Challenges and New Solutions for Live Migration of Virtual Machines in Cloud Computing Environments

Dissertation

for the award of the degree
"Doctor rerum naturalium"
of the Georg-August-Universität Göttingen

within the doctoral Program in Computer Science (PCS)
of the Georg-August University School of Science (GAUSS)

submitted by
Fei Zhang

from Sichuan, China
Göttingen, 2018

Thesis Committee:

Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG)
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Xiaoming Fu
Institut für Informatik, Georg-August-Universität Göttingen

Members of the Examination Board/Reviewer:

Reviewer:
Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG)
Institut für Informatik, Georg-August-Universität Göttingen

Second Reviewer:
Prof. Dr. Xiaoming Fu
Institut für Informatik, Georg-August-Universität Göttingen

Additional Reviewer:
Prof. Dr. Hai Jin
School of Computer Science and Technology, Huazhong University of Science and Technology

Further members of the Examination Board:
Prof. Dr. Jens Grabowski
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Dieter Hogrefe
Institut für Informatik, Georg-August-Universität Göttingen

Jun.-Prof. Dr. Marcus Baum
Institut für Informatik, Georg-August-Universität Göttingen

Date of the oral examination: 03. May 2018

Acknowledgement

During my Ph.D. study, I got a lot of help from so many people. I would say that I could not finish my study without their help.

I will express my great thanks and gratitude to my supervisor Prof. Dr. Ramin Yahyapour for his countless advice, the inspiring discussions and his encouragements. His ample knowledge and experiences give me a deep impression. Many thanks are also given to Prof. Dr. Xiaoming Fu for his kind supervision and the interesting and informative discussions. I also owe my gratitude to Prof. Dr. Hai Jin for reviewing my thesis.

I am grateful to Dr. Philipp Wieder, Martina Brücher, Dr. Kuan Lu, Dr. Edwin Yaqub, Dr. Song Yang for the help and advice during my study. I also thank my colleagues from the eScience group of the GWDG for providing the interesting research environment, especially the guys from the cloud management section, Peter Chronz, Piotr Kasprzak and Maik Srba.

My study is impossible without the financial support from the “China Scholarship Council (CSC)". Best wishes to my country and the people. Also, many thanks are given to the kind Germans and the beautiful German sceneries. They left me many precious memories and I had a lot of fun during leisure time.

Last but not least, I owe my great thanks to my family and my parents for their endless love and encouragements which are always the motivations make me go forward.

Abstract

Live Virtual Machine (VM) migration aims to move a VM between hosts without interruption to the services running in the VM. It is a cornerstone technology for cloud management and is critical for decreasing the operating cost of cloud providers and improving service quality. Many efforts have been made to improve the performance of live VM migration and a variety of achievements have been gained. But some key problems still require solutions or improvements. In addition, with the development and evolution of cloud computing, more and more applications are running in a data center, and some new cloud computing paradigms (e.g., Mobile Edge Computing (MEC)) are being developed as well. This brings more new problems and meanwhile new optimization opportunities for live VM migration. The goal of this dissertation is to identify and qualify the performance bottleneck, and then optimizes the performance of live VM migration in different deployment scenarios. Firstly, a comprehensive review of VM migration and optimization technologies is conducted. Then, according to the lessons learned from the review, a series of optimizations are proposed to solve different problems with live VM migration.

For the migration in a Local Area Network (LAN) or within a data center, we build corresponding performance models, and design a new performance control algorithm which takes users' requirements into consideration. This algorithm also solves the convergence problem with pre-copy migration.

For the migration over a Wide Area Network (WAN) or across data centers, several mechanisms are proposed to improve the migration performance of VM storage data which is the bottleneck for across-data-center VM migration. Specifically, a three-layer VM image structure and a central base image repository are designed to increase data sharing between VMs and data centers. Based on these two structures, several optimization strategies are proposed for data deduplication and Peer-to-Peer (P2P) file sharing to further improve VM storage data migration performance.

For the migration in MEC, several algorithms are designed to mitigate the problems with user mobility-induced VM migration. According to the presented migration environments, user mobility is divided into two categories. Based on this classification, two algorithms are designed to improve migration performance, two mechanisms are designed to reduce the service degradation resulting from VM migration and two algo-

rithms are designed to lower the network overhead of constant VM migration due to user mobility.

Contents

Chapter 1

Introduction

With the support of virtualization technologies, a physical server can be divided into several isolated execution environments by deploying a layer (i.e., Virtual Machine Manager (VMM) or hypervisor) on top of hardware resources or operating system (OS). The execution environments on a server, i.e., Virtual Machines (VMs), run without mutual interruption on each other. Each VM has its own OS and applications. At the beginning, virtualization technologies were not widely used due to a variety of reasons. For example, they will occupy a portion of hardware resources (such as, Central Processing Unit (CPU) and memory) [69]. Furthermore, the poor network bandwidth also hindered vendors to lease their idle resources to clients. As the related technologies advance, such as the utilization of Fibre Channel (FC) [37], the performance improvement of an individual server [177], the development of security technology [110], etc., a new service model—cloud computing [18, 180] emerges on the foundation of virtualization technologies [27]. In cloud computing, big companies can piecemeal their spare hardware resources and rent them to customers in a pay-as-you-go manner, and users can quickly start to work on a VM without the huge expense of hardware purchase and maintenance.

Because an increasing number of users are choosing cloud data centers to hold their applications, it has become a very important task to efficiently manage the VMs in a data center. Users request and use resources from a cloud provider, and leave after their tasks are finished. Correspondingly, cloud providers will constantly create and destroy VMs in data centers. Without efficient management, a data center will operate like this: many servers are running with a small workload, while others are overloaded. Also, some cloud providers (such as, Amazon and Microsoft) run several data centers around the world. To provide high-quality services, the collaboration between data centers is inevitable. For example, it is better to put a VM in a data center which is close to the user's current location to reduce the network latency.

All the above problems can be solved by a critical technology—live VM migration. Live VM migration makes a VM not fixed on the server where it is created anymore. A

VM can be moved from one server to another, even from one data center to another data center, without causing interruptions to the applications running in the VM. Many cloud management operations become feasible with the application of live VM migration, such as server consolidation [126], zero-downtime hardware maintenance [18], energy management [12], and traffic management [77] (more details in Section 2.2).

Improving the performance of live VM migration (such as, decreasing the total migration time and reducing service interruption during migration) has been a hot topic since live VM migration was proposed because its performance is closely related to the level of cloud management and the appearance of new applications of cloud computing. After more than ten years' development, many achievements have been gained for live VM migration. However, there are still some problems waiting for solutions or further improvements. This dissertation focuses on the problems in the following three types of VM migrations: migration in Local Area Network (LAN) (i.e., migration within a data center), migration over Wide Area Network (WAN) (i.e., migration across data centers) and migration in Mobile Edge Computing (MEC).

1.1 Motivation

1.1.1 Problems with Migration in LAN

Within a data center, live VM migration is an important technology for cloud management. For example, VMs in a data center can be consolidated to occupy fewer hosts to reduce the operating cost. The VMs which frequently communicate with each other can be migrated to the same host to facilitate the communication efficiency and decrease the network traffic within a data center.

Due to the good migration environments within a data center (such as high network bandwidth, shared storage system), the performances of VM migration in LAN have almost reached the ceiling (e.g., invisible downtime and small total migration time) after a variety of optimizations. However, there still exist some open issues. 1) Currently, a migration process is uncontrollable and transparent to cloud managers and users. Once a migration is started, the only thing we can do is to wait for the message of success or failure from the migration process. The migration process cannot be tunned to meet a specific performance. 2) Too much attention has been paid to the improvement of migration performance while users' requirements are rarely considered. For example, in the scenario of Network Function Virtualization (NFV) where a VM is providing one or several functions of a service chain, the users of this VM have a very high requirement for migration downtime over total migration time. 3) Pre-copy [29] is extensively used for VM migration in LAN, but it encounters a migration convergence problem. When the memory pages of the migrated VM are dirtied faster than the network bandwidth for

migration, a migration process cannot converge to ensure good migration performances (e.g., small downtime and small total migration time).

1.1.2 Problems with Migration over WAN

Migrating VMs across data centers improves cloud management to a higher level and makes the cooperation between cloud providers become possible. For example, the cloud providers which run several data centers can carry out load balancing between data centers instead of only within one data center. A private cloud data center can live migrate some VMs which do not run confidential workloads to a public cloud data center, when facing a sudden peak workload. Therefore, private cloud providers do not need to maintain so many servers to align with the possible peak workload.

However, the relatively large size of the storage data (i.e., virtual disk) of a VM and the small network bandwidth between data centers make storage data migration become the bottleneck of live VM migration over WAN. Regarding this problem, the existing solutions can be divided into two categories: 1) A network file sharing system is deployed between the source and the destination data centers to avoid migrating storage data. This type of solution leads to a high disk I/O latency. Actually, this structure results in the double transfer of the storage data of a VM during migration, from the source data center to the shared storage system and from the shared storage system to the destination data center, which incurs a large amount of network traffic. 2) Each data center uses a local storage system and storage data migration performance is optimized by a variety of techniques, such as data deduplication or snapshotting. However, each optimization faces a trade-off between the newly-introduced overheads and the benefits to migration performances. For example, data deduplication leverages the hash fingerprints of storage data to detect the blocks which are already presented at the target site to reduce migration time. If the deduplication process cannot find enough duplicated blocks at the target site, it may prolong the total migration time since the computation overhead of data deduplication exceeds its contribution to data transfer. Therefore, a systematic rethink is strongly required on storage data migration for significant performance improvement.

1.1.3 Problems with Migration in MEC

With the popularity of User Equipment (UE) (such as, smartphones and tablets) and the evolution of the Internet of Thing (IoT), MEC was proposed to provide low-latency cloud-computing services at the edge of the network for UEs and a variety of sensors. The proximity of cloud resources to UEs reduces the coverage area of each cloud data center. Also, the majority of tasks offloaded to edge cloud data centers have a high requirement for service latency. These conditions together make MEC very sensitive to UE mobility (i.e., user mobility). When a UE is roaming between the coverage areas of

different edge cloud data centers, sometimes its VM has to be migrated along with it to meet the latency requirement.

For the migration in MEC, we focus on the following problems in this dissertation. 1) Migrating a VM in MEC faces the same environment as the migration over WAN, so storage data migration is also a big problem for it. However, MEC introduces some new environment features which can be exploited to design the optimizations for storage data migration. 2) In addition, the inconsistency between VM migration and UE mobility (more details in Section 6.2.1) will lead to a big interruption to the running service. Therefore, a coordination strategy is needed to link UE mobility with VM migration. 3) Furthermore, user mobility incurs a big network burden to MEC because VMs are constantly migrated between edge cloud data centers. It is also a vital issue about how to reduce the network overhead of constant VM migration.

1.2 Contributions

Regarding the aforementioned problems in live VM migration, several strategies and algorithms are proposed in this dissertation for the migration in different environments (LAN, WAN and MEC). The contributions of this dissertation are summarized as follows.

1. To clearly understand the state-of-the-art technologies of live VM migration, a comprehensive review of the existing solutions is conducted. The lessons learned from this study are used to guide the designs of the systems and the algorithms in this dissertation.

2. **VM migration in LAN—Performance Control:** The performance models of VM migration in LAN are established, and then a series of relationships between migration performances and the influence parameters are analyzed based on Xen platform. It is found that VM migration performances are adjustable by changing some parameters, such as memory page dirty rate. Based on these analytical results, a migration control algorithm is designed to tune a migration process to meet the desired performance. The algorithm not only considers user's performance requirements, but also solves the convergence issue of pre-copy migration.

3. **VM migration over WAN—Storage Data Migration:** To eliminate the trade-off of optimization technologies between newly-introduced overheads and achieved migration performance improvement, rethinks are conducted on VM storage data migration from a structural perspective. Two optimizations are designed to improve VM storage data migration performance. **1)**: A three-layer image structure is proposed to improve data sharing between VMs. The storage data of a VM is physically separated and stored in three layers: an Operating System (OS)

layer, a Working Environment (WE) layer and a User Data (UD) layer. Based on this structure, data deduplication is correspondingly optimized to mitigate its side effects in improving VM migration performances. **2)**: A central repository is employed to deploy and store base images (template images) for different data centers to improve data reuse between data centers. In further, two technologies (data deduplication and Peer-to-Peer (P2P) file sharing) are utilized to improve storage data migration performance.

4. **VM migration in MEC—User Mobility-induced VM Migration:** To make the best of the migration environments, UE mobility is broadly classified into two categories: certain moving trajectory and uncertain moving trajectory. Based on this classification, two optimizations are proposed to improve VM migration performance in MEC. **1)**: Two migration algorithms are designed to improve VM storage data migration performance by using the new environment features of MEC. **2)**: Several algorithms are proposed to make a migration plan for each VM to reduce the network overhead resulting from constantly migrating it along with user mobility.

1.3 Dissertation Structure

The contents of this dissertation are organized as follows:

- Chapter 1 introduces the motivations behind our study and the contributions of this dissertation regarding the targeted problems.
- Chapter 2 gives out the basic knowledge of live VM migration, including its possible applications, the differences between VM migration and container migration, performance metrics, etc.
- Chapter 3 presents a comprehensive review of the studies related to our research topics.
- Chapter 4 shows the performance models for pre-copy migration in LAN environments, and then a migration performance control algorithm is designed.
- Chapter 5 elaborates on the optimizations for VM storage data migration in WAN environments.
- Chapter 6 proposes several algorithms to improve VM migration performances in MEC environments.
- Chapter 7 summarizes the work in this dissertation and give an outlook of future research topics based on the contents of this dissertation.

Chapter 2

Background

In this chapter, we present the background knowledge of this dissertation. Firstly, some basic concepts of hardware virtualization and cloud data center are introduced. Then, the possible use cases of live VM migration, how to live migrate a VM, the new paradigm of cloud computing—MEC and a taxonomy of migration schemes are elaborated. At last, the performance metrics for evaluating a migration strategy, the differences between VM migration and container migration are presented.

2.1 Resource Management of Cloud Data Center

By using virtualization technologies, the hardware resources (CPU, memory, Network Interface Card (NIC), etc.) of a server can be abstracted and assigned to several VMs. This is implemented by deploying a layer (Hypervisor/VMM) on the host OS, as shown in Figure 2.1 [1]. Each VM is a complete computer (including OS, applications, etc.), and also has a full view of hardware resources (like running on bare-metal hardware), such as, CPU, memory, disk and NIC. The VMs co-locating on the same server are unaware of each other.

How a hypervisor/VMM manages the physical memory and disk space of the underlying server is highly related to the contents of this dissertation, so next we will give more details of them. There are three levels of memory for each host after virtualization: *host physical memory*, *guest physical memory* and *guest virtual memory*, as shown in Figure 2.2. Host physical memory is the memory of the underlying server, also called machine memory. Guest physical memory is the memory that is visible for the guest OS of each VM. Guest virtual memory is the memory presented by the guest OS for the applications running in the VM. The mapping between guest virtual memory and guest physical memory is done by the guest OS of each VM. The mapping between guest physical memory and host physical memory and the mapping between guest virtual

[1] There are several types of virtualizations. We only show one of them to elaborate on the relationship between the underlying host and VMs.

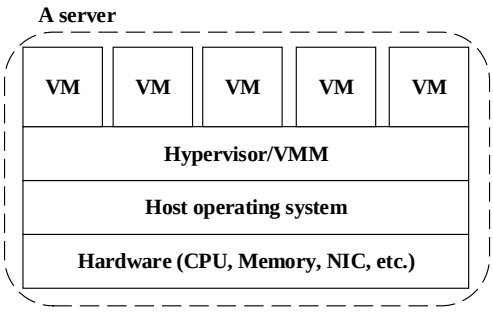


Figure 2.1: Hardware virtualization.

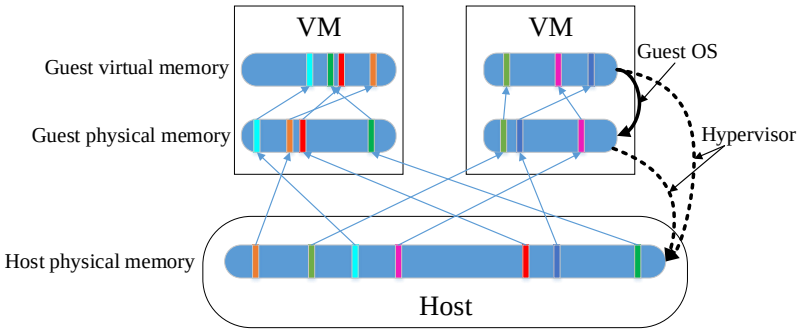
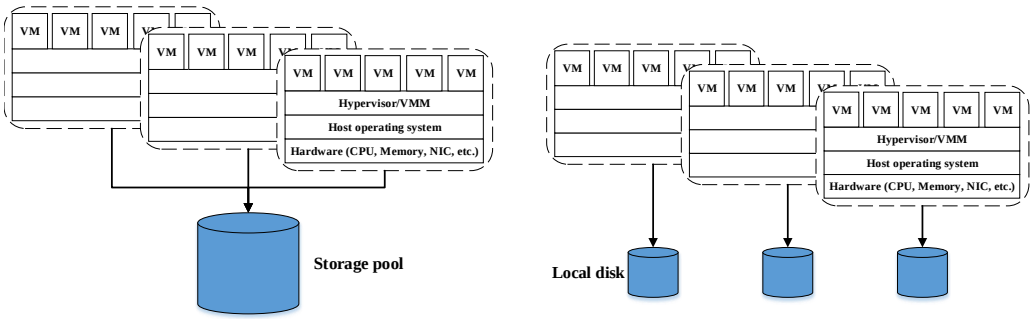


Figure 2.2: Memory management.

memory and host physical memory are managed by the hypervisor/VMM. The mapping between guest virtual memory to host physical memory is implemented by a shadow page table for each VM. It is to remove the virtualization overhead for VM's normal memory access.

The hypervisor/VMM will create a virtual disk for each VM to store all its data. A virtual disk is a large physical file or a set of files. It can be copied, moved and archived, like other normal files. Multiple virtual disks also can be configured for a VM. There are many types of virtual disk formats, such as RAW, QCOW2, VMDK, VDI, etc. Each VM uses a virtual SCSI controller to access its virtual disk(s). Because a virtual disk encapsulates the disk data of a VM into a file, it is also called disk image. There are two types of manners to store the disk images of the VMs on a host. One is to store them in a storage pool (e.g. Network File System (NFS), Network Attached Storage (NAS), Storage Area Network (SAN), etc.), as shown in Figure 2.3(a). VMs access their disk images through a high-speed network. The storage pool is shared by many servers and can be accessed by multiple servers simultaneously. The other one is to store them in the local disk of the underlying host, as shown in Figure 2.3(b).



(a) Storage pool (b) Local disk

Figure 2.3: The storage system structures of cloud data center.

2.2 The Cornerstone of Cloud Management

Based on the introduction in the last section, let's move to the data center scale. There are many servers in a data center, and each server is hosting several VMs. Users apply for resources from a data center and leave after finishing their tasks. The data center will correspondingly create and destroy VMs for users. It will be a mess without a proper management strategy for the VMs in a data center. Also, virtualization is only resource isolation rather than performance isolation, which means that the VMs on a server will mutually influence on performance. Hence, if all the VMs on a server are running resource-eager workloads, all of them will experience performance degradation, even may kill the server. Under this situation, some VMs should be moved to other servers which are with low loads. These problems are a part of motivations for live VM migration which allows a VM to be moved between servers without visible interruptions to the services running in the VM. Many cloud management tasks are carried out based on live VM migration. In this section, we list out some use cases of live VM migration for intra- and inter-cloud managements.

- **Zero-downtime hardware maintenance** [173, 189]. The servers in a data center may have a high possibility of failure after a long-time running or already failed several times. These servers can be replaced with new ones by moving out all the VMs located on them and moving back after replacement. This is applicable for hardware upgrade as well.

- **Load balancing** [65, 25]. Overloaded state not only shortens the lifespan of a server, but also degrades the Quality of Service (QoS). Meanwhile, servers running with underloaded state result in a waste of energy. Live VM migration ensures that all servers in a data center run evenly under the premise of without QoS decrease. Load balancing also can be conducted between several geo-distributed data centers when live VM migration over WAN is enabled.

- **Server consolidation** [198, 67]. VMs are persistently created and destroyed in a data center. In addition, some of them may be suspended or idle. The VMs will be in a mess if the servers in a data center are not properly consolidated. In server consolidation, VMs are live migrated for either energy purpose (using as fewer servers as possible) or communication purpose (locating the VMs communicating heavily with each other on the same server to reduce network traffic).

- **Across-site management** [183]. For the cloud providers with multiple sites, there are more management options to improve the QoS and lower economical cost. For example, *Follow the sun* [173, 183] is a new IT operation strategy which moves compute resources close to the clients to minimize the network latency. Conversely, in order to reduce the cooling cost, Google and Microsoft propose the concept of *free cooling*, also called *follow the moon* [169, 161], which moves VMs and applications to the data center where it is in nighttime or with low temperature.

- **Hybrid cloud** [155, 103]. Hybrid cloud is featured with a couple of advantages [130]: (1) highly cost-effective and elastic; (2) better aligned to the business demands; (3) keeping private services in local. In the hybrid cloud, users can offload some tasks to a public cloud when encountering peak workload, i.e. *cloud bursting* [115].

- **Cloud federation** [23, 21]. The inherent deficiencies of an individual data center are driving cloud providers to cooperate with each other [20], i.e., *cloud federation* [139]. There is still a long way to go for cloud federation in industry, but it is already making benefits in academia. For example, EGI Federated Cloud [43] connects more than 22 computing sites across Europe and the world to provide services to academic researchers. With cloud federation, when the volume of the processed data is huge, we can choose to move computation environment to where data are located to avoid transferring a big amount of data.

- **Breaking vendor lock-in issue** [146, 123]. The worry of being locked in by one vendor is the main obstacle preventing users from choosing cloud computing. The investigation shows that 62% IT organizations regard moving VMs between data centers as an invaluable feature when choosing a cloud vendor [115].

- **Reaction to user mobility**. To avoid violating the Service Level Agreement (SLA), sometimes cloud providers have to migrate VMs to a cloud data center which is close to users. This is also strongly required in the mobile edge computing alike scenarios where each edge cloud data center only covers a small region of users. When a user moves out from the coverage area of an edge cloud data center, the corresponding VMs have to be migrated to the edge cloud data center where the user is currently at.

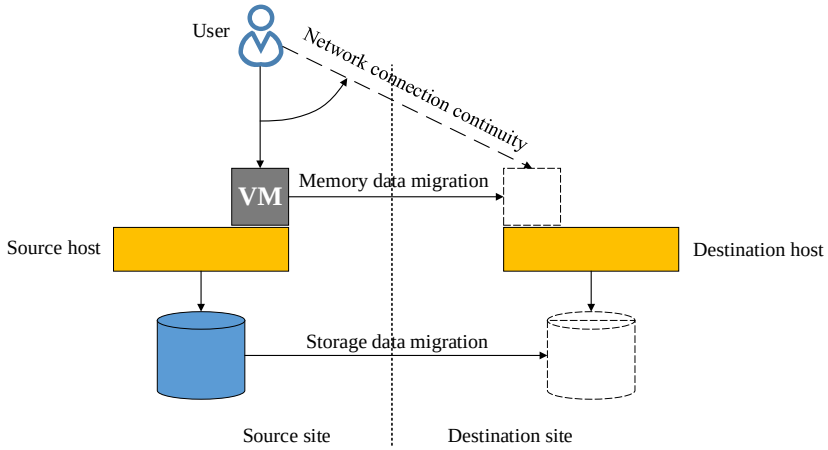


Figure 2.4: Live VM migration.

2.3 How to Live Migrate a VM?

From the perspective of migration, a VM can be divided into three parts: running states (including memory data, CPU states, all external device states, etc.), storage data (i.e., disk data) and the network connections between the VM and its users. Live VM migration is to hand over the three parts from the source site to the destination site. Therefore, it consists of three following tasks, as shown in Figure 2.4.

1. **Memory data migration.** To avoid interrupting the services running in the migrated VM, all real-time states of a VM must be migrated to the new host. These data contain CPU states, memory data, the buffer data of external devices, etc. Generally, the transfer of the running states is called as *memory data migration*.

2. **Storage data migration.** It is to migrate the disk image of a VM to the new location. This task is needed when the source host and the destination host are not sharing a storage pool.

3. **Network connection continuity.** After a VM is moved to a new location, a strategy is required to redirect the network connections of its users to the new location.

In this dissertation, we focus on improving the performance of memory data migration and storage data migration. We will give more details on these two tasks in Section 2.3.1 and 2.3.2, respectively.

2.3.1 Memory Data Migration

According to the VM handover time of a migration process, memory data can be migrated in three patterns: *pre-copy*, *post-copy*, and *hybrid-copy*. Pre-copy [29, 119] firstly copies all memory data to the destination server. Because the VM is still running

on the source host, some memory pages will be dirtied during this period of data transmission. These dirtied pages will be logged and transferred in an iterative way, i.e., the pages transferred in current round are the ones dirtied in the previous round. When the iteration meets the predefined thresholds (called *termination conditions*), the VM is shut down on the source server. The remaining data are copied to resume the VM on the destination server. For example, the termination conditions for stopping the iteration phase can be set as: (1) the transfer round reaches a predefined value; (2) the remaining data size is smaller than a predefined value; (3) the ratio between the size of transferred data and the allocated size of memory space is bigger than a predefined value, etc. Because VMs running different workloads have different memory dirtying features, no termination condition is suitable for all VMs. Therefore, several termination conditions are often combined to design a comprehensive strategy to adapt to as many VMs as possible. Pre-copy is widely used in the mainstream VMMs/hypervisors, such as VMware [119], Xen [29], and KVM [87], because of its robustness. During migration, at least one site (the source site) has all of the data of the migrated VM. When the migration breaks up at halfway due to some reasons (such as network outage) or the VM fails to resume on the destination server, the VM can continue to run on the source server without data loss. The source server releases the data of the migrated VM until it is successfully resumed on the destination server. However, it faces one main problem. When the memory pages of the migrated VM are dirtied (called *memory dirty rate*) faster than the available network bandwidth for VM migration, the migration process cannot be convergent. It means that the iterative copying makes no sense to reduce the remaining data on the source server. This issue is called *migration convergence problem*, which will result in a big migration downtime or a big network traffic.

Contrary to pre-copy, post-copy [59, 62] firstly stops the execution of a VM on the source host. Boot data are scratched and copied to the destination host to resume the VM. The rest of memory data can be transferred by different manners, such as on-demand fetching, active pushing, and prepaging [59]. All memory pages are copied only once in post-copy, so the total migration time is predictable. However, post-copy has a fatal issue. The latest data of the migrated VM are separated to the source host and the destination host, therefore, it may lose data or even destroy the migrated VM when the migration fails at halfway.

Hybrid-copy [66, 86] is a combination of pre-copy and post-copy. It firstly copies memory data with a limited round of iterations, and then hands over the execution of the VM to the destination server. The remaining memory pages are copied in a post-copy manner. Hybrid-copy inherits the strong points from both pre-copy and post-copy. The limited pre-copying iteration reduces network traffic. The majority of memory pages are transferred in the pre-copy phase, which decreases the pages remotely accessed by the destination host, i.e. minimizes the possibility of page faults at the destination site.

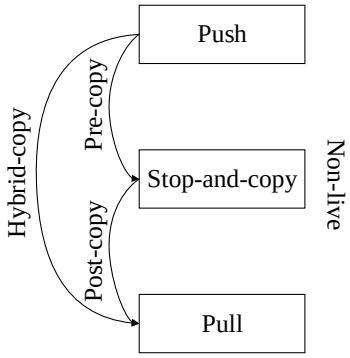


Figure 2.5: The classification of memory data migration patterns.

This is helpful for lowering service degradation. However, it also succeeds the main disadvantage of post-copy, i.e. weak robustness.

For better understanding different memory data migration patterns, Clark et al. [29] divide memory data migration into three basic phases:

- **Push phase:** The migrated VM runs on the source server. Its memory data are iteratively transferred to the destination server.
- **Stop-and-Copy phase:** The execution of the VM is halted on the source host. All or a part of the remaining data on the source host are copied to the destination site to resume the VM.
- **Pull phase:** When the VM is running on the destination server and page faults happen, it remotely fetches these pages from the source server. At the same time, the remaining pages are proactively sent to the destination host to lower the possibility of page fault.

Different migration patterns are achieved by taking one or several phases of this division, as shown in Figure 2.5. Pre-copy takes the push and stop-and-copy phases, while post-copy takes the stop-and-copy and pull phases. But they transfer different data during the stop-and-copy phase. Pre-copy transfers all the remaining data at the source host, while post-copy only transfers the data needed for VM resumption. Hybrid-copy experiences all of these phases. If only the stop-and-copy phase is taken, it is non-live migration.

2.3.2 Storage Data Migration

When both memory data and storage data of a VM will be migrated, according to the migration sequence between memory data and storage data, storage data migration can be also classified into three patterns: *pre-copy*, *post-copy*, and *hybrid-copy*. Pre-copy migrates storage data before memory data, while post-copy transfers storage data after memory data. Hybrid-copy migrates storage data and memory data simultaneously. By

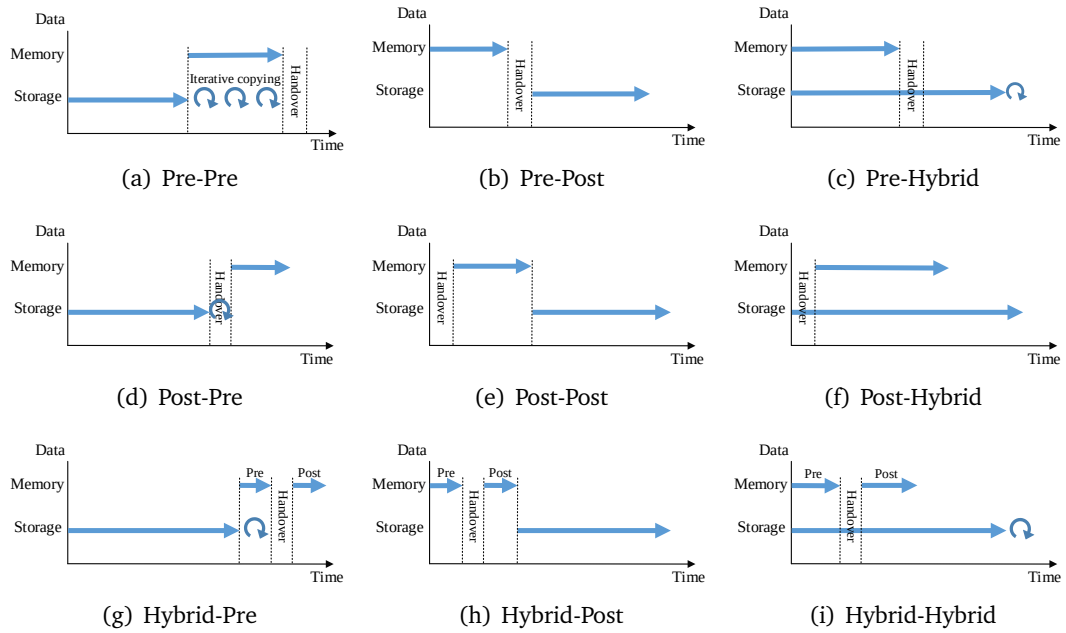


Figure 2.6: The migration sequences between memory data and storage data. The retransmission of dirtied disk blocks can be implemented in different manners. For example, the dirtied blocks can be synchronized to the destination site during migration or transferred at the end of storage data migration in bulk. In these figures ((a), (c), (d), (g), (i)), only the second option is shown.

combining different memory data and storage data migration patterns, nine migration patterns are available when both the memory and storage data of a VM will be migrated: *Pre-Pre*, *Pre-Post*, *Pre-Hybrid*, *Post-Pre*, *Post-Post*, *Post-Hybrid*, *Hybrid-Pre*, *Hybrid-Post* and *Hybrid-Hybrid*, as shown in Figure 2.6.

The two patterns in each name denote memory data migration pattern and storage data migration pattern, respectively. For example, *Pre-Hybrid* migrates memory data in a pre-copy manner and storage data in a hybrid-copy manner. In other words, memory and storage data are concurrently transferred, namely hybrid-copy, and memory data are migrated with the pre-copy pattern. If the VM is running on the source host during storage data migration, two additional mechanisms are required: (1) the dirtied blocks will be logged and retransferred to the destination host for data consistency, such as Pre-Pre and Pre-Hybrid; (2) a strategy is needed to coordinate the write operations from the migrated VM and the read operations from the migration process.

As discussed in Section 2.3.1, post-copy and hybrid-copy have a weak robustness for memory data migration, which is also applicable for storage data migration patterns. Therefore, from migration pattern names, the pattern containing *Post* or *Hybrid* is weak regarding robustness. They may lose data or destroy the migrated VM if the migration fails at halfway. Only the *Pre-Pre* pattern can ensure the migrated VM correct under different situations, and does not need manual intervention even migration outage happens. Therefore, it is widely used for VM migration over WAN.

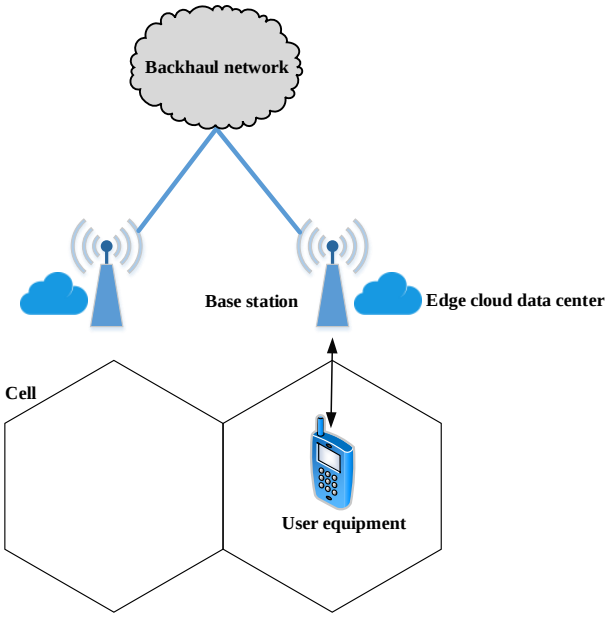


Figure 2.7: The architecture of mobile edge computing.

2.4 Mobile Edge Computing

Mobile devices, such as smart phones and tablets, are becoming more and more important for our daily lives, both work and entertainment. However, the physical device size constraint limits the computation and energy capacities of a UE. Also, with the development of the IoT, a huge amount of sensors are constantly generating data, and some of these data need to be fast processed to response the potential events or accidents, such as smart city, smart home and intelligent transport system. The long distance between sensors and a central remote cloud data center makes it impossible to meet the low-latency requirement.

To provide a high quality of services to mobile devices, some new computing paradigms were proposed, such as fog computing [15], MEC [45], cloudlet [144], etc. All of them have the same structure, i.e., cloud resources (such as compute and storage) are deployed at the network edge to provide low-latency services for UE. MEC is used to denote all these paradigms in this dissertation. As shown in Figure 2.7, MEC moves cloud-computing services to the vicinity of UE, called edge cloud data centers. Edge cloud data centers can be deployed by utilizing the spare hardware resources of the base stations of the Radio Access Network (RAN). UE partially or fully offloads computation tasks to these edge cloud data centers for the purposes of energy saving or computation capacity expansion, i.e., computation offloading [105, 78].

Some basic concepts of MEC are described as follows.

- *Base station*: where transceivers of the RAN are located.

- *Cell*: the signal coverage(service) area of a base station.
- *Edge host*: the servers deployed in base stations. VMs are running on them, and the computation tasks offloaded from UE are executed in these VMs.
- *Edge cloud data center*: the cloud computing platform deployed in each base station.
- *User Equipment(UE)*: a mobile device which offloads computation tasks to an edge cloud data center.
- *Mobile application*: the application which is running on a UE and its computation tasks are partially or fully offloaded to an edge cloud data center.

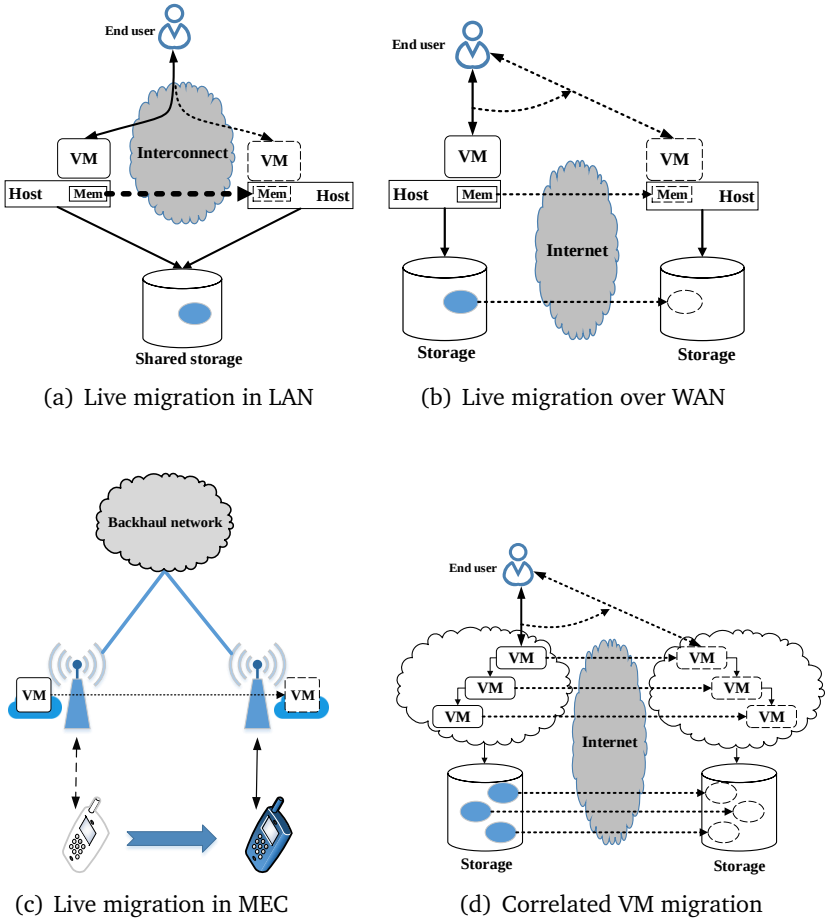
Live VM migration in MEC only involves two concepts: cell and edge cloud data center. Users and UE are moving between cells, while the corresponding VMs are live migrated between edge cloud data centers.

2.5 A Taxonomy of Migration Schemes

The migrations with different conditions confront different challenges. We classify migration schemes from three perspectives: migration manner, migration distance, and migration granularity.

VM migration can be conducted in two manners: *non-live migration* and *live migration*. Live migration is carried out under the precondition of no interruption to the running services, while non-live migration does not have this limitation. With non-live migration, a VM is firstly suspended or shut down before migration depending on whether it will continue the running services after migration or not. If the VM is suspended, the running states will be encapsulated and transferred to the target site. During the migration, no open network connection is kept and all connections are rebuilt after VM resumption. However, memory data migration and network connection continuity are the two problems which must be solved for live migration to avoid service interruption. If the source and the destination sites do not share storage system, storage data migration must be carried out as well. Non-live migration has a significant interruption to the service running in the migrated VM. This dramatically restricts its application field since many applications in a cloud data center are running in a 7×24hours manner. Hence, the majority of researches are focusing on live migration.

According to migration distance, VM migration is divided into two categories: *migration in LAN* and *migration over WAN*. Migrating a VM in LAN means the source and the destination servers are located in the same data center. With the development of network technologies, the difference and boundary between Metropolitan Area Network (MAN) and WAN disappear [89]. *Migration over WAN* in this dissertation refers



(a) Live migration in LAN (b) Live migration over WAN

(c) Live migration in MEC (d) Correlated VM migration

Figure 2.8: Classification of migration schemes. The relatively bigger line width for memory data migration in (a) is to indicate the bigger network bandwidth in LAN environments in comparison with WAN and MEC environments.

to any migration across data centers. The migration mechanisms for LAN environments normally has two basic assumptions. (1) Shared storage system, such as SAN and NAS, is used in a data center. It is accessible from both servers in the migration, which indicates that storage data migration is unnecessary. (2) The source and the destination servers are in the same subnet. The migrated VM will keep its network configurations during the whole migration. Based on these two premises, migrating a VM in LAN only needs to solve the task of memory data migration, as shown in Figure 2.8(a). However, migrating a VM in WAN environments does not have these advantages. There is no shared storage system, and different data centers have different network configurations as well. Furthermore, the network conditions (such as bandwidth, latency) between data centers are much worse than those of LAN, as shown in Figure 2.8(b). Therefore, migrating a VM over WAN not only needs to solve all of the three tasks of live VM migration, they also become much harder in comparison with the migration in LAN environments.

VM migration in MEC not only faces the same problems as the migration over WAN, the proximity of cloud resources to users in MEC also brings both new challenges and optimization opportunities, as shown in Figure 2.8(c). For example, an edge cloud data center in MEC only serves the users in its coverage area. When a user roams between different edge cloud data centers, the corresponding VM has to be migrated to meet the low-latency requirement of mobile applications. This type of VM migration is called *user mobility-induced migration* (more details in Chapter 6).

Nowadays, many applications in a data center consist of a group of VMs [50, 179]. These VMs are closely related to each other and work together to provide a service. For example, three-tier is a typical application architecture. It is composed of a presentation tier, an application tier and a database tier. The number of VMs in each tier can be scaled up or down according to the change of workloads. It is impossible to only migrate one of the correlated VMs to another data center because the long network latency between data centers will severely degrade service performance, even destroy the service, as shown in Figure 2.8(d). Therefore, according to migration granularity, VM migration contains *single migration* and *correlated VM migration*. Single migration migrates one VM each time and this VM is running independently, while correlated VM migration simultaneously moves a bunch of VMs which are communicating with each other.

2.6 Performance Metrics and Overheads

A good migration strategy not only tries to move a VM from one place to another as fast as possible, but also needs to minimize its side-effects. The performance metrics for assessing a migration strategy are summarized as follows.

- **Total migration time**. This refers to the duration between the time when the migration is initiated and the time when the migrated VM is resumed on the destination server and no data remains at the source site.

- **Downtime**. It is the duration that the migrated VM is out of service. This metric determines how transparent the migration is to the users of the migrated VM. For non-live migration, total migration time equals to downtime.

- **Total network traffic**. This metric means the total data transferred during the migration. It is a critical measurement when the migrated VM is running a network-intensive service because it will contend for network bandwidth with the migration process.

- **Service degradation**. It indicates how the service running in the migrated VM is affected by the migration. It can be measured by the changes of throughput, response time, etc.

Sometimes network bandwidth utilization is also used to evaluate a migration strategy. This metric can be gained by combining total migration time with total network traffic. The smaller the total migration time and the less the total network traffic are for a specific migration, the higher the network utilization is.

However, VM migration is not an only-gain-no-pain operation. It may bring interferences to all involved roles in the migration. The side effects of live VM migration can be divided into three categories: computation overhead, network overhead and space overhead.

Computation overhead. Normally, the migration daemon is running in the VMMs of the source and destination hosts (i.e., *managed-migration*). The migration process will occupy a portion of CPU cycles and memory spaces. This will lead to interferences to all VMs on these two hosts. If the migration daemon is running in the migrated VM (i.e., *self-migration*) [55, 29], some computation resources of the migrated VM will be occupied. Besides, some migration optimization technologies also introduce computation overheads, such as data deduplication, data compression.

Network overhead. VM migration is a network-intensive workload. It will compete for network resources with the VMs running on the source and destination hosts. In addition, a migration process reads data from the storage system of the source site and writes them to that of the destination site, which consumes a portion of I/O bandwidth as well.

Space overhead. Compared to other resources (such as CPU cycles, network bandwidth), storage space is less valuable. Inevitably, some optimization technologies will implement migration or improve migration performances at the expense of storage space, such as snapshotting [108].

2.7 VM Migration vs. Container Migration

Container is an unavoidable topic whenever VM is involved due to many common points between them. Meanwhile, there are many differences between them which make them coexist in the "virtualization world" [39, 148]. In this section, they will be differentiated from the migration perspective.

Containers are implemented by OS virtualization, while VMs are implemented by hardware virtualization. The containers on a host are sharing the underlying OS kernel, but VMs are complete and totally isolated execution environments (each VM installed with an OS). This difference makes container migration more close to process migration. Actually, the commonly used migration technology for containers is checkpoint and restart (CR) [113] which saves the memory state of a process into files and resume the

process at the destination host from the checkpoint. A project—CRIU [30], based on CR, has been implemented for container migration.

A container is much more lightweight than a VM, which inherently leads to a smaller challenge for migrating a container than a VM. For example, for the containers which are running stateless services (e.g., RESTful web services), the container can be directly killed on the source host and created again on the destination host. The duration of this operation is tolerable and only the currently running requests will be affected.

Container migration will consider some problems which do not bother VM migration. For example, containers are not only sharing the underlying OS, but also some libraries. Therefore, during container migration, the destination host must prepare these libraries for the migrated container. However, the hosts at the destination site are also running other containers. Therefore, the destination host selection should be an important issue for container migration. In contrast, a VM can run on any host once they are virtualized and managed by the same type of VMM.

2.8 Chapter Summary

In this chapter, the basic knowledge of live VM migration is introduced. Firstly, the resource management of cloud data center and live VM migration are elaborated. Then, the structure of MEC and a taxonomy of different migration schemes are described. Next, the metrics for evaluating a migration strategy and the side effects of VM migration are discussed. At last, we present the differences between VM migration and container migration.

Chapter 3

Related Work

According to the classification in Section 2.5, we show VM migration schemes and the tasks of each type of migration in Figure 3.1. We summarize the existing migration technologies related to the research topics of this dissertation with the following structure in this chapter.

1. Memory data migration is the main task for live VM migration. Firstly, we review the technologies and optimizations for memory data migration. Memory data migration technologies for LAN environments are also suitable for WAN environments, only facing a slower migration bandwidth, and vice versa. Therefore, we review all memory data migration technologies together (Section 3.1).

2. Storage data migration is the bottleneck of the migration in WAN environments, and attracts many attentions from researchers. Like the technologies for memory data migration, storage data migration technologies for WAN environments are also applicable for LAN environments. We summarize all of them in Section 3.2.

3. In the new paradigms of cloud computing (e.g., MEC), VM migration is highly related to user mobility. This type of VM migration will not only solve the same challenges as WAN migration, but also faces some new issues, so we review the state-of-the-art technologies for this topic in Section 3.3.

3.1 Memory Data Migration

3.1.1 Pre-copy

Clark et al. [29] implement the pre-copy mechanism on Xen in two manners: *managed migration* and *self migration*. Managed migration runs the migration daemon in the management VM (*Dom0*), while self migration is conducted by the migrated VM itself. Self migration is more complex than managed migration in terms of implementation. Because the migration process is running with other processes on the migrated

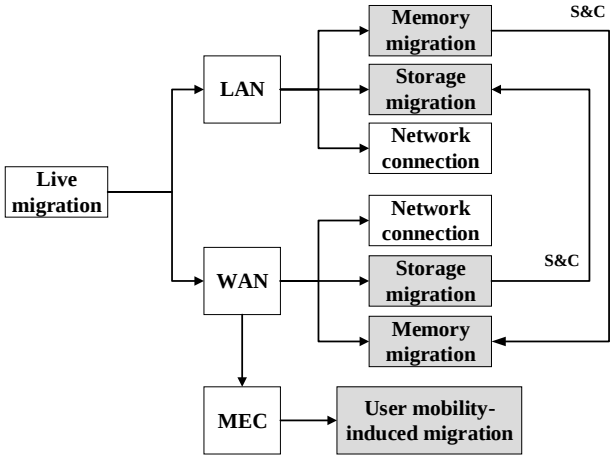


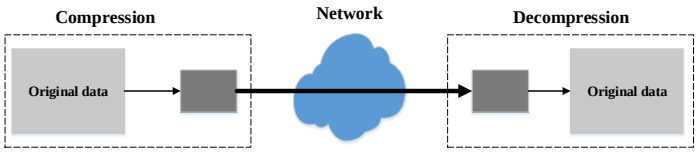
Figure 3.1: The taxonomy of VM migration. The arrows for memory and storage data migration technologies mean that they are mutually compatible in LAN and WAN environments. For both memory and storage data migrations, the technologies for single migration and correlated VM migration are covered. *S* and *C* refer to single migration and correlated VM migration, respectively. In this chapter, we only review the technologies related to our research topics, i.e., the gray boxes in the figure.

VM in self migration, it must solve the migration consistency issue which is out of consideration for managed migration. They design a two-stage stop-and-copy phase to guarantee migration consistency. The first stage stops all processes except the migration process, and scans the dirtied pages. The second stage transfers all dirtied pages in the final scan. Due to the implementation complexity and intrusive deployment for each VM, self migration is rarely used for cloud management.

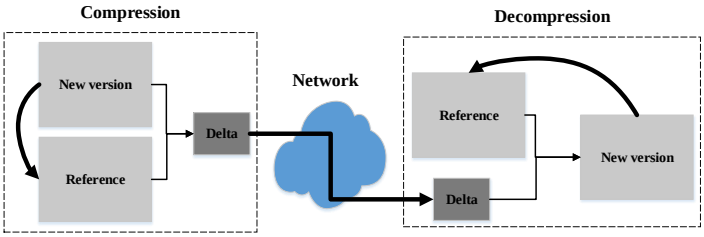
Pre-copy is also used by VMware to carry out its migration system—*VMotion* [119], and integrates it into the VirtualCenter management platform. It sets termination conditions for the iterative copying as: (1) less than 16MB modified pages left; (2) 1MB size decrease of modified pages between two neighbor rounds. They find that scanning dirtied memory pages during the iterative copying phase takes 20% network throughput drops to end users. Their results also show that reserving 30% CPU resource for migrating an idle 512MB Windows 2000 Server VM over a gigabit link minimizes the total migration time.

Compression

Memory pages have strong regularities and also contain many zero bytes [181, 44]. To reduce the total network traffic during migration, compression technology is the first one comes to mind. As shown in Figure 3.2, there are two types of compression manners: normal compression and delta compression. Normal compression takes advantage of data regularities to encode information with fewer bits. The ratio between the size of the representation information and the original data is called compression ratio. Normal compression contains two interleaved phases: modeling and encoding [118].



(a) Normal compression



(b) Delta compression

Figure 3.2: The illustration of normal compression and delta compression.

The modeling phase is to find data regularities, and the encoding phase constructs the representation information. Delta compression reduces data transfer by only sending the difference between current version and its previous version. The difference is called *delta*, while the previous version is called *reference*. In decompression phase, the original data are gained by applying a delta to its reference. After each compression and decompression, the reference is replaced with the new version of data for the next round of delta compression. Delta compression needs space to store the references and introduces additional management efforts for it. Although normal compression does not encounter these disadvantages, it is compute-intensive.

However, compression has to make a trade-off between computational cost and migration performance benefits. A compression algorithm with a higher compression ratio will lead to bigger computational cost and compression time. To this end, Jin et al. [79, 80] propose a Characteristic-Based Compression (CBC) algorithm which adaptively chooses compression algorithms according to data regularities. The memory pages with high similarity are compressed by an algorithm with fast compression speed, such as WKdm [181], while the pages with low similarity are compressed by an algorithm with high compression ratio, such as LZO [122]. The threshold between low similarity and high similarity is also adjustable to adapt to a variety of VMs with different memory dirty rates. Multi-threaded techniques are employed to accelerate the speed of compression and decompression as well.

Hacking and Hudzia [54] propose to use delta compression to shorten the migration duration of a VM running large enterprise applications which are featured by large memory size (can reach tens of GB) and fast memory dirty rate. They introduce a warmup phase before migration to transfer the original memory data to the destination server in the background. During migration, delta compression is conducted to reduce

the network traffic of migration. VMM keeps the previous version of recently dirtied pages in an Adaptive Replacement Cache (ARC) [111]. Before transferring a dirty page, if its previous version is stored in the ARC, only the difference (*delta*) between them is transferred to the target server, and at the same time the ARC is updated to the current version. Otherwise, the dirty page is directly sent to the target server.

Based on the work in [54], Svärd et al. [159] use a two-way associative caching scheme [57] instead of ARC to store the reference pages. This caching scheme is lightweight (small CPU occupation) and has a constant lookup time. XORed Binary Run Length Encoding (XBRLE) compression algorithm [128] is used to compress pages. Their experimental results indicate that their system outperforms the default KVM migration algorithm regarding both total migration time and downtime.

Data Deduplication

Many previous works [48, 197, 51] show that a big amount of identical memory pages exist within a VM or between VMs. These duplicate pages can be eliminated to speedup VM migration. They are partly zero pages and partly result from using the same libraries or applications. There are three types of similarities existing in VM memory pages: *intra-VM similarity*, *inter-VM similarity*, *inter-site similarity*. Intra-VM similarity denotes the duplicate pages within the migrated VM. Inter-VM similarity refers to the identical pages between different VMs at the same data center. This similarity can be used to transfer identical pages only once when multiple VMs are migrated concurrently. Inter-site similarity explores the identical pages between the migrated VM and the VMs located at the destination data center. SHA-1 [41] and SuperFastHash [64] are the two commonly used hashing algorithms to locate duplicate pages. In this section, we review the studies on exploiting intra-VM similarity and inter-site similarity for VM migration, and these on inter-VM similarity are described in Section 3.1.1 on correlated VM migration.

Riteau et al. [137, 138] design a migration system—*Shrinker*, to improve the performance of migrating memory data over WAN. It utilizes distributed content-based addressing to avoid transferring duplicate pages between the migrated VM and the VMs running at the destination site (i.e. inter-site similarity). However, VM memory pages change over time, so a dynamic indexing approach is needed. Shrinker solves this problem with two subsystems: (1) a site-wide Distributed Hash Table (DHT) and (2) a periodic memory indexer. The intra-VM similarity feature is also exploited by them: only the first byte of zero pages is sent to the destination site.

Nowadays, cloud providers pre-deploy template VM images for fast VM creation. Zhang et al. [199] find that many redundant memory blocks are located between the VMs which are cloned from the same VM template by an extensive experiment. To utilize this feature and decrease the footprint size of the VMs on the same host, Content Based

Page Share (CBPS) [51] is widely used in virtualization platforms (such as VMware ESX [175], Xen [112, 88]) to make the VMs on the same physical server share memory pages [182]. Based on these observations, Zhang et al. design a metadata based migration system—*Mvmotion*, which makes the migrated VMs share some redundant memory pages with the VMs running on the destination host (i.e. inter-site similarity) by utilizing CBPS technology. The metadata of a VM contain the hashing values and the block numbers of memory pages. During migration, the metadata of the migrated VM is sent to the destination data center to find the pages already existing at the destination host.

Jo et al. [83] accelerate VM migration by utilizing shared disk blocks rather than memory pages. They observe that many memory pages of a VM are replicas of disk blocks [127]. They propose to only transfer unique memory pages from the source server to the destination server. The information of the memory pages which are the replicas of disk blocks is logged into a list. The destination server gets these memory pages from the shared storage system instead of from the source server.

Li et al. [94] and Zheng et al. [204] propose a template-based migration mechanism. If a page appears n times in a data center which is bigger than the preset threshold, it is called a *template page*. Similar with [137, 138], the fingerprints of the destination data center's template pages are stored in a Distributed Hash Table (DHT). They classify memory pages into three categories: *uniform pages*, *normal pages*, and *duplicate pages*. In their migration system, uniform and normal pages are transferred by using the default migration interface of VMM. Duplicate pages are constructed at the destination data center by copying its identical template pages.

Zhang et al. [197] observe that at least 30% of non-zero memory pages are identical or similar, and design a new migration strategy—*Migration with Data Deduplication (MDD)* which takes advantage of intra-VM similarity. Both data deduplication and delta compression are utilized by Wood et al. [183] to accelerate memory data and storage data migration over WAN. Data deduplication is used to explore the duplicate items for memory pages and disk blocks, and delta compression aims to reduce the transferred bites when a page has been copied before.

Many "free" pages (such as zero pages, cache pages) exist in a VM. These pages will not influence on the correctness after the migrated VM is handed over to the destination server. Clark et al. [29] do not transfer these pages during the first full-transfer round by using memory ballooning mechanism [7]. This mechanism is also combined with QuickAssist Technology (QAT) data compression [131] by Zhang et al. [195] to accelerate VM migration in the NFV scenario. Akane Koto et al. [90] run a process in the migrated VM to record the pages which are unnecessary for VM correctness after migration. These pages will not be transferred during migration and are reproduced after VM resumption. However, even though these pages are not important for the

correctness of VM execution, losing and reconstructing these pages will result in a big service degradation after migration.

RDMA

Many high-speed interconnect technologies, such as InfiniBand [5], Myrinet [13], provide the functionality of Remote Direct Memory Access (RDMA). RDMA allows memory data to be remotely accessed without the involvement of CPU and cache. Huang et al. [68] take advantage of this feature of InfiniBand to minimize the side effects of VM migration and improve migration performance.

Ibrahim et al. [70] also utilize InfiniBand RDMA to migrate the VMs with High Performance Computing (HPC) applications. They comprehensively investigate the performance relationships between VM migration and the applications running in the migrated VM. A series of findings are observed in their experimental results. (1) The monitoring mechanism of migration process introduces a considerable interruption to the workloads running in the migrated VM. The more cores the VM runs with, the bigger the interruption is. They observe that parallelism of the monitoring process is beneficial to migration performance. (2) The memory pages of HPC applications are easily dirtied faster than the available migration bandwidth. Hence, normal migration termination conditions (predefined target downtime and iteration limit) will result in a sub-optimal migration performance. (3) The Writable Working Set (WWS) (the set of frequently dirtied memory pages) varies significantly when a VM is running different workloads. They further evaluate the performance of VM migration when a dedicated network path (InfiniBand RDMA) is employed for migration. They find that: (1) Migration downtime depends on the start time of the migration over the application lifetime. A bigger application dataset or more processors used by the migrated VM leads to a longer downtime. (2) Even though a dedicated migration path is provided, the migration still severely impacts on the quality of service. (3) correlated VM migration will experience a longer downtime than single migration. Based on these observations, they propose an optimized migration termination strategy for HPC VMs (see Section 3.1.1).

Checkpointing/Recovery and Trace/Replay

Normal pre-copy scheme is sensitive to memory dirty rate and also results in a big network traffic. Considering this issue, Liu et al. [98, 97] design and implement a novel migration system—*CR/TR-Motion*, by utilizing Checkpointing/Recovery and Trace/Replay (CR/TR) technology. It is based on a full system trace and replay system—*Revirt* [40]. They record the execution trace of the migrated VM into log files, and iteratively transfer log files rather than dirty pages to the destination server where log files are replayed to recover VM states. This can improve migration performance due to the fact that log file size (growth rate around 0.04GB to 1.2GB per day) is much

smaller than the size of dirty memory pages. Furthermore, migration downtime is also decreased because fewer data is left in the final stop-and-copy phase.

Cully et al. [31] also utilize checkpointing to migrate VM memory data to another host by copying the whole system state rather than only replaying deterministic inputs. They repeat the final stage (the stop-and-copy phase) of pre-copy to transfer the latest memory states to the destination host. To increase the checkpointing speed, two optimizations are conducted on the default Xen live migration mechanism: reducing the number of inter-process requests required to suspend and resume the guest domain and entirely removing *xenstore* from the suspend/resume process. With these optimizations, the checkpoint frequency can reach 40 times per second.

Page Reordering

The memory pages of a VM have different access characteristics. Some pages remain clear during the whole lifetime of the VM, while some are frequently modified. This characteristic can be used to improve migration performance by reordering the pages for transfer in each iteration. Svärd et al. [160] design a mechanism—*dynamic page transfer reordering*, to lower page retransmission during the iterative copying phase. They assign a weight for each page according to its update frequency. Pages are transferred in the order of increasing weight. The most frequently updated pages are postponed to the final stop-and-copy phase. Similarly, Checconi et al. [26] propose two page-reordering mechanisms: a Least Recently Used (LRU) based approach and a frequency-based approach. LRU-based approach prioritizes the transfer of the pages which are least recently used, while frequency-based approach transfers pages in the order of increasing access frequency.

Migration Convergence

The main risk for pre-copy is the migration process cannot converge to an optimal point for the final stop-and-copy phase. This situation happens when the VM dirties its memory pages faster than the migration bandwidth. A plenty of optimization strategies are designed to migrate the VM which has a fast memory dirty rate. They solve the migration convergence problem from two aspects: *tuning memory dirty rate* and *changing migration termination conditions*.

Clark et al. [29] find that some memory pages are dirtied very frequently, i.e., WWS. It is unnecessary to iteratively copy WWS to the target site during migration. Therefore, during the iterative copying phase, the dirtied pages transferred in each iteration are selected like this: only those dirtied in the previous round and not dirtied in the current round again. The paravirtualization feature of Xen also provides some optimization opportunities. A monitoring thread is awakened in the migrated VM when migration begins to stun the rogue processes for migration convergence. It can record the WWS of

each process in the migrated VM, and limits the maximum page faults for each process. In further, to make a trade-off between migration convergence and network bandwidth saving, a dynamic rate-limiting approach is designed by them to control migration bandwidth. Because memory dirty rate changes over time, setting a static network bandwidth for migration is not always optimal. A minimum bandwidth limit and a maximum bandwidth limit are predefined in their approach. Migration starts with the minimum bandwidth, and increases it with a constant increment each round. When the bandwidth reaches the maximum value or the remaining data size is smaller than 256KB, the migration enters the stop-and-copy phase. The upper bandwidth limit is used for the stop-and-copy phase to lower migration downtime.

Jin et al. [81] find that memory dirty rate approximately has a linear relationship with VM execution speed, based on several practical experiments. Assuming a VM runs a specific program, the faster the VCPU frequency is, the faster the I/O (especially write operation) speed is. Therefore, they propose to tune memory dirty rate to a desirable value to lower migration downtime by tuning VCPU execution frequency. Liu et al. [100] also control the CPU resources assigned to the migrated VM to improve migration performance by using the Credit algorithm of Linux kernel. Mashtizadeh et al. [109] design a similar mechanism—*Stun During Page Send (SDPS)*. It does not tune the frequency of a whole VCPU, and only injects delays to page writes to lower page dirtying rate.

Ibrahim et al. [70] aim to migrate the VMs with HPC applications. They propose to switch the iterative copying to the final stop-and-copy phase when no further reduction in downtime is achievable. They define three memory update patterns: (1) iterative copying does not reduce the amount of dirtied pages; (2) the number of dirtied page decreases for a short duration (such as, synchronization and barrier operations) so that a small downtime can be achieved; (3) most of transmitted pages are duplicate pages. For the first pattern, when a stable memory modification rate is detected, the migration will step into the stop-and-copy phase. For the second pattern, the iterative copying is stopped when the dirtied pages are transferred within a preset interval. For the third pattern, the retransmission rate is monitored. When it exceeds 90%, the migration process moves to the stop-and-copy phase. Atif et al. [6] propose a more rigid termination strategy. According to their statistics on Xen's default migration mechanism for HPC applications, iteratively copying the memory data of HPC applications is only a waste of time and CPU cycles. Therefore, they only iterate the pre-copy phase twice. The first iteration copies all memory pages, and the second iteration directly enters into the final stop-and-copy phase. This is to decrease total migration time and total network traffic at the expense of service degradation. In CloudNet [183], it firstly detects the point where the amount of transferred pages is equal or bigger than the dirtied pages. After this point, it stops the execution of the VM when the number of dirtied pages is smaller than any previous iteration.

Correlated VM Migration

In this section, we review the works on optimizing the performances of correlated VM migration. Deshpande et al. [36] name the migration of multiple co-located active VMs (on the same server) as *live gang migration*. They employ both data deduplication and delta compression to eliminate the duplicate memory pages exist between the co-located VMs. At the beginning of migration, all memory pages are transferred to the destination site. Only one copy of identical pages are transferred. The iterative copying phase only sends the page identifiers of newly-found duplicate pages to the destination server. They find that even though two pages are different, they may be partially identical. Therefore, they continue to use delta compression to reduce total network traffic. All duplicate pages are acting as reference pages, and unique pages (no duplicate) are compared with them to generate *deltas*. When the delta size of a page is smaller than a threshold, the delta is transferred to the destination site, otherwise, the entire page is transferred.

Deshpande et al. [35, 34] further propose a new migration mechanism—*gang migration using global (cluster-wide) deduplication* (GMGD), by expanding the migration approach for the VMs on a single host [36] to a server rack which holds many hosts. The GMGD works as: (1) all duplicate pages between the VMs on a rack are identified before migration, (2) only one copy of these duplicate pages are transferred to each target rack, (3) at the target racks, once a server received a duplicate page, it populates this page to other servers in the same rack which need this page instead of fetching it from the source rack. Data deduplication is also applied between the VMs on the source rack and the target rack.

Live VM migration not only imports interferences to both the source server and the destination server, but also to the VMs running on these two servers. Xu et al. [185] extensively analyze migration interference and co-location interference during and after migration. Migration interference refers to the service degradation of the VMs located on the source and the destination servers during migration, while co-location interference denotes the performance losses of the VMs on the destination server after new VMs are migrated in. They created performance models for these two interferences. Based on these models, they propose an interference-aware migration strategy—*iAware*, to minimize both migration interference and co-location interference. For each migration, iAware firstly chooses the migrated VM(s) with the least migration interference, and then chooses the target host(s) by estimating the co-location interference. iAware is lightweight and can be used as a complementary manner for other migration strategies.

Bari et al. [8] try to solve the similar problem but without the selection phase of the migrated VMs and the target servers. They aim at the migration sequence problem to minimize total migration time and downtime when the initial and target VM

placements are given. The key insight of their algorithm is to separate the migrated VMs into Resource Independent Groups (RIGs). The VMs in the same RIG will be migrated between distinct machines. In other words, at any time a server is only running one migration process to reduce the network contention and service interruption to other VMs running on it. Therefore, the VMs in the same RIG can be migrated simultaneously, and RIGs are migrated sequentially. For each RIG, the VM which has the shortest migration time will be firstly migrated.

Multi-tier applications are ubiquitous in cloud data centers. The VMs in a multi-tier application are normally communication-dependent. Some works try to optimize the migration for multiple correlated VMs from different perspectives. Wang et al. [176] make efforts on choosing an optimal migration sequence and a proper migration bandwidth for correlated VM migration. The migration sequence is derived by collecting a variety of information, such as network topology and traffic matrix of the data center, memory sizes and memory dirty rates of VMs. Sarker and Tang [143] design a dynamic bandwidth adaptation strategy to minimize the total migration time for a given number of VMs. The total migration time is controlled by adaptively choosing sequential and parallel migration and changing migration bandwidth.

Liu et al. [96] design an adaptive network bandwidth allocation algorithm to reduce the service interruption of live migrating multi-tier application over WAN. They migrate the correlated VMs concurrently and design a synchronization algorithm to make different migration processes finish at an approximate time. Their migration system consists of a central arbitrator and a migration daemon running in the Dom0 of Xen. The arbitrator dynamically tunes the network bandwidth for each migration process by collecting information from the migration daemon. Moreover, a *wait-and-copy* phase is introduced to synchronize different migration processes to start the final stop-and-copy phase at the same time.

In order to fully utilize network bandwidth to transfer as many VMs as possible in a given period of time, such as for the disaster recovery scenario, Kang et al. [85, 84] propose a feedback-based migration system. It adaptively changes the number of VMs in a migration by drawing experiences from the TCP's congestion control algorithm [73]. A controller starts the migration from a slow start (SS) phase where a small number of VMs (called *VM window*) are migrated in parallel, and increases the size of VM window gradually. When network congestion is detected, the migration enters a congestion avoidance (CA) phase where the VM window is reduced accordingly.

Ye et al. [188] propose to reserve resources (CPU cycles and memory space) for migration. At the target host, several empty VMs with 100% CPU utilization and certain memory spaces are created before VM migration. When the migration starts, these reserved VMs are shut down to leave resources to the migrated VMs. They find that parallel migration is better than sequential migration when enough resources

are available for migration, otherwise, parallel migration is worse. Based on their experimental results, several optimizations are designed. For example, they firstly migrate the VM with small memory size to increase migration efficiency. However, the resource reservation operation increases the migration interference to the destination server.

Others

Besides the common problems discussed above, some migrations conditions require special migration strategies. Liu et al. [99] struggle for the VMM heterogeneity problem, i.e. implementing VM migration between different VMMs. To smooth the heterogeneity of different VMMs, they design a common migration protocol and a common virtual machine abstraction method. The commands and data sent by the source VMM are transformed into intermediate formats which are then intercepted into the formats of the VMM at the target site. Based on this proposal, they implement VM migration between KVM and Xen.

Nathan el al. [117] comprehensively compare the performances of non-adaptive migration and adaptive migration with different parameter values, such as VM size and page dirtying rate. Non-adaptive migration technique migrates a VM at the maximum available bandwidth, while adaptive migration technique changes the migration bandwidth according to memory dirty rate. They find that non-adaptive migration is better than adaptive migration in most scenarios regarding migration performances (such as total migration time, downtime, total network traffic). However, adaptive migration utilizes fewer resources (CPU, network bandwidth) than non-adaptive migration. Based on these benefits of adaptive and non-adaptive migrations, they propose a novel migration technique—*Improved Live Migration (ILM)*. The key idea behind ILM is to use non-adaptive migration but with limited resources (network bandwidth and Dom0 CPU cycles) for migration.

Raghunath et al. [135] make efforts to lower the overhead of VM migration by choosing an appropriate migration triggering point. This is implemented by combining predicted future workloads and migration parameters. The migration-triggering system consists of two components: one is a centralized controller which runs as a resource usage collector, and one is running in the Dom0 of every physical machine. Whenever the central controller detects hotspot problem, it coordinates VM migration tasks according to the resource usage statistics (the utilizations of CPU, memory and network bandwidth) gathered from all individual servers. Baruchi et al. [9] also try to find an appropriate migration triggering point by exploring application resource consumption features. The execution history data of the applications on a VM are analysed with Fast Fourier Transformation (FFT) which is used to identify cyclic patterns in natural events to estimate the cycle size of the applications. Within each cycle, they then find the

moment which is suitable for starting VM migration with the prediction of migration cost.

Xia et al. [184] firstly use linear programming to formulate the path selection and bandwidth assignment problem when multiple VMs will be migrated from different source hosts to different destination hosts in the NFV scenario. Two approaches are proposed to solve this problem: critical edge pre-allocating approach and backtracking approach. The critical edge pre-allocating approach assign bandwidth to each migration process according to the available bandwidth of the edge all migration will pass through. The backtracking approach is a greedy strategy which initially assigns the network bandwidth according to the memory size of the migrated VM and decreases it when network congestion happens.

3.1.2 Post-copy

Post-copy firstly hands over the VM to the destination site. Therefore, the optimizations for it mainly focus on reducing the possibility of page fault after VM handover. In other words, they are to avoid remotely accessing memory pages from the source site when the VM is resumed on the destination host. Hines et al. [59, 58] holistically describe the process of post-copy. To reduce the page faults at the destination site and total migration time, they design four optimization mechanisms to accelerate the transfer of memory pages: demand-paging, active pushing, prepaging, and dynamic self-ballooning (DSB). With demand-paging, when page faults happen after the VM is running on the target server, it fetches these pages from the source server over the network. This access manner will result in a big service degradation. To reduce the possibility of page fault, memory pages are proactively copied by using active pushing and prepaging. Active pushing continuously copies memory pages to the target host in the background. Prepaging is based on the spatial and temporal localities of memory access. Every time when the source host receives a page fault from the destination host, it not only transfers this page, but also the pages surrounding it, to the destination site. DSB aims to avoid transferring free memory pages of the VM. To increase the robustness of post-copy, a periodic incremental checkpointing is suggested to synchronize the updated states back to the source host in case of migration failure. However, this not only neutralizes the advantages of post-copy (such as, transferring all pages only once) in comparison with pre-copy , but also introduces new overheads to the destination server.

Sahni and Varma [142] first iteratively scan the page table to identify the WWS of the migrated VM. The WWS is sent with the running states for VM resumption to the destination host to reduce the possibility of page faults. On-demand fetching, active pushing, prepaging and compression are combined to quickly cut down the dependency on the source host.

Resource overcommitment used to be an attractive point for cloud providers. However, Hirofuchi [62, 63] find that it is rarely utilized by cloud providers in practice. One reason is that pre-copy is widely adopted by VMMs for VM migration. To implement resource overcommitment, the idle and unused VMs in a data center must be consolidated on as fewer servers as possible to spare resources to accommodate more VMs. To achieve a high QoS, these VM must be quickly changed to a new location when they becomes active or are consuming more resources. However, pre-copy cannot meet this requirement due to its long handover time. Hirofuchi et al. propose to utilize post-copy for the instantaneous relocation of VMs for the overcommitment purpose. Two optimizations are used to lower page failures at the destination site: prepaging and active pushing. In prepaging, the neighbor 128 pages are copied for a page fault. Their mechanism can relocate a heavily-loaded VM within one second.

To avoid the migration convergence problem of pre-copy, Shribman et al. [151] propose to employ post-copy to migrate the VM with a high memory dirty rate. They design several optimizations to lower the service degradation resulting from remotely accessing memory pages: RDMA, Pre-paging and Linux Memory Management Unit (MMU) Integration. MMU Integration uses OS management tool to only pause the threads in the destination host which are waiting for memory pages from the source server and continue to run other threads.

3.1.3 Hybrid copy

Different hybrid-copy strategies can be gained by combining pre-copy with post-copy in different manners. Hu et al. [66] firstly transfer all memory data to the destination server. During this period, the newly dirtied pages are logged in a bitmap. Then the bitmap and the running states are sent to the destination server to resume the VM. The dirtied pages are on-demand fetched from the source server according to the bitmap. They further utilize delta compression to lower network traffic during the transfer of dirtied memory pages.

Kim et al. [86] list out the weaknesses of both pre-copy and post-copy migration schemes. To this end, they design a novel migration mechanism—*guide-copy*. It works like: firstly, the running states of the VM are copied to the destination server to resume the VM there; at the same time, the VM continues to run on the source host, and the memory access patterns are recorded; then, the memory pages are transferred to the destination server according to the memory access pattern to reduce page faults of the destination server; when the source server experiences a non-memory-intensive period, the execution of the VM on the source server is terminated; now the migration becomes a typical post-copy scheme. The VM context on the source host is called *guide context*, and that on the destination host is called *migrated context*. The guided context is used to guide the page transfer. However, their method has two problems. (1) When the

execution on the source server is slower than that on the destination server due to some reasons or the network gets stuck, the memory access patterns will not alleviate the page faults of the destination server. (2) During migration, the migrated VM is simultaneously running on both the source server and the target server, which results in a big resource waste.

Deshpande et al. [32] propose a traffic-sensitive technique to migrate co-located VMs. They believe that when the migration traffic has the same direction with the traffics of the applications running in the VM, they will contend for the network bandwidth. When their directions are opposite, there is no competition. Therefore, pre-copy will contend for network bandwidth with the applications on the source host which have outbound network traffic, while post-copy competes for network bandwidth with the applications on the destination host which have inbound network traffic. Based on these observations, their migration mechanism combines pre-copy with post-copy to lower the overall network contention for correlated VM migration. Some of the co-located VMs are migrated by pre-copy, whereas some are migrated by post-copy.

The WWS of a VM is much smaller than the full memory fingerprint. By utilizing this feature, Deshpande et al. [33] further propose to move non-WWS memory pages to a portable per-VM swap device before migration. Only the WWS pages are migrated through the direct TCP connection in pre-copy manner, while the non-WWS pages are remotely fetched from the swap device by the destination host on demand.

3.1.4 Summary of Memory Data Migration Technologies

The migration technologies for memory data are summarized in Table 3.1. The majority of optimizations are designed for pre-copy mechanism due to its robustness. Additionally, almost all optimization technologies are implemented on Xen or KVM because of the open-source feature. We also conclude the benefits and overheads of different optimizations according to the criterion discussed in Section 2.6. Total migration time, downtime, total network traffic and service degradation are denoted by T, D, N and I, respectively. The computation overhead, network overhead and space overhead of live VM migration are denoted by c, n and s, respectively. This denotation is applicable for the rest of this chapter. From the table, we can see that total migration time is the main consideration of migration performance. The improvements for other three metrics are evenly distributed. Different optimization strategies also introduce various overheads to the source or the destination hosts. Some optimizations introduce computation overhead, such as compression, data deduplication, CR/TR, page reordering, etc. Some bring in network overheads, such as distributed data deduplication, feedback-based migration, heterogeneous migration. Additional spaces are also required by some strategies, such as storing the reference data of delta compression and the hash tables of data deduplication.

Table 3.1: The summary of memory data migration technologies.

Pa	Reference	G	Tec.	VMM	Metrics				Overheads		
					T	D	N	I	c	n	s
Pr	[29]	S	BA,PWWS	Xen	✓	✓	✓		✓		
	[119]	S	-	VMw					✓		
	[195]	S	FE,NC	KVM	✓	✓			✓		✓
	[79, 80]	S	NC	Xen	✓	✓	✓		✓		
	[54, 159]	S	DC	KVM	✓	✓	✓		✓		✓
	[137, 94, 204]	S	DD	KVM	✓		✓		✓	✓	✓
	[197, 199, 83]	S		Xen	✓		✓		✓		
	[35, 34]	C		KVM	✓		✓		✓		
	[183]	S	DD,DC	Xen	✓		✓		✓		✓
	[36]	C		KVM	✓		✓		✓		✓
	[68, 70]	S	RDMA	Xen	✓	✓		✓			
	[98, 97]	S	CR/TR	-	✓	✓	✓		✓		✓
	[31]	S	CRe	Xen	✓			✓			
	[160, 26]	S	PR	KVM	✓		✓	✓	✓		✓
	[29, 81, 100]	S	MC	Xen	✓	✓	✓		✓		
	[109]	S		VMw	✓	✓			✓		
	[70]	S		KVM	✓	✓	✓	✓	✓		
	[96, 185]	C	IR,BA	Xen				✓	✓	✓	
	[8, 176, 143]	C	MS	-	✓	✓		✓	✓		
	[85, 84]	C	FB	KVM	✓				✓		
	[188]	C	RR	Xen	✓	✓			✓		✓
	[152]	C	AN	-			✓	✓	✓		
	[99]	S	HM	KVM Xen					✓	✓	
	[117]	S	RM	Xen	✓		✓	✓			
	[135, 9]	S	TP	Xen	✓	✓	✓	✓	✓		✓
	[107]	S	TS,BA	VMw	✓			✓	✓		
Po	[59, 58]	S	-	Xen	✓			✓	✓	✓	
	[142]	S	PWWS	KVM	✓			✓			
	[62, 63]	S	RO	KVM	✓		✓	✓	✓		
	[151]	S	MC	KVM	✓		✓		✓		
Hy	[66]	S	DC	Xen	✓		✓	✓	✓		✓
	[86]	S	GC	KVM	✓			✓	✓		
	[32]	C	NA	KVM	✓			✓	✓		
	[33]	S	PSD	KVM	✓		✓		✓		✓

Abbreviations: *Pa*: migration pattern. *Pr*: pre-copy. *Po*: post-copy. *Hy*: hybrid-copy. *VMw*: VMware. *G*: migration granularity (*S*: single, *C*: Correlated). *Tec.*: techniques. *FE*: free page elimination. *CRe*: checkpointing and replication. *PR*: page reordering. *MC*: migration convergence. *IR*: interference reduction. *BA*: bandwidth allocation. *TS*: target selection. *PS*: path selection. *MS*: migration sequence. *FB*: feedback-based. *RR*: resource reservation. *DD*: data deduplication. *DC*: delta compression. *NC*: normal compression. *AN*: application and network aware. *HM*: heterogeneous migration. *RM*: resource management. *TP*: triggering point. *PWWS*: postponing WWS transfer. *RO*: resource overcommitment. *GC*: guide-copy. *NA*: network contention alleviation. *PSD*: using portable swap devices.

As shown in Figure 3.3, for a better summary of memory data migration optimization mechanisms, we classify them along different points of the migration path: the migrated

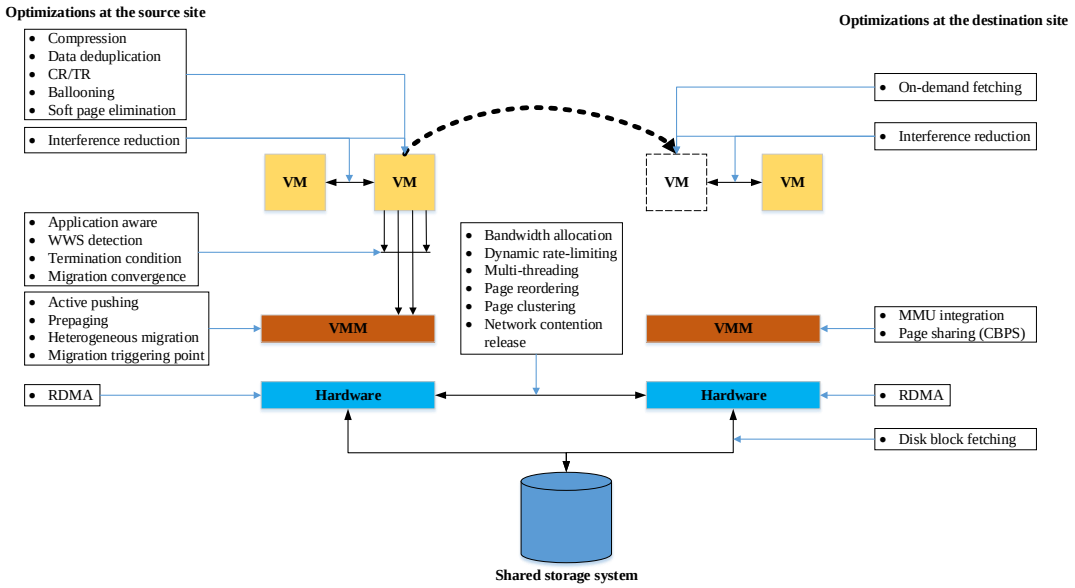


Figure 3.3: The optimizations at different points of memory migration path.

VM(s), other co-located VMs, I/O operation, VMM, hardware, network link, storage system. The optimizations for the migrated VM(s) are to reduce the original data size, such as compression [79, 80, 54], data deduplication [137, 199, 197], CR/TR [98, 97], ballooning [29, 59, 58, 66], free page elimination [29, 90], etc. For post-copy and hybrid-copy, the technology on this level is on-demand fetching. Some efforts aim to lower the impact of migration on the migrated VM and other VMs on the source and the destination hosts [79, 80, 185]. To decrease the data transferred in the iterative copying phase and make migration convergent, the I/O features of the migrated VM are analyzed and utilized by some migration mechanisms, such as, application-aware migration [185, 81, 6, 70], WWS detection [29, 160, 142], different termination conditions [2, 170, 183], migration convergence strategies [81, 6, 70]. The improvements on the VMM level contain active pushing, prepaging, heterogeneous migration [99], migration triggering point [135], MMU integration [151], and page sharing [199, 94, 204]. The RDMA functionality of interconnects can be also used for VM migration [68]. Many efforts are made to increase the utilization of network bandwidth, such as bandwidth allocation strategies for correlated VM migration [185, 8], dynamic rate-limiting [29, 70], multi-threading [94], page reordering [160, 26], page clustering [68], network contention alleviation [32]. The destination server also can directly fetch the clear memory pages (replicas of disk blocks) from the shared storage system to reduce the data transferred from the source site [83].

3.2 Storage Data Migration

Storage data migration has both similarities and differences with memory data migration. Both of them are to transfer data between two sites, so some memory data

migration mechanisms are also suitable for storage data migration, such as data deduplication. Storage data migration also faces different challenges. (1) Low migration bandwidth. Storage data migration normally happens between two data centers where the network bandwidth is much smaller than the interconnect of a data center. (2) Big data size. The size of the virtual disk of a VM ranges from several to hundreds of gigabytes. (3) Memory data have a closer relationship with the QoS of the migrated VM than storage data. With these conditions, some special optimization technologies different from those for memory data migration are proposed for storage data migration.

3.2.1 VMware Strategies

VMware Inc. [174] is the most productive enterprise about virtualization and cloud management technologies. Snapshotting, Dirty Block Tracking (DBT) and IO Mirroring are successively proposed for VMware ESX to migrate the storage data of a VM [108].

VM snapshotting works as: when a snapshot is taken for the storage data of a VM, the snapshot becomes read-only and all new writes are redirected to a new file. Based on this characteristic of VM snapshotting, snapshots are iteratively created and copied to the destination data center. DBT is similar with pre-copy memory data migration mechanism. Firstly the entire disk data are copied to the destination site by a full transfer phase, concurrently a bitmap is created to track the dirtied blocks. After finishing a transfer round, the dirtied blocks are transferred again. Until the amount of dirtied blocks becomes stable or a threshold is reached, the VM is suspended and the rest of dirtied blocks are copied to the destination site. Different from snapshotting, DBT operates in a smaller granularity, block level rather than snapshot level, which provides more optimization possibilities. For example, to lower the efforts of block tracking, only the blocks already copied to the destination site will be tracked, named *incremental DBT*. IO Mirroring mirrors all new writes from the migrated VM to the destination data center while the original storage data is copied in bulk. The copying process is based on VMKernel data mover (DM) [93].

They further integrate IO Mirroring with pre-copy memory migration mechanism and implement a live migration system—*XvMotion* [109]. A variety of optimizations are designed to improve storage data migration performance. Multiple TCP connections are created to accelerate data transfer, and a write barrier is used to ensure data consistency. To lower the impact of IO Mirroring on the running service, the writes are asynchronously instead of synchronously mirrored to the target site. XvMotion also supports to migrate a VM with multiple virtual disks which are separated on different volumes. To smooth the performance disparities of different volumes, it limits each disk to queue a maximum of 16 MB data into the shared transfer buffer. VMware also implements storage data migration by utilizing Cisco SAN extension technology [173].

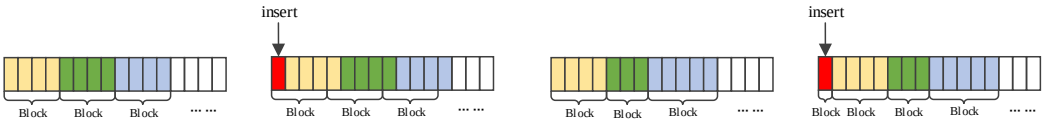
They extend the storage system to be shared by the source and the destination data centers. The FCIP I/O Acceleration of Cisco switches is enabled to decrease the time of accessing storage system through data center interconnect.

3.2.2 Replication

Replication consists of two concurrent parts: bulk transfer and I/O redirection. It is similar with IO Mirroring. Bulk transfer moves the original disk of the migrated VM, while I/O redirection asynchronously or synchronously sends the new writes from the migrated VM to the destination site. Both synchronous and asynchronous replications face advantages and disadvantages [136, 141]. Synchronous replication guarantees data consistency between the source and the destination sites, without the risk of losing data. Therefore, it is applicable for migrating the VM which is running applications with a high-security requirement, such as financial system. However, it cannot benefit from write coalescing to lower network traffic, even though a block is dirtied very frequently. Also, it leads to a bad service performance due to the long disk write latency. In contrast, asynchronous replication marks write operations as complete without the necessity of waiting for the responses from the destination site, therefore, it does not impact on application performance. It provides more optimization spaces for the synchronization of newly-dirtied blocks. For example, the frequently-dirtied blocks can be sent to the destination site periodically to decrease the data transferred over network. Different write operations can be also batched and pipelined to shorten synchronization time. Nevertheless, asynchronous replication may lose data (these have not been synchronized to the destination site) when the source server fails during migration.

By balancing the benefits and weaknesses of asynchronous and synchronous replication, Ramakrishnan et al. [136] propose to asynchronously replicate new writes during transferring the original disk data and change to synchronous replication after finishing the transfer, to avoid long I/O latency due to the bandwidth contention between the copying and the synchronization processes. The replication functionality is also implemented in some software (such as DRBD [38], HAST [56]) for high availability. Directly using the software to migrate storage data is also an option. For example, Wood et al. [183] employ DRBD to implement storage data migration in a similar manner as [136].

Liu et al. [98, 97] utilize Copy-On-Write (COW) technology to store the disk data of a VM in two layers: base image and COW image. The root file system of a VM are stored in the base image which remains read-only, and a COW image is used to store new data from the running VM. They asynchronously copy the base image to the target server before an anticipated migration, and the COW image is transferred with other data (memory, VCPU states) during VM migration. However, they assume the size of



(a) Fixed-size chunking (b) Variable-size chunking

Figure 3.4: The bit-shifting problem with file chunking.

COW image is considerably smaller than the basic image which is not the case in current data centers anymore. Therefore, a huge COW image will lead to a long migration time. Bradford et al. [19] only utilize asynchronous replication to migrate storage data. The new writes during migration are intercepted into *deltas* which are recorded in a queue. After finishing the bulk transfer of the original disk, these deltas are copied and applied to the disk image at the destination site. When the growth speed of deltas is bigger than the network bandwidth, write throttling is taken to slow down the VM execution speed.

3.2.3 Data Deduplication

Similar with memory data, there are also many duplicate blocks between different VM images [187, 74, 51]. Therefore, storage data migration can be accelerated by data deduplication as well. Data deduplication firstly cuts a VM image into blocks, and calculates a fingerprint for each block. The fingerprints are utilized to locate duplicate blocks within an image or between different images. These duplicate blocks are eliminated or transferred only once to reduce data transfer time. Each memory page is an identification unit in data deduplication for memory data migration. However, for storage data, data deduplication faces how to cut a big file into small pieces. There are two options: fixed-size chunking or variable-size chunking. Fixed-size chunking is simple and has a small computation overhead, but it suffers from bit-shifting problem. As shown in Figure 3.4(a), inserting one or several bits at the beginning of a block, all of the following blocks will be changed. Variable-size chunking [134] is not bothered by this problem. However, it is compute-intensive, which limits its benefits to migration performance improvement. Hence, fixed-size chunking is more popular for storage data migration [161, 82, 74]. Same as memory data migration, SHA-1 and SuperFastHash are still the two popular fingerprint-calculating algorithms [194, 204, 3, 199, 197].

As depicted in Section 3.1.1, data deduplication for storage data migration also can utilize three types of similarities: *intra-image similarity*, *inter-image similarity*, and *inter-site similarity*. Intra-image similarity denotes the duplication situation within the image of the migrated VM. Inter-image similarity is the duplicate blocks located between the images within a data center, while inter-site similarity is the duplication condition between the images at the source site and these at the destination site.

Different migration mechanisms exploit different types of similarities to facilitate VM storage data migration according to migration conditions. Bose et al. [17, 16] propose to decrease migration time at the expense of storage space. They keep several replicas of a VM image at different cloud data centers, and choose one replica as the primary copy. The changes from the primary copy are propagated to other replicas periodically. The selection of data center takes long-term average computation cost and end-user latency requirements into consideration. The Content Based Redundancy (CBR) technique with Rabin fingerprints [134] is employed to decrease the data transferred over the network during propagating changes. With this image distribution structure, they name the movement of VM storage data in their system (*CloudSpider*) as *hiberwake* (short for hibernate-awake) to differentiate from normal VM *migration*. In normal *migration*, all storage data must be transferred from the source site to the target site, while in the *hiberwake* only the new changes of storage data need to be copied to the target site. However, it is also accompanied with a big management cost and a big space consumption. The periodical change updates import additional network traffic as well.

Sometimes, we need to frequently migrate a VM between two or several fixed locations. For example, a personal VM system is commutatively used between home and office [156]. In such scenarios, many disk blocks are reusable. Takahashi et al. [161] combine data deduplication with DBT mechanism to speed up the migration between two fixed places. When a VM will be migrated back to a location where its previous version of disk data are located, only the newly dirtied blocks are transferred.

3.2.4 Software-based Approach

Some works carry out storage data migration by directly utilizing existing solutions or a software implementation. Hirofuchi et al. [61, 60] migrate VM storage data based on a block-level I/O protocol—*Network Block Device (NBD)* [101]. Their migration system consists of two NBD storage servers through which the source and the destination hosts access storage data, respectively. Virtual disks of a VM are block device files (e.g. */dev/nbd0*) on a host OS. At the beginning of migration, the memory data of the migrated VM are firstly transferred to the destination server, and then the storage data are migrated in a post-copy manner through the NBD connection between the two storage servers. Disk blocks are directly fetched from the NBD storage server at the source site, which lowers the interruption to the source host and other VMs on this host. On-demand fetching and background copying are combined to accelerate storage data transfer.

Tang [165] designs a new virtual machine image format–*Fast Virtual Disk (FVD)*, which supports a series of functionalities, such as Copy-On-Read (COR), adaptive prefetching, Copy-On-Write (COW), internal snapshot. Some of the functionalities

are beneficial to VM migration. For example, COR and adaptive prefetching can be combined to gradually copy the virtual disk in the background to the target host in a post-copy manner. COR transfers data blocks on demand, and adaptive prefetching transfers data during resource idle time. Adaptive prefetching even can be paused and its transfer rate is adjustable as well.

3.2.5 I/O-aware Migration

Storage data migration encounters the same problem as memory data migration. If storage data are migrated in post-copy pattern, the I/O feature of the migrated VM determines the frequency of remotely accessing disk blocks. If pre-copy is employed, the dirtying rate of disk blocks is critical for migration performance. Some migration mechanisms take the I/O features of the migrated VM into consideration for a better control of migration process. Zheng et al. [203] design a scheduling algorithm for storage data migration. Instead of transferring storage data from the beginning to the end of a disk, their algorithm considers the I/O characteristics of the workloads running in the migrated VM to arrange the migration sequence of disk blocks. It records a short history of the disk I/O operations to predict the future I/O characteristics in terms of temporal locality, spatial locality, and popularity (read/write frequency). According to their experiments, these I/O characteristics are predictable. The migration technology can be used to optimize different migration schemes (pre-copy, post-copy, and hybrid-copy) and has a strong adaptability for different workloads. It is beneficial to reduce the data iteratively transferred in the pre-copy migration pattern, to decrease the blocks remotely accessed in the post-copy migration pattern, and to improve both of these two aspects in the hybrid-copy migration pattern.

Similarly, Nicolae and Cappello [121] mainly aim at improving the storage data migration performance of I/O-intensive VMs. By utilizing the spatial and temporal localities of disk access, they propose a hybrid active push/prioritized prefetch strategy. They monitor and record how many times a block has been written, and only the blocks which have been written more than a preset threshold are marked as *dirty*. During VM migration, they avoid transferring these frequently written blocks and fetch them in decreasing order of access frequency from the destination site after VM handover.

3.2.6 Correlated VM Migration

Many VMs in a data center are correlated with several others rather than running independently [196, 11, 35]. Migrating one of them to another data center will lead to a severe service degradation due to the big network latency between data centers which decreases the communication performance between these VMs. Therefore, besides the three challenges of VM migration, correlated VM migration faces some new problems, such as migration sequence and the interruption to the inter-VM communication.

Al-Kiswany et al. [3] call the VM images belonging to the same application as a *VMFlock*. They design a migration system—*VMFlockMS*, to migrate a VMFlock. VMFlockMS consists of three components: *VM Profiler*, *VM Migration Appliance*, and *VM Launch Pad*. VM Profiler logs the blocks which are necessary for VM boot. These blocks are prioritized to transfer to the destination site. VM Migration Appliance utilizes distributed data deduplication and transfer approach to migrate storage data. Several nodes are used to parallelize data deduplication and transfer. Each node deduplicates and transfers the blocks with hashes in a specified range. Both inter-image and inter-site similarities are exploited. VM Launch Pad resumes the VMs once all blocks logged by the VM Profiler are received. The remaining data is migrated in post-copy manner.

A lack of progress management brings many issues to correlated VM migration. For example, how long will each migration process take?, how is the trade-off between application performance degradation and migration time?, how do correlated VM migration processes avoid splitting application components across data centers?, etc. In order to lower the impacts of migrating multiple correlated VMs over WAN on application performance, Zheng et al. [202] design a migration progress management system—*Pacer*. They firstly design models to predict the dirtying rates of memory and storage data and total migration time. On the basis of these models, Pacer manages migration by controlling the migration time of each migration process and coordinating them to finish at a close time to alleviate the component-split issue.

For the same problem, Zheng et al. [201] propose a communication-impact-driven coordination algorithm to decrease the service interference of VM migration. They formulate multi-tier application migration as a problem of minimizing performance impact. They implement a migration system—*COMMA*, which consists of a central controller and a local process in each VM's hypervisor. It migrates VMs in two steps. In the first step, it coordinates the migration of all VM's storage data to make them finish at the same time. In the second step, they put VMs into different *valid groups*. The sum of the dirtying rates of the VMs in a valid group is smaller than the available network bandwidth. Then inter-group scheduling and intra-group scheduling are combined to lower the impact of migration on the communications between the migrated VMs and improve the network utilization, respectively. Regarding the migration sequence problem, Cerroni [23, 24] compares sequential and parallel migration strategies when a group of VMs will be migrated in cloud federation. The results illustrate that sequential migration has fewer influence on network performance and parallel migration results in a smaller downtime.

3.2.7 Others

Apart from the general issues with storage data migration, many researchers try to solve some problems in special situations. Luo et al. [104] propose a Three-Phase

Migration (TPM) algorithm which is composed of pre-copy, freeze-and-copy, and post-copy. In the pre-copy phase, storage data and memory data are iteratively transferred to the destination site. The dirty information of storage data is recorded in a block-bitmap. In the freeze-and-copy phase, the VM is suspended and the dirtied memory data and the block-bitmap are sent to the destination server. In the post-copy phase, the VM is resumed on the target server, and the modified storage data blocks are moved in a pull and push manner according to the block-bitmap. They also use write throttling for the I/O intensive workloads to ensure migration convergence. After the VM is moved to the target server, a new block-bitmap is created to record the new changes of disk data. Because in some scenarios (such as hardware maintenance) the VM will be migrated back to the original server, they further propose an Incremental Migration (IM) scheme. When the VM is being migrated back to the original server, only the blocks are dirtied in the destination server are synchronized to the source site according to the new block-bitmap.

Nowadays, Solid State Drive (SSD) is widely used in data centers. They have faster I/O speed than mechanical Hard Disk Drive (HDD). Zhou et al. [206] try to solve the I/O speed disparity problem when migrating VM images between SSD and HDD. They design three migration strategies for different situations: (1) Low Redundancy (LR) mechanism. Because all disk data of the migrated VM will be eventually moved to the destination site, LR mechanism directly writes data to the destination host during migration. (2) Source-based Low Redundancy (SLR) mechanism. It is based on LR mechanism, but it keeps the I/O operations to the to-be-copied region at the source site while issues write operations to the copied region to the destination site. (3) Asynchronous I/O Mirroring (AIO) mechanism. It is derived from IO Mirroring [109]. The original IO Mirroring writes data to both the source and the destination sites, but AIO marks the write operation as complete when the faster disk accomplishes the write operation and the slower disk conducts the write operation in the background. The first and third strategies are for the migration from a slow disk (HDD) to a fast disk (SSD), while the second is for the migration from a fast disk (SSD) to a slow disk (HDD).

Normal migration approaches only transfer the memory and storage data of the migrated VM to the target site, but without moving the host-side cache data. Therefore, the VM will suffer from a big performance degradation after resumption on the destination server. Lu et al. [102] design a cache warm-up mechanism–*Successor*, to recover the cache data at the destination host during VM migration. They parallelize the warm-up process with the migration process. *Page preloading* is utilized to warm up the cache of the destination host from the storage server when a VM is migrated within LAN.

Shen et al. [150] design a geo-replicated image file storage to support efficient migration of VM storage data based on Supercloud [76]. It cuts an image file into constant size blocks (4KB) and stores several replicas for each block in other cloud data

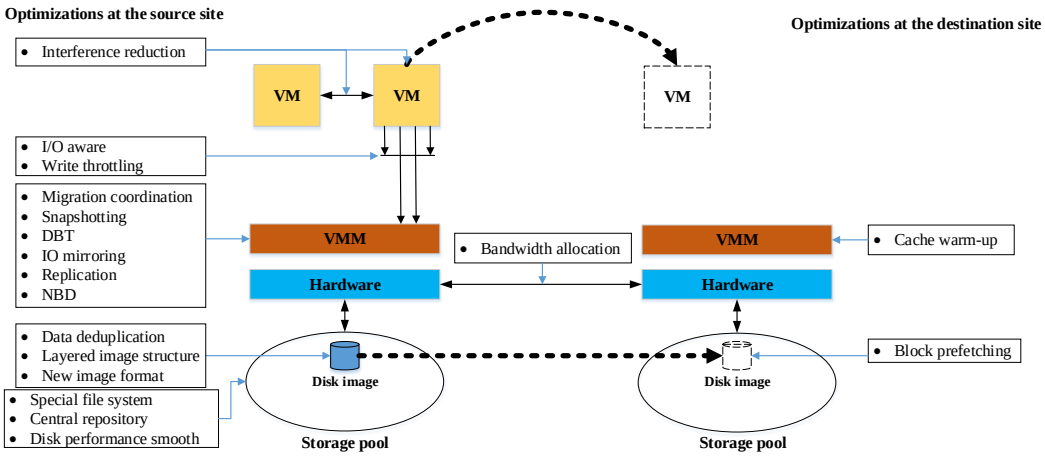


Figure 3.5: The optimizations at different points of storage data migration path.

centers. The primary replica (i.e., where the VM is running) propagates block updates to other replicas according to a transition probability table which is added into the meta-data of each image file. The transition probability table records the probability that a VM is migrated from current cloud data center to another. It can be created by users or trained on the fly. They also assign a priority for each block according the read/write frequency. During VM migration, the blocks with high priority are firstly propagated to the destination site and then update other blocks. Besides, many blocks of a VM image file (e.g., the blocks of base image) remain unmodified during their whole lifetime, which will be further beneficial to storage data migration.

3.2.8 Summary of Storage Data Migration Technologies

The migration technologies for storage data are summarized in Table 3.2. For the *Pattern* column, one pattern denotes the paper only considers storage data migration, a pattern pair means the paper considers both memory data and storage data migrations, and *All* denotes the proposed optimization is suitable for all migration patterns. From the table, we can find many common phenomena with memory data migration. (1) Same as memory data migration, data deduplication is still an important optimization technology for storage data migration. (2) Pre-copy pattern is also widely utilized for storage data migration due to its robustness, and the majority of optimizations are designed for it. Combining with pre-copy memory migration pattern, Pre-Pre becomes the most popular migration pattern for live VM migration over WAN. (3) KVM and Xen are the two popular experimental platforms. (4) Most of the studies are concentrating on single migration. Actually, optimizing the performances of correlated VM migration is strongly desirable as well. (5) The total migration time is still the main concern among all performance metrics. (6) Computation overhead to the source and the destination servers are the main side effect. Only a small part of optimization technologies bring in network and space overheads.

Table 3.2: The summary of storage data migration technologies.

Main	Others	Reference	Pa	G	VMM	Metrics			Overheads		
						T	N	I	c	n	s
SNT	-	[108]	P-P	S	VMw				✓	✓	✓
DBT	iDBT					✓	✓		✓		✓
IOM	MPTCP BC	[108, 109]	P-P	S	VMw	✓		✓		✓	
eSAN	CIOA	[173]	P-P		VMw	✓	✓	✓		✓	
Rep	A-S	[136]	P	S		✓	✓				
	BRDB	[183]	P-P	S	Xen		✓	✓			
	LIS	[98, 97]	H	S	-			✓			✓
	A,WT	[19]	P-P	S	Xen	✓	✓			✓	
DD	DCBA	[138]	-	S	KVM	✓			✓		✓
	HW	[17, 16]	-	S	-	✓				✓	✓
	DBT	[161]	-	S	KVM	✓	✓		✓		✓
	-	[3]	O	C	-	✓	✓		✓		
SA	NBD	[61, 60]	P-O	S	Xen KVM	✓	✓				
	FVD	[165]	-	S	KVM	✓	✓				
IOA	BS	[203]	All	S	-	✓	✓	✓	✓		✓
	PPP	[121]	H	B	KVM	✓	✓	✓	✓		✓
MPM	PM	[202]	P-P	B	KVM			✓	✓	✓	
IR	PIM	[201]	P	C	KVM			✓	✓		
TPM-IM	WT	[104]	P-P	S	Xen	✓	✓		✓		✓
HM	LC,SLR AIO	[206]	-	S	Xen	✓		✓	✓		
CWU	PP PWM	[102]	P-P	S	KVM			✓	✓	✓	
ML	PPr	[4]	All	B	-	✓	✓		✓		✓
GRFS	PBM	[150]	H	B	Xen	✓		✓	✓	✓	✓
DHS	-	[92]	All	B	Xen		✓	✓	✓		✓

Abbreviations: *Main*: main migration strategy. *Pa*: pattern. *P*: pre-copy. *O*: post-copy. *H*: hybrid-copy. *G*: granularity (*S*: single, *C*: Correlated, *B*: both). *VMw*: VMware. *Others*: other optimizations. *SNT*: snapshotting. *DBT*: dirty block tracking. *iDBT*: incremental DBT. *IOM*: IO Mirroring. *eSAN*: extended SAN. *Rep*: replication. *A-S*: asynchronous and synchronous replication. *A*: asynchronous replication. *IOA*: I/O-Aware. *MPM*: migration process management. *SA*: software-based. *IR*: interference reduction. *RM*: resource management. *MS*: migration sequence. *TPM-IM*: Three-Phase Migration and Incremental Migration. *HM*: heterogeneous migration. *CWU*: cache warm-up. *BC*: buffer congestion control. *CIOA*: Cisco FCIP I/O Acceleration. *WT*: write throttling. *DCBA*: distributed content-based addressing. *HW*: hiberwake. *DD*: distributed data deduplication. *BS*: block scheduling. *PPP*: push/prioritized prefetch. *PM*: performance predict models. *MC*: migration coordination. *PIM*: performance impact model. *PP*: page preloading. *PWM*: Piggyback warm-up on migration. *ML*: machine learning. *PPr*: performance prediction. *GRFS*: geo-replicated file storage system. *PBM*: priority-based block migration. *DHS*: device hot-swapping.

We also illustrate different migration technologies and optimizations through the migration path in Figure 3.5. The majority of migration technologies take the synchronization-like manner, i.e., the original disk image is copied in bulk and the new disk writes are recorded and iteratively transferred to the destination site, such as snapshotting, DBT [108], IO mirroring [108, 109], replication [183, 136, 98, 97, 19]. Other available

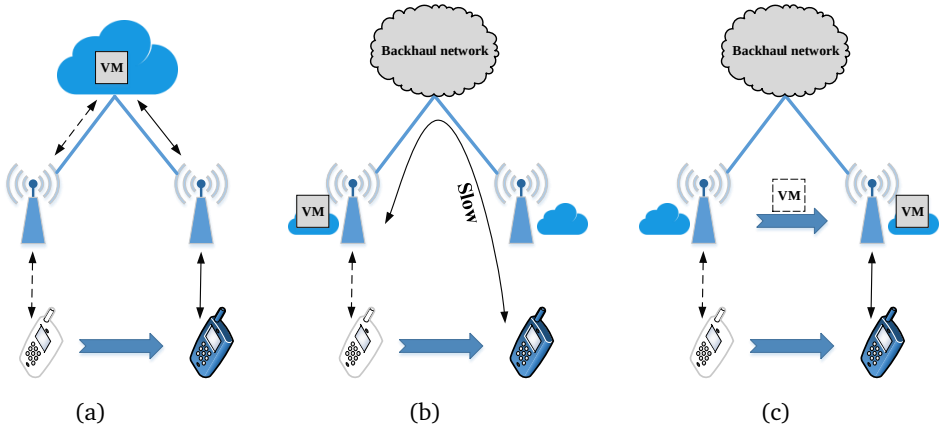


Figure 3.6: The comparison between VM migration in cloud computing and MEC. (a) Cloud computing, (b) No VM migration in MEC, (c) VM migration in MEC. The double-arrowed lines denote the communication between a UE and its VM.

migration mechanisms include NBD [61, 60], central base image repository [192], etc. On the basis of these technologies, many optimization strategies are designed to further improve the migration performance, such as data deduplication [194, 204, 3, 199, 197], write throttling [104], layered image structure [194, 192, 98, 97, 21], new image format [165], cache warm-up [102], bandwidth allocation [172, 171], etc.

3.3 User Mobility-induced VM Migration

MEC has two basic differences from conventional cloud computing: (1) an edge cloud data center is close to users (sometimes only one-hop away); (2) users are with a high mobility. Conventional cloud computing denotes that users offload tasks to a remote central cloud data center (as shown in Figure 3.6(a)). These differences introduce new problems for live VM migration in MEC besides solving the three tasks of live migration in WAN environments. For example, when a user moves out the coverage area of the current edge cloud data center, two options are available for how to tackle with the corresponding VM. The first one is that the VM is not migrated and the user remotely accesses it (as shown in Figure 3.6(b)). The second one is that the VM is migrated to the edge cloud data center where the user is currently at (as shown in Figure 3.6(c)). Both options also face a trade-off between migration costs and gains to decide whether a VM will be migrated or not. In this section, we will review the works on live VM migration in MEC from two perspectives: *migration performance improvement* and *migration performance analysis*. At last, these studies are summarized and discussed with the metrics extracted from the literature.

3.3.1 Migration Performance Improvement

Ha et al. [53] create a VM in an edge cloud data center by generating an overlay which is the compressed binary difference between a base VM image and the launch VM image (both memory and disk data). Based on this VM provisioning manner, they only migrate the overlay while migrating a VM with the assumption that the base image is present at the destination site [52]. In further, a pipeline of processing stages (delta-encoding, deduplication, compression) are designed to reduce the overlay size. The settings of these stages are adaptively configured by dynamically monitoring the migration performance to achieve the shortest total migration time.

Taleb and Ksentini [163, 164] proposed a Follow-Me-Cloud (FMC) concept, which aims to ensure a UE to always be connected to and served by the optimal data center. To achieve this goal, they introduce two additional components to the conventional network architecture, namely FMC controller and data center/gateway (DC/GW) mapping entity. The DC/GW mapping entity is to update the network anchor point of a UE while it is moving from location 1 to location 2. The FMC controller is in charge of selecting the optimal data center according to a UE's current location and migrating the UE's service to this data center. Based on this architecture, several mechanisms are designed to guarantee the service continuity during migration, such as replacing data anchoring at the network layer by service anchoring, replacing IP addressing by service/data identification, etc.

Teka et al. [166] solve the network connection problem when a UE is moving between edge cloud data centers by using Multipath TCP (MPTCP) [114]. Each VM in an edge cloud data center is assigned two network interfaces. One is operated in the regular VM operation mode, and the other is activated only after the VM is migrated. Similarly, MPTCP is utilized by Qiu et al. [133] between two edge cloud data centers to improve the connection fault tolerance and reduce migration time.

To both improve VM migration speed and reduce the cost of improper VM migration due to the uncertainty of user mobility, Saurez et al. [147] choose the target node according to the workload running on the node and initialize VM migration according to the latency between a VM and its user. A VM is migrated in two steps: (1) when the latency is bigger than a threshold, all persistent data (such as disk data) generated by the VM is migrated to the target node, called *state migration*; (2) when the latency continues to increase and becomes bigger than another bigger threshold, the remaining data of the VM is migrated, called *computation migration*.

Secci et al. [149] link VM mobility to user mobility by leveraging one of the following three options: (1) moving the VM to an edge cloud data center which is closer to the user; (2) just switching the data-center routing locator to lower the network access latency; or (3) performing both the previous operations. These operations also must

be transparently conducted with respect to the underlying network infrastructure. To these ends, they propose a cloud access overlay protocol architecture based on LISP protocol. A controller is introduced to adaptively determine the best entry DC and the best VM location on a per-user basis.

Ottenwälder et al. [125, 124] propose a placement and migration method for the VMs of a mobile Complex Event Processing system which is running on infrastructures which contain both cloud and fog resources. They take both the gains and the costs of VM migration into consideration. By using the predicted future movement of users, they design a migration plan for each VM in advance from a systematic perspective to lower the overall network utilization and meet the user-defined latency requirements.

Sun and Ansari [157] propose to place a number of replicas for each VM virtual disk at several edge cloud data centers which are selected by a LatEncy Aware Replica placemeNt (LEARN) algorithm. With this manner, only the newly dirtied disk blocks need to be migrated to the destination site during VM migration. However, it incurs a big storage space consumption and a big overhead on VM image management. Machen et al. [106] store VM images in layers (similar to [194, 192]) and use cloning and incremental synchronization to recreate the missing layers of the migrated VM at the destination site.

3.3.2 Migration Performance Analysis

Some studies mainly focus on quantitatively analyzing the VM migration procedure in MEC, especially the costs and the benefits. They are critical for the design of a better migration strategy. Gkatzikis and Koutsopoulos [49] list the parameters which may influence on the efficiency of migration mechanisms in MEC, such as workload uncertainty, unpredictability of multi-tenancy effects, unknown evolution of accompanying data volume, etc. They propose that migration strategies can be made at three different levels: cloud-wide migration policy, server-initiated migration policy and task-initiated migration policy.

Ksentini et al. [91] formulate the migration procedure in FMC as an Markov Decision Process (MDP). To solve the trade-off between migration costs and gains, a decision policy is proposed for whether to migrate a service when a UE is at a certain distance from the source DC. Wang et al. [178] also model the migration procedure as an MDP. But they only consider the situation where a UE follows a one-dimensional asymmetric random walk mobility model. Under this situation, the optimal policy for VM migration is a threshold policy. Then they propose an algorithm for finding the optimal thresholds. In comparison with [91], they design an optimal threshold policy to find the optimal action of the MDP. Taleb and Ksentini [162] use Markovian models to analyze the performance of FMC. The evaluated metrics include the probability of a user to be

always served by the optimal DC, the average distance from the optimal DC and the cost of VM migration.

Nadembega et al. [116] propose two estimation schemes for live VM migration in MEC. (1) Data transfer throughput estimation scheme, i.e., the data flow size between a UE and its VM during VM migration. (2) VM handoff time estimation scheme. With the support of these two schemes, a VM migration management scheme is further proposed. Sun and Ansari [158] create models for the gains and costs of VM migration in MEC, respectively. Based on these two models, they design a VM placement strategy to maximize the profit of live VM migration which is the difference between migration gains and costs.

3.3.3 Summary of User Mobility-induced Migration Technologies

Table 3.3: The summary of user mobility-induced migration technologies.

Per	Reference	Technique	Others	Metrics			Overheads		
				T	N	I	c	n	s
PI	[53]	VM overlay	DE,DD,DC	✓	✓		✓		
	[163, 164]	FMC	DC/GW			✓	✓		
	[166]	MPTCP	-			✓		✓	
	[133]	MPTCP	-	✓		✓		✓	
	[147]	TSM	-	✓		✓		✓	✓
	[149]	CAOP	-			✓		✓	
	[125, 124]	MP	-	✓	✓	✓			
	[157]	DR	-	✓			✓	✓	✓
	[106]	LIS	-	✓	✓				
Per	**Reference**	**Technique**	**Others**	**CIF**			**PM**		
PA	[49]	TLMS	IF	WL,DS,BW MLT,DCC			T,I		
	[91]	MDP	DP	UMF,BW,DCU			MG,MC		
	[178]		TP	UMF			MG,MC		
	[162]		-	UMF			T,ADCU		
	[116]	PES	MMS,MDP	UMF,DS			I,MC		
	[158]	GPM	VMP	BW,T,DCU			MG,MC		

Abbreviations: *Per*: perspective. *PI*: performance improvement. *PA*: performance analysis. *Others*: other mechanisms proposed in the paper. *CIF*: considered influence factors. *PM*: performance metrics. *TSM*: two-stage migration. *CAOP*: cloud access overlay protocol. *MP*: migration plan. *DR*: disk replicas. *LIS*: layered image structure. *TLMS*: three-level migration strategy. *PES*: performance estimation schemes. *GPM*: gerneal performance model. *DE*: delta-encoding. *DC/GW*: data center/gateway (DC/GW) mapping entity. *IF*: influence factors. *DD*: data deduplication. *DP*: decision policy. *TP*: threshold policy. *MMS*: migration management scheme. *VMP*: VM placement strategy. *WL*: workload type. *MLT*: multitenancy effect. *DS*: data size. *BW*: network bandwidth. *DCC*: data center capacity. *DCU*: the distance between data center and users. *MG*: migration gain. *MC*: migration cost. *UMF*: user mobility feature. *ADCU*: average distance between a user to the optimal DC.

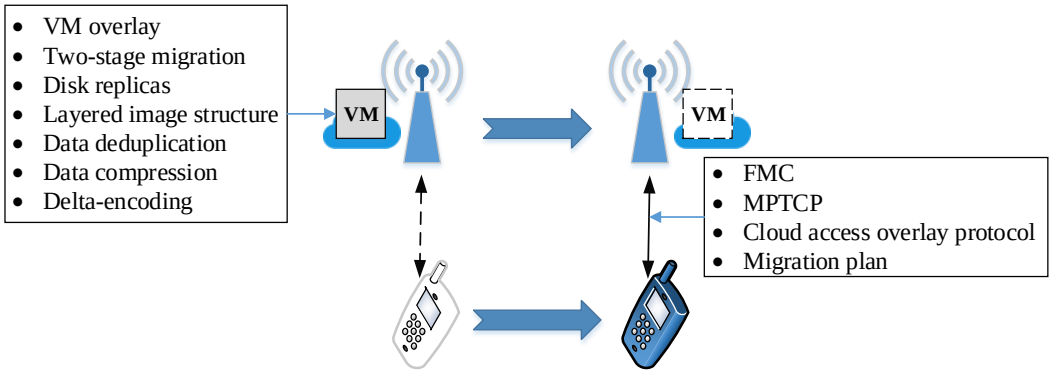


Figure 3.7: The summary of migration optimization technologies in MEC.

The studies on user mobility-induced migration technologies are summarized in Table 3.3. From the perspective of improving migration performance, the optimizations can be divided into two categories: (1) reducing the data transferred during VM migration and (2) keeping a smooth connection between the migrated VM and the moving user, as shown in Figure 3.7. Regarding the first one, the proposed technologies include VM overlay [53], two-stage migration [147], disk replicas [157], layered image structure [106], data deduplication, compression, delta-encoding [53]. Those for the second issue contain FMC [163, 164], MPTCP [166], cloud access overlay protocol [149], migration plan [125, 124].

As discussed before, the researches on migration performance mainly concentrate on analyzing the trade-off between migration gains and costs, which is critical for migration decision making and destination site selection. MDP [91, 178, 162, 116] is widely used to formulate the procedure of live VM migration in MEC where both users and VMs may change their locations.

3.4 Chapter Summary

This chapter summarizes the state-of-the-art technologies of memory data migration, storage data migration and user mobility-induced migration. All the technologies are compared with each other by using the performance metrics and overheads discussed in Section 2.6. From the review, we get the following conclusions: (1) some mechanisms are needed to improve the controllability of memory data migration (such as, solving the convergence problem of pre-copy); (2) many optimization technologies are available for storage data migration, but it is difficult to significantly improve the migration performance with these optimizations because of the physical conditions (the huge size of storage data and the slow network bandwidth between data centers); (3) many outstanding problems are with the migration in MEC due to its novelty.

Chapter 4

VM Migration in LAN—Performance Control

According to the review on memory data migration technologies in Section 3.1, in this chapter we focus on the problems with VM migration in LAN (as presented in Section 1.1.1). Regarding these problems, we firstly create performance models for VM migration in LAN, and then a series of relationships between migration performances and the influence parameters are analyzed based on Xen platform. We find that the memory dirty rate of the migrated VM and the network bandwidth for migration are the two important factors which can be tuned to control the performance of a migration process. Then, a migration control algorithm is designed by using the analytical results.

4.1 Performance Analysis for Pre-copy

As discussed in Section 2.3.1, pre-copy [29, 119] is a dominant manner for VM migration in LAN due to its robustness. It can be divided into three phases: *full transfer*, *iteration* and *stop-and-copy*, as shown in Figure 4.1. In the full transfer phase, all memory data are sent to the destination host. In the iteration phase, newly dirtied memory pages are logged and migrated in rounds. The data transferred in current round i are the pages dirtied during the data transmission of the previous round $i-1$. When the iteration meets one of termination conditions, the VM is suspended on the source host and all remaining data are copied to the destination host to resume the VM there, i.e. the stop-and-copy phase.

4.1.1 Performance Model

The notations used in this chapter are listed in Table 4.1. According to the migration procedure of pre-copy, the performance for each migration phase can be calculated as follows.

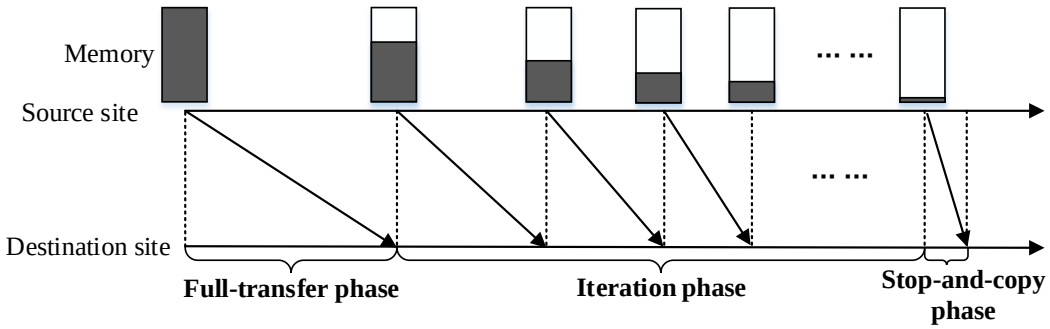


Figure 4.1: The workflow of pre-copy.

Table 4.1: The notations used in this chapter.

Symbol	Description
M	The allocated memory size
B	Network bandwidth available for migration
d	Memory dirty rate
ρ	Dirty bandwidth ratio, $\rho = \frac{d}{B}$
n	The total number of iteration during the iteration phase
T	Total migration time
D	Migration downtime
V	Total migration network traffic

The full-transfer phase: $v_0 = M$, $t_0 = \frac{M}{B}$. v_0 and t_0 are the transferred data and the transmission time, respectively.

The iteration phase:

$v_1 = d \cdot t_0 = \frac{Md}{B} = M\rho$, $t_1 = \frac{v_1}{B} = \frac{M}{B}\rho$;

$v_2 = d \cdot t_1 = d \cdot \frac{M}{B}\rho = M\rho^2$, $t_2 = \frac{v_2}{B} = \frac{M}{B}\rho^2$;

.....

$v_n = d \cdot t_{n-1} = \frac{Md^n}{B^n} = M\rho^n$, $t_n = \frac{v_n}{B} = \frac{M}{B}\rho^n$.

Here, v_i and t_i are the transferred data and the transmission time of the ith round of iteration, respectively.

The stop-and-copy phase: This phase consists of three portions: the time of suspending the VM, the transmission time of the remaining data, and the time of resuming the VM. The time for suspending and resuming a VM almost is a constant regardless of the VM memory size [200], denoted by C and set as $0.1s$ [29, 119]. The remaining data for the stop-and-copy phase are the dirtied memory pages in the nth round of iteration. Hence, the time t_s and the transferred data v_s in the stop-and-copy phase are calculated as follows:

$$v_s = d \cdot t_n = M\rho^{n+1}$$

$$t_s = \frac{v_s}{B} + C = \frac{M}{B}\rho^{n+1} + C$$

If we ignore the time for logging and scanning the dirtied memory pages during each iteration round, the total migration time T, downtime D and the total network traffic V can be gained.

$$T = t_0 + t_1 + t_2 + ... + t_n + t_s = \frac{M(1-\rho^{n+2})}{B(1-\rho)} + C \tag{4.1}$$

$$D = t_s = \frac{M}{B}\rho^{n+1} + C \tag{4.2}$$

$$V = v_0 + v_1 + v_2 + ... + v_n + v_s = \frac{M(1-\rho^{n+2})}{1-\rho} \tag{4.3}$$

4.1.2 Performance Features

In this chapter, we base our study on Xen platform. Xen sets three termination conditions to stop an iteration phase and start the stop-and-copy phase [2]:

- **C1:** Less than 50 pages are dirtied during the last pre-copy iteration, denoted as R_{tc};
- **C2:** 29 iterations have been carried on, denoted as I_{tc};
- **C3:** More than 3 times the memory size allocated to the VM has been copied to the destination host, denoted as V_{tc}.

From the performance models of pre-copy migration in Section 4.1.1, we can see that pre-copy performance is highly related to the dirty bandwidth ratio ρ. To understand the relationship between ρ and migration performances, in this section a series of features are found from the performance models for a specific VM, i.e., the VM memory size M is fixed.

Feature 1: *A bigger ρ requires more iterations to decrease the remaining data to a preset threshold Thd ($Thd < M$).*

We assume that the remaining data of a VM are smaller than Thd after k rounds of iterations. In other words, the data will be transferred in the $(k+1)$th round will be smaller than Thd. Then, the value of k can be calculated as follows.

$$v_{k+1} < Thd, \Longrightarrow M \cdot \rho^{k+1} < Thd, \Longrightarrow k = \lceil \frac{\lg \frac{Thd}{M}}{\lg \rho} - 1 \rceil \tag{4.4}$$

When $\rho \geqslant 1$, it means that memory dirty rate is bigger than migration bandwidth. It will be a full transfer of the whole memory data in each iteration round, so it is impossible to reduce the remaining data to Thd under this situation. We can represent the number of iteration by ∞. When $\rho < 1$, we have $\lg \rho < 0$. Therefore, k is monotonically increasing with ρ in Equation (4.4). By combining these two situations, **Feature 1** is proofed. ■

Feature 2: *When the migration process of a VM is terminated by C1, C2, and C3, the corresponding dirty bandwidth ratios are ρ_1, ρ_2 and ρ_3, respectively. Then we have $\rho_1 < \rho_2 < \rho_3$.*

If an iteration phase is terminated by C1, the total number of iterations must be smaller than 29. Otherwise, it will be terminated by C2. According to **Feature 1**, we can get $\rho_1 < \rho_2$.

Similarly, the number of iteration for C3 also is smaller than 29, denoted by i. From Equation 4.3, we have the following relationship: the migration process terminated by C3 transfers more data than that by C2 with fewer iteration rounds ($i < 29$), as shown in inequality (4.5).

$$\begin{aligned} V_2 < V_3 \Longrightarrow & \frac{M(1-{\rho_2}^{31})}{1-\rho_2} < \frac{M(1-{\rho_3}^{i+2})}{1-\rho_3}; (i < 29) \\ \Longrightarrow & 1+\rho_2+{\rho_2}^2+...+{\rho_2}^{31} < 1+\rho_3+{\rho_3}^2+...+{\rho_3}^i \end{aligned} \tag{4.5}$$

Here, V_2 and V_3 are the total network traffic when the migration process is stopped by C2 and C3, respectively. From inequality (4.5), we can easily get $\rho_2 < \rho_3$. In summary, we have the relationship $\rho_1 < \rho_2 < \rho_3$. ■

Furthermore, the ranges of ρ_1, ρ_2, and ρ_3 can be calculated.

(1) For C1, the total number of iteration n satisfies the relationship $0 \leqslant n < 29$. Then, the range of ρ_1 can be gained by using the Equation (4.4).

$$0 \leqslant \lceil \frac{\lg \frac{R_{tc}}{M}}{\lg \rho_1} - 1 \rceil < 29 \Longrightarrow 0 \leqslant \rho_1 < \sqrt[28]{R_{tc}/M} \tag{4.6}$$

(2) From the definition of C3, we can get that the data transferred during the iteration phase is bigger than or equal to $3M$ and smaller than $4M$ and the data transferred during the stop-and-copy phase is at most M. Hence, the total migration

traffic when a migration is terminated by C3 is bigger than or equal to $4M$ and smaller than $6M$. Then we can get the following inequality:

$$\begin{aligned} 4M < V < 6M \Longrightarrow 4M < \frac{M(1-{\rho_3}^{n+2})}{1-\rho_3} < 6M \\ \Longrightarrow 4 < \frac{1-{\rho_3}^{n+2}}{1-\rho_3} < 6 \end{aligned} \tag{4.7}$$

It is obvious that $\rho_3 \geqslant 1$ is a solution of inequality (4.7) according to the migration procedure of pre-copy. The iteration phase will be terminated after 3 rounds of iterations. According to **Feature 2**, actually we only need to find the smallest value for ρ_3. Then all values which are bigger than this value will make the migration process terminated by C3. According to **Feature 1**, we can get that the smallest value for ρ_3 is the one that corresponds to 28 rounds of iterations. Then the inequality (4.7) is changed to:

$$4 < \frac{1-{\rho_3}^{28+2}}{1-\rho_3} < 6 \tag{4.8}$$

Then we can get:

$${\rho_3}^{30} - 4\rho_3 + 3 < 0 \tag{4.9}$$

$${\rho_3}^{30} - 6\rho_3 + 5 > 0 \tag{4.10}$$

We represent the left side of inequalities (4.9) and (4.10) with functions and calculate their derivatives as follows.

$$y_1 = {x_1}^{30} - 4x_1 + 3 \quad (0 < x_1 < 1) \tag{4.11}$$

$$y_2 = {x_2}^{30} - 6x_2 + 5 \quad (0 < x_2 < 1) \tag{4.12}$$

$${y_1}' = 30{x_1}^{29} - 4 \tag{4.13}$$

$${y_2}' = 30{x_2}^{29} - 6 \tag{4.14}$$

From the derivative functions, we can get the graphs of Function (4.11) and (4.12) by using monotonic feature, as shown in Figure 4.2.

By using brute force, we get: $a \approx 0.75$ and $b \approx 0.835$. Therefore, according to Figure 4.2, the solutions for inequalities (4.9) and (4.10) are $0.75 < \rho_3 \leqslant 1$ and $\rho_3 < 0.835$, respectively. Then, the solution for inequality (4.8) is $0.75 < \rho_3 < 0.835$. From **Feature 1** and **2**, we can get that the migration process also will be terminated by C3 when

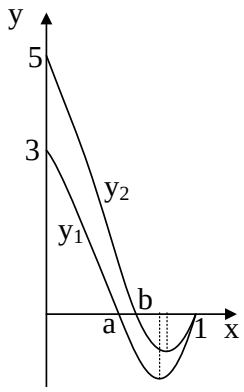


Figure 4.2: The rough diagram of Function (4.11) and (4.12).

$0.835 \leqslant \rho_3 < 1$, which only corresponds to a smaller number of iteration round. In summary, a migration process which will be terminated by C3 must satisfy: $\rho_3 > 0.75$.

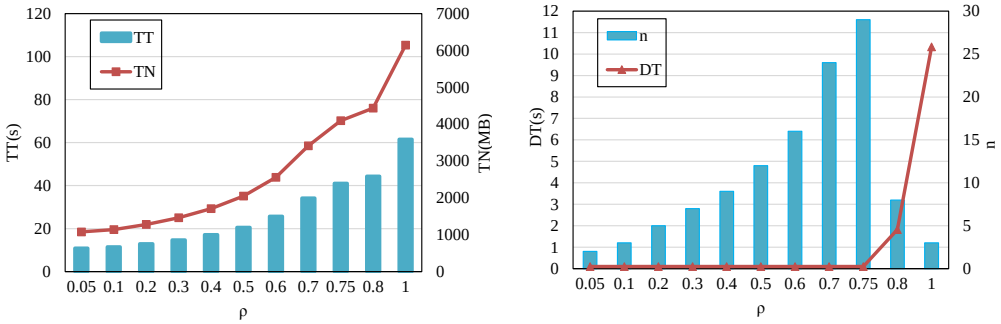
(3) For C2, the dirty bandwidth ratio ρ_2 should be somewhere between ρ_1 and ρ_3 according to **Feature 2**. Theoretically, $\sqrt[28]{R_{tc}/M}$ may be bigger than 0.75. Under this situation, the migration process will be terminated by both C1 and C3, and never by C2. In real case, R_{tc} will be a very small value in comparison with M, so $\sqrt[28]{R_{tc}/M}$ won't be bigger than 0.75.

Therefore, in conclusion, the relationship between which termination condition will be met by a migration process and the dirty bandwidth ratio is:

- When $0 \leqslant \rho < \sqrt[I_{tc}-1]{R_{tc}/M}$, the iteration phase will be terminated by C1;
- When $\sqrt[I_{tc}-1]{R_{tc}/M} \leqslant \rho \leqslant max\{0.75, \sqrt[I_{tc}-1]{R_{tc}/M}\}$, the iteration phase will be terminated by C2;
- When $\rho > 0.75$, the iteration phase will be terminated by C3.

Feature 3: *If an iteration phase is stopped by C1, the corresponding migration process can achieve the best migration performance in comparison with that stopped by C2 or C3.*

Obviously, when a migration process is terminated by C2, it will have the biggest number of iterations. According to **Feature 2** ($\rho_1 < \rho_2$) and the performance model of v_i, we can get that more data will be transferred in each iteration for C2 than C1. Therefore, a migration process terminated by C2 will result in a longer total migration time and a bigger network traffic than C1. Similarly, when a migration process is terminated by C3, it leads to the biggest network traffic, which in turn incurs a bigger total migration time that C1. For both C2 and C3, the remaining data of the stop-and-copy phase will be bigger than R_{tc} (otherwise, the iteration phase will be terminated by C1), so the downtimes of them are bigger than that of C1. ■



(a) Total migration time (TT) and total migration network traffic (TN).

(b) Total number of iterations (n) and downtime (DT).

Figure 4.3: The effects of ρ on migration performances.

To clearly show these three features, we illustrate the migration performances with different ρ in Figure 4.3. We only show the performances within [0,1]. Because when $\rho > 1$, it has the same performance as that when $\rho = 1$. The memory size allocated to the migrated VM is 1024MB. Hence, we can get: when $\rho \in [0, 0.736)$, the migration process will be terminated by C1; when $\rho \in [0.736, 0.75]$, the migration process will be terminated by C2; otherwise, it will be terminated by C3. From the figure, we can see that except the number of iteration round, all migration performances (total migration time, downtime and total network traffic) decrease with the increase of ρ. In other words, for a specific VM, it can achieve a relatively better migration performance when the iteration phase is terminated by C1.

4.2 Migration Control Algorithm

According to the analysis in Section 4.1, once the termination conditions are fixed, VM migration performances strongly depend on ρ. From the opposite perspective, we can control ρ to achieve the desired migration performance. In this section, a migration control algorithm is proposed to find a proper ρ for a migration process according to user's requirements and to solve the convergence issue of pre-copy, as shown in Algorithm 1.

Before migration, we can calculate the migration performances, called *default performances*, (total migration time T_0, Downtime D_0 and total network traffic V_0) by using the monitored parameters (such as the network bandwidth for migration). We only consider the situation that user' requirements of migration performances (denoted by T_u, D_u and V_u) are higher than the default performances (i.e., $T_u \leqslant T_0$, $D_u \leqslant D_0$, $V_u \leqslant V_0$). If the desired performances are worse than the default performances, the migration is started without adjustment. Otherwise, we check whether the migration process will be terminated by C2 or C3 or not. If $\rho_0 \geqslant \sqrt[I_{tc}-1]{R_{tc}/M}$, to make a migration process convergent, we firstly change ρ to a value smaller than and close to $\sqrt[I_{tc}-1]{R_{tc}/M}$,

Algorithm 1 Migration performance control algorithm

Calculate T_0,V_0,D_0;
if $T_0 \leqslant T_u$ and $V_0 \leqslant V_u$ and $D_0 \leqslant D_u$ **then**
    Return;
end if
if $\rho_0 \geqslant \sqrt[I_{tc}-1]{R_{tc}/M}$ **then**
    $\rho_1 = \rho_0 - \triangle$;
    $n = I_{tc} - 1$;
else
    $n = n_0$;
end if
Calculate ρ_t, ρ_v, ρ_d;
$\rho_f = min\{\rho_1, \rho_t, \rho_v, \rho_d\}$;
Return;

denoted by $\rho_1 = \rho_0 - \triangle$. Meanwhile, the total number of iteration n is set as the biggest available value by C1 (i.e., $I_{tc} - 1$) to lower the adjustment to ρ according to **Feature 1**. If a migration process will be terminated by C1, we do not adjust the number of iteration (i.e., $n = n_0$).

After n is fixed, we can gain the corresponding dirty bandwidth ratio (denoted by ρ_t, ρ_d and ρ_v) to meet user's requirements on each performance metric (T_u, D_u and V_u) by using Equation (4.1), (4.2) and (4.3), respectively. Then, according to **Feature 1**, the minimal one among the available values is selected as the final dirty bandwidth ratio ρ_f.

The desired value for ρ (i.e., ρ_f) can be achieved by three manners: only changing memory dirty rate, only changing migration bandwidth and changing both of them. The former two manners only introduce one variable (memory dirty rate or migration bandwidth), hence it is easy to calculate ρ_t, ρ_d and ρ_v by using Equation (4.1), (4.2) and (4.3), respectively. However, when a cloud provider plans to achieve the desired migration performances by tuning both of them, it is impossible to solve Equation (4.1), (4.2) and (4.3) because there are two variables in each equation. Under this situation, we can firstly tune one parameter (e.g., memory dirty rate) to a value and then calculate the other one (i.e., migration bandwidth), and vice versa.

4.3 Implementation

In this section, we will elaborate on how to adjust memory dirty rate and migration bandwidth and how to provide interfaces for users to set performance requirements.

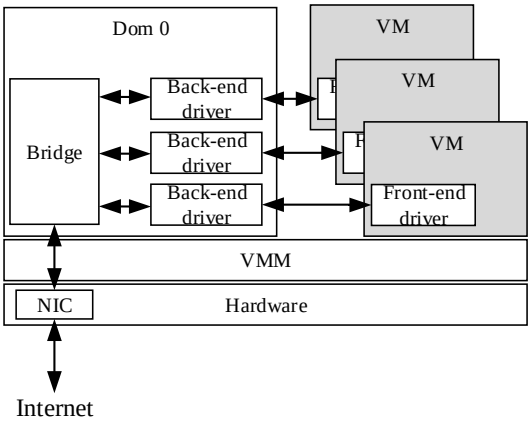


Figure 4.4: The network virtualization in Xen.

4.3.1 Memory Dirty Rate Adjustment

The original memory dirty rate of a VM before migration (i.e., d_0) can be got by using the average of a period of logged historical records or analyzing the regularities of the workloads running in the VM [2].

A VM's memory dirty rate shows a roughly linear relationship with CPU speed [81], so we can adjust memory dirty rate by changing the CPU time allocated to the migrated VM. Xen leverages a Credit scheduler to control the co-located VMs to use the CPU resources of the underlying host. Each VM on a host has a weight. How many CPU time can be gained by a VM is determined by its weight. A bigger weight means that the corresponding VM can get more CPU time. We take the same method in [81] to tune the migrated VM weight to get the desired dirty bandwidth ratio ρ_f.

When the CPU time assigned to the migrated VM decreases, the network bandwidth occupied by the applications may decrease correspondingly. It then results in more bandwidth available for the migration process. We do not consider this mutual influence in our algorithm, because it positively affects the final migration performances.

4.3.2 Migration Bandwidth Adjustment

There are two types of migration networks. One is that a dedicated link is built between the source and the destination hosts for VM migration. The other is that VM migration shares the same network link with the applications running in the migrated VM. For the former situation, it is easy to monitor and control the network bandwidth.

In this section, we mainly depict how to adjust network bandwidth for the latter situation. Xen implements network virtualization with two components: front-end driver and back-end driver, as shown in Figure 4.4. Each VM runs a front-end driver, and a corresponding back-end driver is running in the management domain (i.e., Domain

0). All back-end drivers connect to the NIC of the host through bridging, routing or NAT. With this structure, we can log the network utilization in the back-end driver, and use the average value N_a as the network bandwidth occupied by the applications running in the VM. The difference between the max available network bandwidth N_m for this VM and N_a can be regarded as the original bandwidth (i.e., $B_0 = N_m - N_a$) for migration. We change the overall network bandwidth N_m to get the desired migration performances. However, the applications in the migrated VM will compete for the network bandwidth with the migration process, so the network bandwidths used by the applications and the migration process do not follow the relationship ($N_a : (N_m - N_a)$) after we changed N_m.

To this end, we adjust the network bandwidth with two steps. In the first step, the overall network bandwidth is set as the sum of the bandwidth needed by the applications and the bandwidth needed by the migration process (denoted by B_u), i.e., $N_m{}' = N_a + B_u$. In the second step, VMM tunes the bandwidth to the value required by migration process with an iterative manner. The VMM monitors the real bandwidth B_r used by the migration process after the first step. If it is smaller than B_u, the VMM increases the overall network bandwidth by $B_u - B_r$. Otherwise, it decreases by $B_r - B_u$. This operation is repeated until the desired migration bandwidth has been achieved. To solve the network jitter problem, we assign more network bandwidth than the required to satisfy the desired migration performances. The margin can be set according to the variance of the historical bandwidth utilization of the applications running in the migrated VM.

4.3.3 User Requirement Setting

Users do not know and also do not want to know the values of related parameters for VM migration (e.g., memory dirty rate, available migration bandwidth, etc.), hence it is hard for them to provide reasonable migration performance requirements. For example, a user may think highly of total migration time. Because he/she does not know what the default migration performances are and how well the migration performance can be, it is impossible for him/she to set a proper value for the total migration time. To this regard, cloud platforms can provide a range for each performance metric and users make their decisions within these ranges, i.e., $T_u \in (T_s, T_0]$, $D_u \in (D_s, D_0]$ and $V_u \in (V_s, V_0]$. T_s, D_s and V_s are the best performances the cloud data center can achieve for total migration time, downtime and total migration traffic, respectively.

It is easy to calculate T_0, D_0 and V_0 with the performance models in Section 4.1 and the logged parameters. The smallest values for total migration time and total network traffic are the ones corresponding to non-live migration which directly suspends a VM on the source host and sends its memory data to the destination host, so we can get $T_s = \frac{M}{B_m}$ and $V_s = M$. B_m is the maximal bandwidth available for VM migration

Table 4.2: The information on the migrated VMs and user's performance requirements. Total migration time and downtime are calculated in seconds (*s*), and total network traffic is calculated in megabytes (*MB*). Each requirement follows the structure: (*total migration time, downtime, total network traffic, adjustment manner*). "M" and "B" denote to use memory dirty rate adjustment and migration bandwidth adjustment to achieve the desired performance, respectively.

Parameter \ VM	Web server	Compilation
d_0(Mbps)	15	53.6
(T_0,D_0,V_0)	(21.7,0.15,1347)	(61.8,16.48,5120)
(T_s,D_s,V_s)	(11.8,0.1,1024)	(11.8,0.1,1024)
R1	(13,0.15,1347,B)	(30,16,5120,M)
R2	(21.7,0.11,1347,B)	(60,1,5120,M)
R3	(21.7,0.15,1030,B)	(60,16,2048,M)
R4	(15,0.15,1200,B)	(40,3,3000,M)

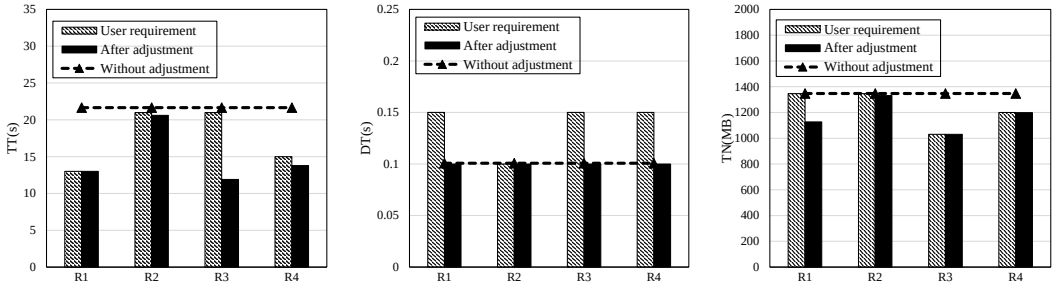
between the source host and the destination host. When the remaining data in the stop-and-copy phase is zero, it will get the smallest downtime. Under this situation, downtime refers to the time needed by VM suspend and resumption. Therefore, the lower threshold for downtime is $D_s = C$.

According to the adjustment methods for memory dirty rate and migration bandwidth in Section 4.3.1 and 4.3.2, lowering memory dirty rate will degrade the quality of the service running in the migrated VM and increasing migration bandwidth faces to assign more resources to the migrated VM. The former adjustment is painful for the users of the VM, and the latter makes cloud providers unpleasant and also may introduce interruptions to the co-located VMs. This problem can be solved from the commercial perspective. For example, when users choose to adjust network bandwidth to achieve the desired migration performance, there will be an additional payment with this operation, and it is free for the selection of memory dirty rate adjustment.

4.4 Evaluation

4.4.1 Experimental Setup

We test the migration control algorithm in an environment with three DELL Precision T1700 SFF workstations and a PC. Each workstation is equipped with one Intel 8-Core i7 3.6GHz processor, 16GB RAM and 2TB 7200rpm SATA device. The PC is with one Intel 8-Core i7 3.4GHz processor, 8GB RAM and 250GB disk space. The three workstations act as the source host, the destination host and the shared storage pool, respectively. The PC simulates the users of the migrated VM. To eliminate the interruption of other co-located VMs to the migration performance, we only run the migrated VM on the source and the destination hosts. All migrated VMs are configured with 1GB RAM and 1 VCPU. The default network bandwidth between the source and the destination hosts



(a) Total migration time (TT) (b) Downtime (DT) (c) Total network traffic (TN)

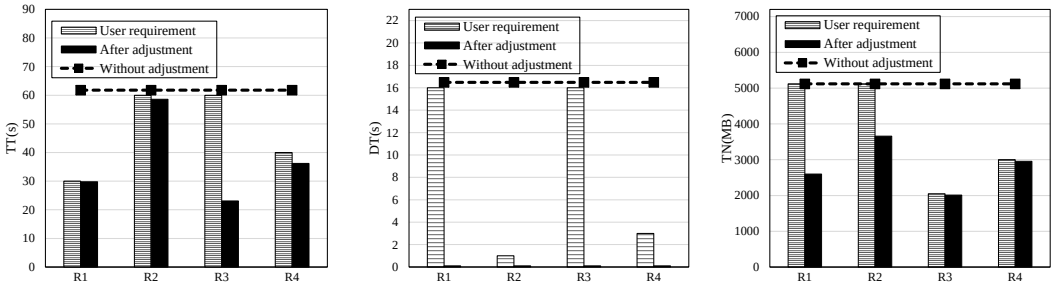
Figure 4.5: The migration performance control of a static web server VM by adjusting network bandwidth.

are configured with 500Mbps by using network interface shaping tool (*TC* [71]). The max available migration bandwidth is 1000Mbps. The hosts are installed with Ubuntu 16.04 LTS, and the Xen version is 4.5.0. We do not test the situation where a dedicated link is deployed for VM migration, because it is a simple version of the situation using shared network link.

We deploy two types of VMs to evaluate the migration control efficiency. One is running a static web server, and 100 clients are downloading a variety of 1MB files. The other one is installed with Ubuntu 14.04, and is compiling Linux kernel 4.13.11. The parameters and user's performance requirements for these two VMs are listed in Table 4.2. We define four types of requirements (denoted by R1, R2, R3 and R4). R1, R2 and R3 have a high requirement on total migration time, downtime and total network traffic, respectively, and R4 pays attentions to all of the three metrics. There is no big difference between the three adjustment manners, so we randomly use migration bandwidth adjustment and memory dirty rate adjustment for the static web server VM and the Linux kernel compilation VM, respectively. We firstly test the efficiency of our migration control algorithm (Section 4.4.2), and then evaluate the influence of migration performance control on the services running in the migrated VM (Section 4.4.3).

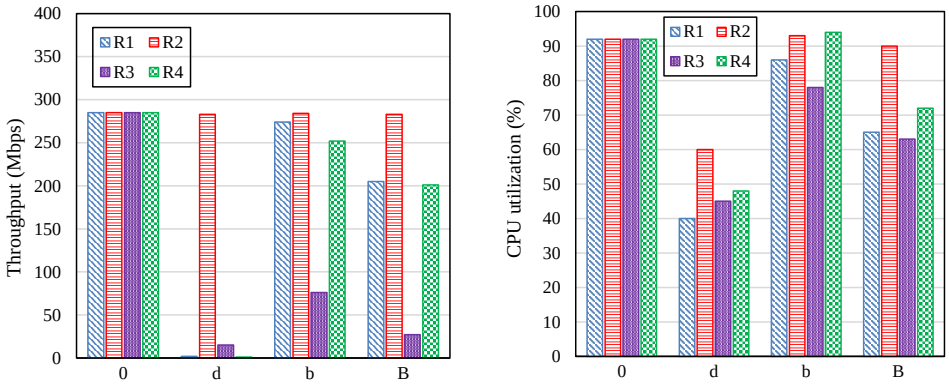
4.4.2 Efficiency of Migration Performance Control

The controls on migration performances for the static web server VM and the Linux kernel compilation VM are shown in Figure 4.5 and 4.6. We have the following observations: (1) Migration performances can be controlled to satisfy user's requirements. Due to the fluctuations of memory dirty rate and migration bandwidth, it is impossible to tune the performances to exactly identical with the desired performances. (2) Even though a user only cares about one performance metric (e.g., total migration time), other metrics will be automatically lowered as well because of a smaller ρ. (3) For the VM which already has a good migration performance, the improvement space is small since each migration has upper performance limits (i.e., T_s, D_s and V_s). For



(a) Total migration time (TT) (b) Downtime (DT) (c) Total network traffic (TN)

Figure 4.6: The migration performance control of a Linux kernel compilation VM by adjusting memory dirty rate.



(a) Static web server VM (b) Linux kernel compilation VM

Figure 4.7: Service degradation during migration performance control. “0”, “b”, “d” and “B” represent without adjustment, only migration bandwidth adjustment, only memory dirty rate adjustment and adjusting both of them, respectively.

example, the migration process for the static web server VM can converge and results in a small service downtime. Then the times for VM suspend and resumption become the bottleneck of downtime. (4) Our algorithm avoids the migration convergence problem of pre-copy. As shown in Figure 4.6(b), the migration for the Linux kernel compilation VM without adjustment cannot converge and has a big downtime (16.5s) which is unbearable for all kinds of network applications. After adjustment, all migration processes for this VM have a small downtime regardless of user’s performance requirements.

4.4.3 Service Degradation During Performance Control

In this section, we try to understand how migration performance control will influence the service running in the migrated VM. Because the static web server VM is running a network-intensive application and Linux kernel compilation is a compute-intensive application, we use the throughput observed by clients and the CPU utilization to reflect the QoS of these two VMs, respectively. All the three adjustment manners are evaluated with the four performance requirements.

From Figure 4.7, we can see that different performance requirements will lead to different extents of adjustments, which in turn results in different service degradations. As expected, lowering memory dirty rate results in a big service degradation for the both VMs. Specifically, the degradation with memory dirty rate adjustment is on average 74% and 48%, respectively. In particular, to satisfy R1 and R4, the migration of the web server VM almost is close to non-live migration since the VM is allocated very small CPU time. Therefore, cloud providers or users should take the workload features into consideration while using memory dirty rate adjustment to control migration performances.

To achieve these performance goals, migration bandwidth adjustment incurs a smaller service degradation in comparison with memory dirty rate adjustment. The average interruption is 22% and 5% for those two VMs, respectively. One of the reasons why the static web server VM has a bigger QoS decrease in comparison with the Linux kernel compilation VM is that it is running a network-intensive application and the migration process competes for the network bandwidth with it. For the requirements R2 and R4, the Linux kernel compilation VM even sees an increase of CPU utilization. This is because it is running a compute-intensive application and the network-accessing tasks of the application get benefits from the increased network bandwidth.

Additionally, we find that sometimes only changing memory dirty rate or migration bandwidth cannot meet user's requirements due to the hardware performance constraint. Under this situation, we have to adjust both memory dirty rate and network bandwidth. For example, in the experiments, even though we tune the network bandwidth to the max value for the migration of the static web server VM, we still cannot get the desired ρ for R4. Hence, we have to simultaneously lower the memory dirty rate of the VM.

Compared with only adjusting one parameter, changing both memory dirty rate and migration bandwidth is more gentle. It does not incur a significant decrease of QoS, and also does not introduce a big network burden to the source and the destination hosts. With this manner, the static web service and the Linux kernel compilation only see a degradation of 37% and 21% on average, respectively.

4.5 Chapter Summary

In this chapter, we thoroughly analyze the relationship between migration performances of pre-copy and its influence factors and find that ρ is an important factor for migration performance. Based on this study, we propose a migration performance control algorithm by adjusting the memory dirty rate of the migrated VM and the network bandwidth for migration. The experimental results validate the efficiency of the migration control algorithm. It not only can tune a migration process to meet

the desired performance, but also solves the convergence problem of pre-copy. The evaluation on the service degradation during performance control can guide users to choose a proper adjustment manner according to migration environment and the workloads in the migrated VM.

Chapter 5

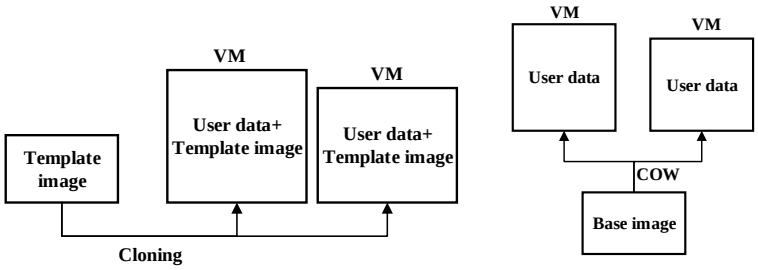
VM Migration over WAN—Storage Data Migration

Based on the review on storage data migration technologies in Section 3.2, in this chapter we target to solve the problems with VM migration over WAN (as presented in Section 1.1.2). We make two contributions to improve the performance of VM migration over WAN, precisely, improving the performance of storage data migration which is the bottleneck part. In the first contribution, to lower the side effects of data deduplication for migration performance improvement, we propose a three-layer image structure to physically separate the storage data of a VM into three layers. This structure also improves data sharing between VMs, which is beneficial to correlated VM migration. To achieve further performance improvement, a central base image repository is introduced for data centers in the second contribution. All data centers download base images from the central repository to deploy VMs. This makes data reuse become possible between data centers during VM storage data migration. Also, optimized data deduplication and Peer-to-Peer (P2P) file sharing are utilized to accelerate storage data migration speed when base images cannot be reused.

5.1 Three-layer Image Structure

5.1.1 VM Image Structure

Fast creating a VM with a basic running environment is an important task for Infrastructure as a Service (IaaS) providers. A running environment is a basic installation of an OS or a combination of an OS and a software stack. A software stack denotes a couple of applications which work together to provide a specific service. For example, MEAN (short for MongoDB, Express, AngularJS and Node.js) is a full javascript stack for web application development.



(a) One-layer image structure (b) Two-layer image structure

Figure 5.1: VM image structures.

At the beginning of cloud computing, cloud providers create VMs by cloning a template image. The user of a VM directly operates with a replica of the template image. All data from users are stored in the replica, as shown in Figure 5.1(a). We name it as *one-layer image structure* (without considering snapshots). However, the cloning operation is time-consuming. The deployment time span is unbearable when a large number of VMs are requested by a user. Furthermore, each VM will generate a copy of a template image, which is space-consuming as well.

To tackle the issues with one-layer image structure, a novel VM deployment approach—COW—was proposed. With the COW, a new empty image is created on the top of a template image (also called *base image*) during VM creation. The base image remains read-only to the upper image, and all new data and the modifications to the base image are redirected to the new image file, as shown in Figure 5.2. The storage data of a VM are split into two image files. We name it as *two-layer image structure*, as shown in Figure 5.1(b). This structure dramatically improves VM deployment speed because only a link between a new empty image and a base image is needed for each VM creation. Furthermore, a base image can be shared by multiple VMs, which lowers the total space consumption of a data center. With this structure, a software stack will be installed in either the base image layer or the upper layer. Installing software stacks in the base image layer leads to repeatedly storing OS data because many software stacks are running on the same OS. As shown in Figure 5.3(a), an OS must run with each software stack despite two software stacks are using the same OS. If a base image only is installed with an OS, the software stack will be installed in the upper layer by users. Repeated data store is also unavoidable when two VMs are running the same software stack, as shown in Figure 5.3(b).

The problems with one-layer and two-layer image structures are due to the fact that VM images are stored in a coarse granularity which leads to limited data sharing between VMs. The statistics from Cisco [28] on cloud application workloads show that compute and collaboration (such as conferencing, content management) are the two main workloads from enterprise users, while social networking and video/media

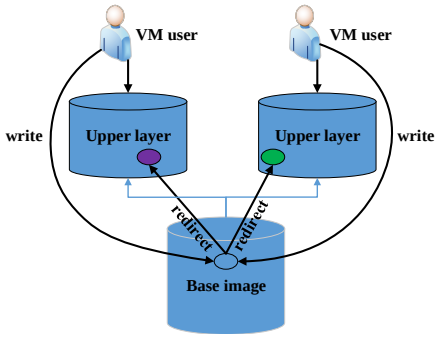
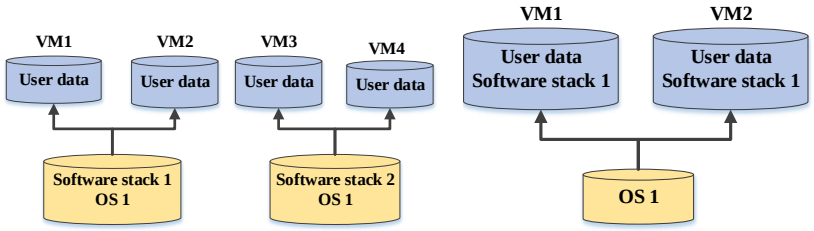


Figure 5.2: COW. The modifications from VMs to base images are redirected to the corresponding upper image files.



(a) (b)

Figure 5.3: Different software stack placements in two-layer image structure. (a)Software stacks are installed in the base image layer. (b)Software stacks are installed in the UD layer.

streaming are the biggest workloads from normal users. Also, another study [42] shows that 56.4% VMs in Amazon EC2 are running on Ubuntu. Actually, with the support of COW technology, storing VM images in a finer granularity to achieve a higher data sharing is possible. To solve the problem with one-layer and two-layer image structures, we propose a *three-layer image structure*. As shown in Figure 5.4, the storage data of a VM are physically separated into three layers: Operating System (OS) layer, Working Environment (WE) layer and User Data (UD) layer. With this structure, the OS layer only contains a basic installation of an OS. Each WE image is deployed with a software stack and runs on the top of an OS image. In other words, OS images run as the base images of WE images. A cloud provider prepares a variety of combinations of OS images and WE images as template images. New VMs are created on the top of WE images, i.e., WE images in turn run as the base images of UD images. Both OS images and WE images are read-only, and all new data from a VM are stored in the UD layer.

Three-layer image structure enhances data sharing between VMs. Furthermore, in most cases, the version of an OS is not critical for the software stack running on it. Therefore, from the perspective of engineering, cloud providers can share an OS image with WE images as much as possible to decrease the total space consumption of base images.

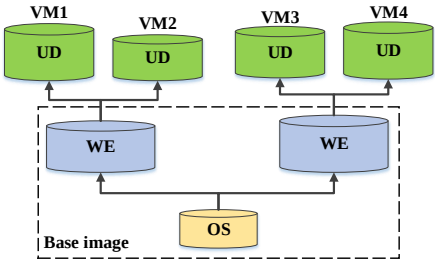


Figure 5.4: Three-layer VM image structure.

5.1.2 Space Consumption Analysis

In this section, we analyze the space consumption when a cloud data center uses different structures to store VM images, and the space benefits resulting from three-layer image structure. For a better discussion, we set aliases as follows.

- *1L*: One-layer image structure.
- *2O*: Two-layer image structure, and a base image only contains an OS, as shown in Figure 5.3(b).
- *2OA*: Two-layer image structure, and a base image contains an OS and a software stack, as shown in Figure 5.3(a).
- *3L*: Three-layer image structure.

The parameters used in this section are listed in Table 5.1. Meanwhile, we have the following assumptions:

- A data center only uses one image structure to store VM storage data.
- All users do not have specific requirements for OS distribution and version. It means that VMs can share an OS base image as much as possible.
- There is no upper limit on the number of VMs sharing a base image.

Ideal Situation

When VM images are deployed in layers, the modifications to base image(s) are stored in the top layer. Taking three-layer image structure as an example, the changes to the OS layer and the WE layer are stored in the UD layer, since base images are read-only. Therefore, layered image structure will lead to a bigger total image size for a VM, because both the original data of base images and new modifications to them are kept. The ideal situation is VMs do not modify base image data during their lifetime. Then for an individual VM, it has the same space consumption when it is stored in

Table 5.1: The notations on storing VM images in different structures.

Symbol	Description
S_1	The total storage space consumption of a data center when VM images are stored in *1L* structure.
S_{2O}	The total storage space consumption of a data center when VM images are stored in *2O* structure.
S_{2OA}	The total storage space consumption of a data center when VM images are stored in *2OA* structure.
S_{3L}	The total storage space consumption of a data center when VM images are stored in *3L* structure.
V_{1L}	The image size of a VM when it is stored in *1L* structure.
V_{2O_b}	The base image size of a VM when it is stored in *2O* structure.
V_{2O_u}	The UD size of a VM when it is stored in *2O* structure.
V_{2OA_b}	The base image size of a VM when it is stored in *2OA* structure.
V_{2OA_u}	The UD size of a VM when it is stored in *2OA* structure.
V_{3L_o}	The OS image size of a VM when it is stored in *3L* structure.
V_{3L_w}	The WE image size of a VM when it is stored in *3L* structure.
V_{3L_u}	The UD image size of a VM when it is stored in *3L* structure.
n	The total number of VMs in a data center.
k	The total type of software stacks in a data center.
R_o	The size of the OS portion when a VM is stored in *1L* structure.
R_w	The size of the software stack portion when a VM is stored in *1L* structure.
R_u	The size of the user data portion when a VM is stored in *1L* structure.

layered or non-layered structure. In this section, we will analyze the storage cost of a data center under this situation.

Due to without modification to base images, there are some basic relationships between the three image structures for a specific VM *i*, as shown in Equation (5.1), (5.2), (5.3) and (5.4).

$$V_{1L_i} = V_{2O_{b_i}} + V_{2O_{u_i}} = V_{2OA_{b_i}} + V_{2OA_{u_i}} = V_{3L_{o_i}} + V_{3L_{w_i}} + V_{3L_{u_i}} \tag{5.1}$$

$$V_{2OA_{b_i}} = V_{3L_{o_i}} + V_{3L_{w_i}} \tag{5.2}$$

$$V_{2O_{u_i}} = V_{3L_{w_i}} + V_{3L_{o_i}} \tag{5.3}$$

$$V_{2O_{b_i}} = V_{3L_{o_i}}, \quad V_{2OA_{u_i}} = V_{3L_{u_i}} \tag{5.4}$$

Table 5.2: The space differences between the *3L* structure and others.

	Difference	Value
ideal	$S_{1L} - S_{3L}$	$\sum_{i=1}^{n-1} V_{3L_{o_i}} + \sum_{i=k+1}^{n} V_{3L_{w_i}}$
	$S_{2O} - S_{3L}$	$\sum_{i=k+1}^{n} V_{3L_{w_i}}$
	$S_{2OA} - S_{3L}$	$\sum_{i=1}^{k-1} V_{3L_{o_i}}$
real	$S_{1L} - S_{3L}$	$\sum_{i=1}^{n-1} R_{o_i} + \sum_{i=k+1}^{n} R_{w_i} - \sum_{i=1}^{k} \beta_i R_{o_i} - \sum_{i=1}^{n} \alpha_i R_{w_i} - \sum_{i=1}^{n} \gamma_i R_{o_i}$
	$S_{2O} - S_{3L}$	$\sum_{i=k+1}^{n} R_{w_i} + \sum_{i=k+1}^{n} \beta_i R_{o_i} - \sum_{i=1}^{n} \alpha_i R_{w_i}$
	$S_{2OA} - S_{3L}$	$\sum_{i=1}^{k-1} R_{o_i} - \sum_{i=1}^{k} \beta_i R_{o_i}$

It is easy to calculate the space consumption of a data center when VM images are stored in *1L* structure, as shown in Equation (5.5).

$$S_{1L} = \sum_{i=1}^{n} V_{1L_i} \tag{5.5}$$

The space consumption for *2O* structure and *2OA* structure are calculated in Equation (5.6) and (5.7), respectively. With *2O* structure, only one OS is needed for all VMs. However, software stacks are stored n times even though some VMs are running with the same software stack. With *2OA* structure, the OS has to be saved for k times with different software stacks.

$$S_{2O} = V_{2O_b} + \sum_{i=1}^{n} V_{2O_{u_i}} \tag{5.6}$$

$$S_{2OA} = \sum_{i=1}^{k} V_{2OA_{b_i}} + \sum_{i=1}^{n} V_{2OA_{u_i}} \tag{5.7}$$

The space consumption of *3L* structure is shown in Equation (5.8). Only one OS is needed, and each software stack also is stored only once.

$$S_{3L} = V_{3L_o} + \sum_{i=1}^{k} V_{3L_{w_i}} + \sum_{i=1}^{n} V_{3L_{u_i}} \tag{5.8}$$

By using the relationships shown in Equation (5.1), (5.2), (5.3) and (5.4), the space differences between *3L* and others can be calculated, as shown in Table 5.2. Compared with *1L*, *3L* can save the space of $(n-1)$ OS images and $(n-k)$ WE images. It gains space benefits of $(n-k)$ WE images and $(k-1)$ OS images in comparison with *2O* structure and *2OA* structure, respectively. The space saved in ideal situation indicates the upper limit which we can achieve from *3L* structure.

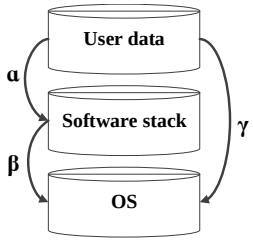


Figure 5.5: Modification percentages.

Real Situation

In real cases, when a VM image is stored in layers, the modifications from the top layer to base image(s) are inevitable. These new data will consume additional space. We use percentages α, β and γ to denote how many base image data are modified by upper image(s), as shown in Figure 5.5. For example, α of the WE image data are modified by users and stored in the UD image. In addition, layered image structure will lead to more meta-data because of reference management. They are very small, compared to the image size of a VM. We do not consider them in our analysis.

We use R_o, R_w and R_u to denote the size of the different parts of a VM when it is stored in *1L* structure, as shown in Table 5.1. Then the total image size of a VM i with *1L* structure is calculated in Equation (5.9).

$$V_{1L_i} = R_{o_i} + R_{w_i} + R_{u_i} \tag{5.9}$$

By using the modification percentages, the image sizes when the VM i is stored with *2O*, *2OA* and *3L* structures can be calculated according to R_o, R_w and R_u, as shown in Equation (5.10), (5.11) and (5.12).

$$\begin{cases} V_{2O_{b_i}} = R_{o_i} \\ V_{2O_{u_i}} = R_{w_i} + R_{u_i} + (\beta_i + \gamma_i)R_{o_i} \end{cases} \tag{5.10}$$

$$\begin{cases} V_{2OA_{b_i}} = R_{o_i} + R_{w_i} \\ V_{2OA_{u_i}} = R_{u_i} + \alpha_i R_{w_i} + \gamma_i R_{o_i} \end{cases} \tag{5.11}$$

$$\begin{cases} V_{3L_{O_i}} = R_{o_i} \\ V_{3L_{w_i}} = R_{w_i} + \beta_i R_{o_i} \\ V_{3L_{u_i}} = R_{u_i} + \alpha_i R_{w_i} + \gamma_i R_{o_i} \end{cases} \tag{5.12}$$

We substitute the corresponding parameter in Equation (5.5), (5.6), (5.7) and (5.8) for Equation (5.9), (5.10), (5.11) and (5.12). Then we can get the real space cost of a

data center when adopting different image structures in Equation (5.13), (5.14), (5.15) and (5.16).

$$S_{1L} = \sum_{i=1}^{n} R_{o_i} + \sum_{i=1}^{n} R_{w_i} + \sum_{i=1}^{n} R_{u_i} \tag{5.13}$$

$$S_{2O} = R_o + \sum_{i=1}^{n} R_{w_i} + \sum_{i=1}^{n} R_{u_i} + \sum_{i=1}^{n} (\beta_i + \gamma_i) R_{o_i} \tag{5.14}$$

$$S_{2OA} = \sum_{i=1}^{k} R_{o_i} + \sum_{i=1}^{k} R_{w_i} + \sum_{i=1}^{n} R_{u_i} + \sum_{i=1}^{n} \alpha_i R_{w_i} + \sum_{i=1}^{n} \gamma_i R_{o_i} \tag{5.15}$$

$$S_{3L} = R_o + \sum_{i=1}^{k} R_{w_i} + \sum_{i=1}^{k} \beta_i R_{o_i} + \sum_{i=1}^{n} R_{u_i} + \sum_{i=1}^{n} \alpha_i R_{w_i} + \sum_{i=1}^{n} \gamma_i R_{o_i} \tag{5.16}$$

The space benefits achieved from *3L* structure are shown in Table 5.2. They are smaller than the counterparts in the ideal situation because of storing the copies of modified base image blocks. Compared with *1L* structure and *2OA* structure, smaller modification to base images will lead to more spaces saved from *3L* structure. *3L* structure has a smaller space consumption in comparison with *2OA* structure when a bigger percentage of modification to the OS image and a smaller percentage of modification to the WE image are made.

5.1.3 The Trade-off of Data Deduplication

As discussed in Section 3.2.3, data deduplication is a widely used method for storage data migration performance improvement. However, data deduplication is a compute-intensive operation. An improper usage will lead to an even worse migration performance. For example, existing deduplication approaches [3, 138, 94] optimize VM migration by eliminating the duplicated blocks for the whole disk image of a VM. This is vulnerable because a portion of the image data of a VM has high similarity with other VM images due to using the same or similar libraries, such as OS, while other portions have rare identical blocks, such as user data. If the size of the user data portion of a VM disk image is huge, current data deduplication manners may even prolong rather than shorten VM migration time.

To understand the overheads and benefits of data deduplication to VM storage data migration, we formulate the migration process with data deduplication. The total migration time of the storage data of a VM after using data deduplication can be calculated as Equation (5.17). Herein, S is the total image size of the migrated VM, S_d is the duplicated block size, B is the available network bandwidth for migration, and C_d is the additional computation time resulting from data deduplication. From Function (5.17), we can see that data deduplication creates benefits to storage data migration only when the number of duplicated blocks between the migrated VM image

and the image(s) at the target site is big enough to compensate for the computation overhead.

$$T = \frac{S - S_d}{B} + C_d = \frac{S}{B} + (C_d - \frac{S_d}{B}) \tag{5.17}$$

Additionally, if we do not consider other network transmission data between the source server and the destination server, such as protocol overhead, and chunk VM image into fixed-size blocks, the network traffic n_i for each data block can be calculated as Equation (5.18). S_f is the size of a fingerprint and S_b is the block size. If a block is duplicated, only its fingerprint is transferred through the Internet, otherwise, firstly the fingerprint and then the block will be transferred. A *unique block* refers to a block without replica at both the source data center and the destination data center.

$$n_i = \begin{cases} S_f, & \text{duplicated block;} \\ S_f + S_b, & \text{unique block.} \end{cases} \tag{5.18}$$

From the total duplicated data size, we can get the duplication ratio $\mu = \frac{S_d}{S}$. Then, the real network traffic N_i for each block is calculated in Equation (5.19).

$$N_i = \mu S_f + (1 - \mu)(S_f + S_b) \tag{5.19}$$

The total network traffic of migrating an image file is gained in Equation (5.20). There, n is the total number of blocks of the image file.

$$N = nN_i = n(\mu S_f + (1 - \mu)(S_f + S_b)) \tag{5.20}$$

Data deduplication benefits VM migration on network traffic only when the total network traffic N is smaller than the total image size S, i.e. $N < S$. Then, we can derive an inequality (5.21) as follows.

$$\begin{aligned} n(\mu S_f + (1 - \mu)(S_f + S_b)) < S &\Rightarrow \mu S_f + (1 - \mu)(S_f + S_b) < \frac{S}{n} \leqslant S_b \\ &\Rightarrow S_f < \mu S_b, \quad (\mu S_b \approx \frac{S_d}{S} \cdot \frac{S}{n} = \frac{S_d}{n}) \\ &\Rightarrow nS_f < S_d \end{aligned} \tag{5.21}$$

Equation (5.21) indicates that data deduplication improves migration performances only when the total fingerprint size is smaller than the size of duplicated blocks. In other words, we can get more benefits when more duplicated blocks are found in an image file. When the duplicated blocks of an image file is small, data duplication will transfer more data than that without data deduplication. Also, data deduplication

will bring in computational overhead to the source and the destination hosts of VM migration.

5.1.4 Migration System

Three-layer image structure not only reduces space consumption, but also separates the storage data of a VM with different data features into different image files. According to previous studies [74, 82], there are many intra- and inter-image duplicated blocks between OS images. There are also many repeated blocks between WE images due to using the same libraries and applications (see the test in Section 5.1.6). However, UD images are tightly related to users. They are different from VM to VM, therefore, the similarity between them is small.

Based on this separation and data features, we design a migration system—LayerMover, to improve storage data migration performance by optimizing data deduplication. In order to decrease the side effects of data deduplication, according to the image similarity features in different layers and the analysis in Section 5.1.3, LayerMover only eliminates the duplicated blocks for OS images and WE images. UD images are migrated without data deduplication operation.

LayerMover firstly migrates VM storage data and then memory data (i.e., *pre-pre*, as discussed in Section 2.3.2). With this migration sequence, at least one site (the source site) has all the latest data of the migrated VM without the risk of splitting VM storage data into the source and the destination sites, which is robust against migration failure. Furthermore, migrating storage data before memory data avoids remotely-accessing disk blocks from the source data centers which will result in a big disk I/O latency or even destroying the VM, after the VM is resumed at the destination server.

Based on above analysis, LayerMover migrates a VM as follows.

- **Base image migration.** Before OS image and WE image are transferred to the destination data center, the blocks which are identical with these at the destination site are eliminated, i.e. data deduplication. The source host only transfers the blocks which are not located at the target site to reduce the data transferred over the Internet.

- **UD layer migration.** UD image is writable to users, so it is directly synchronized to the destination site. The three layers of the storage data of the migrated VM are migrated concurrently.

- **Memory data migration.** After finishing base image migration and the first round synchronization of UD image, LayerMover starts to migrate memory data in a pre-copy manner [29].

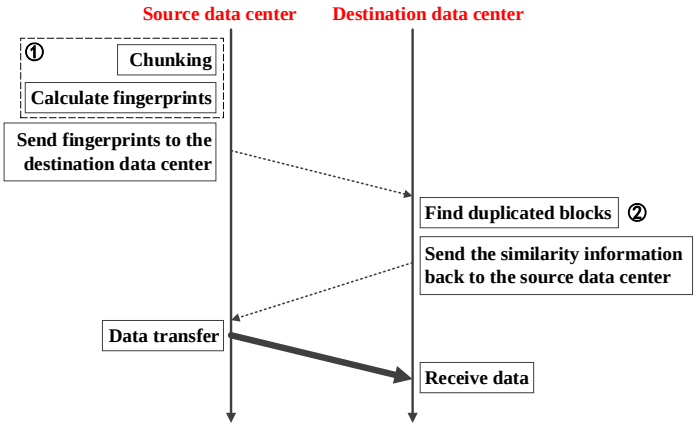


Figure 5.6: The general steps of data deduplication for VM storage data migration. Our optimizations mainly focus on parts ① chunking and fingerprint calculation and ② duplicated block elimination.

- **Handover.** The resources of this VM are released at the source site after it is resumed on the destination host.

As shown in Figure 5.6, data deduplication firstly chunks the images into blocks and then calculates a fingerprint for each block. The fingerprints are used to find the blocks which already are located at the destination data center. At last, the source data center only sends unique blocks to the destination data center. To reduce the side effects of data deduplication, we design optimizations for the chunking and fingerprint calculation step and the duplicated block elimination step in the following two sections, respectively.

Fingerprint Calculation

Since OS image and WE image are unmodified during their whole lifetime, we can change the chunking and fingerprint calculation step from an *inline* manner to an *off-line* manner, i.e. the fingerprints for all base images are pre-calculated and saved to avoid repetitive calculation operation. Then, during VM migration, the fingerprints of base images can be immediately sent to the destination site without the chunking and fingerprint-calculating operations.

To further improve the efficiency of the next step (duplicated block elimination), we design two two hashtable structures to store the fingerprints of base images: *individual hashtable* and *group hashtable*, as shown in Figure 5.7. The fingerprints of a base image are stored in an individual hashtable. At the same time, LayerMover puts the fingerprints of several similar images into a group hashtable. These two hashtable structures decrease the total blocks for comparison. An individual hashtable can eliminate the duplicated blocks located in the image, and a group hashtable eliminates these located between the images of this group.

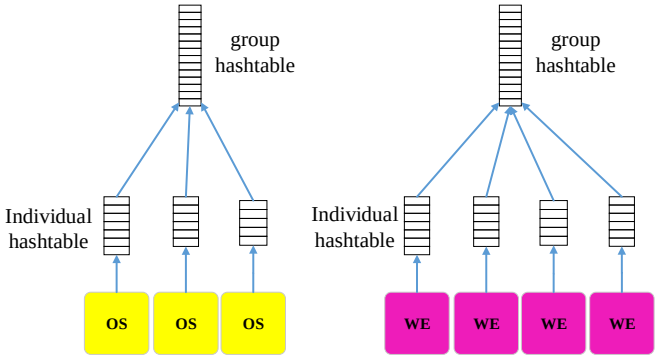


Figure 5.7: Hashtable structure of base images.

Duplicated Block Elimination

During data deduplication, the main work for the destination data center is to find duplicated blocks for the source image. This operation faces a trade-off between computational cost and the amount of duplicated block found by it. Comparing the source image with more images at the destination site will find more duplicated blocks, but the comparison time also is longer.

If the destination site uses a deduplication system for its storage system, it is easy to find the duplicated blocks for the source image during VM migration. This means to compare the source image with all the data at the destination site. Even though it can find all duplicated blocks, the comparison time also is long. Another implementation is to compare the source image with one image which is highly similar to the source image. The *highly similar* means this image contains the same or a close version of the OS or the same software stack as the source image. This reduces the comparison time with the risk of transferring duplicated blocks to the destination site.

With the support of these two hashtable structures, when LayerMover conducts data deduplication at the destination site, it takes the following rules to select image(s) for comparison:

- When there is an image at the destination site which is highly similar to the source image. The individual hashtable of this image is chosen for data deduplication. To be precise, for an OS image the highly similar image denotes it is running the same OS as the source OS image, and for a WE image it refers to an image is installed with the same software stack and running on the same type of OS image as the source WE image.
- Otherwise, a group hashtable which contains the most similar OS or applications to the source image will be selected for comparison.

As the hashtable contents are fixed once they are created, the similarity information between the base images in a data center can be preserved in advance. Furthermore,

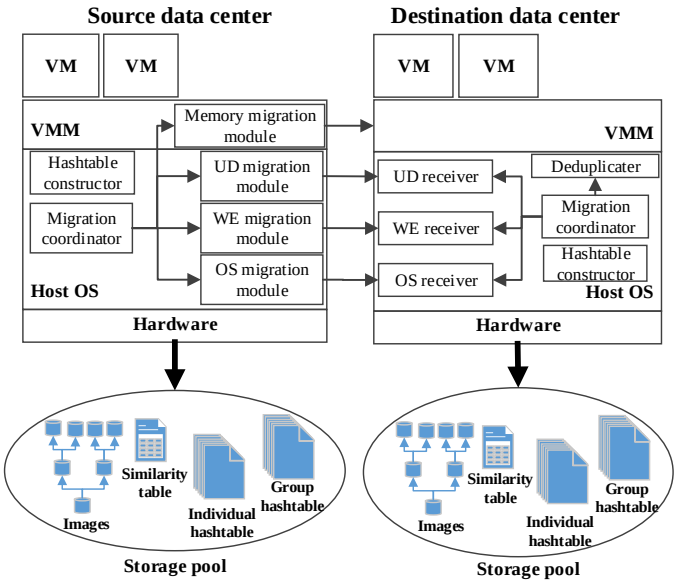


Figure 5.8: The structure of LayerMover.

after each VM migration, the similarity information between the source image and the image selected for comparison is saved as well. This information can guide next migration to choose a better hashtable at the destination site for data deduplication.

5.1.5 Implementation

To indicate the feasibility of our proposals, we implement a prototype based on KVM. LayerMover has no intrusive modification to VMM. Constructing three-layer image structure is an engineering task. It can be implemented by any VMM which supports COW technology. Because LayerMover uses data deduplication to optimize VM storage data migration, in order to achieve a high deduplication efficiency, it is better for both the source data center and the destination data center to utilize a storage pool (such as NAS, SAN) to store their own VM images. LayerMover composes of several modules which are distributed at the source data center and the destination data center, as shown in Figure 5.8.

Migration coordinator. VM migration starts from the migration coordinators. They control all migration modules according to the steps presented in Section 5.1.4.

Hashtable constructor. Individual hashtables and group hashtables for base images are created by the *hashtable constructor*. When a new base image is deployed, its individual hashtable is firstly built, and then its similarities with other hashtables are calculated and stored in the similarity table. Based on the similarity information, it joins in a group which has the highest similarity with it. At last, hashtable constructor updates the corresponding group hashtable.

Similarity table. It stores the information of inter- and intra-image similarities. In addition, after each VM migration, the similarities between the source image and the target image(s) also are added into the similarity tables of both the source and the destination data centers for potential future migration.

Storage data migration modules. The three layers of a VM image are transferred by three corresponding modules. OS migration module and WE migration module are same, and the destination server runs a receiver for each layer. Base images are constructed by collecting unique blocks and duplicated blocks at the destination data center. UD migration module employs DRBD [38] to asynchronously copy UD image data to the target site.

Memory migration module. The performance of memory data migration [29] has reached the ceiling after more than ten years' development. In this chapter, we do not design optimizations for memory data migration, therefore, LayerMover directly calls the memory migration module of KVM to transfer memory data.

Deduplicater. The comparison operation of data deduplication is conducted by a *deduplicater* running on the target host. It chooses an individual hashtable or a group hashtable for comparison according to the previously stored similarity information in the similarity table.

5.1.6 Evaluation

In this section, we firstly evaluate the performance of three-layer image structure, and then test the migration performance of LayerMover with different conditions.

I/O Performance of Three-layer Image Structure

The three-layer image structure increases the I/O path depth. We explore whether and how it affects VM I/O performance in this section. A VM image is deployed in *1L* structure, *2O* structure and *3L* structure, respectively. Only one of the two types of two-layer structures is tested because they should have the similar I/O performance. We use Iometer [72] to imitate three I/O access models: 100% random read, 100% random write and a mix of 33% random read and 67% random write (represents a typical database workload). Average I/O speed and response time are tested with different data sizes per I/O.

As shown in Figure 5.9, we can see that a bigger data size per I/O leads to a better I/O performance. Layered image structures almost have no influence on the read performance (see Figure 5.9(a)), regarding both access speed and response time. Compared with *1L* structure and *2O* structure, *3L* structure sees the lowest write speed and the longest response time, as shown in Figure 5.9(b). Precisely, its speeds are

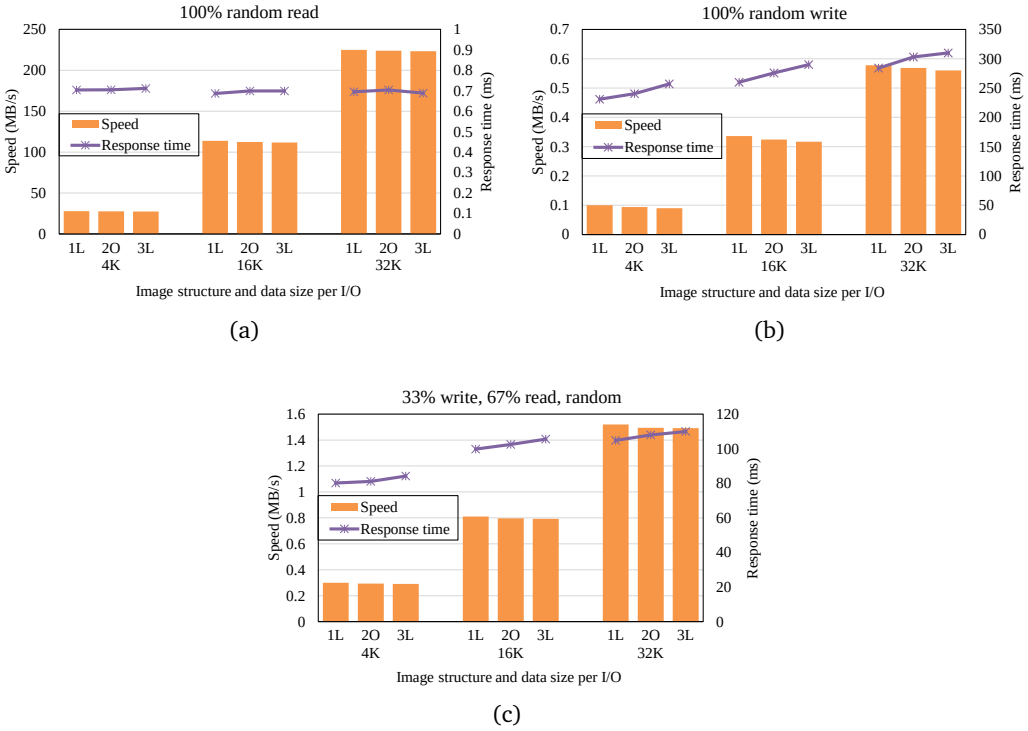


Figure 5.9: I/O performance with different image structures.

slower than *1L* structure and *2O* structure by 4% and 2%, 1.8% and 1.2%, and 1.4% and 0.8% for 4K, 16K, and 32K data size per I/O, respectively. The response time shows a similar trend. However, for the mixed I/O access model, *3L* structure only slightly lowers the I/O performance, as shown in Figure 5.9(c). In comparison with *2O* structure, it only drags down the I/O speed by 1%, 0.5% and 0.2% for 4K, 16K and 32K I/O request size, respectively. Overall, three-layer image structure will not incur a big performance degradation to VM I/O operations.

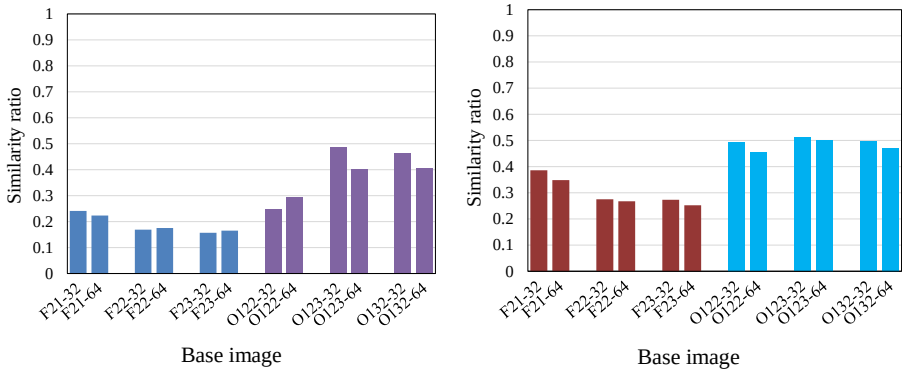
Similarity between WE Images

Jin et al. [82] and Bazarbayev et al. [10] have done experiments for OS image similarity. According to the results, there are many duplicated blocks between different OS images. The similarity ratio between two OS images can be more than 60%. In this section, we further evaluate the similarity feature between WE images.

We deploy two common cloud software stacks (LAMP and MEAN) on different versions of Fedora and openSUSE base images. LAMP is short for Linux, Apache, MySQL and PHP, and MEAN is a software bundle of MongoDB, Express, AngularJS and Node.js. The information of these WE images is shown in Table 5.3. We explore both individual similarity and group similarity according to the hashtable structure designed in Section 5.1.4.

Table 5.3: The size (in MB) of WE images.

Base Image	Kernel Version	Size	
		LAMP	MEAN
Fedora 21(32bit)	3.17.4	424	1,282
Fedora 21(64bit)	3.17.4	470	1,499
Fedora 22(32bit)	4.0.4	340	922
Fedora 22(64bit)	4.0.4	342	948
Fedora 23(32bit)	4.2.3	366	925
Fedora 23(64bit)	4.2.3	369	971
openSUSE 12.2(32bit)	3.4.6	189	776
openSUSE 12.2(64bit)	3.4.6	189	746
openSUSE 12.3(32bit)	3.7.10	235	717
openSUSE 12.3(64bit)	3.4.6	295	731
openSUSE 13.2(32bit)	3.4.6	309	929
openSUSE 13.2(32bit)	3.4.6	384	690

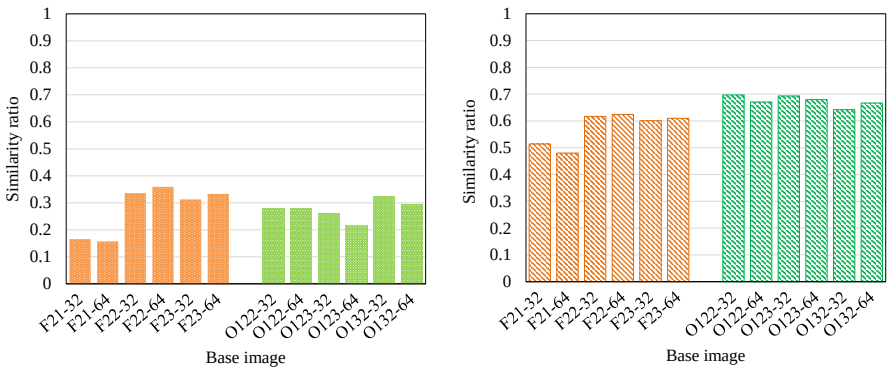


(a) LAMP (b) MEAN

Figure 5.10: Individual similarity between the WE images based on the same version of OS but for different architectures. Regarding the denotations on the x-axis, "F" is short for "Fedora", and "O" for "openSUSE". The first number is the version of the OS, and the second one is the CPU architecture it aims for. For example, "O122-32" denotes "openSUSE 12.2(32bit)". Each bar pair indicates a comparison between these two WE images.

Individual Similarity. Individual similarity means the ratio of duplicated blocks between two images. When there is a highly similar image located at the destination site, the source image will compare with it for data deduplication. In this section, we only test the similarity between the WE images which are based on the same OS version but for different CPU architectures, such as LAMP on Fedora 21 (32bit) vs. LAMP on Fedora 21 (64bit).

As shown in Figure 5.10, for each comparison, the two images will get different final similarity ratios, because they have different total image sizes and also will get a different amount of duplicated blocks. For example, there is one block *A* in one image, but five block *A*s in another. That is why each comparison has two bars in the graphs.



(a) LAMP (b) MEAN

Figure 5.11: Group similarity. The x-axis uses the same denotations as Figure 5.10.

We can see that many duplicated blocks exist in these WE image pairs. For both LAMP and MEAN images, these based on openSUSE show a higher similarity ratio than these on Fedora, but the similarity between the MEAN images on Fedora also can be around 40%, such as between the two MEAN images on Fedora 21(32bit) and Fedora 21(64bit). Additionally, for both types of WE images, the similarity ratio can reach approximately 50%.

Group Similarity. Group similarity is the duplicated blocks between one image and a group of images. As is discussed in Section 5.1.4, a group hashtable is chosen for data deduplication when no highly similar image exists at the target site. We take the images in Table 5.3 one by one as the source image, and put the rest of WE images which are based on the same OS distribution (except the one utilized for individual comparison) into a group as the counterpart for each comparison. For example, when the LAMP image on Fedora 21(32bit) is the source image, the target group contains the LAMP images on Fedora 22(32bit), Fedora 22(64bit), Fedora 23(32bit) and Fedora 23(64bit).

As shown in Figure 5.11, for LAMP images the source image also can discover more than 30% duplicated blocks from a group hashtable, such as the LAMP image on Fedora 22(32bit). However, all MEAN images find more duplicated blocks from group comparison than individual comparison. It indicates that even though no highly similar image at the destination data center with the source image, group comparison also can keep deduplication benefits at a relatively high or even better level.

Choosing images at the destination site for data deduplication is an important step, which is not our main focus of this paper. It is impractical to evaluate the similarities for all possible combinations. We only test one possibility for individual comparison and group comparison. It aims to show that VM storage data migration can get benefits from data deduplication for WE images.

Migration Performance

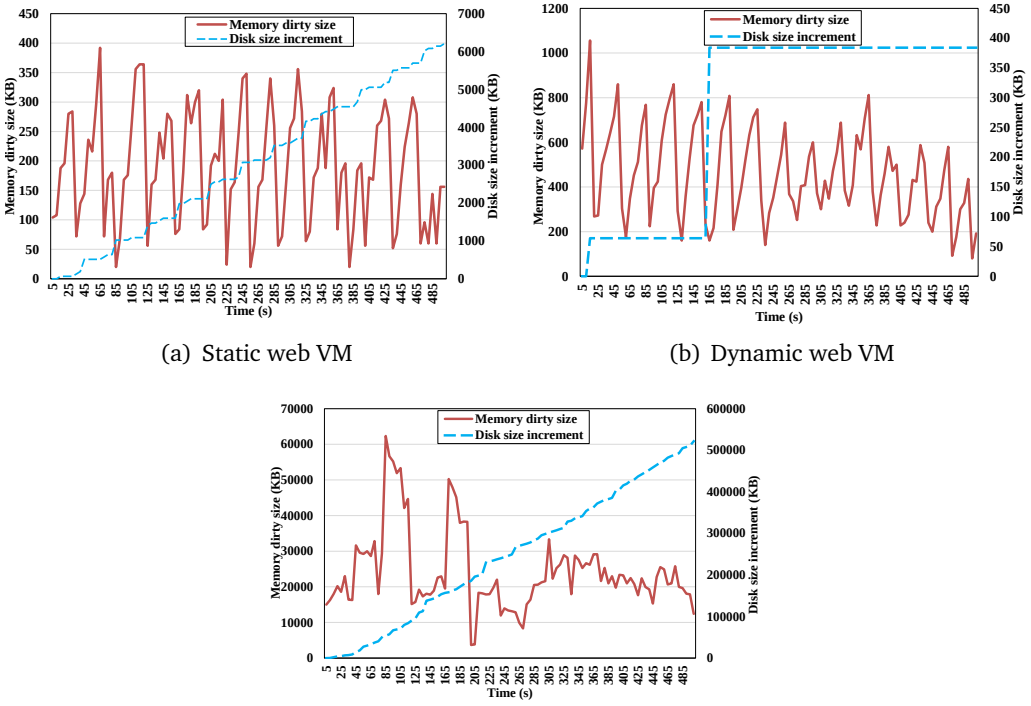
Experimental Setup. We evaluate the migration performance of LayerMover on two DELL Precision T1700 SFF workstations. Each is equipped with one Intel 8-Core i7 3.6GHz processor, 16GB RAM and 2TB 7200rpm SATA device. They are connected with 1Gbit/s HP ProCurve switch. We do not consider the network connection problem of VM migration for LayerMover, therefore, we evaluate the performances of LayerMover in LAN by simulating a WAN environment. The source server and the destination server are in the same subnet and the VM will keep its IP address after migration. The network performance is shaped as 400Mbit/s and 50ms round-trip latency by using Linux Traffic Control [71]. All migrated VMs are configured with 1 CPU and 1GB RAM. LayerMover is compared with the following migration approaches:

- ***1L***. VM images are deployed in *1L* structure. VM storage data are migrated without data deduplication. This runs as the baseline.
- ***2O***. VM images are deployed in *2O* structure. Data deduplication is conducted on the whole image data during VM migration. Current data deduplication methods do not take image structure into consideration, so we only test one type of two-layer image structures.

1L is implemented by directly calling the migration API of KVM, and ***2O*** adds data deduplication functionality on it. In order to thoroughly evaluate migration performances under different situations, we adapt the *Deduplicater* to generate different percentages of duplicated blocks at the destination site for the source image during migration. Data deduplication use 4KB fixed block chunking. The hashtable at the destination data center contains around ten million records (for around 50GB image data). The similarity conditions for the source base images are denoted as (a,b). a and b represent the similarity ratio for OS image and WE image of the migrated VM, respectively. We assume that the same amount of duplicated blocks can be found at the destination site for a specific VM regardless of how the VM image is deployed (in *1L* structure, *2O* structure or *3L* structure). We test the migration performances of LayerMover under two similarity conditions, (5%,5%) and (70%,70%).

Single Migration. Single migration refers to migrating one VM for each migration. The VM works independently without connection with other VMs. The following four type of VMs are deployed to measure the migration performance. The information on these four VMs is shown in Table 5.4.

- **Idle**. This VM is deployed for daily use and in idle status during migration.
- **Static web application**. The VM runs as a static web server. 100 concurrent users are downloading a 512KB file from it.



(a) Static web VM

(b) Dynamic web VM

(c) Linux kernel compile VM

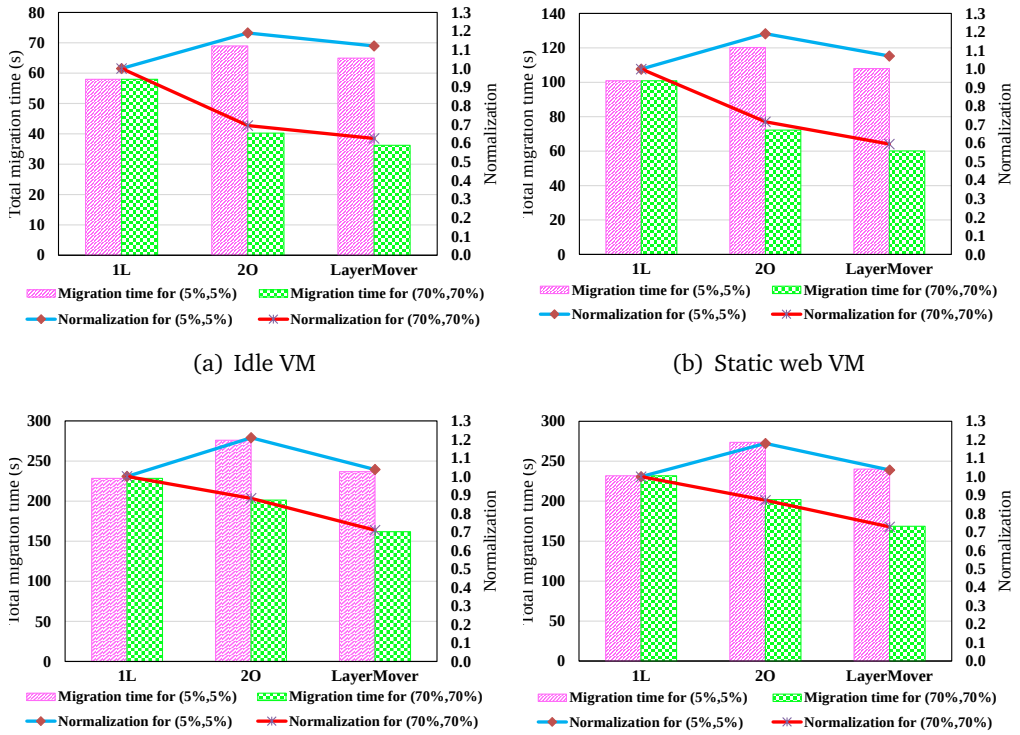
Figure 5.12: Memory and storage data dirty features of different applications.

Table 5.4: The image size (in MB) of different workloads with different image structures.

VM Type	Two-layer Structure			Three-layer Structure			
	OS	WE+UD	Total	OS	WE	UD	Total
Idle	1,340	158	1,498	1,340	145	24	1,509
Static	1,340	1,368	2,708	1,340	1,138	235	2,713
Dynamic	1,340	5,068	6,408	1,340	2,528	2,563	6,431
Compile	1,340	4,287	5,627	1,340	2,356	1,945	5,641

- **Dynamic web application**. A transactional web e-Commerce benchmark—TPC-W [168]—is running on this VM.
- **Linux kernel compile**. Linux kernel 4.8.1 is compiled in this VM. It is a compute and I/O intensive application.

Both UD image data and memory data are migrated in a pre-copy manner, therefore, dirtied data during migration will be retransferred. To better understand these migrated VMs, the dirty features of memory data and storage data are shown in Figure 5.12. We test the size of dirtied memory pages and the increase of disk size for a period of 500s with a 5s interval. The increment of disk size is used to approximately show disk dirty rate. The *idle* VM has almost no dirty data, and its results are abridged. There are regular points with a small memory dirty rate for the static web VM and the dynamic web VM, which is good for migration convergence. The Linux kernel compile VM has



(a) Idle VM

(b) Static web VM

(c) Dynamic web VM

(d) Linux kernel compile VM

Figure 5.13: Migration performances for different type of VMs under different similarity situations.

big memory and disk dirty rates, and the memory data also show a complicated dirty feature.

As shown in Figure 5.13, both *2O* and *LayerMover* result in longer migration times in comparison with *1L* when the similarity ratio is small, at (5%,5%). *2O* prolongs the total migration time by 19%, 19%, 21% and 18% for the four VMs, respectively. However, LayerMover only experiences 12%, 7%, 4% and 4% longer migration times for them than *1L* due to the optimizations for data deduplication. When the similarity ratio reaches (70%,70%), both of them reduce the total migration time, but *LayerMover* shows a better performance than *2O*. It achieves 7%, 8%, 17% and 14% of improvements for the four VMs, respectively, compared with *2O*. This further proves that LayerMover lowers the overhead of data deduplication. In other words, the deduplication for the UD layer of a VM decreases migration performance.

The last round of UD layer data synchronization will compete for the network bandwidth with the last iteration of memory data migration (i.e., VM handover). To evaluate the influence of the final stop-and-copy phase on the running services, the downtime of LayerMover is tested. As shown in Figure 5.14, all four VMs only experience less than one second downtime. It is invisible to almost all kinds of applications.

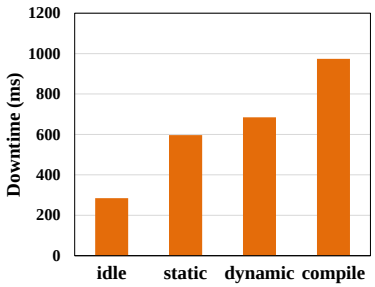
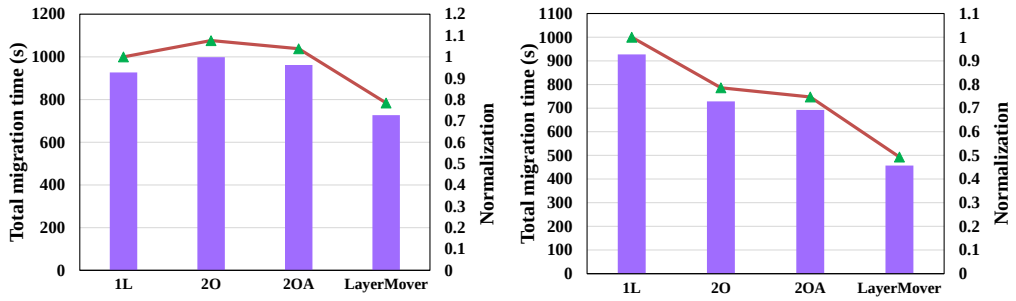


Figure 5.14: The downtime of LayerMover for different applications.

Table 5.5: The image size (in MB) of the VMs in a three-tier application with different image structures. The image sharing structures are also indicated in the table.

	Layer	Frontend	Web 1	Web 2	Database 1	Database 2
1L	-	2,195	2,224	3,229	7,398	8,876
	Total	23,922				
2O	Base	1,340				
	UD	934	2,018	2,025	6,129	7,630
	Total	20,076				
2OA	Base	2,061	2,907		3,350	
	UD	199	429	526	4,094	5,616
	Total	19,182				
3L	OS	1,340				
	WE	893	1,934		1,246	
	UD	68	109	98	4,856	6,415
	Total	16,948				



(a) Similarity ratio: (5%,5%) (b) Similarity ratio: (70%,70%)

Figure 5.15: Migration performance of multiple VMs under different similarity conditions.

Correlated VM Migration. Three-layer image structure increases the data sharing between VMs, LayerMover, therefore, should have better performance while migrating multiple correlated VMs which share base images. To investigate this performance of LayerMover, a typical three-tier application is deployed. It contains a frontend VM, two web VMs and two database VMs. They are deployed in *1L* structure, *2O* structure, *2OA* structure and *3L* structure, and share base images as much as possible. In this section, we also consider the *2OA* structure, because it has a different data sharing model as the *2O* structure, which will lead to different migration performances. The image structures

and sizes are listed in Table 5.5. From this table, we can see that *3L* has the smallest total image size.

We can get from Figure 5.15 that when the similarity ratio is (5%,5%), both *2O* and *2OA* get a longer migration time than *1L*, because the deduplication benefits are smaller than its computational cost. However, at this situation, LayerMover still can benefit the migration due to significantly reducing the total image size of the migrated VMs and a smaller deduplication cost. It is faster than *1L*, *2O* and *2OA* by 22%, 29% and 25%, respectively. When the similarity ratio is (70%,70%), data deduplication makes contributions to all migrations. LayerMover sees a shortest total migration time and improves migration performance by 51%, 27% and 24% in comparison with *1L*, *2O* and *2OA*, respectively.

5.2 Central Base Image Repository

5.2.1 System Design

The base images of a VM are read-only during their whole lifetime with three-layer image structure, so we further advocate an architectural rethink on storing VM storage data and propose a novel migration system—*CBase*—to improve VM storage data migration performance. A central repository is introduced to deploy and store base images (OS and WE images) based on the three-layer image structure, as shown in Figure 5.16. Data centers directly download base images from the repository to its local storage system rather than deploy them locally during VM creation. Because base images can be shared between VMs, each cloud data center only needs to download the needed base images once from the central repository. This not only reduces the workload of deploying base images from n (the number of data centers) to 1 (only deploy once in the central repository), but also improves base image reusability between data centers. The reusability denotes that if the base images of a VM are present at the target data center during VM migration, they can be directly used and do not need to be transferred from the source data center. We assume that data centers and the central repository use the same type of VMM and there is no compatibility issue. In CBase, the roles involved in cloud computing and the service provisioning procedure are correspondingly changed.

Cloud Roles

Currently, IaaS generally involves three roles [75]: *cloud provider*, *cloud user* and *end user*, as shown in Figure 5.17(a). A cloud provider maintains a set of hardware resources and prepares some base images for fast VM deployment. A cloud user leases resources from a cloud provider to host applications which are then offered to end users. End users are at the bottom of the service chain. They generate requirements and

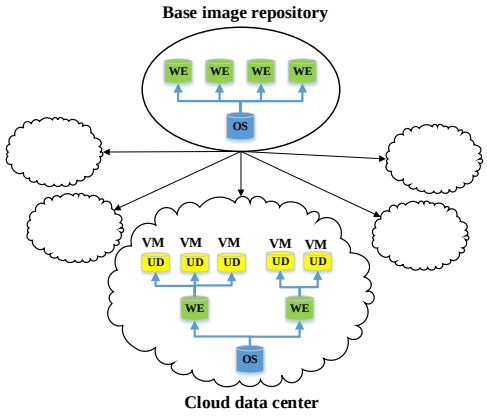


Figure 5.16: The architecture of CBase.

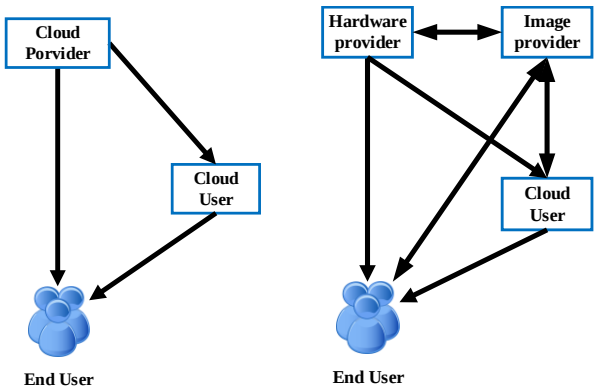


Figure 5.17: Cloud computing roles. (a) Current roles in cloud computing. (b) Roles in CBase. The double-arrow lines in (b) indicate the functionality of uploading base images to the central repository.

workloads and get services either directly from a cloud provider or a cloud user. We call cloud user and end user as user for short in the remaining part of this dissertation.

In CBase the responsibilities of a cloud provider are split into two parts: hardware maintenance and base image preparation. Now, cloud providers are only in charge of hardware maintenance, and the task of base image preparation is moved to the central repository. We name these two roles as *hardware provider* and *image provider*, respectively, as shown in Figure 5.17(b).

Furthermore, the central repository can act as an image-sharing platform as well. Base images are not only deployed by the image provider, but also can be uploaded from the other three roles according to the rules set by the image provider (such as image format, image description, etc.). This feature is valuable for academic researchers. They can quickly reproduce an experimental environment by directly downloading the images shared by others. Each image uploader can determine the access model of

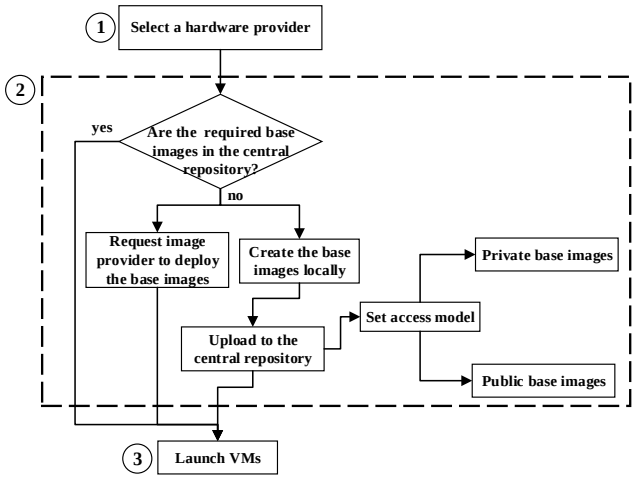


Figure 5.18: Service provisioning procedure of CBase.

the images he/she uploads: public or private. A public-access image can be used by everyone, while a private-access image is only available for some nominated users.

Service Provisioning Procedure

Correspondingly, the service provisioning procedure of cloud computing changes with the new structure of CBase. A user gets services through the following three steps: ① *select a hardware provider*, ② *prepare base images* and ③ *launch VMs*, as shown in Figure 5.18. The detailed procedure is presented as follows.

① **Select a hardware provider**. Users choose a hardware provider according to their requirements and reserve the required hardware resources at the selected data center.

② **Prepare base images**. Users check whether the base images they need exist in the central repository. If not, users can create the necessary base images locally and upload to the central repository, or pay the image provider to deploy them.

③ **Launch VMs**. Users offer the information of the required base images to the selected hardware provider. The hardware provider downloads the base images which are not present in its local storage system from the central repository and launches VMs on them.

5.2.2 Storage Data Migration

In CBase the base images of the migrated VM have replicas in the central repository and may also be used at the destination data center, which provides new optimization options for storage data migration. If the base images exist at the target site, they can be used directly without the necessity of transferring them from the source site, a

situation called *base image reuse*. For the three-layer image structure, there are two types of reuse: OS reuse and WE reuse. We assume that the OS image also is located at the destination site when WE reuse is possible since the OS image is the backing file of the WE image. If base images cannot be reused at the destination data center, two migration mechanisms are designed for CBase to accelerate the transfer speed of base images by using data deduplication and P2P file sharing technology, respectively. Additionally, we also devise a strategy for migrating UD image and an orchestration algorithm for coordinating the migration processes of the central repository and the source site to improve the robustness of CBase.

Base Image Migration with Data Deduplication

Same as *LayerMover*, CBase also only make data deduplication for base images. With the central repository, we further design a mechanism to lower the overheads of data deduplication besides the optimizations of *LayerMover*. In CBase, base images are migrated from the central repository rather than from the source data center. With this manner, it is unnecessary to transfer the hash table of the migrated image during VM migration because the central repository has all of the information. Then, Equation (5.19) and (5.20) are correspondingly changed into Equation (5.22) and (5.23). Because $1-\mu < 1$, there must be $N' < S$, which indicates that our optimizations eliminate the risk of increasing the total network traffic no matter how many duplicated blocks are found during migration.

$$N_i' = (1-\mu)S_b \tag{5.22}$$

$$N' = nN_i' = (1-\mu)nS_b = (1-\mu)S \tag{5.23}$$

Based on above discussion, base images are migrated by data deduplication as follows. The migration processes of OS and WE images are identical, so we take the migration of an OS image as the example. We assume that the Universally Unique Identifier (UUID) of the migrated OS image is X. The destination data center has a list $\langle X_1, X_2, ..., X_k \rangle$ which is the UUIDs of all the OS images in its local storage system.

1. The source VMM sends X to the destination VMM.

2. If there is $X \in \langle X_1, X_2, ..., X_k \rangle$, the destination VMM informs the source VMM that the OS image is ready (i.e. OS reuse). Otherwise, it sends X and $\langle X_1, X_2, ..., X_k \rangle$ to the central repository.

3. The central repository chooses the base image $X_i \in \langle X_1, X_2, ..., X_k \rangle$ which has the highest similarity ratio with X according to the similarity table, and sends the duplicated block ID list to the destination VMM.

4. Then destination VMM constructs X by receiving the unique blocks from the central repository and copying the duplicated blocks from the local image X_i.

Base Image Migration with P2P File Sharing

As depicted before, base images are not only stored in the central repository, but also have replicas in different data centers, which provides another migration option for base images—P2P file sharing. The destination data center can get base image blocks simultaneously from the central repository and other peer data centers.

Firstly, we elaborate on some basic concepts about P2P file sharing. We take BitTorrent as an example, which is the most popular P2P protocol [129]. With BitTorrent, files are cut into chunks which are the communication unit between peers. All metadata of a file is saved in a *.torrent* file. This file is distributed to many *.torrent* file servers. Each *.torrent* file contains the information of a special server, i.e. a *tracker*, where the uploader, all downloaders and seeds of this file are registered. Each node in a P2P file sharing system is both a downloader and an uploader. When a node has downloaded a file and is willing to upload some chunks of this file to other peers, this node is called a seed of this file. When a user wants to download a file, he/she firstly gets the *.torrent* file of the requested file from a *.torrent* file server. From the *.torrent* file, the user is directed to the tracker where the seeds which the user can connect to are gained. Then the user assembles the file by receiving chunks from these seeds. P2P file sharing can be also run in a trackless model with a DHT method, but it is accompanied by a slow seed-discovering speed.

In CBase, the central repository is an ideal place to run as a *.torrent* file server and a tracker because it has all the metadata of base images and the information of the data centers connecting with it. In particular, with this migration manner, when a data center deletes a base image from its local storage system, it should inform this operation to the tracker (i.e. the central repository) to update the seed list of this base image.

A basic characteristic of P2P file sharing is: the more seeds a file has, the faster the downloading speed is. However, downloading is delightful, while uploading is painful. Therefore, P2P file sharing faces another issue–free riding [1], which means that a user only downloads files without uploading chunks for others. There are many studies to solve this problem [1, 46]. We design an example incentive mechanism for CBase regarding this issue. The central repository can periodically count the data size of downloading operation D and uploading operation U for each data center, and then punishes or rewards each data center based on the value of $D - U$. If $D_i - U_i > 0$, the data center i should pay money in proportion to the value of $D_i - U_i$; otherwise, it gets money. To reduce the interference of P2P file sharing to the migration of UD image and memory data, the source data center will be eliminated from the seed list. The destination data center downloads base image chunks from other peer data centers

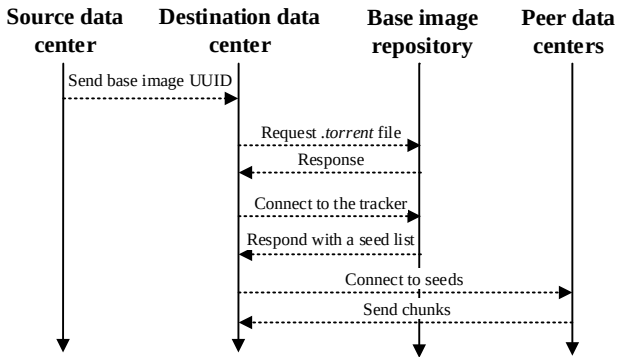


Figure 5.19: Migrating base images with P2P file sharing.

except for the source data center. Additionally, to take security and privacy issues into consideration, it is better to apply P2P file sharing for base images in a private community, such as in the scenario of cloud federation.

In sum, CBase migrates base images by using P2P file sharing with the following steps, as shown in Figure 5.19.

1. The source VMM sends X to the destination VMM.
2. If there is $X \in \langle X_1, X_2, ..., X_k \rangle$, the destination VMM informs the source data center that the OS image is ready. Otherwise, it sends the request for X to the base image repository.
3. The base image repository responds with the *.torrent* file of X to the destination VMM.
4. After opening the *.torrent* file, the destination VMM connects to the tracker, which also is the base image repository.
5. The base image repository replies to the destination VMM with a list of data centers $\langle DC_1, DC_2, ..., DC_n \rangle$ which have the replicas of X.
6. The destination VMM connects to these data centers except for the source data center and downloads chunks from them to assemble X.

UD Layer Migration

The UD layer of the migrated VM is writable to users and is constantly changing during the whole migration process. CBase migrates this layer in a pre-copy manner, similar to that for memory data migration. We firstly migrate the whole UD image data to the destination site. During this period, the dirtied and newly-written blocks are logged in a bitmap. The next round of transmission migrates these blocks logged in the bitmap. The operation loops until the VM is suspended at the source server. Then all remaining dirtied blocks are copied to the destination site to resume the VM. This

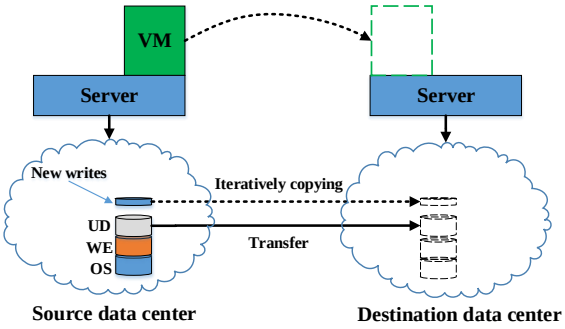


Figure 5.20: UD layer migration.

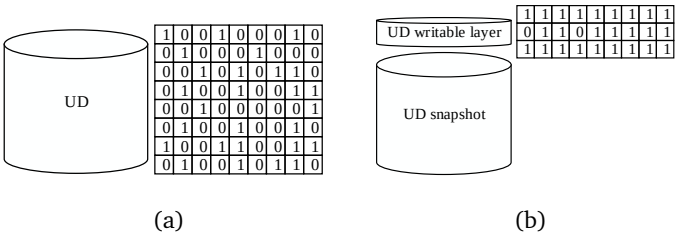


Figure 5.21: The comparison between two types of bitmaps for a UD image. (a) A bitmap for the whole UD image, (b) A bitmap only for the writable layer of a UD image.

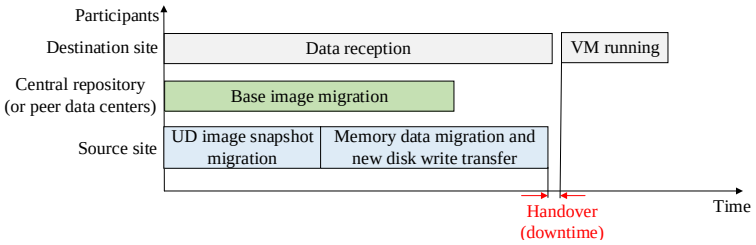
migration can efficiently take advantage of I/O spatial locality. In other words, because some blocks are frequently dirtied, this approach allows to only transfer the latest write of a block rather than transfer every write, i.e. enables write coalescing.

To lower the workload of logging dirtied blocks and scanning the bitmap in each round, snapshotting technology is used to condense newly-dirtied blocks together, as shown in Figure 5.21. When the UD image migration is initialized, an external snapshot is taken for the UD image. This will save all original UD image data into a snapshot, and creates an empty new image file to hold all new data written from the VM. We only need to build a bitmap for the new image file because the UD snapshot is read-only. The snapshot is migrated in the first round of transfer, and then the new image file is migrated in a pre-copy manner with the support of the bitmap.

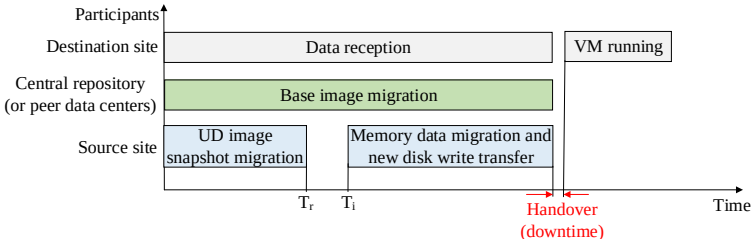
Migration Orchestration

In CBase, base images are migrated from the central repository or other peer data centers, the UD image and memory data are migrated from the source data center, as shown in Figure 5.22. To guarantee the pre-pre migration manner, these two migration processes are initialized as follows:

- The source data center firstly migrates the UD snapshot, and then migrates memory data and the writable layer of the UD image.



(a)



(b)

Figure 5.22: The orchestration of different migration processes. (a) $T_b < T_u + T_m$, (b) $T_b > T_u + T_m$.

- The destination data center initializes the preparation of base images simultaneously with the migration process of the source data center.

The two migration processes may finish with different durations. We denote the time of UD image snapshot migration as T_u, the time of memory data migration and the transfer of new disk writes as T_m, and the time of base image migration as T_b. All of them are predictable by monitoring the critical parameters [2], such as available migration bandwidth, memory dirty rate, disk dirty rate, etc. When base images can be reused at the destination site, $T_b \approx 0$. There are two possible relationships between the two migration processes. (1) $T_b < T_u + T_m$, i.e. the destination site prepares base images before the VM is handed over to it, as shown in Figure 5.22(a). (2) $T_b > T_u + T_m$, as shown in Figure 5.22(b).

Under the first situation, everything will go well as we expected (migrating storage data before memory data). However, for the second situation, if memory data migration is started at T_r in the Figure 5.22(a), it will lead to the problem of remotely-accessing base image blocks. To this end, CBase initializes memory data migration and new disk write transfer at T_i to make base image and memory data migrations finish at the approximate time. From Figure 5.22(b), we can get that $T_i = T_u + (T_b - T_m)$.

5.2.3 Evaluation

In this section, the performance of CBase is evaluated under a variety of conditions and compared with existing related migration mechanisms. There are many metrics used for assessing a migration strategy, such as total migration time, total network

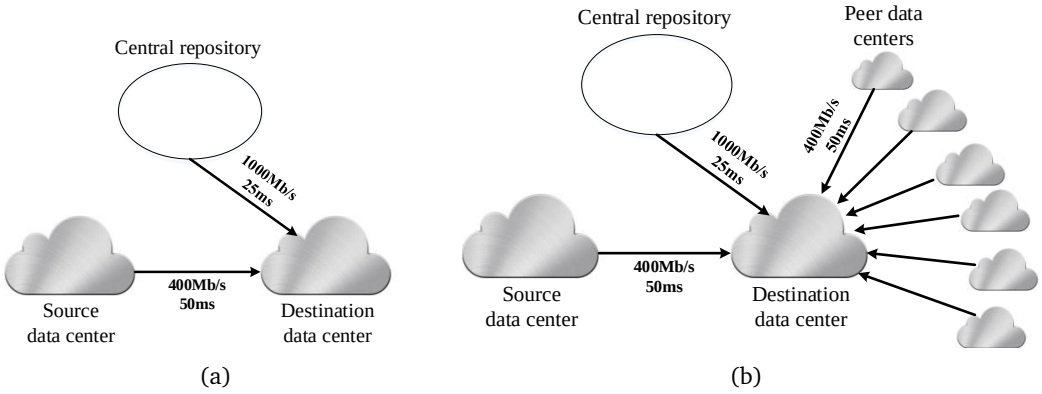


Figure 5.23: The experimental testbeds. (a) The testbed for the migration with data deduplication, (b) the testbed for the migration with P2P file sharing.

traffic, downtime, service degradation level, etc. For storage data migration, total migration time and total network traffic are the two main performance metrics. Since our optimizations are focusing on storage data migration, we will evaluate CBase from these two aspects.

Experimental Setup

We evaluate the performance of CBase in an environment with three DELL Precision T1700 SFF workstations and 50 virtual servers. Each workstation is equipped with one Intel 8-core i7 3.6GHz processor, 16GB RAM and 2TB 7200rpm SATA device. One of them is running as the central repository, and the other two are virtualized and act as the source and the destination data centers, respectively. The 50 virtual servers are the peer nodes to test the migration performance of P2P file sharing. As shown in Figure 5.23, the network performance between data centers is shaped as 400Mb/s bandwidth and 50ms latency. Because each data center gets base images from the central repository, we assume that there is a high-speed link between the central repository and the destination data center. The performance is set as 1000Mb/s bandwidth and 25ms latency. All migrated VMs in our test are configured with 1 CPU and 1GB RAM. To eliminate the interference of other co-located VMs on migration performance, the source host and the destination host only run the migrated VM(s).

All tests are conducted 10 times, and the average is set as the final result. Because the one-layer image structure (storing the storage data of a VM in a single image file) is rarely used by cloud provider at present, we evaluate our proposals by comparing with the following migration mechanisms.

- **NN2.** No data deduplication operation is conducted on VM storage data during migration. It indicates that the whole virtual disk of the migrated VM will be transferred to the destination site. There is no central repository for base images.

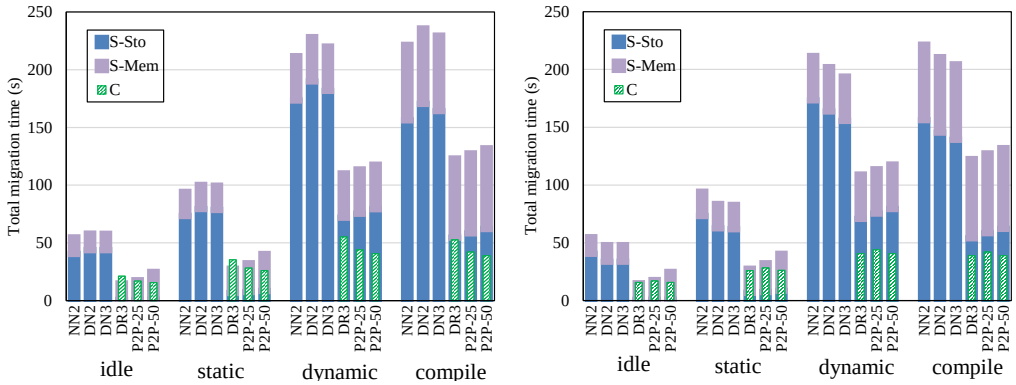
Different data centers deploy and use their own base images. VM images are stored in the two-layer structure. This is the baseline of our experiment.

- **DN2.** Data deduplication is conducted on the whole storage data of a VM during migration. There is no central repository for base images, and VM images are stored in the two-layer structure.
- **DN3.** The migration method of LayerMover. Data deduplication is only conducted on base images during VM migration. The fingerprints and similarities of base images are pre-treated as discussed in Section 5.2.2. There is no central repository for base images, and VM images are stored in the three-layer structure.
- **DR3.** The system designed in this chapter, and base images are migrated with optimized data deduplication.
- **P2P-25.** The system designed in this chapter, and base images are migrated with P2P file sharing. 25 peer nodes (seeds) exist in the P2P file sharing system.
- **P2P-50.** The system designed in this chapter, and base images are migrated with P2P file sharing. 50 peer nodes (seeds) exist in the P2P file sharing system.

Data deduplication uses 4KB fixed block size [120, 52]. We assume that for a specific VM it can find the same amount of duplicated data blocks at the destination site no matter how its image is stored (in two-layer structure or three-layer structure). The similarity situation between the migrated VM image and the images at the destination site is denoted as (a,b). a and b indicate the duplication ratios of the OS image and the WE image of the migrated VM, respectively, when the image of the migrated VM is stored in three layers. We assume that there are 1% duplicated blocks located in the UD portion of the migrated VM image for NN2 and DN2. We generate different amount of duplicated blocks at the destination site to simulate different image similarity conditions. For P2P file sharing, the destination site is the only one node which is downloading, which denotes that it does not need to upload blocks to other peer data centers.

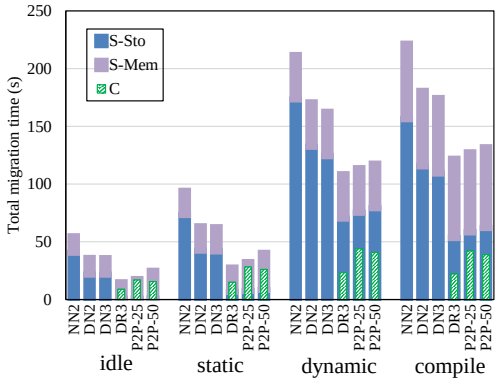
Single Migration

For the two types of deployment methods for two-layer image structure: applications are installed with OS image and applications are stored with user data, we only take the latter one in our experiments. Because existing migration methods with data deduplication do not take image structure into consideration, different image structures have no impact on migration performance. The same four types of VMs as Section 5.1.6 are deployed for evaluation (as shown in Table 5.4).



(a) Similarity ratio=(5%,5%)

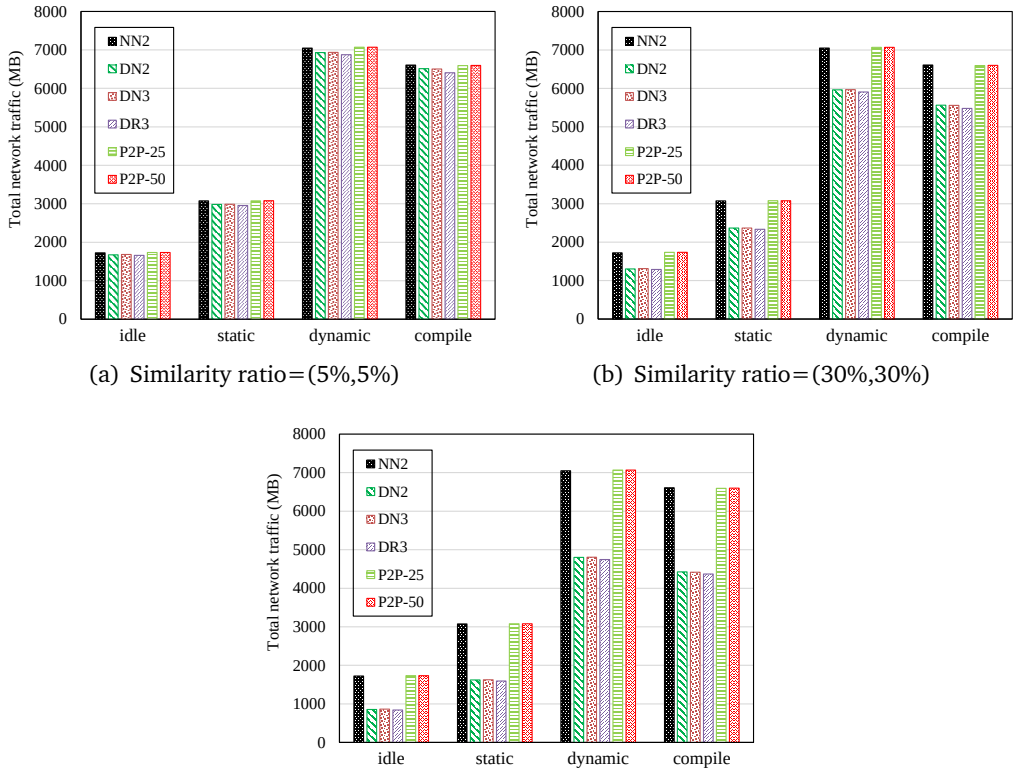
(b) Similarity ratio=(30%,30%)



(c) Similarity ratio=(60%,60%)

Figure 5.24: The total migration time with different migration mechanisms and different VMs. *S-Sto* is the migration time of the storage data of the migrated VM, and *S-Mem* is that of memory data migration. Especially, for DR3, P2P-25, and P2P-50, *S-Sto* denotes the migration time of the UD image from the source site, and *C* is the migration time of base images from the central repository. This denotation is applicable for all figures in this chapter.

Three similarity conditions are used to evaluate the migration performance of DN2, DN3, DR3. According to Figure 5.24, we have the following observations: (1) NN2, P2P-25 and P2P-50 are not affected by image similarity because of not using data deduplication. (2) DN2 and DN3 almost show the same trend, but DN3 is better than DN2. This illustrates that the optimizations to data deduplication proposed in Section 5.2.2 lower the overheads of data deduplication and increases deduplication efficiency. Both DN2 and DN3 prolong the total migration time when the image similarity is small (i.e. (5%,5%)). The bigger the VM image size is, the more they will prolong. It indicates that a big image size and a small amount of duplicated blocks found for it results in a big overhead and a small benefit to migration performance. When the similarity ratio reaches (30%,30%) and (60%,60%), DN2 and DN3 get a better migration performance than NN2. These results show that the optimization of existing data deduplication to the total migration time is highly related with the image similarity condition, even though only conducting data deduplication for base images (DN3). (3) DR3 can reduce the total migration time despite the similarity is small, at (5%,5%). This is because



(a) Similarity ratio=(5%,5%)

(b) Similarity ratio=(30%,30%)

(c) Similarity ratio=(60%,60%)

Figure 5.25: The total network traffic when migrating different applications with different migration methods.

the deduplication operation is moved to the central repository, and its side effects are overlapped by the migration of UD layer data and memory data from the source data center. Compared to NN2, DR3 improves the migration performance for different applications on average by 39%, 45%, and 49% under the three similarity situations, respectively. (4) When the similarity ratio is (5%,5%), P2P-25 and P2P-50 can finish migrating base images faster than DR3, while DR3 becomes faster than P2P-25 and P2P-50 as the similarity ratio increases. However, even though when P2P-25 and P2P-50 are better than DR3, they lead to longer total migration times. This is due to the fact that P2P file sharing takes up too much network bandwidth of the destination host, which decreases the migration speed of UD image from the source site. This is verified by the longer migration time for UD image with P2P-25 and P2P-50 in comparison with DR3. (5) Due to the same reason, P2P-50 exhibits a worse performance than P2P-25. We assume that there is a value in the number of seeds which will result in a proper migration speed for base images to achieve a shorter total migration time than DR3. (6) DR3, P2P-25 and P2P-50 see a small performance improvement as the similarity ratio increases. This results from the fact that the migration process of the source site becomes the bottleneck of VM migration. (7) Overall, the separation of base image migration from the source data center can significantly improve the total migration time, and migrating base images with data deduplication is better than P2P file sharing.

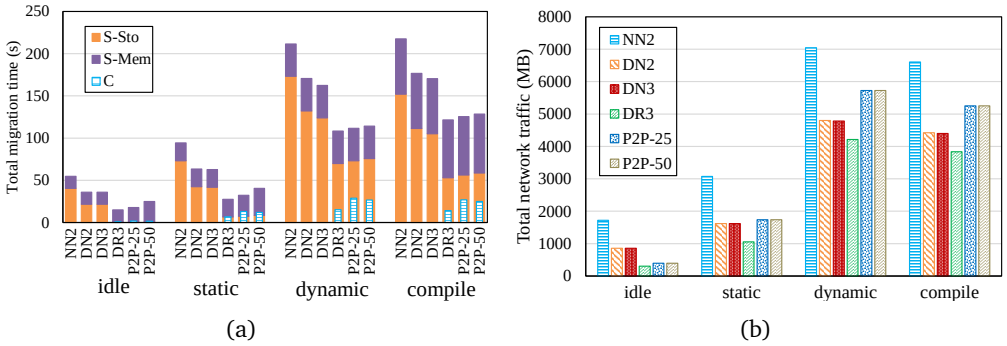


Figure 5.26: The total migration time and total network traffic for OS reuse.

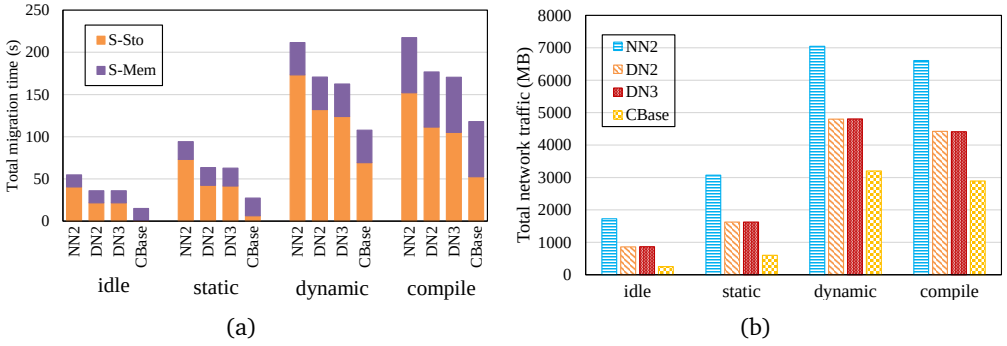


Figure 5.27: The total migration time and total network traffic for WE reuse.

The total network traffic of live VM migration is affected by a variety of factors, such as memory dirty rate, disk dirty rate and total migration time, etc. It is an important metric to measure the interruption of VM migration to the services running in the source data center and the destination data center. The results regarding the total network traffic of migrating the four types of VMs with different migration methods are shown in Figure 5.25. Straightforwardly, for all of the four applications, NN2 has the same network traffic under different similarity ratios. We can see that even though DN2 increases the total migration time when the similarity ratio is (5%,5%), it still reduces the network traffic, compared with NN2. It indicates that existing migration mechanisms with data deduplication can only guarantee lowering network traffic rather than total migration time. Obviously, the network traffic decreases with the increase of similarity ratio. DN3 is better than DN2, and in turn DR3 is better than DN3. This is because a shorter migration time leads to less newly-dirtied data transferred from the source site. For P2P file sharing, no optimization is observed in comparison with NN2. In our experiment, the destination host does not upload base image blocks to other peer data centers. We can predict that when there are other peers which are also downloading the base image, P2P will result in more network traffic than NN2.

Base Image reuse

One advantage of the central base image repository is that base images can be reused between different data centers. In this section, we explore the performance gains resulting from reusing base images at the destination site during VM migration. However, it is impossible to reuse base images for NN2, DN2 and DN3 because base images are created in individual data centers. Even though there is a base image which runs the same OS or software stack with the migrated VM at the target site, they still need to use data deduplication to construct the base images of the migrated VM.

For a fair comparison, when the base images of the migrated VM can be reused at the destination site for CBase, we generate a high ratio of duplicated blocks for DN2 and DN3 at the destination site. As is discussed in Section 5.2.2, there are two situations for base image reuse: OS image reuse and WE image reuse. We set the similarity ratio as (60%,60%) [10, 194] for DN2 and DN3 under these two reuse situations. For the first situation, the similarity ratio of the WE image is set as 60% for DR3. For the second situation, there is no data transferred from the central repository for DR3 because both OS and WE images exist at the destination site.

For OS image reuse, both the total migration time and the total network traffic see the similar trend as the results in Figure 5.24 and 5.25, respectively. But because the OS image will not be migrated in CBase, DR3, P2P-25 and P2P-50 show a further better performance in comparison with NN2, DN2 and DN3. For WE image reuse, CBase only transfers data from the source site without base image migration, so DR3, P2P-25 and P2P-50 have the same migration performance. We combine and denote them as *CBase*, as shown in Figure 5.27. From the results, we can see that CBase is much better than other migration mechanisms regarding both the total migration time and the total network traffic. Furthermore, the migration for the VM which has a bigger base image size can get more performance benefits from CBase.

UD Image Migration

To evaluate the performance of the optimization to UD image migration in Section 5.2.2, we compare CBase with DN3. The migration of the newly written disk blocks is overlapped with memory data migration. We can use the total migration time of these two parts to deduce the performance of the UD image migration, because for a specific VM which is running a stable workload the migration time of memory data almost is a constant. As shown in Figure 5.28, we pre-migrate OS and WE images and the original UD image to the destination site for DN3, and assume OS and WE images can be reused at the destination site and the snapshot of UD image is also transferred to the destination site for CBase. Now, both DN3 and CBase only need to iteratively copy the newly-dirtied disk blocks and simultaneously migrate memory data to the destination site.

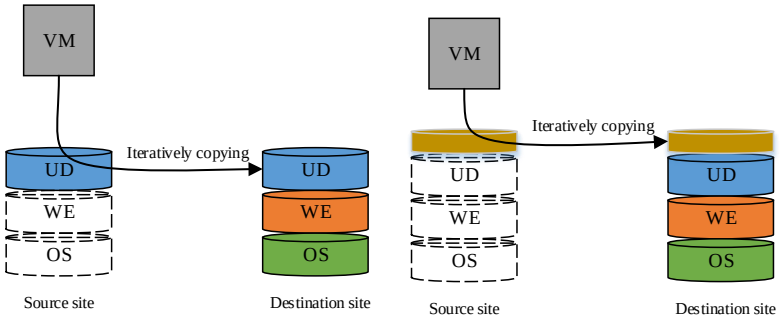


Figure 5.28: The experimental environments of DN3 and CBase for testing the performance of UD image migration.

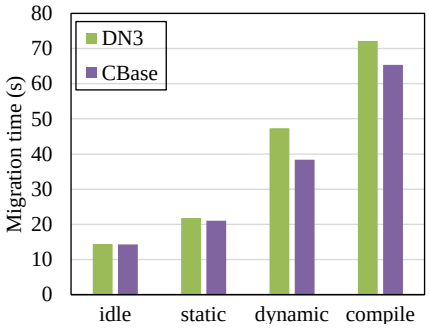


Figure 5.29: The migration time of DN3 and CBase for newly-dirtied disk blocks and memory data.

From Figure 5.29, we can observe that CBase has a smaller migration time for the four types of VMs than DN3, which indicates that CBase has a better migration performance for the new-written disk blocks than DN3. In other words, CBase has a faster migration speed for UD image than DN3. Furthermore, the bigger the size of the original UD image is, the better CBase performs compared to DN3. In particular, CBase increases the migration performance in comparison with DN3 by 0.5%, 3.7%, 18.9% and 9.4% for these four VMs, respectively.

Correlated VM Migration

In this section, the performance of migrating multiple correlated VMs is explored. We deploy a popular three-tier web benchmark, the Rice University Bulletin Board System (RUBBoS) [140]. The presentation tier, the web tier and the database tier consist of 1 VM, 2 VMs and 2 VMs, respectively. Because the two types of two-layer image structures will incur different total storage data size for these VMs, we compare CBase with the migration performances of these two image structures. For a better description, we name the two types of two-layer image structures as *21* and *22*, respectively. The image sizes of these VMs and the data sharing structure of them are shown in Table 5.6. Sequentially or simultaneously migrating multiple VMs is an issue orthogonal to our focus of this paper. We simultaneously migrate the VMs of RUBBoS. The total migration

Table 5.6: The image size (in MB) of the VMs in RUBBoS. The data sharing of base images is also illustrated in this table. For example, for the second type of two-layer image structure, the base image of web server is shared by two VMs.

	Layer	Frontend	Web 1	Web 2	DB 1	DB 2
Two-layer(21)	OS	1,340				
	WE+UD	5,053	2,309	2,540	20,769	21,202
	Total	53,213				
Two-layer(22)	OS+WE	6,210	3,505		5,889	
	UD	167	134	128	16,176	16,269
	Total	48,478				
Three-layer	OS	1,340				
	WE	4,908	2,204		4,583	
	UD	156	114	123	16,195	16,258
	Total	45,881				

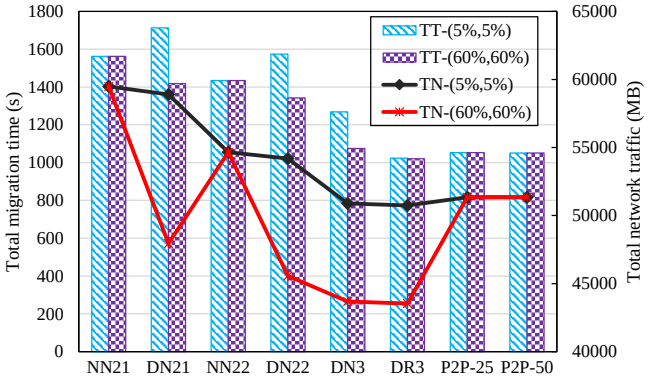


Figure 5.30: The total migration time of different approaches for RUBBoS. *TT*: total migration time, *TN*: total network traffic.

time is the duration between migration initialization and the time when all VMs are handed over to the destination site. The total network traffic is the data received by the destination site during the entire migration time.

We test the total migration time and total network traffic under two image similarity conditions ((5%,5%) and (60%,60%)) for data deduplication and two peer conditions for P2P file sharing (25 seeds and 50 seeds). From Figure 5.30, when the similarity is (5%,5%), same as the performance of single migration, DN21 and DN22 lead to a longer migration time in comparison with NN21 and NN22, respectively. The migration for the VMs with 21 image structure has a worse migration performance than that with 22 image structure. This is because 22 image structure has a smaller total image size than 21 image structure, as shown in Table 5.6. Under both similarity conditions, DR3 is better than DN3, DN3 is better than DN2, and in turn DN2 is better than NN2 on both performance metrics. DR3 increases the total migration time by 34%, 40%, 28%, 35% and 19% when the similarity condition is (5%,5%), compared with NN21, DN21, NN22, DN22 and DN3, respectively. When the similarity condition is (60%,60%), they are 35%, 28%, 29%,24% and 5%. DR3 also results in a significant decrease in the total

migration network traffic. For example, when the similarity is (60%,60%), it reduces the network traffic by 27%, 9%, 20%, 4% and 0.3% in comparison with NN21, DN1, NN22, DN22 and DN3, respectively. Similar to the migration of a single VM, both P2P-25 and P2P-50 have worse performances than DR3 on total migration time and total network traffic. However, P2P-50 has a slightly shorter total migration time than P2P-25, which indicates that P2P file sharing can show its strength when a huge file will be shared. Both P2P-25 and P2P-50 gain better migration performance than other migration mechanisms except DR3.

5.3 Chapter Summary

Many issues accompany with live VM migration over WAN. The most difficult one is how to fast migrate the big storage data of a VM through the Internet. In this chapter, we propose two optimizations for storage data migration: a three-layer image structure and a central base image repository.

In the first optimization, the storage data of a VM is physically separated into three layers. Combining the new image structure and the similarity feature of VM images, we design a migration system—LayerMover. Several mechanisms are designed for data deduplication to improve VM storage data migration performance.

In the second optimization, based on the three-layer image structure, a central repository is introduced to store base images for cloud data centers. With this structure, two technologies (data deduplication and P2P file sharing) are utilized for storage data migration. A migration strategy also is proposed to lower the overhead of UD image migration, and a mechanism is designed to improve the robustness of migration process.

Chapter 6

VM Migration in MEC—User Mobility-induced VM Migration

According to the review of the migration technologies for MEC environments in Section 3.3, many problems are outstanding for live VM migration in MEC. Regarding the problems discussed in Section 1.1.3, two contributions are made in this chapter. Specifically, the first one (Section 6.2) is to decrease storage data migration time and the service degradation during migration when users have high latency requirements. The second one (Section 6.3) is to make a migration plan to reduce the network overhead of constant VM migration when users' latency requirement is not so high (i.e., a VM can be several hops away from its users).

European Telecommunications Standards Institute (ETSI) only proposed a reference architecture for MEC [45]. Different deployment variants are possible. Our study is based on the following deployment structure [105].

- *Each UE only offloads computation tasks to one VM.* A VM in an edge cloud data center can be shared by several users [52]. Also, one UE may request several VMs in an edge cloud data center. We assume that a UE only corresponds to one VM, and each VM also only provides services to one UE.
- *A UE only offloads one type of computation task to an edge cloud data center at any time.* This is based on the users' habitats of using UE. It is seldom to open two or more compute-intensive mobile applications simultaneously on a UE. In addition, some UE, such as sensors, has a very simple and fixed function.

6.1 UE Mobility

According to the presented environments for VM migration, we classify UE mobility into two categories: *certain moving trajectory* and *uncertain moving trajectory*. The mobility with certain moving trajectory means that the cells which a UE will visit are

known. For example, the moving trajectory of a UE can be gained from a navigation application (e.g., Google maps) running on the same UE. Also, some UE may have a fixed moving trajectory, such as, the sensors on a bus. A trajectory is denoted by $\langle S_1, S_2, S_3, ..., S_n \rangle$, and n is the total number of cells visited by a UE. The corresponding edge cloud data centers on the path are denoted by $\langle E_1, E_2, E_3, ..., E_n \rangle$. However, sometimes only the start point S_1 of a UE is known before initializing VM migration. We have no idea about which cell a UE will access next and where its destination is, i.e., uncertain moving trajectory. To take full advantage of the environment features, we design optimization mechanisms for VM migration in MEC based on this classification of UE mobility in this chapter.

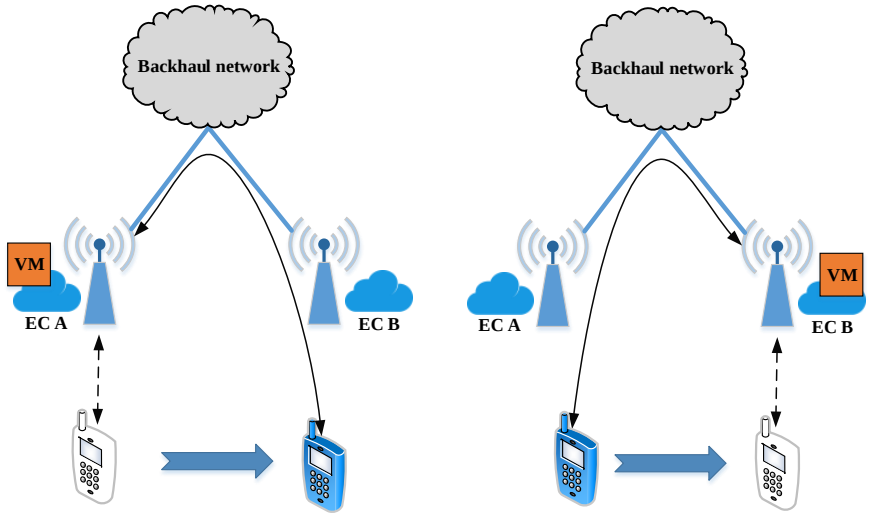
6.2 Migration Performance Improvement

6.2.1 Problem Statement

As discussed in Section 2.5, VM migration in MEC faces the same challenges as the migration over WAN. However, the new environment features of MEC also bring new challenges and optimization opportunities for VM migration. VM migration over WAN is a one-time operation, while the migration in MEC is a constant operation. When a UE is roaming in the RAN, its VM must be constantly migrated between edge cloud data centers to meet the latency requirement, which imposes a higher requirement for migration performance, especially storage data migration. According to this migration characteristic, a question comes to mind: *can we let edge cloud data centers cooperate with each other to improve VM migration performance?*

Another problem with live VM migration in MEC is: *when will the migration of a VM be initialized when its UE is moving between cells*. Some mobile applications (such as online gaming) have a very high requirement on service latency so that the VM must be located in the edge cloud data center where the UE is located at. In other words, the VM must be migrated along the movement of the UE. Under this situation, a too-early or too-late VM migration start-up will hand over a VM to the next edge cloud data center before and after the UE moves into the corresponding area, which in turn leads to an increase of latency and a degradation of service quality, a problem called *inconsistency between UE mobility and VM migration*. As shown in Figure 6.1(a), when the UE moves into the service area of edge cloud data center (EC) B and its VM has not been handed over to EC B, we denote this situation as *late VM handover*. The other situation is the VM is handed over to EC B before the UE arrives at the service area of EC B, as shown in Figure 6.1(b), called *early VM handover*.

Based on above discussions, we illustrate VM migration steps and the relationship between VM migration and UE mobility in Figure 6.2. The goals of our first contribution are twofold.



(a) Late VM handover (b) Early VM handover

Figure 6.1: The inconsistency problem between VM migration and UE mobility.

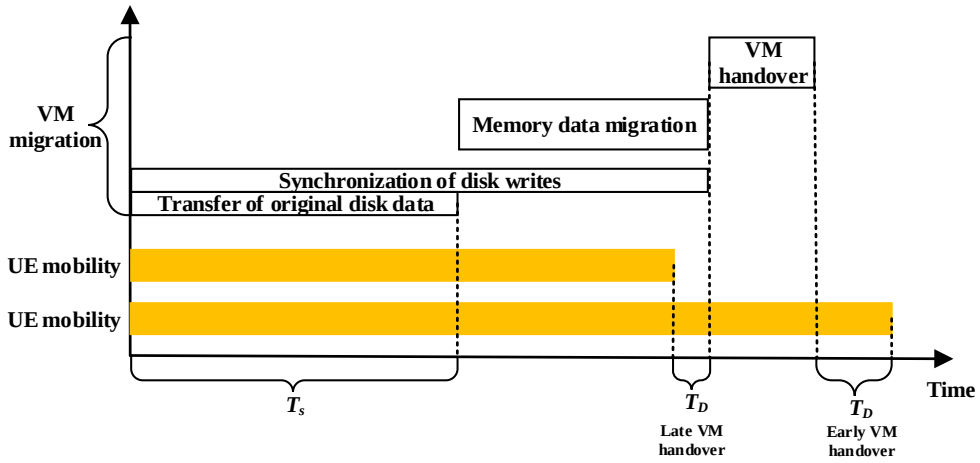


Figure 6.2: The relationship between VM migration and UE mobility. Both early and late VM handovers are shown in this figure.

- Firstly, based on the optimizations in Chapter 5, we further try to improve the storage data migration speed (i.e., T_s in Figure 6.2) between edge cloud data centers in the MEC environments by taking full advantage of UE mobility features.
- Secondly, the tasks of a VM migration process are carefully orchestrated to alleviate the service degradation resulting from the inconsistency between UE mobility and VM migration, i.e., decreasing T_D in Figure 6.2. In other words, it is to make VM handover and UE entrance to a new cell close to each other.

6.2.2 Algorithm Design

We deploy and store VM storage data with three-layer structure. The storage data of a VM is denoted by $\langle B_o, B_w, W \rangle$. B_o, B_w and W represent the OS layer, the WE layer and the UD layer of a VM, respectively. Base images (OSes and WEs) can be pre-deployed and stored in a repository which is accessible to a region of edge cloud data centers, such as, in the place of Base Station Controller (BSC) or a central cloud data center. All edge cloud data centers download base images from the repository to their local storage systems to create VMs, like the structure of *CBase*.

With this deployment structure, we assume that both the OS image and the WE image (B_o, B_w) of the migrated VM exist at the target site for each migration. This assumption is based on the following facts. In MEC environments, what users mainly care about is whether an edge cloud data center can return computing results as soon as possible to meet the low-latency requirement. In most instances, there is no constraint on the selection of the OS image which the VM is running on [145]. Therefore, an OS image can be shared by as many WE images as possible to improve the reuse possibility of OS images between different edge cloud data centers during VM migration. Furthermore, each edge cloud data center is serving a variety of users within its coverage area. It is highly possible that the users in different cells are using the same application [14]. Therefore, WE images have a high reuse possibility among edge cloud data centers as well. In addition, even though the base images of the migrated VM do not exist at the destination site, the source site can inform it to download them from the base image repository before or during VM migration. With this assumption, the step *transfer of original disk data* in Figure 6.2 only needs to transfer the UD layer W of a VM.

In this section, two migration algorithms are designed for the two types of UE mobility to improve storage data migration performance, specifically, a Jump and Delta Migration (JDM) algorithm for the certain moving UE and a Speculation and Snapshot Migration (SSM) algorithm for the uncertain moving UE. At last, two migration initialization mechanisms corresponding to the two migration algorithms are proposed to alleviate the inconsistency problem.

Jump and Delta Migration Algorithm

Since the edge cloud data centers the UE will pass are known before migration, $\langle E_1, E_2, E_3, ..., E_n \rangle$ ($n \geqslant 2$), the core idea behind Jump and Delta Migration (JDM) algorithm is to let these edge cloud data centers cooperate with each other to improve migration speed, as shown in Algorithm 2.

We take three successive edge cloud data centers E_i, E_{i+1}, $E_{i+2} \in \langle E_1, E_2, E_3, ..., E_n \rangle$ to elaborate on the migration procedure of the JDM algorithm. The corresponding cells of the three edge cloud data centers are S_i, S_{i+1}, S_{i+2}. When the VM is run-

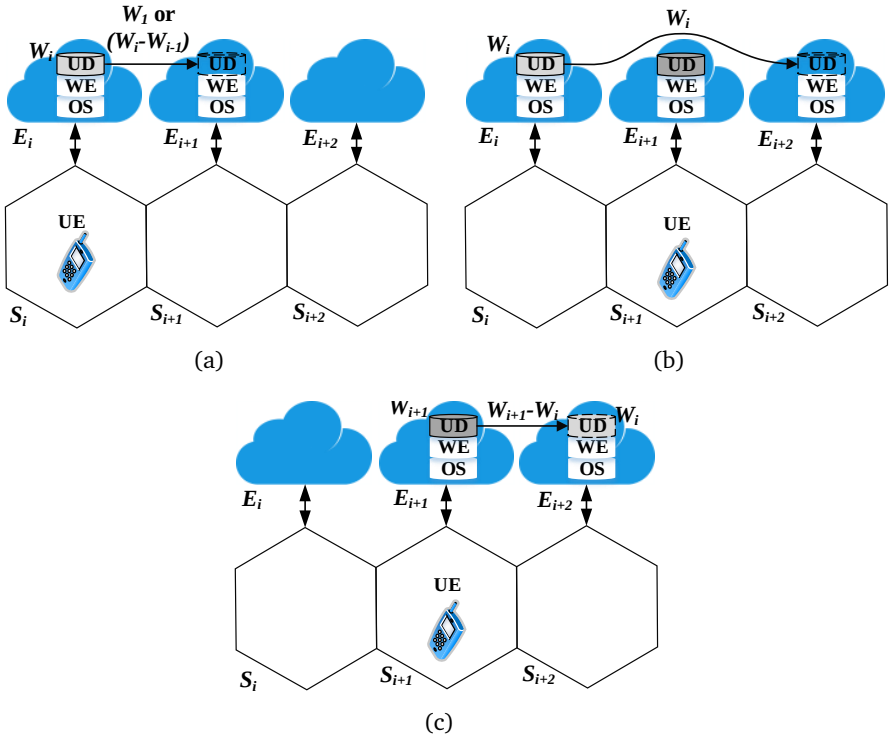


Figure 6.3: Migration procedure of JDM algorithm. (a) Delta migration or the migration between the first and the seond edge cloud data centers. (b) Jump migration. (c) Delta migration.

ning in an edge cloud data center, some disk blocks will be changed and new data will be constantly written to the UD image. We denote the UD image versions in E_i, E_{i+1} and E_{i+2} by W_i, W_{i+1} and W_{i+2}, respectively. The extreme situation is that no new data is written to the UD image in each edge cloud data center, we will have $W_i = W_{i+1} = W_{i+2}$. Otherwise, W_{i+2} is the latest version of the UD image. The JDM algorithm contains two operations: *jump migration* and *delta migration*.

- *Delta migration*: at the beginning, a UE is moving in cell S_i and its VM runs in the corresponding edge cloud data center E_i. Once VM migration is initialized, if a previous version of the UD image W_{i-1} is present at the next edge cloud data center E_{i+1}, only the difference $W_i - W_{i-1}$ between W_i and W_{i-1} is synchronized to E_{i+1}, as shown in Figure 6.3(a). Otherwise, it will be a full transfer of W_i to E_{i+1}, which will happen between the first and the second edge cloud data centers (E_1 and E_2).

- *Jump migration*: after the VM is handed over to E_{i+1}, E_i continues to migrate the UD image W_i to E_{i+2}, as shown in Figure 6.3(b). Starting this step after VM handover is to prevent it from competing for the network bandwidth with the migration process between E_i and E_{i+1}. After this step, the resources of the VM at E_i can be released. When E_{i+1} starts to migrate the UD image W_{i+1} which

Algorithm 2 JDM algorithm

if $n = 2$ **then**
  Transfer W_1 from E_1 to E_2;
  Return;
else
  for all $E_i \in \langle E_1, E_2, E_3, ..., E_{n-1} \rangle$ **do**
    /*Delta migration*/
    Transfer W_i from E_i to E_{i+1}; /*If W_{i-1} is not present in E_{i+1}, it will be a full transfer of W_i. Otherwise, only $W_i - W_{i-1}$ is synchronized to E_{i+1}*/
    /*Jump migration*/
    if $(i < n - 2)$ **and** (the VM is handed over to E_{i+1}) **then**
      Transfer W_i from E_i to E_{i+2};
    end if
  end for
end if

contains some newly-dirtied blocks, a previous version W_i is already located in E_{i+2}.

From the procedure of JDM, we can see that when the UE only moves from one cell to another, i.e. $n = 2$, UD image migration cannot get benefits from JDM. In addition, when a UE frequently moves between two cells, such as office and home, we can keep the copy of its UD image at both these two edge clouds. The copy which is using by the UE is a primary copy, and another acts as a secondary copy. When the UE stops computation offloading in the primary site, the changes of the UD image are synchronized to the secondary site. When the UE moves to the secondary site, a VM with the latest data can be fast provided.

Speculation and Snapshot Migration Algorithm

When the next cell a UE will arrive at is unknown, JDM is not applicable anymore. If no optimization technology is adopted, a VM only can be migrated after the corresponding UE moves into the next cell, which will result in the late-handover problem. Actually, in most cases, the movement of a UE is not stochastic [190], and it is predictable by combining different information, such as, movement history, living habitats, road topology, etc. There are many available algorithms to predict the movement of a UE [154, 95, 191, 153].

The prediction error and computation overhead of a mobility prediction algorithm increase in proportion to the distance between the predicted location and the UE's current location. Therefore, we only take advantage of the possibilities of a UE moving in its neighbor cells. By using a mobility prediction algorithm, we denote the achieved possibilities of a UE moving from its current location to the six neighbor cells as $\langle P_1, P_2, P_3, P_4, P_5, P_6 \rangle$.

Algorithm 3 SSM algorithm

Make an external snapshot for the UD image;
while $S \neq 0$ or $P_i = 100\%$ **do** /*loop until the UE stops or the next cell is fixed*/
  while 1 **do**
    Update $\langle P_1, P_2, P_3, P_4, P_5, P_6 \rangle$;
    Select the two biggest possibilities P_i and P_j. /*$P_i \geqslant P_j$*/;
    if $P_i \geqslant Td_0$ or ($P_i + P_j \geqslant Td_0$ and $P_i - P_j \leqslant Td_1$) **then**
      break;
    end if
  end while
  if SL is not at P_i or P_j **then**
    Migrate SL to the selected edge cloud data center(s);
  end if
end while
if $S = 0$ **then**
  Return;
end if
if $P_i = 100\%$ **then**
  Migrate WL to this edge cloud data center;
end if

In this section, a Speculation and Snapshot Migration (SSM) algorithm is designed to improve the migration speed of UD images based on UE mobility prediction. The key insight of SSM is to migrate a portion of UD image data to the possible next edge cloud data center(s) as early as possible to reduce the service degradation time resulting from a late initialization of VM migration.

The procedure of SSM is shown in Algorithm 3. At the beginning, the UE is moving in the cell S_1, and its VM is running in the corresponding edge cloud E_1. An external snapshot [132] is taken for the UD image. With this operation, an image file will be created to save all new disk writes from the VM, and the original disk data becomes a snapshot file (read-only). Now the UD image is changed into two layers (files): a snapshot layer and a writable layer, denoted by SL and WL, respectively. We have $\langle B_o, B_w, W \rangle \Rightarrow \langle B_o, B_w, SL, WL \rangle$.

The SSM algorithm contains three steps: selection of the possible next cell(s), migration of the snapshot layer and migration of the writable layer, as shown in Figure 6.4. In the first step, a mobility prediction algorithm periodically updates the possibilities $\langle P_1, P_2, P_3, P_4, P_5, P_6 \rangle$. After each update, the two cells with the highest possibility are selected. Their possibilities are denoted by P_i and P_j ($P_i \geqslant P_j$). When the two highest possibilities meet one of the following two conditions: (1) $P_i \geqslant Td_0$, (2) $P_i + P_j \geqslant Td_0$ and $P_i - P_j \leqslant Td_1$, SSM moves into the second step. In the second step, the snapshot layer is migrated to the corresponding edge cloud data center(s) of the selected cell(s) if the snapshot layer was not migrated to them before.

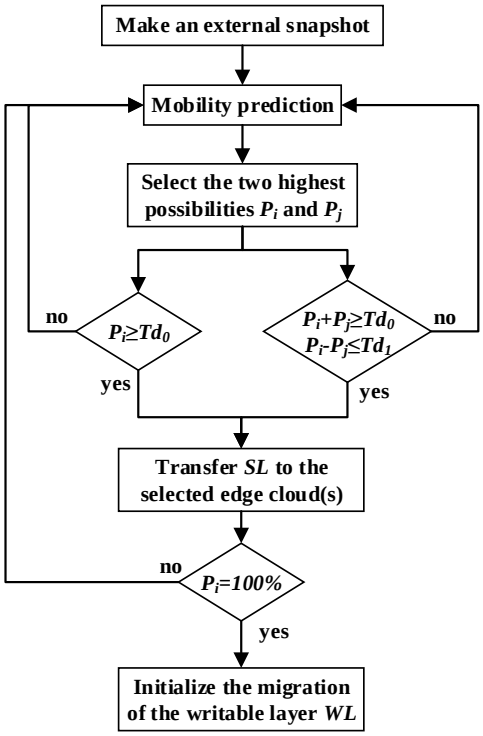


Figure 6.4: The procedure of SSM algorithm.

However, the possibilities for next cells may vary with the movement of the UE. The selected cell(s) (P_i, or, P_i and P_j) may be not the one the UE will arrive at next. Therefore, SSM repeats the above two steps until the next cell is fixed (getting the possibility of 100%) or the UE stops moving. During the loop, each neighbor edge cloud data center at most gets one copy of SL. If the UE stops moving (i.e., $S = 0$), VM migration will be terminated correspondingly. Otherwise, one of the neighbor cells must get the possibility of 100% as the UE moves. The extreme situation is that the next cell is fixed when the UE moves to the border of the current cell. Once the next cell is fixed, SSM begins to transfer the writable layer WL to the corresponding edge cloud data center (more details in Section 6.2.3).

During the selection of the possible next cell(s), Td_0 and Td_1 can be adjusted to meet different migration requirements. For example, a bigger Td_0 means migrating the snapshot layer to the next edge cloud data center with a high confidence. This is beneficial to save network traffic, but it may result in a late migration initialization. A smaller Td_1 makes the snapshot layer be prepared at two neighbor edge cloud data centers which have similar possibilities to guarantee a good migration performance.

6.2.3 Migration Initialization Mechanism

Both too-early and too-late migration start-up will prolong the service degradation time. In this section, we design migration initialization mechanisms for JDM and SSM

algorithms to minimize the service degradation time resulting from the inconsistency between VM migration and UE mobility.

We set the duration that a UE moves from its current location to the cell border and enters a neighbor cell as T_b, the time the UE has stayed in the current cell as T_p, and the total migration time of its VM as T_m. If a UE does not stop in the current cell, T_b will decrease and T_p will increase along with its movement. The sum of T_b and T_p is the total time T_u a UE stays in a cell ($T_u = T_p + T_b$). The key idea behind the initialization mechanisms is to minimize $|T_m - T_b|$, as discussed in Section 6.2.1. We assume that the sojourn time of a UE in a cell is bigger than the total migration time of its VM, i.e., $T_u > T_m$. Because when there is $T_u \leqslant T_m$, it indicates that the UE stays in a cell a very short time or it has a cumbersome VM. Under these situations, the UE can change more computation tasks to local execution to improve its VM's mobility.

When the UE's moving trajectory is certain, T_p and T_b can be calculated by using the moving speed of the UE. As discussed in Section 4.1, the migration time T_m can be calculated with monitored parameters as follows [2, 193].

$$T_m = \frac{S_u}{B} + \frac{M(1-\rho^{n+2})}{B(1-\rho)}, \quad (\rho = \frac{D_m}{B - D_s}) \tag{6.1}$$

S_u and M are the sizes of the UD image and the allocated memory space of the migrated VM, respectively. B is the available network bandwidth for VM migration. ρ is the ratio between memory dirty rate D_m and the available migration bandwidth $B - D_s$ for memory data migration. D_s is the dirty rate of storage data. Based on above analysis, for the UE with a certain moving trajectory, the migration of its VM is initialized when there is $T_m = T_b$ for each visited cell to alleviate the inconsistency issue of UE mobility and VM migration.

When the moving trajectory of a UE is uncertain, the snapshot layer of a UD image is migrated as early as possible once a possible next cell is selected according to the SSM algorithm. Hence, we separate a migration process into two parts: the migration of the snapshot layer and the migration of the writable layer and memory data. The times for these two parts are denoted by T_{sl} and T_{wl}, respectively. T_{sl} and T_{wl} can be gained by using Equation (6.2) and (6.3), respectively.

$$T_{sl} = \frac{S_u}{B} \tag{6.2}$$

$$T_{wl} = \frac{T_t * D_s}{B} + \frac{M(1-\rho^{n+2})}{B(1-\rho)} \tag{6.3}$$

Here, T_t is the total time of a UE offloading computation tasks to edge cloud data centers after a snapshot layer has been created for the UD image of its VM.

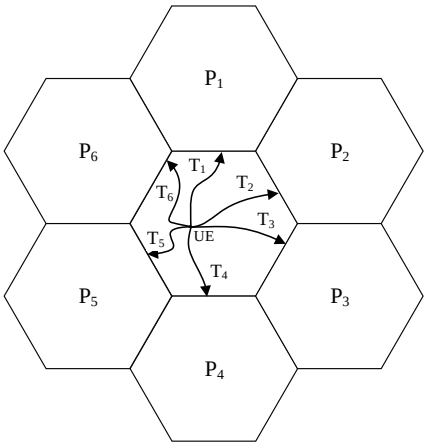


Figure 6.5: The times of a UE moving into its neighbor cells by using mobility prediction.

We represent the times of a UE moving from the current location to a neighbor cell by $\langle T_1, T_2, T_3, T_4, T_5, T_6 \rangle$, as shown in Figure 6.5. They can be achieved with the predicted routes and the moving speed of the UE. Under this situation, T_b is the one calculated from the route with the biggest possibility, i.e.:

$$T_b = T_i; \quad T_i \in \langle T_1, T_2, T_3, T_4, T_5, T_6 \rangle \, and P_i = max\{P_1, P_2, P_3, P_4, P_5, P_6\} \tag{6.4}$$

As shown in Figure 6.6, we set the time that the next cell is fixed as T_f in the SSM algorithm. The time that a UE has stayed in this cell until T_f and the time that it needs to move into the next cell from T_f are denoted by T'_p and T'_b, respectively. There will be four possible relationships between T'_p, T'_b and T_{sl}, T_{wl}, as shown in Figure 6.6.

1. Snapshot layer migration finished before T_f and $T_{wl} \leqslant T'_b$ (Figure 6.6(a)).
2. Snapshot layer migration finished after T_f and $T_{wl} \leqslant T'_b$ (Figure 6.6(b)).
3. Snapshot layer migration finished before T_f and $T_{wl} > T'_b$ (Figure 6.6(c)).
4. Snapshot layer migration finished after T_f and $T_{wl} > T'_b$ (Figure 6.6(d)).

To minimize the inconsistency between VM migration and UE mobility, the migration of the writable layer and memory data is started as follows:

- When $T'_b > T_{wl}$, it is started when $T_b = T_{wl}$;
- When $T'_b \leqslant T_{wl}$, it is started at T_f.

According to the initialization mechanism, for (a) and (b), the migration of the writable layer and memory data will be initialized when $T_b = T_{wl}$; and for (c) and (d), it is started at T_f. When the next cell is fixed too late to leave enough time for the migration of the writable layer and memory data, i.e., the relationships (c) and (d), it

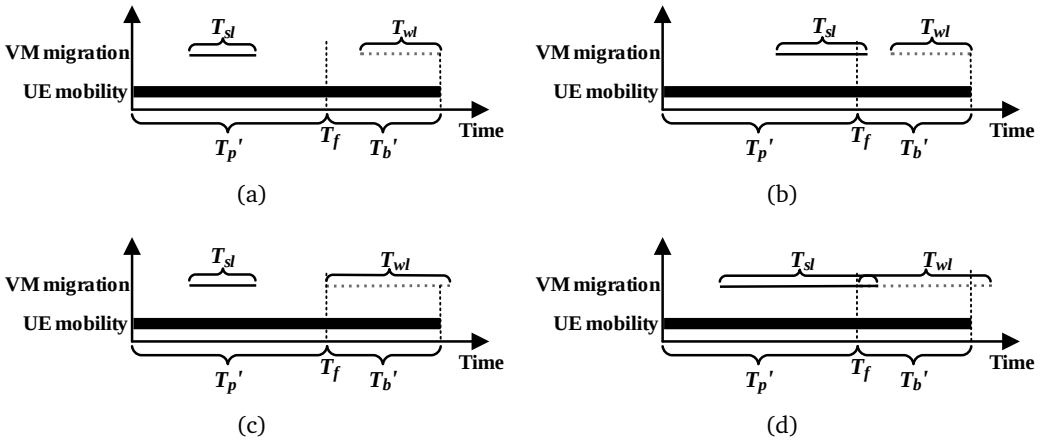


Figure 6.6: The possible relationships between VM migration and UE mobility with uncertain moving trajectory.

will lead to the inconsistency problem between VM migration and UE mobility. This can be solved from two aspects: shorten the migration time T_{wl} and reduce the moving speed of the UE.

T_{wl} is determined by two dynamic parameters: the increase rate of newly generated data (D_s and D_m) and the network bandwidth for migration (B), as shown in Equation (6.3). Therefore, T_{wl} can be shortened by lowering the increase rate of newly generated data or increasing the network bandwidth, like the performance control algorithm in Section 4.2. The increase rate of newly generated data can be controlled from both the UE's and the cloud data center's sides. The UE can decrease the number of tasks offloaded to the edge cloud data center and the edge cloud data center can assign less CPU time to lower the execution speed of the VM [81]. However, both these two manners will introduce a big service interruption, which is opposite to our objective of lowering service degradation. Also, it is impractical and unfriendly to ask a user to lower his/her moving speed to wait for the handover of his/her VM. In summary, the optimal solution is to increase the network bandwidth for migration, such as using MPTCP or assigning more bandwidth to the migration process [167, 109].

6.2.4 Algorithm Performance

In this section, we evaluate the performances of our proposals by numerical studies, firstly the algorithm for the UE with a certain moving trajectory and then the algorithm for the UE with an uncertain moving trajectory. The sojourn time of a UE in a cell is exponentially distributed with an average of 200 seconds. We set three types of VMs, as shown in Table 6.1. Because only one type of computation task is offloaded to an edge cloud data center within a period of time by a UE, we assume that the VMs have a stable memory dirty rate D_m and disk dirty rate D_s. They are configured based on some previous practical tests [203, 206, 2]. The network bandwidth available for VM

Table 6.1: The parameters of the migrated VMs.

VM	UD (MB)	RAM (MB)	D_m (MB/s)	D_s (MB/s)
VM1	500	512	5	1
VM2	1000	1024	15	6
VM3	2000	2048	25	10

migration is 100MB/s. The termination conditions of Xen [2] are used to stop the iteration phase of pre-copy and start the stop-and-copy phase.

We compare our migration algorithms and mechanisms with two simple migration strategies: *M-enter* and *M-leave*. *M-enter* means to start to migrate a VM from the current edge cloud data center E_i to the next edge cloud data center E_{i+1} once the UE enters the cell S_i, while *M-leave* initializes the migration when the UE enters the next cell S_{i+1}.

Migration Performance with Certain Moving Trajectory

According to the JDM algorithm, there are two types of conditions for UD image migration. One is that there is no previous version of the migrated UD image at the destination edge cloud data center, i.e., the migration between the first and the second edge cloud data centers, denoted by *JDM-1*. The other is that a previous version of the migrated UD image is pre-transferred to the destination site and only newly-dirtied blocks will be synchronized, denoted by *JDM-n*.

From Figure 6.7, we can see that *M-leave* gets the worst performance among these migration algorithms. This is due to the fact that a late initialization of VM migration needs to transfer more newly-written disk blocks. Note that migration start time has no effect on the total migration time of memory data due to the volatility feature of memory data. Only the memory states during VM migration will be transferred to the destination site. *JDM-1* has longer storage data migration times in comparison with *M-enter* for these three VMs due to the same reason for *M-leave*. *JDM-n* is better than both *M-enter* and *M-leave* since a previous version of UD image is located at the destination site except that of VM2. *JDM-n* gets a longer storage data migration time than *M-enter* for VM2, which indicates that the size of newly-dirtied disk blocks exceeds the size of the original UD image. For these three VMs, *JDM-n* decreases the storage data migration time by 21% and 59% on average in comparison with *M-enter* and *M-leave*, respectively. In further, we can get that for *JDM*, the more cells a UE visits, the closer the migration benefit is to that of *JDM-n*, because only the migration between the first and the second edge clouds is *JDM-1* and the rest of migrations are *JDM-n*.

As shown in Figure 6.8, the migration initialization mechanism for *JDM* significantly reduces the service degradation time. Due to a careful orchestration of migration

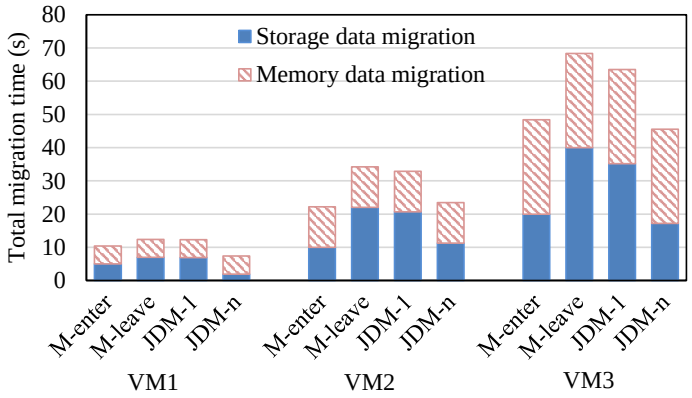


Figure 6.7: Total migration time for the UE with certain moving trajectory.

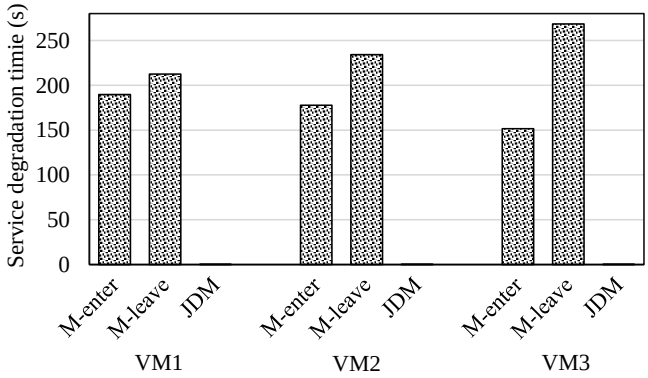


Figure 6.8: Service degradation time during VM migration for the UE with certain moving trajectory.

process, the service degradation time only contains the downtime of memory data migration which has been optimized to the millisecond level by existing technologies [29, 119, 17, 81]. However, both *M-enter* and *M-leave* have a big service degradation time because of a remote connection between a UE and its VM, even though *M-enter* experiences a shorter storage data migration than *JDM* for VM2. In further, *M-enter* is better than *M-leave* because less newly-dirtied data will be transferred, as shown in Figure 6.7.

In summary, *JDM* is beneficial to reduce the migration time of storage data in most cases except that the migrated VM has a big disk dirty rate (such as VM2). More importantly, it can dramatically lower the service degradation duration when a user is roaming in the cellular network.

Migration Performances with Uncertain Moving Trajectory

Since the next cell a UE will arrive at is unknown, it is impossible to use *M-enter* to migrate its VM under this situation. We only compare the performances of *SSM* with *M-leave*. *SSM* prepares the snapshot layer of a VM to the possible next edge cloud data center before VM migration, so we use the duration of migrating the writable layer

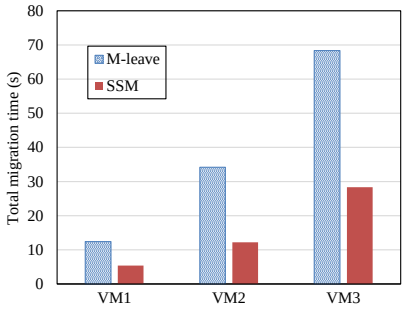


Figure 6.9: Total migration time for the UE with uncertain moving trajectory.

and memory data as the total migration time of each VM. As shown in Figure 6.9, *SSM* greatly decreases the total migration time. Specifically, it is better than *M-leave* by 56%, 64% and 59% for these three VMs, respectively.

According to the migration start-up mechanism for *SSM* algorithm (Section 6.2.3), the service degradation time is determined by two factors: 1) the time T_f when the next cell is fixed and 2) the remaining data of the snapshot layer when the migration of the writable layer will be initialized, as shown in Figure 6.6. Therefore, we divide *SSM* into two situations: *SSM-f* and *SSM-l*. *SSM-f* denotes that the transmission of the snapshot layer is finished when the migration of the writable layer will be initialized (i.e., Figure 6.6(a), 6.6(b) and 6.6(c)), and *SSM-l* refers to that the snapshot layer is not fully prepared at the fixed next edge cloud data center at that time (i.e., Figure 6.6(d)). *SSM* constantly transfers the snapshot layer to the possible next edge cloud according to the predicted possibilities. We assume that half of the snapshot layer data are left for *SSM-l*. Then, we analyze the service degradation time of *SSM* with different values of T_f.

From Figure 6.10, we can observe that the service degradation time of *M-leave* is a constant value for each VM. When the next edge cloud data center is fixed early, *SSM* can guarantee a small service degradation time. For example, if the next edge cloud data center is known before 190s, both *SSM-f* and *SSM-l* have a small service degradation duration for VM1. Furthermore, a VM with a bigger total size (the size of memory data and disk data), a bigger memory dirty rate or a bigger disk dirty rate needs a smaller value of T_f to ensure a small service degradation time, such as 190s for VM1 while 160s for VM3. This is because these three factors determine the total migration time of a VM. A migration with a longer migration time must be started earlier to avoid the inconsistency between UE mobility and VM migration. This is also applicable between *SSM-f* and *SSM-l*. For all these three VMs, *SSM-l* always sees an increase of service degradation time before *SSM-f*. We also can see that even though the next edge cloud data center is fixed very late (such as 195s), *SSM* still can get a smaller service degradation time, compared to *M-leave*, due to a pre-migration of the snapshot layer to the next edge cloud data center.

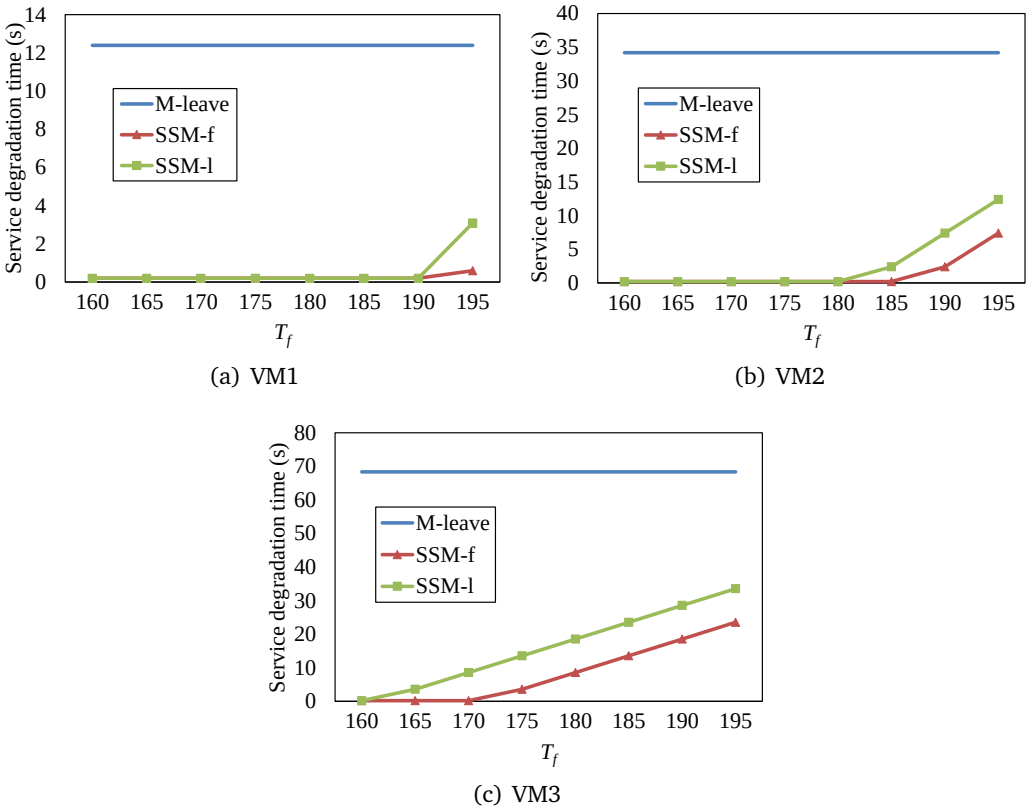


Figure 6.10: Service degradation time during VM migration for the UE with uncertain moving trajectory.

To guarantee a small service degradation time in the situation where the next cell is fixed too late, the additionally required network bandwidth for migration is shown in Figure 6.11 according to the discussion in Section 6.2.3. A bigger service degradation time needs more additional network bandwidth to solve the inconsistency problem between VM migration and UE mobility.

In conclusion, *SSM* significantly decreases both the total migration time and the service degradation time in comparison with *M-leave* under different migration conditions. If required, additional network bandwidth can keep VM migration and UE mobility consistent.

6.3 Network Overhead Alleviation

6.3.1 System Formulation

A cellular network is composed of many hexagonal cells. Each cell is the signal coverage area of a base station. For better discussion, cells in the RAN for each UE are labeled as follows: the start cell of a UE is denoted by $S_{0,0}$, and the other cells are numbered in rings and clockwise, as shown in Figure 6.12. For each cell $S_{i,j}$, the

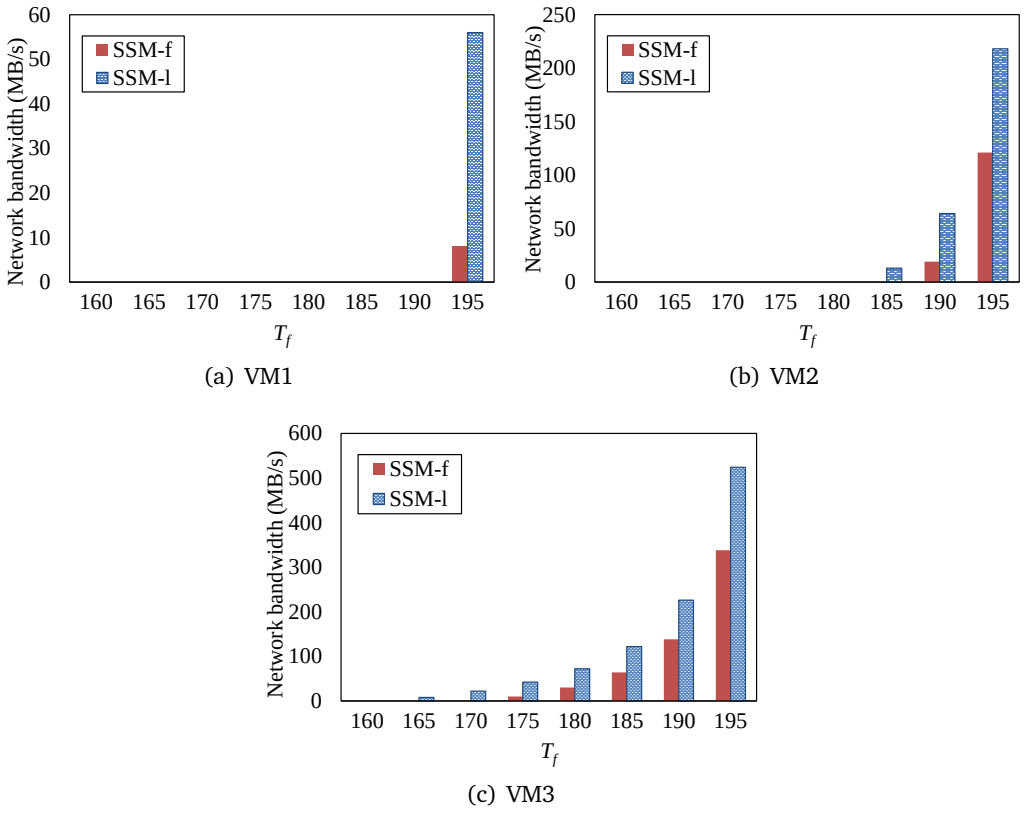


Figure 6.11: The required additional network bandwidth to solve the inconsistency problem between VM migration and UE mobility.

corresponding edge cloud data center is $E_{i,j}$. The distance between two edge cloud data centers ($E_{i,j}$ and $E_{p,q}$) and that between a UE and its VM are denoted by d_{ij}^{pq} and d, respectively. They are the minimum number of cells transversed while a UE moves from one to another rather than Euclidean distance. For example, in Figure 6.12, $d_{00}^{13} = 1$ and $d_{11}^{23} = 2$.

Each UE has a latency requirement l_u for the tasks offloaded to edge cloud data centers, which is the maximum latency the UE can tolerate. Based on this requirement, we can calculate the biggest available value for d, denoted by k. Then each edge cloud data center $E_{i,j}$ has a coverage area $A_{i,j}$. In other words, when a UE's VM is running in the edge cloud data center $E_{i,j}$ and the UE is moving between the edge cloud data centers of $A_{i,j}$, there is no necessity to migrate the VM, as shown in Figure 6.12. Otherwise, the VM must be migrated to an edge cloud data center which is closer to the UE to reduce the latency. The coverage area $A_{i,j}$ of an edge cloud data center $E_{i,j}$ is formulated as:

$$A_{i,j} = \left\{ E_{m,n} | E_{m,n} \in E, d_{ij}^{mn} \leqslant k \right\} \tag{6.5}$$

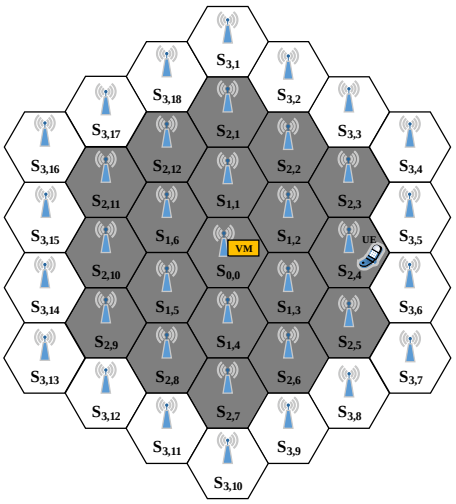


Figure 6.12: An MEC structure based on a cellular network. Herein, $k = 2$ and $d = 2$. The cells with gray color belong to $A_{0,0}$.

When a UE is roaming in the cellular network, its movement and the migration of its VM are denoted by sequences $\Gamma = \{SA_0, SA_1, SA_2, ...\}\,(SA_0 = S_{0,0})$ and $\Pi = \{EC_0, EC_1, EC_2, ...\}\,(EC_0 = E_{0,0})$, respectively. EC_i is the destination edge cloud data center of the ith migration. Hence, the total number t_s of cells visited by a UE equals to $\parallel \Gamma \parallel$, and the total number t_m of migration for its VM is $\parallel \Pi \parallel -1$. A migration plan is represented by both Π and π. π is a cell sequence and each element in it is the trigger point of the corresponding migration in Π. For example, the ith element in π is $S_{m,n}$ and the ith and $(i+1)$th elements in Π are $E_{o,p}$ and $E_{s,r}$, respectively. It means that the VM will be migrated from $E_{o,p}$ to $E_{s,r}$ when the UE is moving in $S_{m,n}$. We define the addition operation between a sequence and an element as appending the element at the end of the sequence, e.g., $\{SA_0, SA_1\} + SA_2 = \{SA_0, SA_1, SA_2\}$.

6.3.2 Network Overhead of Live VM Migration

To calculate the network overhead of each VM migration, we divide VM migration into four steps, as shown in Figure 6.13.

- *S1*: Migrate the original disk data and synchronize the new disk writes to the destination site.
- *S2*: Migrate the original memory data.
- *S3*: Iteratively log and migrate the newly dirtied memory pages to the destination host. The "iteratively" means that the data transferred in current round are the pages dirtied in the previous round of data transmission.

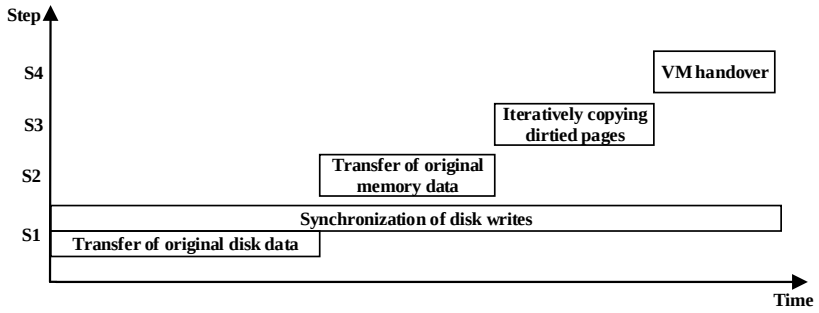


Figure 6.13: The schematic graph of VM migration steps.

- *S4*: When the remaining data is smaller than a preset threshold, the VM is suspended and the remaining data are copied to the destination host for VM resumption, i.e., VM handover.

We denote the data transferred in each step by v_1, v_2, v_3 and v_4, respectively. According to the migration sequence, they can be calculated as follows:

S1: $v_1 = S + \frac{S}{B} * D_s$. S and B are the size of the original disk data and the available network bandwidth for migration, respectively. D_s is the disk write rate of the migrated VM, i.e., disk dirty rate.

S2: $v_2 = M$. M is the allocated memory size.

S3: $v_3 = \frac{M}{B-D_s} * r_m + ... + \frac{M}{(B-D_s)^n} * D_m^n = \frac{M(1-\rho^n)}{1-\rho}$ [193, 96]. It is calculated by changing the network bandwidth to $B - D_s$ since the synchronization of disk write will occupy a portion of network bandwidth. D_m is memory dirty rate and $\rho = \frac{D_m}{B-D_s}$. n is the total number of iterations and can be calculated as follows [193].

$$n = \lceil \frac{lg\frac{Thd}{M}}{\lg \rho} - 1 \rceil \tag{6.6}$$

Here, Thd is the preset threshold for the size of remaining data. We only consider the situation where the migration process can converge, i.e., the remaining data of the migrated VM at the source data center can be reduced to smaller than Thd. It means that the increased speed ($D_s + D_m$) of the newly generated data is smaller than the available network bandwidth (B) for migration.

S4: if we ignore the running states (e.g., CPU registry) due to the relatively small size, the remaining data for *S4* is the memory pages dirtied in the last round of iteration in *S3*. Therefore, we can get the data transferred in this step as: $v_4 = \frac{M}{(B-D_s)^{n+1}} * D_m^{n+1} = M\rho^{n+1}$

In summary, the total network traffic V for a VM migration is:

$$\begin{aligned} V &= v_1 + v_2 + v_3 + v_4 \\ &= S + \frac{SD_s}{B} + M + \frac{M\left(1-\rho^n\right)}{1-\rho} + M\rho^{n+1} \\ &= S + \frac{SD_s}{B} + \frac{M\left(1-\rho^{n+2}\right)}{1-\rho}, \quad \left(\rho = \frac{D_m}{B-D_s}\right) \end{aligned} \tag{6.7}$$

In MEC, the network bandwidth between the source and the destination edge cloud data centers decreases with the distance between them. Because the source edge cloud data center connects to the destination edge cloud data center through the backhaul network, a bigger distance indicates that more hops will be transversed for each data transmission. We assume the network resources (i.e., network bandwidth) are evenly distributed to all base stations and the network bandwidth between two neighbor base stations is D_0. According to previous studies [47, 186], the network bandwidth between two edge cloud data centers ($E_{i,j}$ and $E_{p,q}$) roughly follows the following relationship.

$$B_{ij}^{pq} = \gamma^{d_{ij}^{pq}} B_0 \tag{6.8}$$

γ is the bandwidth degradation coefficient. It is set as 0.85 according to the practical measurements in [47, 186]. For a better description, we set $\rho_i = \frac{D_m}{B_i - D_d}$ and V_k is the network overhead when the migration distance is k for the rest of chapter. B_i and i are the network bandwidth and the distance between the source and the destination edge cloud data centers, respectively.

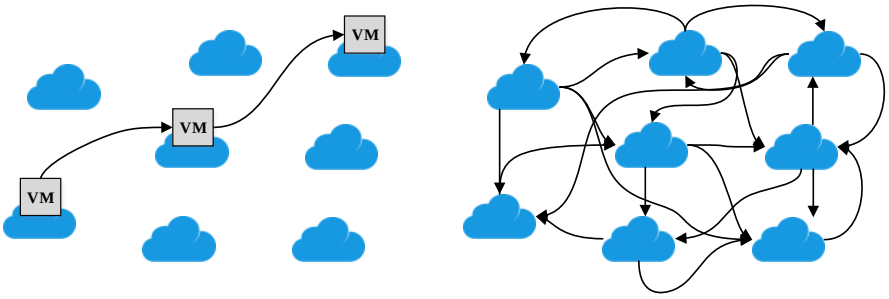
6.3.3 Problem Statement

When a UE is roaming in a cellular network, the constant migrations of its VM will incur a huge amount of network overheads, as shown in Figure 6.14(a). The total network traffic N resulting from constant migration of a VM can be calculated as Equation (6.9). i and j are the ith migration and the migration distance of this migration, respectively.

$$N = \sum_{i=1}^{t_m} V_{ij} \tag{6.9}$$

When we consider this issue in a wider view, all the VMs in edge cloud data centers face the same situation, which derives the network overheads of MEC to a further worse level, as shown in Figure 6.14(b).

Our objective in this section is to minimize the network overhead N without SLA violation (i.e., meet the latency requirement l_u). From Equation (6.9), we can see



(a) The migration of one VM (b) The VM migrations in MEC

Figure 6.14: The network overheads resulting from VM migrations in MEC.

Table 6.2: The environment settings for understanding the effects of migration distance and migration frequency on total network overhead.

S**(MB)**	D_s**(MB/s)**	M**(MB)**	D_m**(MB/s)**	B_0**(MB/s)**
4096	2	1024	10	100

that for a specific movement Γ_i the total network overhead N can be reduced from two aspects: decreasing the total number of migration t_m and lowering the overhead of each migration V_i. To satisfy the first aspect, we should migrate the VM of a UE as far as possible for each migration, i.e., reducing *migration frequency*. To satisfy the second aspect, it is the best to always migrate a VM from one edge cloud data center to its neighbor edge cloud data center according to Equations (6.7) and (6.8) because of the biggest network bandwidth, i.e., reducing *migration distance*. If the total number of cells t_s visited by a UE is fixed, these two aspects are contradictory. A smaller migration frequency means a relatively bigger migration distance, and a smaller migration distance indicates that the VM will be frequently migrated. Ideally, the minimal valve for the total network overhead can be achieved by tuning both these two factors. However, in real case, it is impossible to calculate this value due to the uncertainty and the diversity of UE mobility.

To understand the impacts of migration distance and migration frequency on the total network overhead, we quantitatively analyze the network overheads with different migration distances (per migration every 1 cell, 2 cells, 4 cells and 6 cells, respectively). The configuration on the migrated VM and the migration environment is shown in Table 6.2 [29, 2]. The total number t_s of cells visited by a UE is set as 20, 50, 100, respectively. From Figure 6.15, we can observe that the network overhead per migration is increasing with the migration distance. The migrations over 2 cells, 4 cells and 6 cells have 0.7%, 2.6% and 5.7% more network overhead in comparison with the migration to a neighbor cell, respectively. However, a smaller migration distance does not correspondingly result in a smaller total network overhead. Specifically, the smaller the migration distance is, the bigger the total network overhead is. In further, a bigger t_s can get more benefits from a smaller migration frequency.

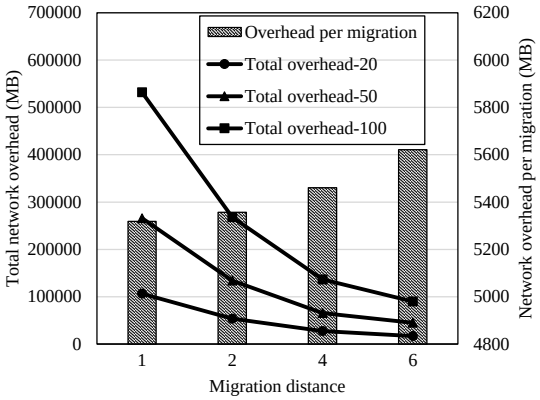
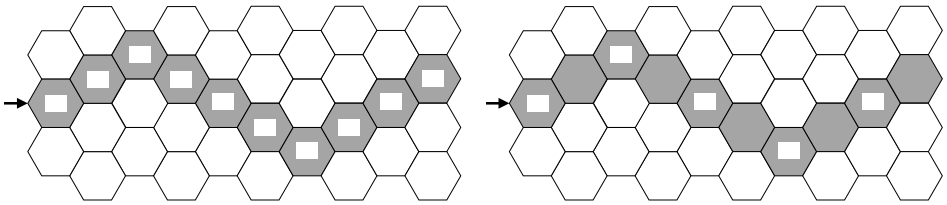


Figure 6.15: The network overheads with different migration distances.



(a) M-All (b) M-Edge

Figure 6.16: Two simple migration algorithms when $k = 2$. The cells with gray color are the moving trajectory of the UE (Γ) and those of which are with a rectangle are the migration trajectory of the corresponding VM (Π).

Even though we migrate the VM with a constant distance for each migration, it is enough for us to understand the significance of these two factors for the total network overhead. Migration frequency has a bigger influence on the total network overhead than migration distance. Actually, the additional network overhead resulting from a bigger migration distance can be easily compensated by a better selection of the destination edge cloud data center for each migration (e.g., the algorithm in Section 6.3.4). Based on above analysis, we try to decrease the total network overhead N by lowering migration frequency.

6.3.4 Algorithm Design

In this section, we firstly discuss two simple migration algorithms (Section 6.3.4). Then two optimized algorithms are respectively designed to minimize the network overhead of VM migration for the two types of UE mobility, specifically, a weight-based algorithm for certain moving UE and a heuristic algorithm for uncertain moving UE.

Algorithm 4 M-Edge algorithm

1: $\Gamma = \{S_{0,0}\}$, $\Pi = \{E_{0,0}\}$, $\pi = \varnothing$;
2: **while** a UE moves into a new cell $S_{i,j}$ **do**
3: $\Gamma = \Gamma + S_{i,j}$;
4: Calculate d;
5: **if** $d < k$ **then**
6: Continue to monitor UE movement;
7: **else** ▷ $d = k$.
8: $\pi = \pi + S_{i,j}$;
9: $\Pi = \Pi + E_{i,j}$;
10: $d = 0$;
11: **end if**
12: **end while**

Two Simple Migration Algorithms

One naive migration algorithm is to migrate a VM along with the UE. A VM is always migrated to the cell where the UE is located at, as shown in Figure 6.16(a). We call this migration strategy as *M-All*. With this strategy, each migration is conducted between two neighbor cells, so they have the same migration bandwidth B_0. According to Equation (6.7) and (6.8), each migration will result in the same network overhead:

$$V_1 = D + \frac{SD_s}{B_0} + \frac{M\left(1-\rho_1^{n+2}\right)}{1-\rho_1} \tag{6.10}$$

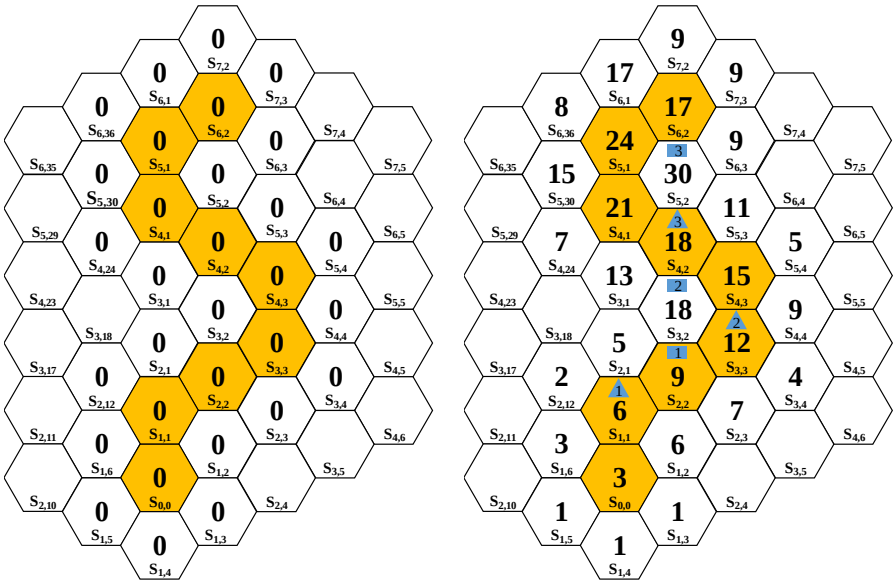
Then the total network overhead N_a of *M-All* for constantly migrating a VM is:

$$N_a = t_m * V_1 \tag{6.11}$$

Another simple migration algorithm is to migrate a VM when its UE moves to the edge of the coverage area of the edge cloud data center where the VM is running, called *M-Edge*. The VM is migrated to the corresponding edge cloud data center of the cell where the UE is located at, as shown in Algorithm 4. Note that d is not always increasing with UE movement. It can increase or decrease or remain unchanged as the UE roams in a cellular network. If it increases, the incremental must be 1. That is why line 7 in Algorithm 4 only can be $d = k$ rather than $d > k$.

With this migration algorithm, the migration distance alway is k. According to Equation (6.8), the network bandwidth for each migration is $\gamma^k B_0$. Then, we can get the total network overhead of *M-Edge* for constantly migrating a VM as follows.

$$N_e = t_m * V_k = t_m * \left(S + \frac{SD_s}{\gamma^k B_0} + \frac{M\left(1-\rho_k^{n+2}\right)}{1-\rho_k}\right) \tag{6.12}$$



(a) (b)

Figure 6.17: The illustration of the *M-Weight* algorithm, here $k = 1$. The cells with color are the moving trajectory of a UE. (a) The initialization phase. The number on a cell is its weight. (b) The weight calculation and migration plan making phases. The cell with a triangle is a migration trigger point and the one with a rectangle with the same number is the corresponding destination edge cloud data center for this migration. The migration plan for this UE mobility is: $\pi = \{S_{1,1}, S_{3,3}, S_{4,2}\}$ and $\Pi = \{E_{0,0}, E_{2,2}, E_{3,2}, E_{5,2}\}$

Migration Algorithm for Certain Moving Trajectory

When a UE's moving trajectory is known, i.e., $\Gamma = \{SA_0, SA_1, SA_2, ..., SA_{t_s-1}\}$, this information can be used to make a migration plan for its VM to minimize the total network overhead. For each movement $SA_i = S_{m,n} \in \Gamma$, we denote the coverage area of the corresponding edge cloud data center $E_{m,n}$ by CA_i, i.e., $CA_i = A_{m,n}$. Then, we can get the corresponding coverage area sequence $\Theta = \{CA_0, CA_1, CA_2, ..., CA_{t_s-1}\}$ for a Γ.

According to the discussion in Section 6.3.2, we get the following conclusions: (1) For each moving step of a UE, we do not need to migrate the VM correspondingly. Once the UE is still in the coverage area of the edge cloud data center where the VM is running at, it is unnecessary to migrate the VM. (2) If an edge cloud data center can cover more cells in Γ, it should be preferable to migrating the VM to there to reduce migration frequency, which in turn can lower the network overhead.

Based on above analysis, we design a migration algorithm (called *M-Weight*) in this section, as shown in Algorithm 5. The key idea behind *M-Weight* is to assign each edge cloud data center in Θ a weight according to the moving trajectory of a UE and migrate the VM to the edge cloud data center with the highest weight for each migration. It

Algorithm 5 M-Weight algorithm

/*Initialization*/
$\Gamma = \{SA_0, ..., SA_{t_s-1}\}$, $\Pi = \{E_{0,0}\}$, $\pi = \varnothing$, $\Theta = \{CA_0, CA_1, ..., CA_{t_s-1}\}$;
for each element $CA_i \in \Theta$ **do**
    for each element $E_{m,n} \in CA_i$ **do**
        $w_{m,n} = 0$;
    end for
end for
/*Weight calculation*/
$loc = 1$; ▷ To mark the searching position
for the locth element $CA_i \in \Theta$ **do**
    for each element $E_{m,n} \in CA_i$ **do**
        $w_{m,n} = w_{m,n} + loc$;
    end for
    $loc = loc + 1$;
end for
/*Migration plan making*/
$x = 0$,$y = 0$; ▷ The current location of the VM
$Sel = \varnothing$; ▷ The possible destination edge cloud data center for each migration.
$loc = 1$;
while $loc < t_s$ **do**
    Select the locth element $S_{i,j}$ in Γ;
    if $d_{ij}^{x,y} = k$ **then**
        Select the edge cloud data center with the highest weight in $A_{i,j}$ and put them into Sel;
        procedure EC-SELECTION;
        /*If $Sel \neq \varnothing$, assume the selected edge cloud data center is $E_{p,q}$, i.e., $Sel = \{E_{p,q}\}$*/
        if $Sel \neq \varnothing \ \ and \ \ p \neq x \ \ and \ \ q \neq y$ **then**
            $x = p$, $y = q$;
            $\pi = \pi + S_{i,j}$;
            $\Pi = \Pi + E_{p,q}$;
        end if
    end if
    $loc = loc + 1$;
end while

consists of three phases: initialization, weight calculation and migration plan making. In the initialization phase, the weights w of all the edge cloud data centers in Θ are set as zero, as shown in Figure 6.17(a).

In the weight calculation phase, the ith element of Θ increases the weights of all the edge cloud data centers within its coverage area by i, as shown in Figure 6.17(b). For example, CA_0 increases the weights of its elements by 1, CA_1 increases by 2, etc. The weight incremental is increasing along the moving direction of the UE, which is to make the migration plan making phase opt to select the edge cloud data center which is far away from the current location of the UE in each migration step to potentially

Algorithm 6 M-Weight algorithm—edge cloud data center selection

```
1: procedure EC-SELECTION
2:     if E_{x,y} ∈ Sel and d_n ⩽ k then
3:         Sel = ∅;                                          ▷ no migration
4:         return;
5:     end if
6:     if E_{x,y} ∈ Sel and d_n > k then
7:         Delete E_{x,y} from Sel;
8:     end if
9:     if ‖ Sel ‖> 1 then
10:        Only leave the edge cloud data centers in Sel which cover the most un-
   scanned SAs in Γ;
11:        if ‖ Sel ‖> 1 then
12:            Leave the edge cloud data centers which incurs the smallest migration
   distance in Sel;
13:            if ‖ Sel ‖> 1 then
14:                Randomly leave one element in Sel.
15:            end if
16:        end if
17:    end if
18: end procedure
```

reduce migration frequency. For example, if two edge cloud data centers cover the same number of cells of Γ, the one which covers more cells in the later part of Γ will get a bigger weight.

The main task of the migration plan making phase is to find the trigger point and the destination edge cloud data center for each migration. It scans the elements of Γ in order. Once a cell $S_{i,j}$ makes $d = k$, it is a possible migration trigger point. We then select the edge cloud data centers which have the highest weight among the edge cloud data center within $A_{i,j}$. The set of the selected edge cloud data centers is denoted by Sel. The final destination edge cloud data center for each migration is fixed with the following steps, as shown in Algorithm 6 and Figure 6.18.

1. If the edge cloud data center where the VM is currently running at is in Sel ($E_{x,y} \in Sel$) and the distance between this edge cloud data center and the next cell the UE will visit is smaller than or equal to k ($d_n \leqslant k$), which means that this edge cloud data center is still an optimal location for the VM, there is no migration.

2. If the edge cloud data center where the VM is currently running at is in Sel ($E_{x,y} \in Sel$) and the distance between this edge cloud data center and the next cell the UE will visit is bigger than k ($d_n \geqslant k$), the VM has to be migrated to a new location. We delete $E_{x,y}$ from Sel.

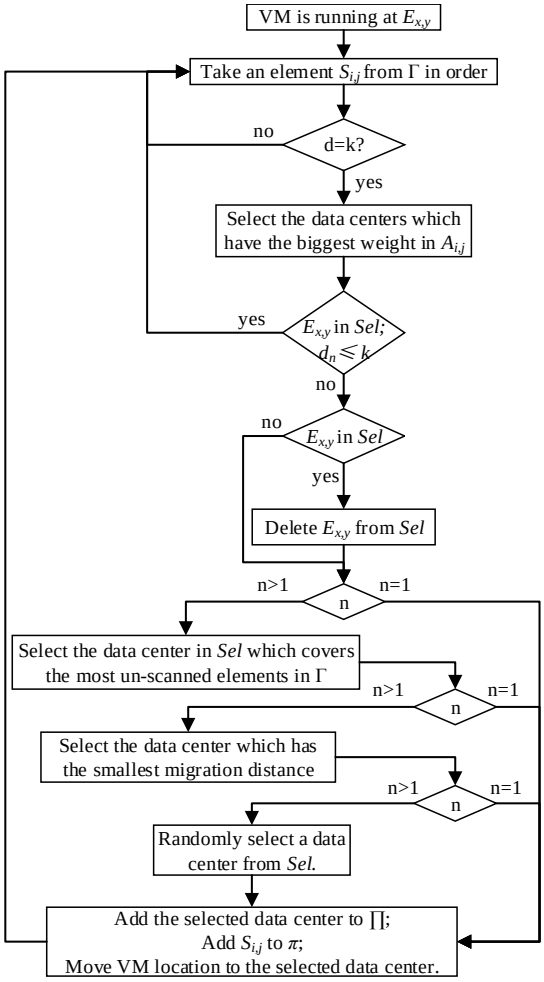


Figure 6.18: The procedure of the migration plan making phase. $n = \| Sel \|$.

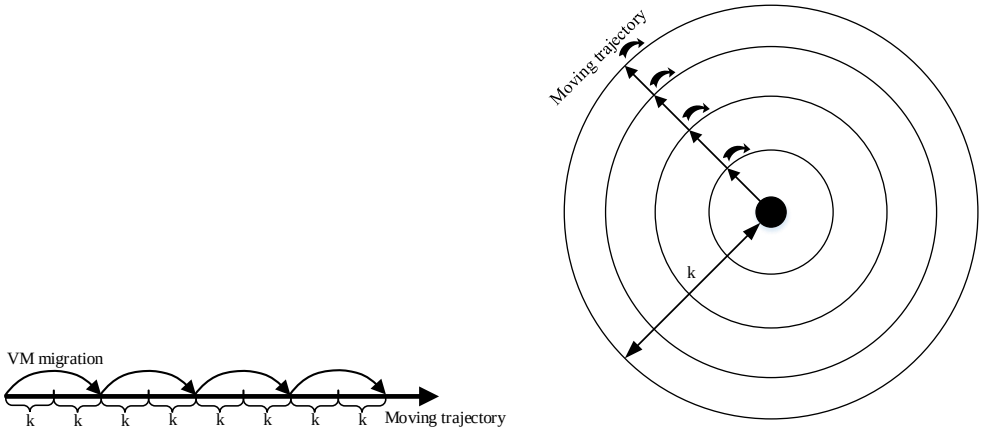
3. If $\| Sel \| > 1$, the edge cloud data center in Sel which covers the most un-scanned elements of Γ will be left.

4. If there is still $\| Sel \| > 1$, the edge cloud data center in Sel which will result in the smallest migration distance is left.

5. If there is still $\| Sel \| > 1$, an edge cloud data center in Sel is randomly selected.

The above operations shrink the elements of Sel step by step. After these steps, if $Sel = \varnothing$ or the selected edge cloud data center $E_{p,q}$ is same as the one where the VM is currently running at, we do nothing and continue to make migration plan for the next element of Γ. Otherwise, $E_{p,q}$ and $S_{i,j}$ are added into Π and π, respectively. The location of the VM is moved to $E_{p,q}$.

Different UE moving trajectories present different environments for VM migration. It is impossible to create a uniform model for the network overhead of *M-Weight*. According to the migration procedure of *M-Weight*, we can calculate the overhead range once the total number t_s of cells visited by a UE is known. If the time of a UE staying



(a) Move in line (b) Move in circle

Figure 6.19: The two extreme situations of UE movement.

in the coverage area $A_{i,j}$ of an edge cloud data center $E_{i,j}$ is s_t, we define the leaving speed of the UE from $A_{i,j}$ as $\frac{1}{s_t}$. A UE will has the biggest leaving speed when it moves straight away from the edge cloud data center $E_{i,j}$ where the VM is running (as shown in Figure 6.19(a)) and the smallest leaving speed when it moves around $E_{i,j}$ (as shown in Figure 6.19(b)). For the first situation, the VM will be migrated once for every $2k$ cells. For the second situation, once the UE is moving in $A_{i,j}$, it is unnecessary to migrate the VM. Correspondingly, these two situations will result in the biggest and the smallest network overheads for a specific UE mobility, respectively.

The total number t_{m1} of migration for the first situation is[1]:

$$t_{m1} = \begin{cases} t_s/(2k), & t_s\%2k \leqslant k; \\ t_s/(2k)+1, & t_s\%2k > k; \end{cases} \tag{6.13}$$

When $t_s\%2k \leqslant k$, the distance for each migration is $2k$. When $t_s\%2k > k$, the distance for all migration except the last one is $2k$. The distance of the last migration is $t_s\%2k - k$. To simplify the problem, we assume that the VM is also migrated to the edge cloud data center which is $2k$ away from the current edge cloud data center in the last migration step. Then, the network overhead for the first situation can be gained.

$$n_w = t_{m1} * V_{2k} = t_{m1} * \left(S + \frac{SD_s}{\gamma^{2k} B_0} + \frac{M\left(1-\rho_{2k}^{n+2}\right)}{1-\rho_{2k}} \right) \tag{6.14}$$

[1]"/" represents modulo operation rather than division operation here.

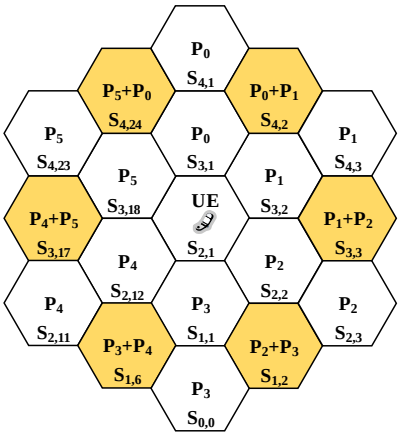


Figure 6.20: The possibility calculation when $k \geqslant 2$.

The overhead for the second situation is zero. Then we can get the network overhead range of *M-Weight* as follows.

$$0 \leqslant N_w \leqslant t_{m1} * \left(S + \frac{SD_s}{\gamma^{2k} B_0} + \frac{M\left(1-\rho_{2k}^{n+2}\right)}{1-\rho_{2k}} \right) \tag{6.15}$$

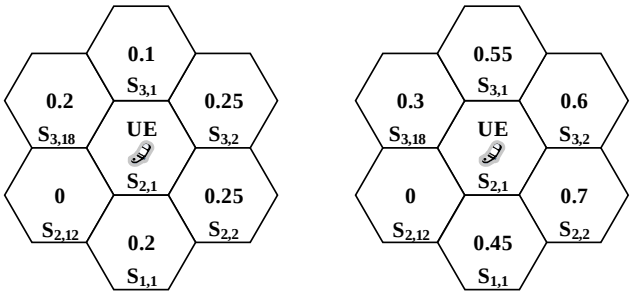
Migration Algorithm for Uncertain Moving Trajectory

Most of the time we do not know about how a UE will move, or users do not want to share their mobility information with a third party. When the moving trajectory of a UE is unknown, it is infeasible to make a holistic migration plan for its VM.We still use a mobility prediction algorithm to predict the future movement of a UE. In this section, we design a heuristic migration algorithm (called *M-Predict*) based on UE mobility prediction to lower the network overhead of VM migration.

Same as the discussion in Section 6.2.2, we only take advantage of the prediction possibilities of a UE moving from the current cell $S_{i,j}$ to its neighbor cells, denoted by $P_{i,j} = \{P_0, P_1, P_2, P_3, P_4, P_5\}$, as shown in Figure 6.20. If a UE does not stop in a cell, there is $P_0 + P_1 + P_2 + P_3 + P_4 + P_5 = 1$, i.e., it must move to a neighbor cell.

If the moving trajectory of a UE is known, we can migrate the VM as far as possible to reduce migration frequency, like *M-Weight*. However, we only predict the next cell a UE will visit. The predicted result also is not fully credible, so migrating the VM too far from the cell where the UE is located at is risky. When the UE changes its moving direction or the predicted movement is wrong, an additional migration may be needed again very soon, which may increase the total network overhead.

To balance the benefits and risks, *M-Predict* only considers the destination cell within the following scopes: the edge cloud data center where the UE is at, its neighbor data centers (the first ring of data centers) and the neighbor neighbor data centers (the



(a) The original possibilities (b) The updated possibilities.

Figure 6.21: The possibility calculation of the edge cloud data centers in the first ring.

Algorithm 7 M-Predict algorithm

```
1: while speed ≠ 0 and d = k do
2:     Calculate {P0, P1, P2, P3, P4, P5};
3:     if k = 1 then
4:         procedure SELECT1;
5:     else
6:         Calculate the possibilities of the edge cloud data center in the second ring;
7:         Select the edge cloud data centers in the second ring with the highest
   possibility;
8:         if x = 1 then
9:             Select this edge cloud data center;
10:        else
11:            if x = 2 and the distance between them is 2 then
12:                Select the small edge cloud data center between them;
13:            else
14:                procedure SELECT1;
15:            end if
16:        end if
17:    end if
18:    if d' > d then
19:        Migrate the VM to the selected edge cloud data center.
20:    end if
21: end while
```

second ring of data centers). The ones in the second ring are further divided into two categories: *big EC* and *small EC*. A big EC is a neighbor of two edge cloud data centers in the first ring, while a small EC is a neighbor of only one edge cloud data center in the first ring. The possibilities of the edge cloud data centers in the second ring are the sum of the possibilities of its neighbor edge cloud data centers in the first ring, as shown in Figure 6.20. They are used to indicate the moving direction of a UE. We can see that the possibility of a big EC must be bigger than its two neighbor small ECs. This is reasonable because a big EC covers more edge cloud data centers in the first ring in comparison with a small EC, which means that a big EC is more resilient to prediction error.

Algorithm 8 M-Predict algorithm—edge cloud data center selection ($k = 1$)

```
1: procedure SELECT1
2:     Select the edge cloud data center in the first ring with the highest possibility;
3:     if x = 1 then
4:         Select this edge cloud data center;
5:     else
6:         Update the edge cloud data center possibilities in the first ring;
7:         Select the edge cloud data center in the first ring with the highest possibility;
8:         if x = 1 then
9:             Select this edge cloud data center;
10:        else
11:            Select the edge cloud data center where the UE is;
12:        end if
13:    end if
14: end procedure
```

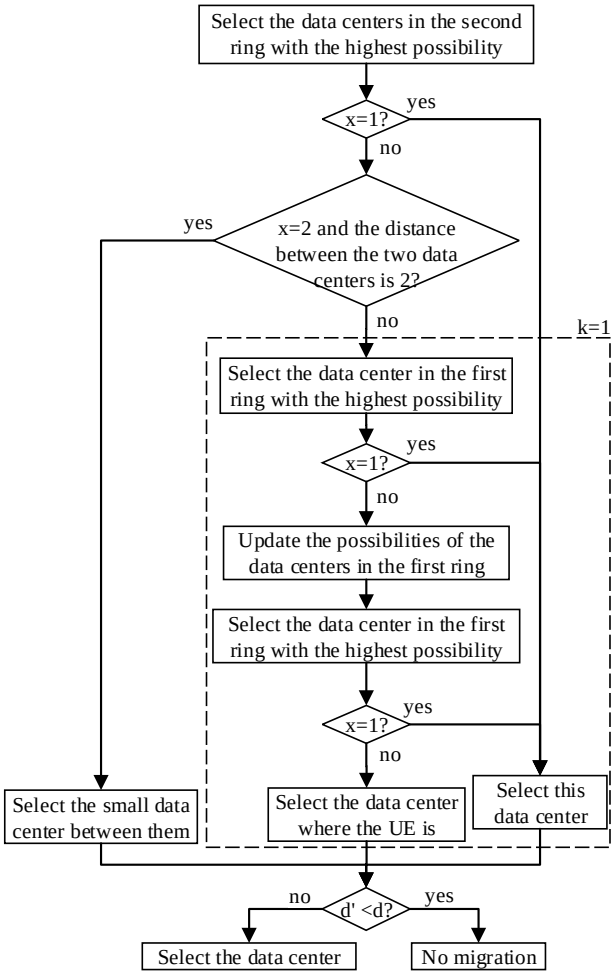


Figure 6.22: The destination edge cloud data center selection for each migration in *M-Predict*.

The core idea behind *M-Predict* is to migrate a VM as far as possible within these edge cloud data center scopes along the predicted moving direction. As shown in Algorithm

7, *M-Predict* firstly calculates the possibilities of the edge cloud data centers in the first and the second rings. Then it selects the destination edge cloud data center for VM migration from the second ring to where the UE is. During the selection procedure, possibility comparison is only conducted between the edge cloud data centers in the same ring. We denote the number of selected edge cloud data centers in each ring by x.

In the second ring, the big ECs which have the highest possibility are selected. If $x = 1$, this edge cloud data center is finally selected. If $x = 2$ and the distance between these two edge cloud data centers is 2 (i.e., there is only one small EC between them), this small EC is finally selected. Otherwise, the algorithm begins to search the first ring. In the first ring, the edge cloud data centers with the highest possibility are selected. If $x = 1$, this edge cloud data center is selected. If $x > 1$, we update the possibilities of the first ring as follows: the possibility of each edge cloud data center is updated by adding the possibilities of its two neighbor edge cloud data centers in the same ring, i.e., $\{P_0, P_1, P_2, P_3, P_4, P_5\} \Rightarrow \{P_5 + P_0 + P_1, P_0 + P_1 + P_2, P_1 + P_2 + P_3, P_3 + P_4 + P_5, P_4 + P_5 + P_0\}$, as shown in Figure 6.21. Then the edge cloud data centers with the highest possibility are selected. If $x = 1$, this edge cloud data center is selected. Otherwise, it indicates that the mobility prediction algorithm cannot accurately predict the UE's moving direction. The edge cloud data center of the cell where the UE is at is selected. We represent the distance between the finally selected edge cloud data center and the one where the VM is currently running by d'. If $d' < d$, it means that the UE will move toward the edge cloud data center where the VM currently is in the next step, so there will be no migration for the VM. Otherwise, the VM is migrated to the finally selected edge cloud data center.

In particular, when $k = 1$, it is infeasible to migrate the VM to the second ring of edge cloud data centers. The latency between the VM and the UE will exceed the latency limit l_u. Therefore, when $k = 1$, *M-Predict* starts to search the edge cloud data centers in the first ring instead of the ones in the second ring. The selection procedure of *M-Predict* is shown in Figure 6.22.

M-Predict has a strong resilience to mobility prediction error. When $k \geqslant 3$, even though a VM is migrated to an edge cloud data center in the second ring, all the edge cloud data centers in the first ring are still in the coverage area of the new edge cloud data center no matter which neighbor cell the UE moves to. For example, in Figure 6.20, if the VM is migrated to $E_{4,2}$ based on Algorithm *M-Predict*, but the UE changes its moving direction or a mobility predict error happens and the UE finally moves into $S_{2,12}$ or $S_{1,1}$, these two edge cloud data centers are still in the coverage area of $E_{4,2}$ and the latency between the VM and the UE still can meet user's requirement.

However, when $k = 1$ and the VM is migrated to an edge cloud data center in the first ring, and when $k = 2$ and the VM is migrated to an edge cloud data center in the

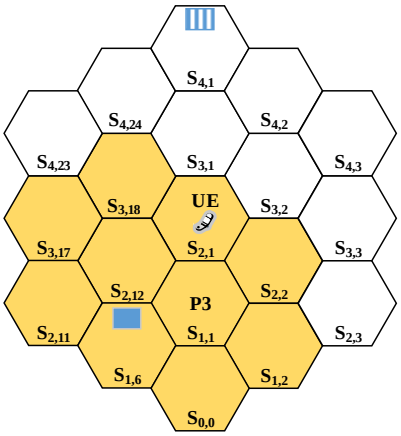


Figure 6.23: The method to solve the prediction error when $k = 2$. The rectangle with color represents the previous location of the VM, and the rectangle with lines is the new location selected by *M-Predict*. The cells with color are in the coverage area of the edge cloud data center $E_{1,6}$.

second ring, the latency between the VM and the UE may exceed the requirement l_u when the above mentioned errors happen. To this end, we do not immediately delete the VM data at the source edge cloud data center after migration for the above two situations and delete it only when the VM moves out the coverage area of the source edge cloud data center. Meanwhile, the predict algorithm constantly monitors the movement of the UE. Once it moves in an adverse direction from the predicted one, the VM begins to synchronize VM data back to the source edge cloud data center. When the UE moves out of the coverage area of the new edge cloud data center, the VM can be fast handed over to the source edge cloud data center. Taking Figure 6.23 as an example, where $k = 2$ and the selected new edge cloud data center for the VM is $E_{4,1}$, we do not delete the VM data at the edge cloud data center $E_{1,6}$ after migration. According to the last step of *M-Predict*, the finally selected edge cloud data center must be out of the coverage area of $E_{1,6}$. From Figure 6.23, we can see that no matter which edge cloud data center is selected as the new location of the VM, all the neighbor edge cloud data centers of $E_{2,1}$ are in the coverage areas of the source and the new edge cloud data centers ($A_{1,6}$ and $A_{4,1}$). However, when the VM is migrated to $E_{4,23}$ and the UE moves to $S_{3,2}$, $E_{3,2}$ will not be in the coverage areas of $E_{1,6}$ and $E_{4,23}$. This also will happen when $k = 1$. Regarding this situation, we change the selected edge cloud data center to its neighbor edge cloud data center. In the example, if the selected edge cloud data center is $E_{4,23}$ or $E_{2,3}$, we change them to $E_{4,24}$ and $E_{3,3}$, respectively.

Same as *M-Weight*, it is also impossible to build a network overhead model for *M-Predict*. The two extreme situations which lead to the biggest and smallest network overheads for *M-Weight* are applicable for *M-Predict* as well. Therefore, the smallest network overhead of *M-Predict* also is zero, i.e., when a UE always moves in the coverage area $A_{0,0}$ of the edge cloud data center $E_{0,0}$ where the VM is running.

The worst situation for *M-Predict* is: a UE leaves the coverage area of each edge cloud data center where its VM runs at with the biggest speed (as shown in Figure 6.19(a)) and the mobility prediction algorithm cannot predict the moving direction for each migration. According to the edge cloud data center selection procedure, a VM will alway be migrated to the edge cloud data center where the UE is located at. VM migration is conducted once every k cells. With this situation, *M-Predict* becomes *M-Edge*. Hence, the biggest network overhead of *M-Predict* is N_e, as shown in Equation (6.12).

By combining these two situations, the range of the network overhead N_p of *M-Predict* is:

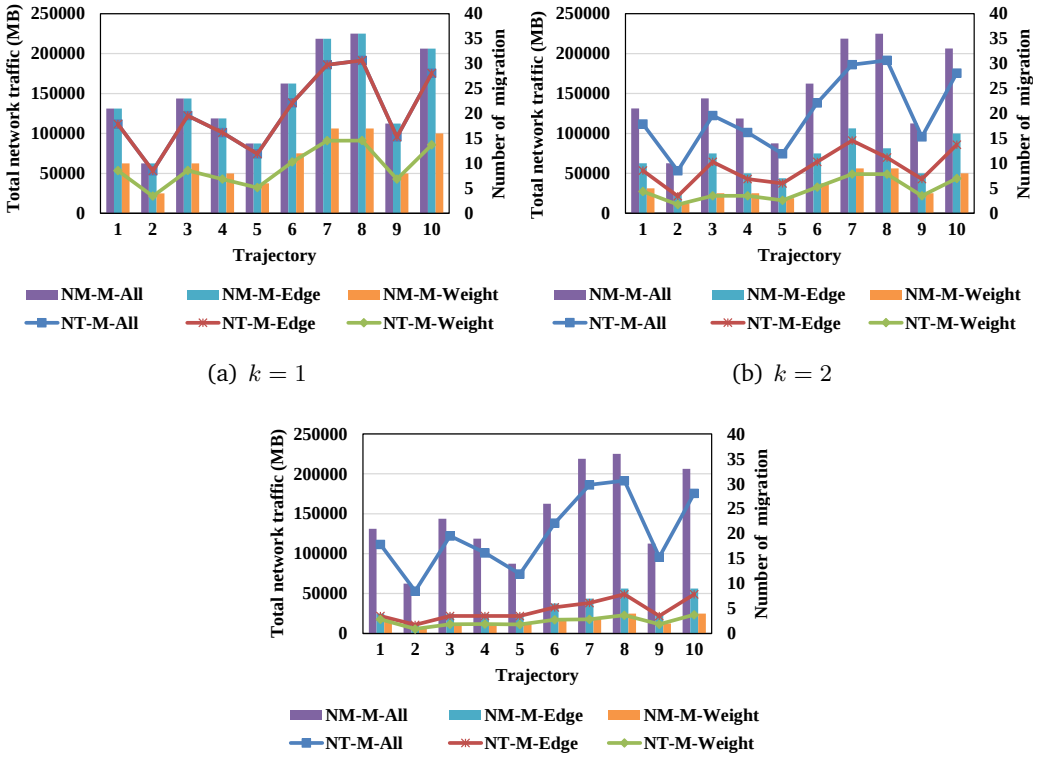
$$0 \leqslant N_p \leqslant N_e \tag{6.16}$$

6.3.5 Algorithm Performance

We evaluate the algorithms by numerical studies in this section. Specifically, the performances of *M-Weight* and *M-Predict* are analyzed by taking *M-All* and *M-Edge* as baselines. As stated in Section 6.3.3, we mainly focus on two performance metrics: total network traffic and the number of migrations. We select ten UE moving trajectories from Microsoft Geolife Trajectories 1.3 [205]. This dataset does not record the telecommunication provider information of each UE. We assume all the selected ten UE are using the China Mobile. By combining the base station topology of the China Mobile in Beijing [22], we can get the corresponding edge cloud data centers a UE transverses during its movement. We set VM parameters for all the ten UE and migration bandwidth as shown in Table 6.2. The performances of the algorithms are also studied with different UE service latency requirements: $k = 1$, $k = 2$ and $k = 4$.

Performances of *M-Weight*

The performances of *M-Weight* are evaluated and compared with *M-All* and *M-Edge* in this section. According to the requirements of *M-Weight*, the moving trajectories are known before VM migration. As shown in Figure 6.24, we can get the following results. (1) When $k = 1$, *M-All* and *M-Edge* have the same migration performances, because *M-Edge* becomes *M-All* under this service latency requirement. Also, *M-All* has a constant migration performance for different service latency requirements because this parameter is not considered in this algorithm. (2) *M-Weight* significantly outperforms *M-All* and *M-Edge* for all the ten moving trajectories. Specifically, it reduces the total network traffic on average by 55% and 55% (when $k = 1$), 77% and 50% (when $k = 2$) and 88% and 46% (when $k = 4$), in comparison with *M-All* and *M-Edge*, respectively. From this statistic, it is easy to understand the percentages for *M-All* are increasing with k. However, the percentages for *M-Edge* are decreasing with k. This is due to two reasons: 1) when k is big, a VM does not need to be frequently migrated because



(a) $k = 1$ (b) $k = 2$

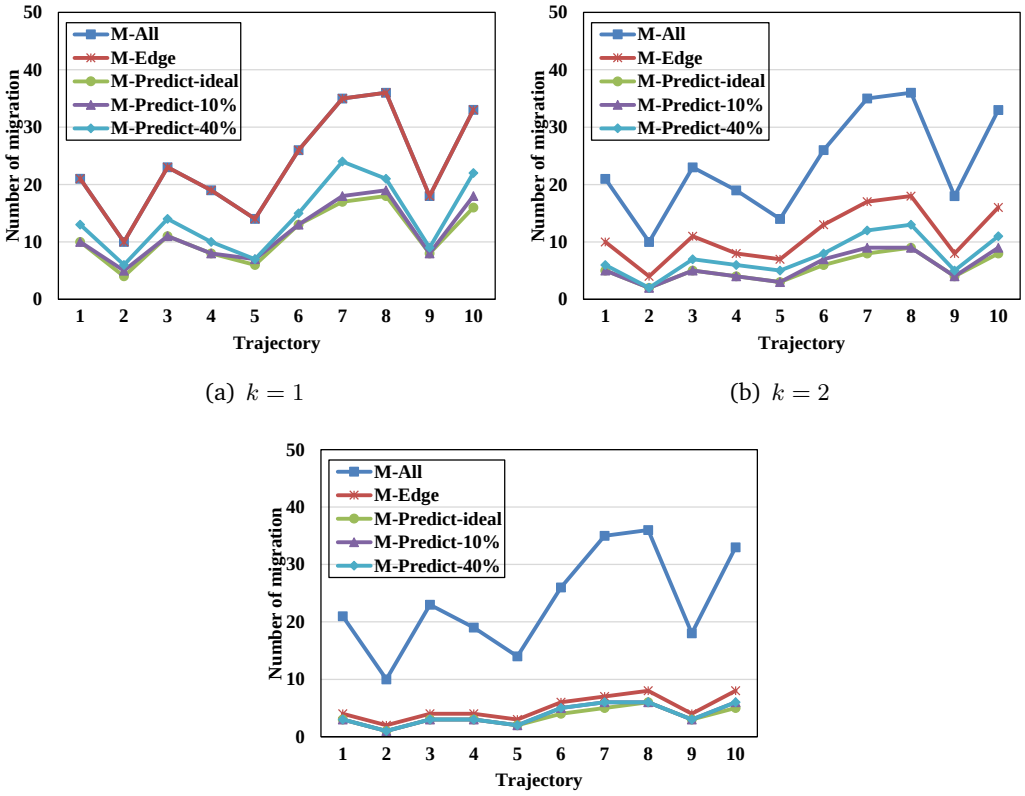
(c) $k = 4$

Figure 6.24: The performances of *M-Weight*. "NM": number of migration, "NT": total network traffic.

each edge cloud data center has a big coverage area, so the performance improvement space for *M-Weight* is small; 2) a bigger k means that there will be a bigger migration distance difference between *M-Edge* and *M-Weight* for each migration, which pushes down the performance improvement of *M-Weight*. (3) *M-Weight* lowers migration frequency, which means that it has a bigger average migration distance compared with *M-All* and *M-Edge*. The total network traffic is correspondingly decreased rather than increased. This further verifies the result in Section 6.3.3: migration frequency is more important than migration distance for reducing total network traffic when a VM has to be constantly migrated.

Performances of *M-Predict*

M-Predict heuristically migrates a VM to a location in advance based on mobility prediction. Therefore, we take prediction error into consideration during evaluating the performances of *M-Predict*. Three conditions are considered: no prediction error (i.e., ideal situation), 10% prediction error and 40% prediction error. For each mobility prediction error, we migrate the VM to the farthest edge cloud data center in the opposite direction of the real moving direction of the UE. According to the edge cloud data center selection procedure of *M-Predict*, the selected one is from the second ring of



(a) $k = 1$ (b) $k = 2$

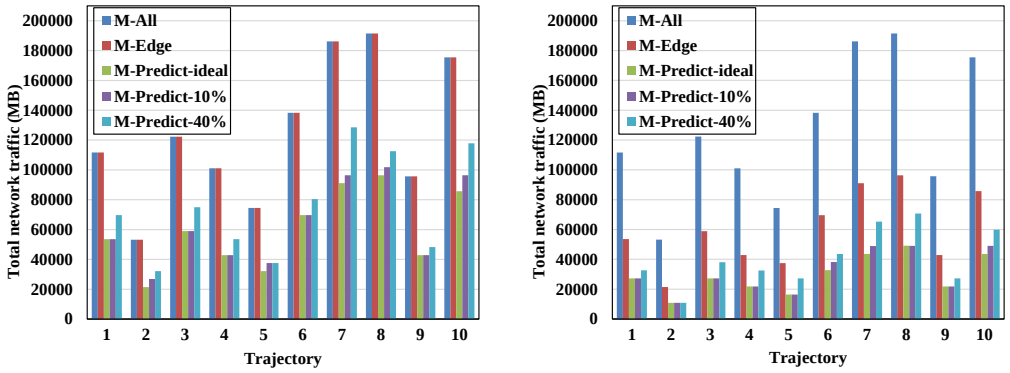
(c) $k = 4$

Figure 6.25: The number of migration of *M-All*, *M-Edge* and *M-Predict*.

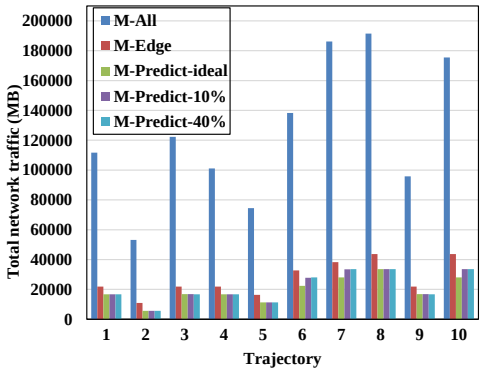
edge cloud data centers when $k \geqslant 2$ and from the first ring of edge cloud data centers when $k = 1$. This is the worst migration decision for each prediction error. We ignore the data synchronized back to the old edge cloud data center from the new edge cloud data center when prediction errors happen, because they are small in comparison with the total network traffic of VM migration.

Regarding the total number of migration, the differences between *M-Predict* and *M-Edge* are decreasing with k. This is due to the fact that a bigger coverage area of each edge cloud data center reduces the number of migration, which in turn lowers the performance improvement of *M-Predict*. The differences between the three conditions of *M-Predict* are very small, even though the latency requirement of the UE is high. Because of the uncertainty of UE moving trajectory, a bad migration decision for current step may be a good decision for the future movement of the UE. From Figure 6.25, we also can get that a smaller service latency requirement further strengthens the resilience of *M-Predict* to prediction errors. This can be seen from the decrease of the differences between the three conditions.

Corresponding to the number of migration, *M-Predict* results in a smaller network traffic for all the ten moving trajectories, compared to *M-All* and *M-Edge*. Ideally, *M-Predict* is better than *M-All* and *M-Edge* on average by 54% and 54%, 77% and 51%,



(a) $k = 1$ (b) $k = 2$



(c) $k = 4$

Figure 6.26: The total network traffic of *M-All*, *M-Edge* and *M-Predict*.

85% and 29% under the three types of service latency requirements, respectively. The decrease of the improvement percentages between *M-Predict* and *M-Edge* is due to the same reasons as those for *M-Weight*. 10% and 40% mobility prediction errors only lead to on average 5% and 12%, 3% and 7% and 1% and 2% more network traffic for the three types of service latency requirements, compared to the ideal situation, respectively.

6.4 Chapter Summary

In this chapter, two optimizations are designed for VM migration in MEC. Different UE mobility presents different environments for VM migration. To make the best of environment features, we broadly divide UE mobility into two categories: certain moving trajectory and uncertain moving trajectory. Both the optimizations are proposed based on this classification of UE mobility.

The first optimization is designed for the situation where a UE has a high latency requirement so that its VM has to be migrated with along its movement (i.e., $k = 0$). Two algorithms are proposed to improve VM storage data migration performance. To

decrease the service degradation time resulting from the inconsistency between UE mobility and VM migration, two migration initialization mechanisms are designed.

The second optimization is to make a migration plan for the UE which does not have a high latency requirement (i.e., $k \geqslant 1$) to lower the network overhead resulting from constant VM migration due to UE mobility. We firstly formulate this issue and figure out the most influential factor for network overhead. Then, two naive migration algorithms (M-All and M-Edge) are described. Then, two optimized migration algorithms (M-Weight and M-Predict) are designed to lower the network overhead of VM migration. Specifically, M-Weight is for the UE with certain moving trajectories, and M-Predict is for the UE with uncertain moving trajectories based on mobility prediction.

Chapter 7

Conclusion

7.1 Summary

In this dissertation, several optimizations are proposed to improve VM migration performance under different environments. The contributions of this dissertation can be divided into four parts: literature review, optimization of VM migration in LAN, optimization of VM migration over WAN and optimization of VM migration in MEC.

In the literature review part, we comprehensively summarize the existing migration technologies, from single migration to correlated VM migration, from LAN migration to WAN migration, from normal migration to user mobility-induced migration. These technologies are compared with each other by the metrics extracted from the literature as well. From the review, we found a series of problems with live VM migration, which are the motivations of this dissertation. Also, we draw on many experiences from the existing technologies of live VM migration for our proposals.

In the optimization of VM migration in LAN part, we mainly focus on the controllability of a migration process and the convergence problem with pre-copy migration. Firstly, performance models are created for the migration procedure and several performance features are derived. Based on the features, a migration control algorithm is designed. It not only takes users' requirements on migration performance into consideration, but also solves the convergence problem of pre-copy. Controlling a migration process to meet user's performance requirement was not considered before. The feasibility was presented with our algorithm by changing the memory dirty rate of the migrated VM or/and the network bandwidth for migration. A migration process is not transparent to cloud managers and users anymore. This is helpful to improve the level of cloud management. For example, users are clear about what performances they can get before migrating their VMs. Cloud managers can make a more detailed plan for hardware maintenance (such as, server replacement).

In the optimization of VM migration over WAN part, we make two architectural rethinks on storage data migration. In the first optimization, a three-layer image structure is designed to increase the data sharing between VMs. The storage data of a VM is physically stored in three layers: OS layer, WE layer and UD layer, by using the COW technology. Based on this structure, several optimizations are proposed for data deduplication to improve VM storage data migration performance. In the second optimization, a central base image repository is introduced for data centers to increase the data sharing between data centers. Base images (i.e., OS images and WE images) are shared by different data centers. With this structure, data deduplication and P2P file sharing are utilized to accelerate VM storage data migration speed. According to the experimental results, those two contributions significantly improve storage data migration performance. Specifically, the first one can decrease the total migration time up to 14%, and the second one can reach 44%. With the development of cloud computing, the cooperations between the data centers of a cloud provider and between the data centers from different cloud providers are becoming more and more frequent, such as hybrid cloud, cloud federation and across-data-center load balancing. It is impossible to implement these cooperations without the support of live VM migration over WAN. Our contributions improve the performance of storage data migration which is the bottleneck part of live VM migration over WAN. Although we only test them on KVM in our experiments, they are generic solutions and can be integrated with different hypervisors/VMMs to migrate a VM across data centers.

In the optimization of VM migration in MEC part, two optimizations are made to improve the migration performance from different perspectives. In the first optimization, two algorithms and two migration initialization mechanisms are designed to reduce the storage data migration time and mitigate the service degradation during migration, respectively. In the second optimization, two algorithms are proposed to mitigate the network overhead resulting from constant VM migration due to UE mobility. The two optimizations for storage data migration (three-layer image structure and central base image repository) are also utilized for the migration in MEC. Numeric studies validate the efficiency of our algorithms. MEC is an outcome of the popularity of the IoT and mobile devices. However, there is still no practical system at present because of many outstanding problems. How to fast live migrate VMs between edge cloud data centers is one of these problems because service latency is a vital metric for MEC. MEC also provides new optimization opportunities for live VM migration. Our contributions are to utilize the new architectural features of MEC to improve the performance of user mobility-induced VM migration under the premise of no violation to service latency requirement, which is a vital aspect for the implementation of MEC.

7.2 Outlook

The work of this dissertation can be extended from the following directions:

1. The migration performance control algorithm now is only for LAN environments and also is based on Xen platform. It can be improved and generalized to adapt to other environments (such as, WAN and MEC) and other platforms (such as, VMware, KVM). The accuracy of monitoring the related parameters of live VM migration determines the accuracy of the performance control algorithm. Therefore, new tools and methods are desired to improve the accuracies of the monitored parameters and the performance control algorithm.

2. For both LayerMover and CBase, we simply use parallel migration to migrate multiple VMs. A coordination algorithm is required for correlated VM migration to lower the service interruption during migration. In addition, because of the high data sharing of the three-layer image structure and the central base image repository, further optimizations can be designed based on these two structures to improve the performance of correlated VM migration.

3. The optimizations to LAN and WAN environments in this dissertation can be integrated with the algorithms for VM migration in MEC to improve the migration performance, especially, the optimizations to WAN migration because they face many common conditions. Our optimizations to user mobility-induced migration assume that each VM is only used by one user. In real cases, a VM in an edge cloud data center may be shared by several users and a user also may offload computation tasks to several VMs, so some optimizations to our algorithms can be designed by taking these factors into consideration. The SSM algorithm did not consider the situation when the mobility prediction is wrong. Some strategies are needed to improve the robustness of the SSM algorithm.

Chapter 8

Publications

The work of this dissertation is published in the following conferences/journals.

The study of literature review:

- **Fei Zhang**, Guangming Liu, Xiaoming Fu, Ramin Yahaypour. *"A Survey on Virtual Machine Migration: Challenges, Techniques and Open Issues."*[J] IEEE Communications Surveys & Tutorials. 2018. DOI: 10.1109/COMST.2018.2794881.

The study of the optimization to VM migration in LAN:

- **Fei Zhang**, Bo Zhao, Xiaoming Fu, Ramin Yahyapour. *"Controlling Migration Performance of Virtual Machines According to User's Requirements."*[C] In Proceedings of CAN'17: Cloud-Assisted Networking Workshop (CAN@CoNEXT 2017). Incheon, South Korean. December 2017. ACM. DOI: 10.1145/3155921.3160606.

The study of the optimization to VM migration over WAN:

- **Fei Zhang**, Xiaoming Fu, Ramin Yahaypour. *"Layermover: Storage migration of virtual machine across data centers based on three-layer image structure."*[C] //Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on. IEEE, 2016: 400-405. DOI: 10.1109/MASCOTS.2016.27.
- **Fei Zhang**, Xiaoming Fu, Ramin Yahaypour. *"LayerMover: Fast Virtual Machine Migration over WAN with Three-Layer Image Structure."*[J] Journal of Future Generation Computer Systems (FGCS). 2018. Volume: 83, pp. 37-49. DOI: 10.1016/j.future.2018.01.017.
- **Fei Zhang**, Xiaoming Fu, Ramin Yahyapour. *"CBase: A New Paradigm for Fast Virtual Machine Migration across Data Centers."*[C] //Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on. IEEE, 2017: 284-293. DOI: 10.1109/CCGRID.2017.26.
- **Fei Zhang**, Guangming Liu, Bo Zhao, Piotr Kasprzak, Xiaoming Fu, Ramin Yahyapour. *"CBase: Fast Virtual Machine Storage Data Migration with a New*

Data Center Structure."[J] Journal of Parallel and Distributed Computing (submitted).

The study of the optimization to VM migration in MEC:

- **Fei Zhang**, Bo Zhao, Philipp Wieder, Xiaoming Fu, Ramin Yahyapour. *"Optimizing Live Migration of Virtual Machines in Mobile Edge Computing."*[J] Journal of Network and Computer Applications (submitted).
- **Fei Zhang**, Guangming Liu, Bo Zhao, Xiaoming Fu, Ramin Yahyapour. *"Reducing the Network Overhead of User Mobility-induced Virtual Machine Migration in Mobile Edge Computing."*[J] Journal of Software: Practice and Experience (submitted).

Bibliography

[1] Eytan Adar and Bernardo A Huberman. "Free riding on Gnutella". In: *First monday* 5.10 (2000).

[2] Sherif Akoush, Ripduman Sohan, Andrew Rice, Andrew W Moore, and Andy Hopper. "Predicting the performance of virtual machine migration". In: *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*. IEEE. 2010, pp. 37–46.

[3] Samer Al-Kiswany, Dinesh Subhraveti, Prasenjit Sarkar, and Matei Ripeanu. "VMFlock: virtual machine co-migration for the cloud". In: *Proceedings of the 20th international symposium on High performance distributed computing*. ACM. 2011, pp. 159–170.

[4] Moiz Arif, Adnan K Kiani, and Junaid Qadir. "Machine learning based optimized live virtual machine migration over WAN links". In: *Telecommunication Systems* 64.2 (2017), pp. 245–257.

[5] InfiniBand Trade Association et al. *InfiniBand Architecture Specification: Release 1.0*. InfiniBand Trade Association, 2000.

[6] Muhammad Atif and Peter Strazdins. "Optimizing live migration of virtual machines in SMP clusters for HPC applications". In: *Network and Parallel Computing, 2009. NPC'09. Sixth IFIP International Conference on*. IEEE. 2009, pp. 51–58.

[7] Paul Barham, Boris Dragovic, Keir Fraser, et al. "Xen and the art of virtualization". In: *ACM SIGOPS operating systems review*. Vol. 37. 5. ACM. 2003, pp. 164–177.

[8] Md Faizul Bari, Mohamed Faten Zhani, Qi Zhang, Reaz Ahmed, and Raouf Boutaba. "CQNCR: Optimal VM migration planning in cloud data centers". In: *Networking Conference, 2014 IFIP*. IEEE. 2014, pp. 1–9.

[9] Artur Baruchi, Edson Toshimi Midorikawa, and Marco AS Netto. "Improving Virtual Machine live migration via application-level workload analysis". In: *Network and Service Management (CNSM), 2014 10th International Conference on*. IEEE. 2014, pp. 163–168.

[10] Sobir Bazarbayev, Matti Hiltunen, Kaustubh Joshi, William H Sanders, and Richard Schlichting. "Content-based scheduling of virtual machines (VMs) in the cloud". In: *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE. 2013, pp. 93–101.

[11] Jing Bi, Zhiliang Zhu, Ruixiong Tian, and Qingbo Wang. "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center". In: *Cloud Computing (CLOUD), 2010 IEEE 3rd international conference on*. IEEE. 2010, pp. 370–377.

[12] Ricardo Bianchini and Ramakrishnan Rajamony. "Power and energy management for server systems". In: *Computer* 37.11 (2004), pp. 68–76.

[13] Nanette J Boden, Danny Cohen, Robert E Felderman, et al. "Myrinet: A gigabit-per-second local area network". In: *IEEE micro* 15.1 (1995), pp. 29–36.

[14] Matthias Böhmer, Brent Hecht, Johannes Schöning, Antonio Krüger, and Gernot Bauer. "Falling asleep with Angry Birds, Facebook and Kindle: a large scale study on mobile application usage". In: Proceedings of the 13th international conference on Human computer interaction with mobile devices and services. ACM. 2011, pp. 47–56.

[15] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. "Fog computing and its role in the internet of things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM. 2012, pp. 13–16.

[16] Sumit Kumar Bose, Scott Brock, Ronald Skeoch, and Shrisha Rao. "CloudSpider: Combining replication with scheduling for optimizing live migration of virtual machines across wide area networks". In: *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Computer Society. 2011, pp. 13–22.

[17] Sumit Kumar Bose, Scott Brock, Ronald Skeoch, Nisaruddin Shaikh, and Shrisha Rao. "Optimizing live migration of virtual machines across wide area networks using integrated replication and scheduling". In: *Systems Conference (SysCon), 2011 IEEE International*. IEEE. 2011, pp. 97–102.

[18] Greg Boss, Padma Malladi, Dennis Quan, Linda Legregni, and Harold Hall. "Cloud computing". In: *IBM white paper* 321 (2007), pp. 224–231.

[19] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. "Live wide-area migration of virtual machines including local persistent state". In: *Proceedings of the 3rd international conference on Virtual execution environments*. ACM. 2007, pp. 169–179.

[20] Rajkumar Buyya, James Broberg, and Andrzej M Goscinski. *Cloud computing: Principles and paradigms*. Vol. 87. John Wiley & Sons, 2010.

[21] Antonio Celesti, Francesco Tusa, Massimo Villari, and Antonio Puliafito. "Improving virtual machine migration in federated cloud environments". In: *Evolving Internet (INTERNET), 2010 Second International Conference on*. IEEE. 2010, pp. 61–67.

[22] *Cellmapper*. Available: `https://www.cellmapper.net`, Accessed: December 2017.

[23] Walter Cerroni. "Multiple virtual machine live migration in federated cloud systems". In: *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*. IEEE. 2014, pp. 25–30.

[24] Walter Cerroni. "Network performance of multiple virtual machine live migration in cloud federations". In: *Journal of Internet Services and Applications* 6.1 (2015), p. 6.

[25] Zenon Chaczko, Venkatesh Mahadevan, Shahrzad Aslanzadeh, and Christopher Mcdermid. "Availability and load balancing in cloud computing". In: *International Conference on Computer and Software Modeling, Singapore*. Vol. 14. 2011.

[26] Fabio Checconi, Tommaso Cucinotta, and Manuel Stein. "Real-time issues in live migration of virtual machines". In: *European Conference on Parallel Processing*. Springer. 2009, pp. 454–466.

[27] Long Cheng, Ilias Tachmazidis, Spyros Kotoulas, and Grigoris Antoniou. "Design and evaluation of small–large outer joins in cloud computing environments". In: *Journal of Parallel and Distributed Computing* (2017).

[28] *Cisco Global Cloud Index: Forecast and Methodology, 2015-2020 (online)*. `www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf`, November.2016.

[29] Christopher Clark, Keir Fraser, Steven Hand, et al. "Live migration of virtual machines". In: *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association. 2005, pp. 273–286.

[30] *CRIU*. Available: `https://criu.org/Main_Page`.

[31]Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, et al. “Remus: High availability via asynchronous virtual machine replication”. In: *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. San Francisco. 2008, pp. 161–174.

[32]Umesh Deshpande and Kate Keahey. “Traffic-sensitive live migration of virtual machines”. In: *Future Generation Computer Systems* (2016).

[33]Umesh Deshpande, Danny Chan, Ten-Young Guh, et al. “Agile live migration of virtual machines”. In: *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE. 2016, pp. 1061–1070.

[34]Umesh Deshpande, Brandon Schlinker, Eitan Adler, and Kartik Gopalan. “Gang migration of virtual machines using cluster-wide deduplication”. In: *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE. 2013, pp. 394–401.

[35]Umesh Deshpande, Unmesh Kulkarni, and Kartik Gopalan. “Inter-rack live migration of multiple virtual machines”. In: *Proceedings of the 6th international workshop on Virtualization Technologies in Distributed Computing Date*. ACM. 2012, pp. 19–26.

[36]Umesh Deshpande, Xiaoshuang Wang, and Kartik Gopalan. “Live gang migration of virtual machines”. In: *Proceedings of the 20th international symposium on High performance distributed computing*. ACM. 2011, pp. 135–146.

[37]Chris Develder, Marc De Leenheer, Bart Dhoedt, et al. “Optical networks for grid and cloud computing applications”. In: *Proceedings of the IEEE* 100.5 (2012), pp. 1149–1167.

[38]*DRBD*. Available: `https://docs.linbit.com/`, Acessed: Mar. 2017.

[39]Rajdeep Dua, A Reddy Raja, and Dharmesh Kakadia. “Virtualization vs containerization to support paas”. In: *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE. 2014, pp. 610–614.

[40]George W Dunlap, Samuel T King, Sukru Cinar, Murtaza A Basrai, and Peter M Chen. “ReVirt: Enabling intrusion analysis through virtual-machine logging and replay”. In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pp. 211–224.

[41]D Eastlake 3rd and Paul Jones. *US secure hash algorithm 1 (SHA1)*. Tech. rep. 2001.

[42]*EC2 Statistics*. `thecloudmarket.com/stats`. December.2016.

[43]*EGI.* Available: `https://www.egi.eu`, Acessed: Jan. 2017.

[44]Magnus Ekman and Per Stenstrom. “A robust main-memory compression scheme”. In: *ACM SIGARCH Computer Architecture News*. Vol. 33. 2. IEEE Computer Society. 2005, pp. 74–85.

[45]MECISG ETSI. “Mobile Edge Computing (MEC); Framework and reference architecture”. In: *ETSI, DGS MEC* 3 (2016).

[46]Michal Feldman and John Chuang. “Overcoming free-riding behavior in peer-to-peer systems”. In: *ACM sigecom exchanges* 5.4 (2005), pp. 41–50.

[47]Mario Gerla, Ken Tang, and Rajive Bagrodia. “TCP performance in wireless multi-hop networks”. In: Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA’99. Second IEEE Workshop on. IEEE. 1999, pp. 41–50.

[48]Balazs Gerofi, Zoltan Vass, and Yutaka Ishikawa. “Utilizing memory content similarity for improving the performance of replicated virtual machines”. In: *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. IEEE. 2011, pp. 73–80.

[49]Lazaros Gkatzikis and Iordanis Koutsopoulos. “Migrate or not? Exploiting dynamic task migration in mobile cloud computing systems”. In: *IEEE Wireless Communications* 20.3 (2013), pp. 24–32.

[50] Nikolay Grozev and Rajkumar Buyya. “Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments”. In: *The Computer Journal* 58.1 (2015), pp. 1–22.

[51] Diwaker Gupta, Sangmin Lee, Michael Vrable, et al. “Difference engine: Harnessing memory redundancy in virtual machines”. In: *Communications of the ACM* 53.10 (2010), pp. 85–93.

[52] Kiryong Ha, Yoshihisa Abe, Zhuo Chen, et al. “Adaptive vm handoff across cloudlets”. In: (2015).

[53] Kiryong Ha, Padmanabhan Pillai, Wolfgang Richter, Yoshihisa Abe, and Mahadev Satyanarayanan. “Just-in-time provisioning for cyber foraging”. In: *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM. 2013, pp. 153–166.

[54] Stuart Hacking and Benoît Hudzia. “Improving the live migration process of large enterprise applications”. In: *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*. ACM. 2009, pp. 51–58.

[55] Jacob Gorm Hansen. “Virtual Machine Mobility with Self-Migration”. PhD thesis. Citeseer, 2009.

[56] *HAST - Highly Available Storage.* Available: `https://wiki.freebsd.org/HAST`, Acessed: Mar. 2017.

[57] Mark D Hill. *Aspects of cache memory and instruction buffer performance*. Tech. rep. DTIC Document, 1987.

[58] Michael R Hines and Kartik Gopalan. “Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning”. In: *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM. 2009, pp. 51–60.

[59] Michael R Hines, Umesh Deshpande, and Kartik Gopalan. “Post-copy live migration of virtual machines”. In: *ACM SIGOPS operating systems review* 43.3 (2009), pp. 14–26.

[60] Takahiro Hirofuchi, Hidemoto Nakada, Hirotaka Ogawa, Satoshi Itoh, and Satoshi Sekiguchi. “A live storage migration mechanism over wan and its performance evaluation”. In: *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*. ACM. 2009, pp. 67–74.

[61] Takahiro Hirofuchi, Hirotaka Ogawa, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. “A live storage migration mechanism over wan for relocatable virtual machine services on clouds”. In: *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society. 2009, pp. 460–465.

[62] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. “Enabling instantaneous relocation of virtual machines with a lightweight vmm extension”. In: *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society. 2010, pp. 73–83.

[63] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. “Reactive consolidation of virtual machines enabled by postcopy live migration”. In: *Proceedings of the 5th international workshop on Virtualization technologies in distributed computing*. ACM. 2011, pp. 11–18.

[64] Paul Hsieh. *Hash functions.* Available: `http://www.azillionmonkeys.com/qed/hash.html`.

[65]Jinhua Hu, Jianhua Gu, Guofei Sun, and Tianhai Zhao. “A scheduling strategy on load balancing of virtual machine resources in cloud computing environment”. In: *Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on*. IEEE. 2010, pp. 89–96.

[66]Liang Hu, Jia Zhao, Gaochao Xu, Yan Ding, and Jianfeng Chu. “HMDC: live virtual machine migration based on hybrid memory copy and delta compression”. In: *Appl. Math* 7.2L (2013), pp. 639–646.

[67]Rongdong Hu, Guangming Liu, Jingfei Jiang, and Lixin Wang. “A new resources provisioning method based on QoS differentiation and VM resizing in IaaS”. In: *Mathematical Problems in Engineering* 2015 (2015).

[68]Wei Huang, Qi Gao, Jiuxing Liu, and Dhabaleswar K Panda. “High performance virtual machine migration with RDMA over modern interconnects”. In: *Cluster Computing, 2007 IEEE International Conference on*. IEEE. 2007, pp. 11–20.

[69]Nikolaus Huber, Marcel von Quast, Michael Hauck, and Samuel Kounev. “Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments.” In: *CLOSER*. 2011, pp. 563–573.

[70]Khaled Z Ibrahim, Steven Hofmeyr, Costin Iancu, and Eric Roman. “Optimized pre-copy live migration for memory intensive applications”. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM. 2011, p. 40.

[71]*Introduction to Linux Traffic Control.* 2006.

[72]*Iometer*. `http://www.iometer.org/`. December.2016.

[73]Van Jacobson. “Congestion avoidance and control”. In: *ACM SIGCOMM computer communication review*. Vol. 18. 4. ACM. 1988, pp. 314–329.

[74]KR Jayaram, Chunyi Peng, Zhe Zhang, et al. “An empirical analysis of similarity in virtual machine images”. In: *Proceedings of the Middleware 2011 Industry Track Workshop*. ACM. 2011, p. 6.

[75]Brendan Jennings and Rolf Stadler. “Resource management in clouds: Survey and research challenges”. In: *Journal of Network and Systems Management* 23.3 (2015), pp. 567–619.

[76]Qin Jia, Zhiming Shen, Weijia Song, Robbert Van Renesse, and Hakim Weatherspoon. “Supercloud: Opportunities and challenges”. In: *ACM SIGOPS Operating Systems Review* 49.1 (2015), pp. 137–141.

[77]Joe Wenjie Jiang, Tian Lan, Sangtae Ha, Minghua Chen, and Mung Chiang. “Joint VM placement and routing for data center traffic engineering”. In: *INFOCOM, 2012 Proceedings IEEE*. IEEE. 2012, pp. 2876–2880.

[78]Lei Jiao, Roy Friedman, Xiaoming Fu, et al. “Cloud-based computation offloading for mobile devices: State of the art, challenges and opportunities”. In: Future Network and Mobile Summit (FutureNetworkSummit), 2013. IEEE. 2013, pp. 1–11.

[79]Hai Jin, Li Deng, Song Wu, Xuanhua Shi, and Xiaodong Pan. “Live virtual machine migration with adaptive, memory compression”. In: *Cluster Computing and Workshops, 2009. CLUSTER’09. IEEE International Conference on*. IEEE. 2009, pp. 1–10.

[80]Hai Jin, Li Deng, Song Wu, et al. “MECOM: Live migration of virtual machines by adaptively compressing memory pages”. In: *Future Generation Computer Systems* 38 (2014), pp. 23–35.

[81]Hai Jin, Wei Gao, Song Wu, et al. “Optimizing the live migration of virtual machine by CPU scheduling”. In: *Journal of Network and Computer Applications* 34.4 (2011), pp. 1088–1096.

[82] Keren Jin and Ethan L Miller. "The effectiveness of deduplication on virtual machine disk images". In: *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM. 2009, p. 7.

[83] Changyeon Jo, Erik Gustafsson, Jeongseok Son, and Bernhard Egger. "Efficient live migration of virtual machines using shared storage". In: *ACM Sigplan Notices*. Vol. 48. 7. ACM. 2013, pp. 41–50.

[84] Tae Seung Kang, Maurício Tsugawa, Andréa Matsunaga, Takahiro Hirofuchi, and José AB Fortes. "Design and implementation of middleware for cloud disaster recovery via virtual machine migration management". In: *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE Computer Society. 2014, pp. 166–175.

[85] Tae Seung Kang, Mauricio Tsugawa, Jose Fortes, and Takahiro Hirofuchi. "Reducing the migration times of multiple VMs on WANs using a feedback controller". In: *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE. 2013, pp. 1480–1489.

[86] Jihun Kim, Dongju Chae, Jangwoo Kim, and Jong Kim. "Guide-copy: fast and silent migration of virtual machine for datacenters". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM. 2013, p. 66.

[87] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. "kvm: the Linux virtual machine monitor". In: *Proceedings of the Linux symposium*. Vol. 1. 2007, pp. 225–230.

[88] Jacob Faber Kloster, Jesper Kristensen, Arne Mejlholm, and Gerd Behrmann. "On the feasibility of memory sharing: Content-based page sharing in the xen virtual machine monitor". In: (2006).

[89] Panagiotis Kokkinos, Dimitris Kalogeras, Anna Levin, and Emmanouel Varvarigos. "Survey: Live Migration and Disaster Recovery over Long-Distance Networks". In: *ACM Computing Surveys (CSUR)* 49.2 (2016), p. 26.

[90] Akane Koto, Hiroshi Yamada, Kei Ohmura, and Kenji Kono. "Towards unobtrusive VM live migration for cloud computing platforms". In: *Proceedings of the Asia-Pacific Workshop on Systems*. ACM. 2012, p. 7.

[91] Adlen Ksentini, Tarik Taleb, and Min Chen. "A Markov decision process-based service migration procedure for follow me cloud". In: *Communications (ICC), 2014 IEEE International Conference on*. IEEE. 2014, pp. 1350–1354.

[92] Sanjay Kumar and Karsten Schwan. "Netchannel: a VMM-level mechanism for continuous, transparentdevice access during VM migration". In: *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM. 2008, pp. 31–40.

[93] Nicholas S Lemak. *Data mover*. US Patent 4,296,465. 1981.

[94] Mingyu Li, Mian Zheng, and Xiaohui Hu. "Template-based memory deduplication method for inter-data center live migration of virtual machines". In: *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE. 2014, pp. 127–134.

[95] Yi-Bing Lin, Chien-Chun Huang-Fu, and Nabil Alrajeh. "Predicting human movement based on telecom's handoff in mobile networks". In: *IEEE Transactions on Mobile Computing* 12.6 (2013), pp. 1236–1241.

[96] Haikun Liu and Bingsheng He. "Vmbuddies: Coordinating live migration of multi-tier applications in cloud environments". In: *IEEE Transactions on Parallel and Distributed Systems* 26.4 (2015), pp. 1192–1205.

[97] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen 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*. ACM. 2009, pp. 101–110.

[98] Haikun Liu, Hai Jin, Xiaofei Liao, Chen Yu, and Cheng-Zhong Xu. “Live virtual machine migration via asynchronous replication and state synchronization”. In: *IEEE Transactions on Parallel and Distributed Systems* 22.12 (2011), pp. 1986–1999.

[99] Pengcheng Liu, Ziye Yang, Xiang Song, et al. “Heterogeneous live migration of virtual machines”. In: *International Workshop on Virtualization Technology (IWVT'08)*. 2008.

[100] Zhaobin Liu, Wenyu Qu, Weijiang Liu, and Keqiu Li. “Xen live migration with slowdown scheduling algorithm”. In: *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2010 International Conference on*. IEEE. 2010, pp. 215–221.

[101] Marin Lopez and PTA Arturo Garcia Ares. “The network block device”. In: *Linux Journal* 2000.73es (2000), p. 40.

[102] Tao Lu, Ping Huang, Morgan Stuart, et al. “Successor: Proactive cache warm-up of destination hosts in virtual machine migration contexts”. In: *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE. 2016, pp. 1–9.

[103] Yuqian Lu, Xun Xu, and Jenny Xu. “Development of a hybrid manufacturing cloud”. In: *Journal of Manufacturing Systems* 33.4 (2014), pp. 551–566.

[104] Yingwei Luo, Binbin Zhang, Xiaolin Wang, et al. “Live and incremental whole-system migration of virtual machines using block-bitmap”. In: *Cluster Computing, 2008 IEEE International Conference on*. IEEE. 2008, pp. 99–106.

[105] Pavel Mach and Zdenek Becvar. “Mobile edge computing: A survey on architecture and computation offloading”. In: *IEEE Communications Surveys & Tutorials* 19 (3 2017).

[106] Andrew Machen, Shiqiang Wang, Kin K Leung, Bong Jun Ko, and Theodoros Salonidis. “Live Service Migration in Mobile Edge Clouds”. In: *arXiv preprint arXiv:1706.04118* (2017).

[107] Vijay Mann, Akanksha Gupta, Partha Dutta, et al. “Remedy: Network-aware steady state VM management for data centers”. In: *NETWORKING 2012* (2012), pp. 190–204.

[108] Ali Mashtizadeh, Emré Celebi, Tal Garfinkel, Min Cai, et al. “The design and evolution of live storage migration in VMware ESX”. In: *Usenix Atc*. Vol. 11. 2011, pp. 1–14.

[109] Ali José Mashtizadeh, Min Cai, Gabriel Tarasuk-Levin, et al. “XvMotion: Unified Virtual Machine Migration over Long Distance.” In: *USENIX Annual Technical Conference*. 2014, pp. 97–108.

[110] Tim Mather, Subra Kumaraswamy, and Shahed Latif. *Cloud security and privacy: an enterprise perspective on risks and compliance*. " O'Reilly Media, Inc.", 2009.

[111] Nimrod Megiddo and Dharmendra S Modha. “ARC: A Self-Tuning, Low Overhead Replacement Cache.” In: *FAST*. Vol. 3. 2003, pp. 115–130.

[112] Grzegorz Miłós, Derek G Murray, Steven Hand, and Michael A Fetterman. “Satori: Enlightened page sharing”. In: *Proceedings of the 2009 conference on USENIX Annual technical conference*. 2009, pp. 1–1.

[113] Andrey Mirkin, Alexey Kuznetsov, and Kir Kolyshkin. “Containers checkpointing and live migration”. In: *Proceedings of the Linux Symposium*. Vol. 2. 2008, pp. 85–90.

[114] *MPTCP*. Available: `http://multipath-tcp.org/`, Acessed: September. 2017.

[115] Alan Murphy. *Enabling long distance live migration with F5 and VMware vMotion*. 2011.

[116] Apollinaire Nadembega, Abdelhakim Senhaji Hafid, and Ronald Brisebois. "Mobility prediction model-based service migration procedure for follow me cloud to support QoS and QoE". In: *Communications (ICC), 2016 IEEE International Conference on*. IEEE. 2016, pp. 1–6.

[117] Senthil Nathan, Purushottam Kulkarni, and Umesh Bellur. "Resource availability based performance benchmarking of virtual machine migrations". In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ACM. 2013, pp. 387–398.

[118] Mark Nelson and Jean-Loup Gailly. *The data compression book*. Vol. 2. M&t Books New York, 1996.

[119] Michael Nelson, Beng-Hong Lim, Greg Hutchins, et al. "Fast Transparent Migration for Virtual Machines." In: *USENIX Annual technical conference, general track*. 2005, pp. 391–394.

[120] Chun-Ho Ng, Mingcao Ma, Tsz-Yeung Wong, Patrick PC Lee, and John Lui. "Live deduplication storage of virtual machine images in an open-source cloud". In: *Proceedings of the 12th International Middleware Conference*. International Federation for Information Processing. 2011, pp. 80–99.

[121] Bogdan Nicolae and Franck Cappello. "A hybrid local storage transfer scheme for live migration of I/O intensive workloads". In: *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*. ACM. 2012, pp. 85–96.

[122] MFXJ Oberhumer. "LZO real-time data compression library". In: *User manual for LZO version 0.28, URL: http://www. infosys. tuwien. ac. at/Staff/lux/marco/lzo. html (February 1997)* (2005).

[123] Justice Opara-Martins, Reza Sahandi, and Feng Tian. "Critical review of vendor lock-in and its impact on adoption of cloud computing". In: *Information Society (i-Society), 2014 International Conference on*. IEEE. 2014, pp. 92–97.

[124] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, et al. "Mcep: A mobility-aware complex event processing system". In: *ACM Transactions on Internet Technology (TOIT)* 14.1 (2014), p. 6.

[125] Beate Ottenwälder, Boris Koldehofe, Kurt Rothermel, and Umakishore Ramachandran. "Migcep: Operator migration for mobility driven distributed complex event processing". In: *Proceedings of the 7th ACM international conference on Distributed event-based systems*. ACM. 2013, pp. 183–194.

[126] Pradeep Padala, Xiaoyun Zhu, Zhikui Wang, Sharad Singhal, Kang G Shin, et al. "Performance evaluation of virtualization technologies for server consolidation". In: *HP Labs Tec. Report* (2007).

[127] Eunbyung Park, Bernhard Egger, and Jaejin Lee. "Fast and space-efficient virtual machine checkpointing". In: *ACM SIGPLAN Notices*. Vol. 46. 7. ACM. 2011, pp. 75–86.

[128] Dick Pountain. "Run-length encoding." In: *Byte* 12.6 (1987), pp. 317–319.

[129] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. "The bittorrent p2p file-sharing system: Measurements and analysis". In: *International Workshop on Peer-to-Peer Systems*. Springer. 2005, pp. 205–216.

[130] *Practical Guide to Hybrid Cloud Computing*. Available: `http://www.cloud-council.org/deliverables/CSCC-Practical-Guide-to-Hybrid-Cloud-Computing.pdf`.

[131] *Programming Intel QuickAssist Technology Hardware Accelerators for Optimal Performance*. Available: `https://01.org/sites/default/files/page/332125_002_0.pdf`. 2015.

[132] *QCOW2 backing files & overlays*. 2012. URL: `https://kashyapc.fedorapeople.org/virt/lc-2012/snapshots-handout.html`.

[133] Yuqing Qiu, Chung-Horng Lung, Samuel Ajila, and Pradeep Srivastava. "LXC Container Migration in Cloudlets under Multipath TCP". In: *Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual*. Vol. 2. IEEE. 2017, pp. 31–36.

[134] Michael O Rabin et al. *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[135] Bane Raman Raghunath and B Annappa. "Virtual Machine Migration Triggering using Application Workload Prediction". In: *Procedia Computer Science* 54 (2015), pp. 167–176.

[136] KK Ramakrishnan, Prashant Shenoy, and Jacobus Van der Merwe. "Live data center migration across WANs: a robust cooperative context aware approach". In: *Proceedings of the 2007 SIGCOMM workshop on Internet network management*. ACM. 2007, pp. 262–267.

[137] Pierre Riteau, Christine Morin, and Thierry Priol. "Shrinker: Efficient wide-area live virtual machine migration using distributed content-based addressing". PhD thesis. INRIA, 2010.

[138] Pierre Riteau, Christine Morin, and Thierry Priol. "Shrinker: Improving live migration of virtual clusters over wans with distributed data deduplication and content-based addressing". In: *European Conference on Parallel Processing*. Springer. 2011, pp. 431–442.

[139] Benny Rochwerger, David Breitgand, Eliezer Levy, et al. "The reservoir model and architecture for open federated cloud computing". In: *IBM Journal of Research and Development* 53.4 (2009), pp. 4–1.

[140] *RUBBoS*. `http://jmob.ow2.org/rubbos.html`, Accessed: March, 2016.

[141] Chris Ruemmler and John Wilkes. *UNIX disk access patterns*. Hewlett-Packard Laboratories, 1992.

[142] Shashank Sahni and Vasudeva Varma. "A hybrid approach to live migration of virtual machines". In: *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on*. IEEE. 2012, pp. 1–5.

[143] Tusher Kumer Sarker and Maolin Tang. "Performance-driven live migration of multiple virtual machines in datacenters". In: *Granular Computing (GrC), 2013 IEEE International Conference on*. IEEE. 2013, pp. 253–258.

[144] Mahadev Satyanarayanan, Zhuo Chen, Kiryong Ha, et al. "Cloudlets: at the leading edge of mobile-cloud convergence". In: *Mobile Computing, Applications and Services (MobiCASE), 2014 6th International Conference on*. IEEE. 2014, pp. 1–9.

[145] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. "The case for vm-based cloudlets in mobile computing". In: *IEEE pervasive Computing* 8.4 (2009).

[146] Benjamin Satzger, Waldemar Hummer, Christian Inzinger, Philipp Leitner, and Schahram Dustdar. "Winds of change: From vendor lock-in to the meta cloud". In: *IEEE internet computing* 17.1 (2013), pp. 69–73.

[147] Enrique Saurez, Kirak Hong, Dave Lillethun, Umakishore Ramachandran, and Beate Ottenwälder. "Incremental deployment and migration of geo-distributed situation awareness applications in the fog". In: *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM. 2016, pp. 258–269.

[148] Mathijs Jeroen Scheepers. "Virtualization and containerization of application infrastructure: A comparison". In: *21st Twente Student Conference on IT*. 2014, pp. 1–7.

[149] Stefano Secci, Patrick Raad, and Pascal Gallard. "Linking virtual machine mobility to user mobility". In: *IEEE Transactions on Network and Service Management* 13.4 (2016), pp. 927–940.

[150] Zhiming Shen, Qin Jia, Gur-Eyal Sela, et al. “Follow the Sun through the Clouds: Application Migration for Geographically Shifting Workloads”. In: *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM. 2016, pp. 141–154.

[151] Aidan Shribman and Benoit Hudzia. “Pre-Copy and post-copy VM live migration for memory intensive applications”. In: *European Conference on Parallel Processing*. Springer. 2012, pp. 539–547.

[152] Vivek Shrivastava, Petros Zerfos, Kang-Won Lee, et al. “Application-aware virtual machine migration in data centers”. In: *INFOCOM, 2011 Proceedings IEEE*. IEEE. 2011, pp. 66–70.

[153] Vasilios A Siris and Dimitrios Kalyvas. “Enhancing mobile data offloading with mobility prediction and prefetching”. In: *ACM SIGMOBILE Mobile Computing and Communications Review* 17.1 (2013), pp. 22–29.

[154] Wee-Seng Soh and Hyong S Kim. “Dynamic bandwidth reservation in cellular networks using road topology based mobility predictions”. In: vol. 4. Proceedings of INFOCOM 2004. IEEE. 2004, pp. 2766–2777.

[155] Borja Sotomayor, Rubén S Montero, Ignacio M Llorente, and Ian Foster. “Virtual infrastructure management in private and hybrid clouds”. In: *IEEE Internet computing* 13.5 (2009).

[156] Shivani Sud, Roy Want, Trevor Pering, et al. “Dynamic migration of computation through virtualization of the mobile platform”. In: *Mobile Networks and Applications* 17.2 (2012), pp. 206–215.

[157] Xiang Sun and Nirwan Ansari. “Avaptive Avatar Handoff in the Cloudlet Network”. In: *IEEE Transactions on Cloud Computing* (2017).

[158] Xiang Sun and Nirwan Ansari. “Primal: Profit maximization avatar placement for mobile edge computing”. In: *Communications (ICC), 2016 IEEE International Conference on*. IEEE. 2016, pp. 1–6.

[159] Petter Svärd, Benoit Hudzia, Johan Tordsson, and Erik Elmroth. “Evaluation of delta compression techniques for efficient live migration of large virtual machines”. In: *ACM Sigplan Notices* 46.7 (2011), pp. 111–120.

[160] Petter Svärd, Johan Tordsson, Benoit Hudzia, and Erik Elmroth. “High performance live migration through dynamic page transfer reordering and compression”. In: *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE. 2011, pp. 542–548.

[161] Kazushi Takahashi, Koichi Sasada, and Takahiro Hirofuchi. “A fast virtual machine storage migration technique using data deduplication”. In: *Proceedings of CLOUD COMPUTING* (2012), pp. 57–64.

[162] Tarik Taleb and Adlen Ksentini. “An analytical model for follow me cloud”. In: *Global Communications Conference (GLOBECOM), 2013 IEEE*. IEEE. 2013, pp. 1291–1296.

[163] Tarik Taleb and Adlen Ksentini. “Follow me cloud: interworking federated clouds and distributed mobile networks”. In: *IEEE Network* 27.5 (2013), pp. 12–19.

[164] Tarik Taleb, Peer Hasselmeyer, and Faisal Ghias Mir. “Follow-me cloud: An OpenFlow-based implementation”. In: *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE. 2013, pp. 240–245.

[165] Chunqiang Tang. “FVD: A High-Performance Virtual Machine Image Format for Cloud.” In: *USENIX Annual Technical Conference*. 2011, p. 2.

[166] Fikirte Teka, Chung-Horng Lung, and Samuel A Ajila. “Nearby live virtual machine migration using cloudlets and multipath TCP”. In: *Journal of Cloud Computing* 5.1 (2016), p. 12.

[167] Fikirte Teka, Chung-Horng Lung, and Samuel Ajila. “Seamless Live Virtual Machine Migration with Cloudlets and Multipath TCP”. In: *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*. Vol. 2. IEEE. 2015, pp. 607–616.

[168] *TPC-W*. `http://www.tpc.org/tpcw/`. December.2017.

[169] Franco Travostino, Paul Daspit, Leon Gommans, et al. “Seamless live migration of virtual machines over the MAN/WAN”. In: *Future Generation Computer Systems* 22.8 (2006), pp. 901–907.

[170] Thomas Treutner and Helmut Hlavacs. “Service level management for iterative pre-copy live migration”. In: *Proceedings of the 8th International Conference on Network and Service Management*. International Federation for Information Processing. 2012, pp. 252–256.

[171] Konstantinos Tsakalozos, Vasilis Verroios, Mema Roussopoulos, and Alex Delis. “Live VM Migration under Time-Constrains in Share-Nothing IaaS-Clouds”. In: *IEEE Transactions on Parallel and Distributed Systems* (2017).

[172] Konstantinos Tsakalozos, Vasilis Verroios, Mema Roussopoulos, and Alex Delis. “Time-constrained live VM migration in share-nothing IaaS-clouds”. In: *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE. 2014, pp. 56–63.

[173] *Virtual Machine Mobility with Vmware VMotion and Cisco Data Center Interconnect Technologies*. Available: `http://www.cisco.com/c/dam/en/us/solutions/collateral/data-center-virtualization/data-center-virtualization/white_paper_c11-557822.pdf`.

[174] *VMware Inc.* Available: `http://www.vmware.com/`, Acessed: Dec. 2016.

[175] Carl A Waldspurger. “Memory resource management in VMware ESX server”. In: *ACM SIGOPS Operating Systems Review* 36.SI (2002), pp. 181–194.

[176] Huandong Wang, Yong Li, Ying Zhang, and Depeng Jin. “Virtual machine migration planning in software-defined networks”. In: *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE. 2015, pp. 487–495.

[177] Lizhe Wang, Jie Tao, Marcel Kunze, et al. “Scientific cloud computing: Early definition and experience”. In: *High Performance Computing and Communications, 2008. HPCC’08. 10th IEEE International Conference on*. Ieee. 2008, pp. 825–830.

[178] Shiqiang Wang, Rahul Urgaonkar, Ting He, et al. “Mobility-induced service migration in mobile micro-clouds”. In: *Military Communications Conference (MILCOM), 2014 IEEE*. IEEE. 2014, pp. 835–840.

[179] Christof Weinhardt, Arun Anandasivam, Benjamin Blau, et al. “Cloud computing–a classification, business models, and research directions”. In: *Business & Information Systems Engineering* 1.5 (2009), pp. 391–399.

[180] Aaron Weiss. “Computing in the clouds”. In: *Computing* 16 (2007).

[181] Paul R Wilson, Scott F Kaplan, and Yannis Smaragdakis. “The Case for Compressed Caching in Virtual Memory Systems.” In: *USENIX Annual Technical Conference, General Track*. 1999, pp. 101–116.

[182] Timothy Wood. “Improving data center resource management, deployment, and availability with virtualization”. PhD thesis. University of Massachusetts Amherst, 2011.

[183] Timothy Wood, KK Ramakrishnan, Prashant Shenoy, and Jacobus Van der Merwe. “CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines”. In: *ACM Sigplan Notices*. Vol. 46. 7. ACM. 2011, pp. 121–132.

[184] Jing Xia, Deming Pang, Zhiping Cai, Ming Xu, and Gang Hu. “Reasonably Migrating Virtual Machine in NFV-Featured Networks”. In: *Computer and Information Technology (CIT), 2016 IEEE International Conference on*. IEEE. 2016, pp. 361–366.

[185] Fei Xu, Fangming Liu, Linghui Liu, et al. “iaware: Making live migration of virtual machines interference-aware in the cloud”. In: *IEEE Transactions on Computers* 63.12 (2014), pp. 3012–3025.

[186] Shugong Xu and Tarek Saadawi. “Performance evaluation of TCP algorithms in multi-hop wireless packet networks”. In: *Wireless communications and mobile computing* 2.1 (2002), pp. 85–100.

[187] Xiaolin Xu, Hai Jin, Song Wu, and Yihong Wang. “Rethink the storage of virtual machine images in clouds”. In: *Future Generation Computer Systems* 50 (2015), pp. 75–86.

[188] Kejiang Ye, Xiaohong Jiang, Dawei Huang, Jianhai Chen, and Bei Wang. “Live migration of multiple virtual machines with resource reservation in cloud computing environments”. In: *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE. 2011, pp. 267–274.

[189] Lamia Youseff, Maria Butrico, and Dilma Da Silva. “Toward a unified ontology of cloud computing”. In: *Grid Computing Environments Workshop, 2008. GCE’08*. IEEE. 2008, pp. 1–10.

[190] Zainab R Zaidi and Brian L Mark. “Real-time mobility tracking algorithms for cellular networks based on Kalman filtering”. In: *IEEE Transactions on Mobile Computing* 4.2 (2005), pp. 195–208.

[191] Daqiang Zhang, Min Chen, Mohsen Guizani, Haoyi Xiong, and Daqing Zhang. “Mobility prediction in telecom cloud using mobile calls”. In: *IEEE Wireless Communications* 21.1 (2014), pp. 26–32.

[192] Fei Zhang, Xiaoming Fu, and Ramin Yahyapour. “CBase: A New Paradigm for Fast Virtual Machine Migration across Data Centers”. In: *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press. 2017, pp. 284–293.

[193] Fei Zhang, Bo Zhao, Xiaoming Fu, and Ramin Yahyapour. “Controlling migration performance of virtual machines according to user’s requirements”. In: Proceedings of CAN’17: Cloud-Assisted Networking Workshop. ACM, 2017.

[194] Fei Zhang, Xiaoming Fu, and Ramin Yahyapour. “LayerMover: Storage Migration of Virtual Machine across Data Centers Based on Three-Layer Image Structure”. In: *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on*. IEEE. 2016, pp. 400–405.

[195] Jinshi Zhang, Liang Li, and Dong Wang. “Optimizing VNF live migration via para-virtualization driver and QuickAssist technology”. In: *Communications (ICC), 2017 IEEE International Conference on*. IEEE. 2017, pp. 1–6.

[196] Qi Zhang, Lu Cheng, and Raouf Boutaba. “Cloud computing: state-of-the-art and research challenges”. In: *Journal of internet services and applications* 1.1 (2010), pp. 7–18.

[197] Xiang Zhang, Zhigang Huo, Jie Ma, and Dan Meng. “Exploiting data deduplication to accelerate live virtual machine migration”. In: *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*. IEEE. 2010, pp. 88–96.

[198] Zhaoning Zhang, Ziyang Li, Kui Wu, et al. “VMThunder: fast provisioning of large-scale virtual machine clusters”. In: *IEEE Transactions on Parallel and Distributed Systems* 25.12 (2014), pp. 3328–3338.

[199] Zhenzhong Zhang, Limin Xiao, Mingfa Zhu, and Li Ruan. “Mvmotion: a metadata based virtual machine migration in cloud”. In: *Cluster Computing* 17.2 (2014), pp. 441–452.

[200] Ming Zhao and Renato J Figueiredo. “Experimental study of virtual machine migration in support of reservation of cluster resources”. In: *Virtualization Technology in Distributed Computing (VTDC), 2007 Second International Workshop on*. IEEE. 2007, pp. 1–8.

[201] Jie Zheng, Tze Sing Eugene Ng, Kunwadee Sripanidkulchai, and Zhaolei Liu. “Comma: Coordinating the migration of multi-tier applications”. In: *ACM SIGPLAN Notices*. Vol. 49. 7. ACM. 2014, pp. 153–164.

[202] Jie Zheng, TS Eugene Ng, Kunwadee Sripanidkulchai, and Zhaolei Liu. “Pacer: A progress management system for live virtual machine migration in cloud computing”. In: *IEEE transactions on network and service management* 10.4 (2013), pp. 369–382.

[203] Jie Zheng, Tze Sing Eugene Ng, and Kunwadee Sripanidkulchai. “Workload-aware live storage migration for clouds”. In: *ACM Sigplan Notices*. Vol. 46. 7. ACM. 2011, pp. 133–144.

[204] Mian Zheng and Xiaohui Hu. “Template-based migration between data centers using distributed hash tables”. In: *Fuzzy Systems and Knowledge Discovery (FSKD), 2015 12th International Conference on*. IEEE. 2015, pp. 2443–2447.

[205] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. “Mining interesting locations and travel sequences from GPS trajectories”. In: Proceedings of the 18th international conference on World wide web. ACM. 2009, pp. 791–800.

[206] Ruijin Zhou, Fang Liu, Chao Li, and Tao Li. “Optimizing virtual machine live storage migration in heterogeneous storage environment”. In: *ACM SIGPLAN Notices*. Vol. 48. 7. ACM. 2013, pp. 73–84.

List of Acronyms

ARC Adaptive Replacement Cache

BSC Base Station Controller

CBC Characteristic-Based Compression

CBPS Content Based Page Share

COR Copy-On-Read

COW Copy-On-Write

CPU Central Processing Unit

CR/TR Checkpointing/Recovery and Trace/Replay

DHT Distributed Hash Table

ETSI European Telecommunications Standards Institute

FC Fibre Channel

HDD Hard Disk Drive

HPC High Performance Computing

IaaS Infrastructure as a Service

IoT Internet of Thing

LAN Local Area Network

LRU Least Recently Used

MAN Metropolitan Area Network

MDP Markov Decision Process

MEC Mobile Edge Computing

MMU Memory Management Unit

MPTCP Multipath TCP

NAS Network Attached Storage

NBD Network Block Device

NFS Network File System

NFV Network Function Virtualization

NIC Network Interface Card

OS Operating System

QAT QuickAssist Technology

QoS Quality of Service

RAN Radio Access Network

RDMA Remote Direct Memory Access

SAN Storage Area Network

SDPS Stun During Page Send

SLA Service Level Agreement

SSD Solid State Drive

UD User Data

UE User Equipment

UUID Universally Unique Identifier

VM Virtual Machine

WAN Wide Area Network

WE Working Environment

WWS Writable Working Set

XBRLE XORed Binary Run Length Encoding

List of Figures

List of Tables