *Proceedings of the 5th ACM Workshop on HotPlanet*, 2013, pp. 15–20.

[41] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology," RFC 7426, Jan. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7426.txt

[42] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[43] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *Proceedings of 2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–4.

[44] O. N. Foundation, "Openflow switch specification version 1.5.1," 2015. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[45] ——, "Openflow management and configuration protocol," 2014. [Online]. Available: https://opennetworking.org/sdn-resources/onf-specifications/openflow-config/openflow-config/

[46] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[47] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013, pp. 13–18.

[48] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third Workshop on Hot Topics in Software Defined Networking*, 2014, pp. 1–6.

[49] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE, 2014, pp. 1–6.

[50] Ryu, "Ryu sdn framework," 2019. [Online]. Available: https://github.com/faucetsdn/ryu

[51] Floodlight, "Openflow controller," 2019. [Online]. Available: https://github.com/floodlight/floodlight

[52] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *Proceedings of 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 117–130.

[53] T. L. Foundation, "Data plane development kit," 2021. [Online]. Available: https://www.dpdk.org/

[54] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with sr-iov," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471–1480, 2012.

[55] fd.io, "Vector packet processing," 2021. [Online]. Available: https://wiki.fd.io/view/VPP

[56] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy, "Softnic: A software nic to augment hardware," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-155*, 2015.

[57] J. Son and R. Buyya, "A taxonomy of software-defined networking (sdn)-enabled cloud computing," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–36, 2018.

[58] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[59] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (sd-wan): Architecture, advances and opportunities," in *Proceedings of 2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019, pp. 1–9.

[60] CRIU, "Live migration," 2019. [Online]. Available: https://criu.org/Live_migration

[61] K. Nagin, D. Hadas, Z. Dubitzky, A. Glikson, I. Loy, B. Rochwerger, and L. Schour, "Inter-cloud mobility of virtual machines," in *Proceedings of the 4th Annual International Conference on Systems and Storage*, 2011, pp. 1–12.

[62] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Computing Surveys (CSUR)*, vol. 47, no. 1, pp. 1–47, 2014.

[63] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vnf placement at the network edge," in *Proceedings of IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 693–701.

[64] Huawei, "White paper: 5g network architecture," 2020. [Online]. Available: https://www.huawei.com/en/technology-insights/industry-insights/outlook/mobile-broadband/insights-reports/5g-network-architecture

[65] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 3, pp. 14–26, 2009.

[66] S. Sahni and V. Varma, "A hybrid approach to live migration of virtual machines," in *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. IEEE, 2012, pp. 1–5.

[67] D. Fernando, J. Terner, K. Gopalan, and P. Yang, "Live migration ate my vm: Recovering a virtual machine after failure of post-copy live migration," in *Proceedings of IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 343–351.

[68] A. Shribman and B. Hudzia, "Pre-copy and post-copy vm live migration for memory intensive applications," in *Proceedings of European Conference on Parallel Processing*. Springer, 2012, pp. 539–547.

[69] Z. Wang, G. Xue, S. Qian, G. Liu, M. Li, J. Cao, and J. Yu, "Ada-copy: An adaptive memory copy strategy for virtual machine live migration," in *Proceedings of 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2017, pp. 461–468.

[70] Z. Wang, D. Sun, G. Xue, S. Qian, G. Li, and M. Li, "Ada-things: An adaptive virtual machine monitoring and migration strategy for internet of things applications," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 164–176, 2019.

[71] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya, "Cloudsimsdn: Modeling and simulation of software-defined cloud data centers," in *Proceedings of 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2015, pp. 475–484.

[72] A. Strunk, "Costs of virtual machine live migration: A survey," in *Proceedings of 2012 IEEE Eighth World Congress on Services*. IEEE, 2012, pp. 323–329.

[73] J. Shetty, M. Anala, and G. Shobha, “A survey on techniques of secure live migration of virtual machine,” *International Journal of Computer Applications*, vol. 39, no. 12, pp. 34–39, 2012.

[74] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, “Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions,” *Proceedings of the IEEE*, vol. 102, no. 1, pp. 11–31, 2013.

[75] V. Medina and J. M. García, “A survey of migration mechanisms of virtual machines,” *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, pp. 1–33, 2014.

[76] W. Li and A. Kanso, “Comparing containers versus virtual machines for achieving high availability,” in *Proceedings of 2015 IEEE International Conference on Cloud Engineering*. IEEE, 2015, pp. 353–358.

[77] H. Yamada, “Survey on mechanisms for live virtual machine migration and its improvements,” *Information and Media Technologies*, vol. 11, pp. 101–115, 2016.

[78] M. Noshy, A. Ibrahim, and H. A. Ali, “Optimization of live virtual machine migration in cloud computing: A survey and future directions,” *Journal of Network and Computer Applications*, vol. 110, pp. 1–10, 2018.

[79] D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, “Process migration,” *ACM Computing Surveys (CSUR)*, vol. 32, no. 3, pp. 241–299, 2000.

[80] P. Kokkinos, D. Kalogeras, A. Levin, and E. Varvarigos, “Survey: Live migration and disaster recovery over long-distance networks,” *ACM Computing Surveys (CSUR)*, vol. 49, no. 2, pp. 1–36, 2016.

[81] “Kubernetes kuryr,” accessed 29 June 2021. [Online]. Available: https://docs.openstack.org/kuryr-kubernetes/latest/devref/kuryr_kubernetes_design.html

[82] “Kubernetes cluster networking,” accessed 29 June 2021. [Online]. Available: https://kubernetes.io/docs/concepts/cluster-administration/networking/

[83] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, "Live migration of virtual machine based on full system trace and replay," in *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, 2009, pp. 101–110.

[84] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the performance of virtual machine migration," in *Proceedings of 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2010, pp. 37–46.

[85] D. Breitgand, G. Kutiel, and D. Raz, "Cost-aware live migration of services in the cloud." *SYSTOR*, vol. 10, pp. 1 815 695–1 815 709, 2010.

[86] B. Wei, C. Lin, and X. Kong, "Energy optimized modeling for live migration in virtual data center," in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, vol. 4. IEEE, 2011, pp. 2311–2315.

[87] S. Kikuchi and Y. Matsumoto, "Performance modeling of concurrent live migration operations in cloud computing systems using prism probabilistic model checker," in *Proceedings of 2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 2011, pp. 49–56.

[88] H. Liu, H. Jin, C.-Z. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines," *Cluster Computing*, vol. 16, no. 2, pp. 249–264, 2013.

[89] F. Callegati and W. Cerroni, "Live migration of virtualized edge networks: Analytical modeling and performance evaluation," in *Proceedings of 2013 IEEE SDN for Future Networks and Services (SDN4FNS)*. IEEE, 2013, pp. 1–6.

[90] F. Xu, F. Liu, L. Liu, H. Jin, B. Li, and B. Li, "iaware: Making live migration of virtual machines interference-aware in the cloud," *IEEE Transactions on Computers*, vol. 63, no. 12, pp. 3012–3025, 2014.

[91] Q. Huang, K. Shuang, P. Xu, J. Li, X. Liu, and S. Su, "Prediction-based dynamic resource scheduling for virtualized cloud systems," *Journal of Networks*, vol. 9, no. 2, p. 375, 2014.

[92] M. Aldossary and K. Djemame, "Performance and energy-based cost prediction of virtual machines live migration in clouds." pp. 384–391, 2018.

[93] A. Strunk and W. Dargie, "Does live migration of virtual machines cost energy?" in *Proceedings of 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2013, pp. 514–521.

[94] W. Hu, A. Hicks, L. Zhang, E. M. Dow, V. Soni, H. Jiang, R. Bull, and J. N. Matthews, "A quantitative study of virtual machine live migration," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, 2013, pp. 1–10.

[95] Q. Huang, F. Gao, R. Wang, and Z. Qi, "Power consumption of virtual machine live migration in clouds," in *Proceedings of 2011 Third International Conference on Communications and Mobile Computing*. IEEE, 2011, pp. 122–125.

[96] A. Strunk, "A lightweight model for estimating energy cost of live migration of virtual machines," in *Proceedings of 2013 IEEE Sixth International Conference on Cloud Computing*. IEEE, 2013, pp. 510–517.

[97] M. E. Elsaid and C. Meinel, "Live migration impact on virtual datacenter performance: Vmware vmotion based study," in *Proceedings of 2014 International Conference on Future Internet of Things and Cloud*. IEEE, 2014, pp. 216–221.

[98] Z. Li and G. Wu, "Optimizing vm live migration strategy based on migration time cost modeling," in *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*. ACM, 2016, pp. 99–109.

[99] C. Jo, Y. Cho, and B. Egger, "A machine learning approach to live migration modeling," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 351–364.

[100] M. E. Elsaid, H. M. Abbas, and C. Meinel, "Machine learning approach for live migration cost prediction in vmware environments." pp. 456–463, 2019.

[101] K. Rybina, A. Patni, and A. Schill, "Analysing the migration time of live migration of multiple virtual machines." *CLOSER*, vol. 14, pp. 590–597, 2014.

[102] D. Fernando, P. Yang, and H. Lu, "Sdn-based order-aware live migration of virtual machines," in *Proceedings of IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1818–1827.

[103] K. Tsakalozos, V. Verroios, M. Roussopoulos, and A. Delis, "Live vm migration under time-constraints in share-nothing iaas-clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2285–2298, 2017.

[104] T. K. Sarker and M. Tang, "Performance-driven live migration of multiple virtual machines in datacenters," in *Proceedings of 2013 IEEE International Conference on Granular Computing (GrC)*. IEEE, 2013, pp. 253–258.

[105] V. Mann, A. Vishnoi, A. Iyer, and P. Bhattacharya, "Vmpatrol: Dynamic and automated qos for virtual machine migrations," in *Proceedings of 2012 8th International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualiztion Management (SVM)*. IEEE, 2012, pp. 174–178.

[106] L. Cui, R. Cziva, F. P. Tso, and D. P. Pezaros, "Synergistic policy and virtual machine consolidation in cloud data centers," in *Proceedings of IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[107] L. Cui, F. P. Tso, D. P. Pezaros, W. Jia, and W. Zhao, "Plan: Joint policy-and network-aware vm management for cloud data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1163–1175, 2016.

[108] H. Flores, V. Tran, and B. Tang, "Pam & pal: Policy-aware virtual machine migration and placement in dynamic cloud data centers," in *Proceedings of IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2549–2558.

[109] D. Kakadia, N. Kopri, and V. Varma, "Network-aware virtual machine consolidation for large data centers," in *Proceedings of the Third International Workshop on Network-Aware Data Management*, 2013, pp. 1–8.

[110] F. P. Tso, G. Hamilton, K. Oikonomou, and D. P. Pezaros, "Implementing scalable, network-aware virtual machine migration for cloud data centers," in *Proceedings of 2013 IEEE Sixth International Conference on Cloud Computing*. IEEE, 2013, pp. 557–564.

[111] W. Zhang, S. Han, H. He, and H. Chen, "Network-aware virtual machine migration in an overcommitted cloud," *Future Generation Computer Systems*, vol. 76, pp. 428–442, 2017.

[112] Y. Cui, Z. Yang, S. Xiao, X. Wang, and S. Yan, "Traffic-aware virtual machine migration in topology-adaptive dcn," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3427–3440, 2017.

[113] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2008–2025, 2017.

[114] B. Martini, D. Adami, M. Gharbaoui, P. Castoldi, L. Donatini, and S. Giordano, "Design and evaluation of sdn-based orchestration system for cloud data centers," in *Proceedings of 2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[115] R. Cziva, S. Jouët, D. Stapleton, F. P. Tso, and D. P. Pezaros, "Sdn-based virtual machine management for cloud data centers," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 212–225, 2016.

[116] A. Mayoral, R. Munoz, R. Vilalta, R. Casellas, R. Martínez, and V. López, "Need for a transport api in 5g for global orchestration of cloud and networks through a virtualized infrastructure manager and planner," *Journal of Optical Communications and Networking*, vol. 9, no. 1, pp. A55–A62, 2017.

[117] A. Mayoral, R. Vilalta, R. Muñoz, R. Casellas, and R. Martínez, "Sdn orchestration architectures and their integration with cloud computing applications," *Optical Switching and Networking*, vol. 26, pp. 2–13, 2017.

[118] A. Mayoral, R. Vilalta, R. Muñoz, R. Casellas, R. Martínez, M. S. Moreolo, J. M. Fabrega, A. Aguado, S. Yan, D. Simeonidou *et al.*, "Control orchestration protocol: Unified transport api for distributed cloud and network orchestration," *Journal of Optical Communications and Networking*, vol. 9, no. 2, pp. A216–A222, 2017.

[119] S. Fichera, M. Gharbaoui, P. Castoldi, B. Martini, and A. Manzalini, "On experimenting 5g: Testbed set-up for sdn orchestration across network cloud and iot domains," in *Proceedings of 2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2017, pp. 1–6.

[120] D. Harrington, R. Presuhn, and B. Wijnen, "Rfc3411: An architecture for describing simple network management protocol (snmp) management frameworks," 2002.

[121] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes, "Cisco systems netflow services export version 9," 2004.

[122] P. Phaal, S. Panchen, and N. McKee, "Rfc3176: Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks," 2001.

[123] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments," *Computer Networks*, vol. 62, pp. 122–136, 2014.

[124] P. Wang, K.-M. Chao, H.-C. Lin, W.-H. Lin, and C.-C. Lo, "An efficient flow control approach for sdn-based network threat detection and migration using support vector machine," in *Proceedings of 2016 IEEE 13th international conference on e-business engineering (ICEBE)*. IEEE, 2016, pp. 56–63.

[125] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González, "Towards sflow and adaptive polling sampling for deep learning based ddos detection in sdn," *Future Generation Computer Systems*, vol. 111, pp. 763–779, 2020.

[126] S. U. Rehman, W.-C. Song, and M. Kang, "Network-wide traffic visibility in of@ tein sdn testbed using sflow," in *Proceedings of The 16th Asia-Pacific Network Operations and Management Symposium*. IEEE, 2014, pp. 1–6.

[127] M. Afaq, S. Rehman, and W.-C. Song, “Large flows detection, marking, and mitigation based on sflow standard in sdn,” *Journal of Korea Multimedia Society*, vol. 18, no. 2, pp. 189–198, 2015.

[128] F. Paolucci, F. Cugini, N. Hussain, F. Fresi, and L. Poti, “Openflow-based flexible optical networks with enhanced monitoring functionalities,” in *Proceedings of European Conference and Exhibition on Optical Communication*. Optical Society of America, 2012, pp. Tu–1.

[129] C.-Y. Lin, C. Chen, J.-W. Chang, and Y. H. Chu, “Elephant flow detection in datacenters using openflow-based hierarchical statistics pulling,” in *Proceedings of 2014 IEEE Global Communications Conference*. IEEE, 2014, pp. 2264–2269.

[130] H. Xu, Z. Yu, C. Qian, X.-Y. Li, and Z. Liu, “Minimizing flow statistics collection cost of sdn using wildcard requests,” in *Proceedings of IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[131] S. Bera, S. Misra, and A. Jamalipour, “Flowstat: Adaptive flow-rule placement for per-flow statistics in sdn,” *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 530–539, 2019.

[132] F. Rossigneux, L. Lefevre, J.-P. Gelas, and M. D. De Assuncao, “A generic and extensible framework for monitoring energy consumption of openstack clouds,” in *Proceedings of 2014 IEEE Fourth International Conference on Big Data and Cloud Computing*. IEEE, 2014, pp. 696–702.

[133] A. Uchechukwu, K. Li, Y. Shen *et al.*, “Energy consumption in cloud computing data centers,” *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, vol. 3, no. 3, pp. 31–48, 2014.

[134] J. Zhang, F. Ren, and C. Lin, “Delay guaranteed live migration of virtual machines,” in *Proceedings of IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 574–582.

[135] A. Singh, M. Korupolu, and D. Mohapatra, “Server-storage virtualization: in-

tegration and load balancing in data centers," in *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE, 2008, pp. 1–12.

[136] A. Verma, P. Ahuja, and A. Neogi, "pmapper: power and migration cost aware application placement in virtualized systems," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware'08)*. Springer, 2008, pp. 243–264.

[137] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Sandpiper: Black-box and gray-box resource management for virtual machines," *Computer Networks*, vol. 53, no. 17, pp. 2923–2938, 2009.

[138] M. Forsman, A. Glad, L. Lundberg, and D. Ilie, "Algorithms for automated live migration of virtual machines," *Journal of Systems and Software*, vol. 101, pp. 110–126, 2015.

[139] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1107–1117, 2012.

[140] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[141] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[142] J. N. Witanto, H. Lim, and M. Atiquzzaman, "Adaptive selection of dynamic vm consolidation algorithm using neural network for cloud resource management," *Future Generation Computer Systems*, vol. 87, pp. 35–42, 2018.

[143] X. Li, P. Garraghan, X. Jiang, Z. Wu, and J. Xu, "Holistic virtual machine scheduling in cloud datacenters towards minimizing total energy," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1317–1331, 2017.

[144] J. T. Piao and J. Yan, “A network-aware virtual machine placement and migration approach in cloud computing,” in *2010 Ninth International Conference on Grid and Cloud Computing*. IEEE, 2010, pp. 87–92.

[145] B. Cao, X. Gao, G. Chen, and Y. Jin, “Nice: network-aware vm consolidation scheme for energy conservation in data centers,” in *Proceedings of 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2014, pp. 166–173.

[146] M. A. Khan, A. Paplinski, A. M. Khan, M. Murshed, and R. Buyya, “Dynamic virtual machine consolidation algorithms for energy-efficient cloud resource management: a review,” *Sustainable Cloud and Energy Services*, pp. 135–165, 2018.

[147] U. Deshpande, X. Wang, and K. Gopalan, “Live gang migration of virtual machines,” in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, 2011, pp. 135–146.

[148] U. Deshpande, U. Kulkarni, and K. Gopalan, “Inter-rack live migration of multiple virtual machines,” in *Proceedings of the 6th International Workshop on Virtualization Technologies in Distributed Computing Date*, 2012, pp. 19–26.

[149] U. Deshpande, B. Schlinker, E. Adler, and K. Gopalan, “Gang migration of virtual machines using cluster-wide deduplication,” in *Proceedings of 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 2013, pp. 394–401.

[150] G. Sun, D. Liao, V. Anand, D. Zhao, and H. Yu, “A new technique for efficient live migration of multiple virtual machines,” *Future Generation Computer Systems*, vol. 55, pp. 74–86, 2016.

[151] J. Zheng, T. S. E. Ng, K. Sripanidkulchai, and Z. Liu, “Comma: Coordinating the migration of multi-tier applications,” in *Proceedings of the 10th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2014, pp. 153–164.

[152] H. Liu and B. He, "Vmbuddies: Coordinating live migration of multi-tier applications in cloud environments," *IEEE transactions on parallel and distributed systems*, vol. 26, no. 4, pp. 1192–1205, 2014.

[153] H. Lu, C. Xu, C. Cheng, R. Kompella, and D. Xu, "vhaul: Towards optimal scheduling of live multi-vm migration for multi-tier applications," in *Proceedings of 2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 2015, pp. 453–460.

[154] T. Lu, M. Stuart, K. Tang, and X. He, "Clique migration: Affinity grouping of virtual machines for inter-cloud live migration," in *Proceedings of 2014 9th IEEE International Conference on Networking, Architecture, and Storage*. IEEE, 2014, pp. 216–225.

[155] T. S. Kang, M. Tsugawa, A. Matsunaga, T. Hirofuchi, and J. A. Fortes, "Design and implementation of middleware for cloud disaster recovery via virtual machine migration management," in *Proceedings of 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 2014, pp. 166–175.

[156] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang, "Live migration of multiple virtual machines with resource reservation in cloud computing environments," in *Proceedings of 2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 2011, pp. 267–274.

[157] U. Deshpande, Y. You, D. Chan, N. Bila, and K. Gopalan, "Fast server deprovisioning through scatter-gather live migration of virtual machines," in *Proceedings of 2014 IEEE 7th International Conference on Cloud Computing*. IEEE, 2014, pp. 376–383.

[158] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Proceedings of IEEE International Conference on Cloud Computing*. Springer, 2009, pp. 254–265.

[159] S. Kikuchi and Y. Matsumoto, "Impact of live migration on multi-tier application

performance in clouds," in *Proceedings of 2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*. IEEE, 2012, pp. 261–268.

[160] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "Ofswitch13: Enhancing ns-3 with openflow 1.3 support," in *Proceedings of the Workshop on ns-3*, 2016, pp. 33–40.

[161] D. Klein and M. Jarschel, "An openflow extension for the omnet++ inet framework," in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, 2013, pp. 322–329.

[162] T. Hirofuchi, A. Lèbre, and L. Pouilloux, "Adding a live migration model into simgrid: One more step toward the simulation of infrastructure-as-a-service concerns," in *Proceedings of 2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 1. IEEE, 2013, pp. 96–103.

[163] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[164] M. M. Lopes, W. A. Higashino, M. A. Capretz, and L. F. Bittencourt, "Myifogsim: A simulator for virtual machine migration in fog computing," in *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, 2017, pp. 47–52.

[165] C. Puliafito, D. M. Goncalves, M. M. Lopes, L. L. Martins, E. Madeira, E. Mingozzi, O. Rana, and L. F. Bittencourt, "Mobfogsim: Simulation of mobility and migration for fog computing," *Simulation Modelling Practice and Theory*, vol. 101, p. 102062, 2020.

[166] J. Son, T. He, and R. Buyya, "Cloudsimsdn-nfv: Modeling and simulation of network function virtualization and service function chaining in edge computing environments," *Software: Practice and Experience*, vol. 49, no. 12, pp. 1748–1764, 2019.

[167] C. H. Benet, R. Nasim, K. A. Noghani, and A. Kassler, "Openstackemu—a cloud testbed combining network emulation with openstack and sdn," in *Proceedings of*

2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC). IEEE, 2017, pp. 566–568.

[168] J. Son and R. Buyya, "Sdcon: Integrated control platform for software-defined clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 230–244, 2018.

[169] T. Benjaponpitak, M. Karakate, and K. Sripanidkulchai, "Enabling live migration of containerized applications across clouds," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2529–2538.

[170] X. Song, J. Shi, R. Liu, J. Yang, and H. Chen, "Parallelizing live migration of virtual machines," *ACM SIGPLAN Notices*, vol. 48, no. 7, pp. 85–96, 2013.

[171] J. J. Herne, "Auto-convergence feature," https://wiki.qemu.org/Features/AutoconvergeLiveMigration, accessed 11 Feb 2018, 2015.

[172] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[173] "Libvirt. Virtualization API," https://libvirt.org/, accessed 25 Jan 2018, 2017.

[174] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Measuring control plane latency in sdn-enabled switches," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. ACM, 2015, Conference Proceedings, p. 25.

[175] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *Proceedings of International Conference on Passive and Active Network Measurement*. Springer, 2015, pp. 347–359.

[176] "Open vSwitch," https://www.openvswitch.org/, accessed 15 Jan 2018, 2016.

[177] "QEMU Project. Live block operation documentation," https://kashyapc.fedorapeople.org/QEMU-Docs/_build/html/index.html, accessed 11 Feb 2018, 2017.

[178] A. Mashtizadeh, E. Celebi, T. Garfinkel, M. Cai *et al.*, "The design and evolution of live storage migration in vmware esx," in *Usenix Atc*, vol. 11, no. 2011, 2011, pp. 1–14.

[179] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2011, pp. 111–120.

[180] A. N. Toosi, J. Son, and R. Buyya, "Clouds-pi: A low-cost raspberry-pi based micro data center for software-defined cloud computing," *IEEE Cloud Computing*, vol. 5, no. 5, pp. 81–91, 2018.

[181] "OpenDayLight Carbon release," https://docs.opendaylight.org/en/stable-carbon/index.html, accessed 25 Jan 2018, 2017.

[182] C. King, "Stress-ng," http://kernel.ubuntu.com/~cking/stress-ng/, accessed 24 Feb 2018, 2018.

[183] "Iperf3," https://iperf.fr/, accessed 11 Feb 2018, 2016.

[184] E.-J. Van Baaren, "Wikibench: A distributed, wikipedia based web application benchmark," Master's thesis, VU University Amsterdam, 2009.

[185] T. Guo, U. Sharma, P. Shenoy, T. Wood, and S. Sahu, "Cost-aware cloud bursting for enterprise applications," *ACM Transactions on Internet Technology (TOIT)*, vol. 13, no. 3, p. 10, 2014.

[186] J. Bi, H. Yuan, W. Tan, M. Zhou, Y. Fan, J. Zhang, and J. Li, "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 1172–1184, 2015.

[187] Q. Wu, F. Ishikawa, Q. Zhu, and Y. Xia, "Energy and migration cost-aware dynamic virtual machine consolidation in heterogeneous cloud datacenters," *IEEE transactions on Services Computing*, vol. 12, no. 4, pp. 550–563, 2016.

[188] Q. Peng, Y. Xia, Z. Feng, J. Lee, C. Wu, X. Luo, W. Zheng, S. Pang, H. Liu, Y. Qin *et al.*, "Mobility-aware and migration-enabled online edge user allocation in mobile edge computing," in *Proceedings of 2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 2019, pp. 91–98.

[189] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1272–1288, 2019.

[190] Y. Jiang, J. Wang, J. Shi, J. Zhu, and L. Teng, "Network-aware virtual machine migration based on gene aggregation genetic algorithm," *Mobile Networks and Applications*, vol. 25, pp. 1457—-1468, 2020.

[191] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[192] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[193] E. L. Lawler, J. K. Lenstra, and A. Rinnooy Kan, "Generating all maximal independent sets: Np-hardness and polynomial-time algorithms," *SIAM Journal on Computing*, vol. 9, no. 3, pp. 558–565, 1980.

[194] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Computer Science*, vol. 363, no. 1, pp. 28–42, 2006.

[195] D. R. Lick and A. T. White, "k-degenerate graphs," *Canadian Journal of Mathematics*, vol. 22, no. 5, pp. 1082–1096, 1970.

[196] F. Cazals and C. Karande, "A note on the problem of reporting maximal cliques," *Theoretical Computer Science*, vol. 407, no. 1-3, pp. 564–568, 2008.

[197] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," in *Proceedings of International Symposium on Algorithms and Computation*. Springer, 2010, pp. 403–414.

[198] K. Park and V. S. Pai, “Comon: a mostly-scalable monitoring system for planet-lab,” *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 65–74, 2006.

[199] J. Halpern and C. Pignataro, “Service function chaining (sfc) architecture,” RFC Editor, RFC 7665, October 2015.

[200] J. Son, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, “Sla-aware and energy-efficient dynamic overbooking in sdn-based cloud data centers,” *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 76–89, 2017.

[201] D. Ersoz, M. S. Yousif, and C. R. Das, “Characterizing network traffic in a cluster-based, multi-tier data center,” in *Processings of 27th International Conference on Distributed Computing Systems (ICDCS’07)*. IEEE, 2007, pp. 59–59.

[202] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, “Carpo: Correlation-aware power optimization in data center networks,” in *Proceedings of 2012 IEEE INFOCOM*. IEEE, 2012, pp. 1125–1133.

[203] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder, “Understanding and abstracting total data center power,” in *Workshop on Energy-Efficient Design*, vol. 11, 2009.

[204] F. Checconi, T. Cucinotta, and M. Stein, “Real-time issues in live migration of virtual machines,” in *Proceedings of European Conference on Parallel Processing*. Springer, 2009, pp. 454–466.

[205] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[206] T. V. Doan, G. T. Nguyen, H. Salah, S. Pandi, M. Jarschel, R. Pries, and F. H. Fitzek, “Containers vs virtual machines: Choosing the right virtualization technology for mobile edge cloud,” in *Proceedings of 2019 IEEE 2nd 5G World Forum (5GWF)*. IEEE, 2019, pp. 46–52.

[207] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, “Containers checkpointing and live migration,” in *Proceedings of the Linux Symposium*, vol. 2, 2008, pp. 85–90.

[208] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, “Dynamic service migration in mobile edge-clouds,” in *Proceedings of 2015 IFIP Networking Conference (IFIP Networking)*. IEEE, 2015, pp. 1–9.

[209] L. Ma, S. Yi, and Q. Li, “Efficient service handoff across edge servers via docker container migration,” in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.

[210] L. Ma, S. Yi, N. Carter, and Q. Li, “Efficient live migration of edge services leveraging container layered storage,” *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2020–2033, 2018.

[211] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, “Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach,” *IEEE Transactions on Mobile Computing*, vol. 20, no. 3, pp. 939–951, 2021.

[212] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, “Voyager: Complete container state migration,” in *Proceedings of 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2137–2142.

[213] J. W. Moon and L. Moser, “On cliques in graphs,” *Israel Journal of Mathematics*, vol. 3, no. 1, pp. 23–28, 1965.

[214] R. E. Tarjan and A. E. Trojanowski, “Finding a maximum independent set,” *SIAM Journal on Computing*, vol. 6, no. 3, pp. 537–546, 1977.

[215] J. M. Robson, “Algorithms for maximum independent sets,” *Journal of Algorithms*, vol. 7, no. 3, pp. 425–440, 1986.

[216] ——, “Finding a maximum independent set in time o (2n/4),” Technical Report 1251-01, LaBRI, Université Bordeaux I, Tech. Rep., 2001.

[217] R. Boppana and M. M. Halldórsson, “Approximating maximum independent sets by excluding subgraphs,” *BIT Numerical Mathematics*, vol. 32, no. 2, pp. 180–196, 1992.

[218] S. Sakai, M. Togasaki, and K. Yamazaki, "A note on greedy algorithms for the maximum weighted independent set problem," *Discrete Applied Mathematics*, vol. 126, no. 2-3, pp. 313–322, 2003.

[219] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[220] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2866–2880, 2015.

[221] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, "Scipy 1.0: fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020.

[222] Y. Sun, S. Zhou, and J. Xu, "Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2637–2646, 2017.

[223] Y. Xiao and M. Krunz, "Qoe and power efficiency tradeoff for fog computing networks with fog node cooperation," in *Proceedings of IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[224] R. Stoyanov and M. J. Kollingbaum, "Efficient live migration of linux containers," in *Proceedings of International Conference on High Performance Computing*. Springer, 2018, pp. 184–193.

[225] D. Kim, T. Yu, H. H. Liu, Y. Zhu, J. Padhye, S. Raindel, C. Guo, V. Sekar, and S. Seshan, "Freeflow: Software-based virtual {RDMA} networking for containerized clouds," in *Proceedings of 16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, 2019, pp. 113–126.

[226] Y. Qiu, C.-H. Lung, S. Ajila, and P. Srivastava, "Experimental evaluation of lxc

container migration for cloudlets using multipath tcp," *Computer Networks*, vol. 164, p. 106900, 2019.

[227] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Proceedings of 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2012, pp. 644–651.