Journal of Parallel and Distributed Computing 131 (2019) 55–68

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Performance evaluation of live virtual machine migration in SDN-enabled cloud data centers

TianZhang He [a,*], Adel N. Toosi [b], Rajkumar Buyya [a]

[a] Clouds Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne Parkville, VIC 3010, Australia

[b] Faculty of Information Technology, Monash University Clayton, VIC 3800, Australia

HIGHLIGHTS

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

ARTICLE INFO

Article history:
Received 4 July 2018
Received in revised form 1 February 2019
Accepted 17 April 2019
Available online 26 April 2019

MSC:
00-01
99-00

Keywords:
Live VM migration
Software-Defined Networking
Cloud computing
Virtual machine
Performance measures
OpenStack
OpenDaylight

ABSTRACT

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

1. Introduction

With the rapid adoption of cloud computing environments for hosting a variety of applications such as Web, Virtual Reality, scientific computing, and big data, the need for delivering cloud services with Quality of Service (QoS) guarantees is becoming critical. For cloud data center management, it is important to prevent the violation of Service Level Agreement (SLA) and maintain the QoS in heterogeneous environments with different application contexts. Therefore, there has been a lot of focus on optimizing the service latency and energy efficiency dynamically in order to benefit both cloud computing tenants and providers. Virtual Machines (VMs), as one of the major virtualization technologies to host cloud services, can share computing and networking resources. In order to alleviate SLA violation and meet the QoS guarantees, the placement of VMs needs to be optimized constantly in the dynamic environment. Live VM migration is the key technology to relocate running VMs between physical hosts without disrupting the VMs' availability [8]. Thus, in SDN-enabled data centers, live VM migration as a dynamic management tool facilities various objectives of the resource scheduling [12,22,28], such as load balancing, cloud bursting, resource overbooking, and energy-saving strategy, fault tolerance, scheduled maintenance as well as evacuating VMs to other data centers before the incidents like earthquake and flooding which require VM location adjustment.

The **live VM migration** technologies can be categorized into the **pre-copy memory** [8] and **post-copy memory migration**

* Corresponding author.
E-mail addresses: tianzhangh@student.unimelb.edu.au (T. He), adel.n.toosi@monash.edu (A. N. Toosi), rbuyya@unimelb.edu.au (R. Buyya).

https://doi.org/10.1016/j.jpdc.2019.04.014
0743-7315/

[15]. During the pre-copy live migration, the Virtual Machine Monitor (VMM), such as KVM and Xen, iteratively copy memory (dirty pages produced in the last round) from the running VM at the source host to the VM container at the target host. However, the post-copy live migration first suspends the VM at the source host and resumes the VM at the target host by migrating a minimal subset of VM execution state. At the same time, the source VM still pro-actively pushing the remained pages to the resumed VM. A page-fault happens when the VM attempt to access an un-transferred page which can be solved by fetching the pages from the source VM. However, in many circumstances such as in Wide Area Network (WAN) environment (inter-data centers and between edge and core cloud computing), and some production Data Centers where some servers do not share the same storage system, there is no Network File System (NFS) between the source and the target hosts to share ephemeral disks. The live migration with block storage, also called **block live migration**, is used by combining memory migration with live disk migration. Besides the memory migration of live VM migration, the live disk migration is used [5,23] to transfer the ephemeral disk of the VM instance to the target host.

The goal of this paper is to tackle an important aspect in the field of VM migration, namely understanding the parameters that affect the performance of live VM migration in SDN-enabled cloud computing. The performance of live migration could be evaluated by measuring three metrics:

- **Migration time** is the time duration from the initialization of the pre-migration process on the source host to the successful completion of the post-migration process on both hosts.
- **Downtime** refers to the duration that the VM is suspended due to the stop-and-copy, Commitment and Activation phase. From the client perspective, the service is unavailable.
- **Transferred data** is the amount of data transferred between the source and destination host during the migration process.

There are continuous efforts to improve live VM migration, such as improving the performance of live migration algorithm [26,29], modeling for better prediction of the cost [6,20], network-aware live migration to alleviate the influence of migration on SLA and application QoS [9,10,36], optimizing the multiple live VM migration planning [7,11,30,35], and benchmarking the live migration effects on applications [17,34]. Nonetheless, many parameters, such as downtime adjustment and non-network overheads, that affect the live migration time and downtime are neglected to a large extent. During a live VM migration, the downtime threshold for the last memory-copy iteration could be changed as time elapses. This will affect the memory-copy iteration rounds, which leads to different migration time and downtime. Computing overheads of live VM migration can also constitute a large portion of total migration time, which will affect the performance of multiple VM migrations.

On the other hand, some work focus on the live VM migration in Software-Defined Networking (SDN) scenarios [11,22,35]. By virtualizing the network resources, we could use SDN to dynamically allocate bandwidth to services and control the route of network flows. Due to the centralized controller, SDN can provide a global view of the network topology, states of switches, and statistics on the links (bandwidth and latency). Based on the information, orchestrator can calculate the 'best' path for each flow and call SDN controller Northbound APIs to push the forwarding rules to each switch in the path. However, the latencies of the flow entry installation on the switch and the communication between SDN controller and switches could impact the traffic engineering performance in the SDN-enabled cloud data centers. Thus, the scheduling update rate of choosing the 'best' path will affect the live migration traffic.

Moreover, although some work [17,34] focus on the impacts of live migration on the cloud services, such as multi-tier web application, the worst-case response time pattern as well as the technologies, such as hybrid post-copy and auto-convergence, for a successful live migration need to be investigated further. Hybrid post-copy (H-PC) [15] is the strategy that combines pre-copy and post-copy migration. The post-copy mode will be activated after the certain pre-copy phase where most of the memory has been transferred. Based on the CPU throttling, Auto-convergence (AC) [14] will decrease the workload where the memory write speed is relative to the CPU executing speed.

We evaluate the live migration time, downtime, and total transferred data using OpenStack [25] as the cloud computing platform. OpenStack uses the pre-copy live migration with the default driver Libvirt (virtualization API) [3]. Our study is fundamentally useful to resource scheduling, such as energy-saving strategy, load balancing, and fault tolerant, driven by SLA. The **contributions** are fourfold, and are summarized as follows:

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

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

2. Related work

Clark et al. [8] firstly proposed the live VM migration comparing to the naive stop-and-copy method. During the iterative memory copy phase of live migration implemented in Xen virtualization platform, rapid dirtying pages which updated extremely frequently, called Writable Working Set (WWS), was introduced. These pages will not be transmitted to the destination host in the iteration round in order to reduce the total migration time and transferred data. In addition, the authors elaborated the implementation issues and features with regard to the managed migration (migration daemons of Xen in host and destination hosts), self migration (implementation the mechanism within the

OS), dynamic rate-limiting for each iteration round, rapid page dirtying, and paravirtualized optimizations (stunning rogue process, i.e. limit write faults of each process, and freeing page cache pages, i.e. reclaiming back cold buffer cache pages). Although pre-copy migration is widely used in various virtualization platforms, such as Xen, QEMU/KVM, VMWare, it is worth noting that migration algorithms and performance of different hypervisors are different in terms of dirty pages detection and transmission and stop-and-copy threshold [16]. For instance, the page skip (WWS) mechanism does not be implemented in KVM.

In order to alleviate the overheads caused by live VM migrations, the prediction model is required to estimate the live migration performance in advance. Akoush et al. [6] proposed a model to estimate the migration time and downtime of live VM migration based on the two main functions of migration, i.e. peek and clean. The peek function returns the dirty bitmap and the clean function returns the dirty pages and resets them to clean state. They used both average dirty page rate (AVG) and history based page dirty rate (HIST) in their prediction algorithms. The HIST model could capture the variability of live migration and help to decide the moment at which migration begins to minimize the migration cost. Moreover, Liu et al. [21] introduced the rapid page dirtying in its migration performance prediction model. In order to obtain a more accurate prediction model, the authors refined the previous prediction model of migration performance by estimating the size of WWS. Based on the observation, it is an approximated proportional size of the total dirty pages in each iterative memory copy round with regard to previous iteration time and dirty pages rate. The authors also proposed an energy consumption model of live migration based on the linear regression between total transferred data and measured energy consumption. The synthesized cost for migration decision is based on the estimated values of downtime, migration time, transferred data, and energy cost. Furthermore, based on prediction model of migration cost, different migration strategies for load balancing, fault toleration, and server consolidation are proposed [20]. The algorithms choose the proper migration candidates in order to minimize the total migration cost while satisfying the requirements of rescheduling algorithms. Contrary to their work we focus on the mechanism and performance of proposed parameters and corresponding models.

Prediction models of live migration which assume a static downtime threshold [6,20,21] or constant dirty page rate [35] cannot reflect the real migration time and downtime in OpenStack. The downtime threshold in OpenStack uses a static adjustment algorithm. It is increased monotonically with a certain time interval and steps during the migration in order to reduce the migration time. A misconfigured downtime configuration will lead to a poor performance of live migration, such as unstable downtime which results in the SLA violation and a long-time migration that degrades the network performance. Therefore, in order to dynamically set optimal configurations, we need to have a better understanding of the relationship between **downtime adjustment configurations** and migration performance in OpenStack.

Planning of the sequential and parallel migration in intra and inter-data centers to optimize the server evacuation time and minimize the influence of live migration has attracted interest recently [7,11,35]. However, they only focus on the network aspect of multiple live migration planning to decide the sequence of sequential and concurrent live migration in order to minimize the migration duration. As mentioned in [6], the total migration time includes pre-migration, pre-copy phase, stop-and-copy phase and post-migration overheads. The most proportion of migration time could be the operation overheads. Therefore, in order to have a better algorithm of the multiple VM evacuation planning, we need to pinpoint the impacts of non-network overheads on the **parallel and sequential migration** in the same path.

Moreover, Software-Defined Networking (SDN) [27] as a powerful feature in Cloud computing provides a centralized view of topology and bandwidth on every path. We could flexibly implement network scheduling algorithm and set bandwidth limit for live migration and other application traffics. In a highly dynamic network environment, the 'best' path decided by scheduling algorithm based on an update rate could change frequently. Therefore, not only the bandwidth but traffic pattern, SDN control plan [13] and flow table latency [19] could also affect the live migration performance. Understanding the **SDN latency** in the flow scheduling is very important for achieving better live migration performance.

With different application context, the impacts of live migration on application performance could change dramatically. For instance, the workload of a multi-tier web application with specific write and communication pattern [17,34] is different with the workload in scientific computing applications. The response time should be soft real-time to satisfy the QoS of application. Therefore, network service suffers more from the disruption due to the downtime and the performance degradation due to the live VM migration. As there are few works on this topic, evaluating the live migration effects on the **response time** of different types of network-sensitive applications is desirable. However, current work did not consider the worst-case response time and the situation when the pre-copy migration could not finish in a reasonable time. Thus, we need to evaluate the response time distribution of the web application during the migration, and the impacts of strategies, hybrid post-copy, and auto-convergence, on application response time which perform a successful live migration.

3. System overview

In SDN-enabled data centers, the computing resources are under control of cloud management platform, such as OpenStack, while the networking resources are managed by SDN controller. The management module (orchestrator) coordinates the SDN controller and the OpenStack services by using northbound RESTful APIs to perform VM migration planning in resource scheduling algorithm such as the SLA-aware energy-saving strategy as shown in Fig. 1. In OpenStack, Nova service runs on top of Linux servers as daemons to provide the ability to provision the compute servers. Meanwhile, Neutron component provides 'connectivity as a service' between network interfaces managed by other services like Nova.

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

Therefore, the SDN controller uses OpenFlow [24] protocol through southbound interfaces to manage the forwarding planes on network devices (switches). The open-source virtual switch, Open vSwitch (OVS) [2], provides the virtualization switching

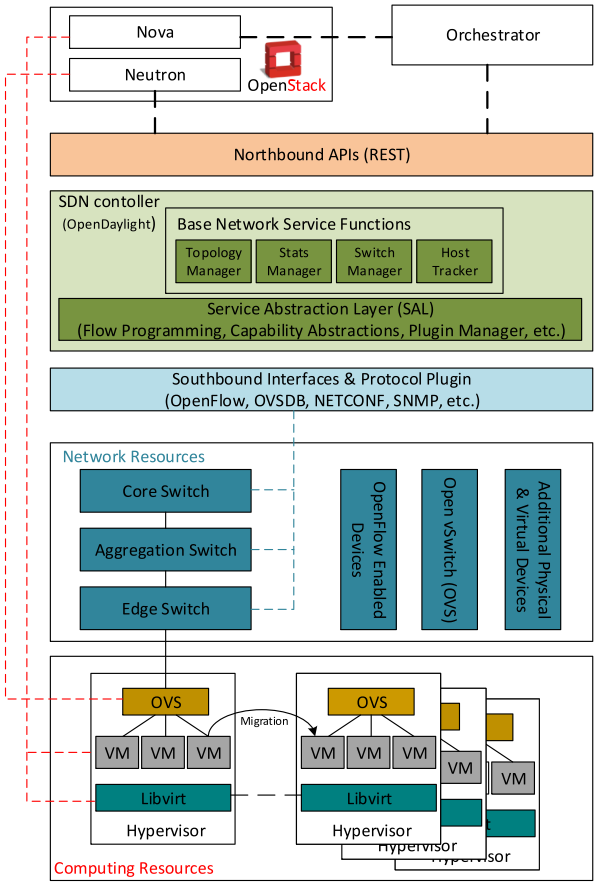


Fig. 1. System overview.

stack supporting OpenFlow and other standard protocols. Therefore, without expensive dedicated switches, we could install OVS in the white box as the OpenFlow switch in SDN-enabled data centers. Based on the link information between OpenFlow devices, the SDN controller calculates the forwarding tables for all network traffics. The OpenFlow switches forward the traffic flow according to the received forwarding tables from SDN controller. It also measures the received and transmitted data size as well as the bandwidth and latency between each other.

3.1. Live migration in OpenStack

In this section, we present the details of block live migration in OpenStack. Providing a comprehensive solution to control the computing and network resources in the datacenter, OpenStack uses Libvirt [3] to manage hosts in order to support different kinds of virtualization. Nova live migration interacts with Neutron to perform the pre- and post-live-migration operations, and uses Libvirt to handle the actual live migration operations. The pre-copy live VM migration is used by default driven by libvirt.

Since libvirt 1.0.3, the QEMU's Network Block Device (NBD) server and "drive-mirror" primitive [5] are used to perform live storage migration (without shared storage setup). Similarly, since VMWare ESX 5.0, it uses VMKernel data mover (DM) and IO mirroring to perform live storage migration [23]. It separates the storage streaming data flows from the instance's RAM and hypervisor's internal state data flows. The disk transmission will perform concurrently with IO mirroring and VM migration. The write operation can be categorized into three types: (1) Into the block has been migrated, the writes will be mirrored to the target.

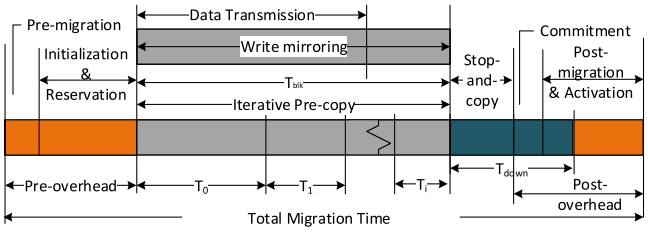


Fig. 2. OpenStack block live migration.

(2) Into the block being migrated, the writes will be sent to the target first and wait in the queue until the region migration finished. (3) Into the block which will be migrated, the writes are issued to the source disk without mirroring. By caching the backing file or instance image when it boot in the Nova compute host, the mirror action could just apply to the top active overlay in the image chain. Thus, the actual disk transmission will be reduced.

Similar to the pre-copy migration described in [8], the block live migration in OpenStack includes 9 steps (Fig. 2):

1. **Pre-live migration** (PreMig): Creates VM's port (VIF) on the target host, updates ports binding, and sets up the logical router with Neutron server.
2. **Initialization** (Init): Preselects the target host to speed the future migration.
3. **Reservation** (Reserv): Target host sets up the temporary share file server; and initializes a container for the reserved resource on the target host.
4. **Disk transmission**: For live storage migration, starts to perform storage migration and synchronizes the disk through IO mirroring.
5. **Iterative pre-copy**: For pre-copy VM migration, sends dirty pages that are modified in the previous iteration round to the target host. The entire RAM is sent in the first round.
6. **Stop-and-copy**: the VM is paused during the last iteration round according to the downtime threshold (remained amount is less than the required).
7. **Commitment** (Commit): Source host gets the commitment of a successfully received instance copy from the target host.
8. **Activation** (Act): Reassigns computing resource to the new VM and delete the old VM on the source host.
9. **Post-live migration** (PostMig): On the target host, updates port state and rebinds the port with Neutron. VIF driver unplugs the VM's port on the source host.

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

Downtime Adjustment Algorithm: Unlike the stop conditions that are used in QEMU or Xen migration algorithm, the downtime threshold in OpenStack live migration increases monotonically in order to minimize the downtime for lower dirty page rate VM while increasing the availability of high dirty page rate VM

migration with a reasonable downtime. The downtime adjustment algorithm used in Libvirt is basically based on three static configuration values (*max_downtime, steps, delay*):

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

For example, the setting tuple (400, 10, 30) means that there will be 10 steps to increase the downtime threshold with 30 s delay for each step up to the 400 ms maximum. With the total 3 GB RAM and Disk data size, the downtime threshold at time t, as $T_{d-thd}(t)$, will be increased at every 90 s starting from 40 ms, i.e. $T_{d-thd}(0) = 40$ ms, $T_{d-thd}(90) = 76$ ms, ..., $T_{d-thd}(900) = 400$ ms, ..., $T_{d-thd}(t > 900) = 400$ ms. The mathematical model of downtime adjustment algorithm is shown in Eq. (9). Although OpenStack only support static downtime adjustment in configuration files, we could use the *virsh* command to interact with the on-going migration based on the elapsed time.

4. Mathematical model

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

4.1. Block live migration

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

$$\begin{aligned} T_{pre} &= \text{PreMig} + \text{Init} + \text{Reserv} \\ T_{post} &= \text{Commit} + \text{Act} + \text{PostMig} \end{aligned} \tag{1}$$

We use D and M to represent the **system disk size** and the **VM memory size**, and let ρ denote the **average compression rate** used in memory compression algorithm [31]. Let ρ' and R' denote the **average disk compression rate** and **mirrored disk write rate**. Let R_i and L_i denote the **average dirty page rate** need to be copied and **bandwidth** in iteration round i. In total n round iterative pre-copy and stop-and-copy stages, T_i denotes the time interval of i_{th} round iteration shown in Fig. 2. Therefore, the **transferred volume** V_i in round i can be calculated as:

$$V_i = \begin{cases} \rho \cdot M & \text{if } i = 0 \\ \rho \cdot T_{i-1} \cdot R_{i-1} & \text{otherwise} \end{cases} \tag{2}$$

As shown in Fig. 2, the time interval of the i_{th} iteration can be calculated as:

$$T_i = \rho \cdot V_i / L_i = \rho \cdot \prod_{j=1}^{i-1} R_j \cdot M \Big/ \prod_{j=0}^{i} L_j \tag{3}$$

In [35], they assume that, when R_i, L_i are constant, the average dirty page rate is not larger than the network bandwidth in every iteration. Let ratio $\sigma = \rho \cdot R/L$. Therefore, $T_i = M \cdot \sigma^i / L$. The total time of iterative memory pre-copy T_{mem} can be calculated as:

$$T_{mem} = \frac{\rho \cdot M}{L} \cdot \frac{1 - \sigma^{n+1}}{1 - \sigma} \tag{4}$$

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

$$T_{blk} \le \rho' \cdot \left(D + R' \cdot T_{blk}\right) / L \tag{5}$$

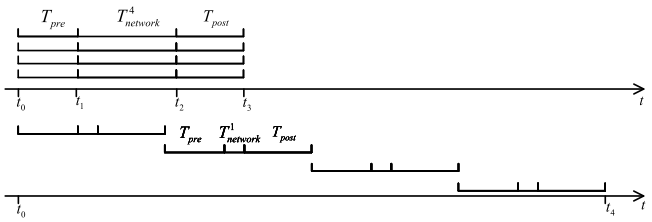


Fig. 3. An example of sequential and parallel migrations.

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

$$T_{blk} \le \frac{\rho' \cdot D}{L - \rho' \cdot R'} \tag{6}$$

For a more accurate T_{blk}, one need to simulate the write behavior based on the actual workload. The network part of block live migration is the maximum value of Eqs. (4) and (6):

$$T_{copy} = Max\{T_{blk}, T_{mem}\} \tag{7}$$

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

$$T_{mig} = T_{pre} + T_{copy} + T_{post} \tag{8}$$

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

$$T_{d-thd}(t) = \lfloor t/(d \cdot (D + M)) \rfloor \cdot (\theta s - \theta)/s^2 + \theta/s \tag{9}$$

The downtime threshold of remained dirty pages accordingly will be

$$V_{d-thd}(t) = T_{d-thd}(t) \cdot L_{n-1} \tag{10}$$

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

The live migration changes to the stop-and-copy phase when remained dirty pages is less than the current threshold, as $V_n \le V_{d-thd}(t)$. Using Eq. (2) in the inequality, the total round of memory iteration can be represented as:

$$n = \left\lceil \log_\sigma \frac{V_{d-thd}(t)}{M} \right\rceil \tag{11}$$

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

4.2. Sequential and parallel migrations

When applying energy-saving policy, hardware maintenance, load balancing or encountering devastating incidents, we need to evacuate part of or all VMs from several physical hosts to others through live VM migrations as soon as possible. In this section, we establish the mathematical model of sequential and parallel live VM migrations which share the same network traffic path. For example, there are 4 same live migrations sharing the same network path as well as source and destination hosts. In Fig. 3, lower graph shows the sequential live migration. Because each migration fully uses the path bandwidth, the network transmission part is much smaller than the part of parallel migration shown in the upper graph at which 4 migrations share the bandwidth evenly. However, in this example, the total network bandwidth is extremely large comparing to the dirty rate and the memory size of each

VM is relatively small. Therefore, the pre and post migration overheads contribute substantially to the total migration time. As the result, even though sharing the same network path could extend the memory iteration, parallel migration running the pre and post migration on multicore in this situation actually outperforms the sequential algorithm.

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

$$T_{seq} = \sum_{1}^{N} T_{mig} = \sum T_{overhead} + \sum T_{network} \tag{12}$$

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

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

$$T^{m}_{network} = Max\left\{m \cdot T_{blk}, \frac{m \cdot \rho \cdot M}{L} \cdot \frac{1-(m\sigma)^{n+1}}{1-m\sigma}\right\} \tag{13}$$

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

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

$$\Sigma^{m} W_{pre}/m \cdot p \geq T^{m}_{network} \tag{14}$$

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

$$\begin{aligned} T'_{par} &= \frac{\sum_{1}^{Xm} W_{pre}}{m \cdot p} + \frac{\sum_{Xm+1}^{N} W_{pre}}{N-Xm+2} + T^{N-X}_{network} + \frac{\sum_{Xm+1}^{N} W_{post}}{N-Xm+2} \\ &= \frac{(\lfloor N/m \rfloor + 1) \cdot W_{pre} + W_{post}}{p} + T^{N-X}_{network} \\ T''_{par} &= \frac{\sum_{1}^{m} W_{pre}}{m \cdot p} + T^{m}_{network} + \frac{\sum_{1}^{Xm} W_{post}}{m \cdot p} + \frac{\sum_{Xm+1}^{N} W_{post}}{N-Xm+2} \\ &= \frac{(\lfloor N/m \rfloor + 1) \cdot W_{post} + W_{pre}}{p} + T^{m}_{network} \end{aligned} \tag{15}$$

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

$$T^{m}_{network} \leq Max\left\{m \cdot T_{blk}, \frac{m \cdot \rho \cdot M}{L \cdot (1-m\sigma)}\right\} \tag{16}$$

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

$$T^{seq}_{response} = (N+1)/2 \cdot \left(W_{overhead}/p + T^{1}_{network}\right) \tag{17}$$

$$T^{par}_{response} = W_{overhead}/p + T^{m}_{network} \tag{18}$$

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

$$\Delta_{network} = T^{m}_{network} - \Sigma^{m} T^{1}_{network} \tag{19}$$

$$\Delta_{workload} = \Sigma^{m} T_{overhead} - \Sigma^{m} T_{overhead}/m \cdot p \tag{20}$$

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

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

5. Performance evaluation

There are several parameters which can influence the live VM migration performance in SDN-enabled data centers from **system view**, such as the flavor, CPU, memory, and static downtime adjustment, **network view**, such as parallel and sequential migrations, available bandwidth, and dynamic flow scheduling update rate, and **application view**, such as response time under different migration strategies. In this section, we explore the impacts of these parameters on migration performance. The migration time, downtime, and transferred data shown in the results are the average values. In OpenStack, we can use the *nova migration-list* to measure the duration of live migration. The downtime of live migration could be calculated by the time stamp difference of VM lifecycle event (VM Paused and VM Resumed) in both Nova log files. Each configured migration experiment is performed 6 times.

5.1. Testbed and its specification

As current production system will not allow users to access or modify the low-level infrastructure elements, such as resource management interfaces and SDN controllers and switches, needed for experiments, we created our own testbed. CLOUDS-Pi [32], a low-cost testbed environment for SDN-enabled cloud computing, is used as the research platform to test virtual machine block live migration. We use OpenStack combined with OpenDayLight [4] (ODL) SDN controller to manage the SDN-enabled Data Centers, which contains 9 heterogeneous physical machines connected through Raspberry Pis as OpenFlow switches whose specifications are shown in Table 1. The Raspberry Pis are integrated with Open vSwitch (OVS) as 4-port switches with 100 Mbps Ethernet Interfaces. The network physical topology is shown in Fig. 4. The OpenStack version we used is Ocata and the Nova version is 15.0.4 and the Libvirt version is 3.2.0. The Ubuntu tool *stress-ng* [18] is used as the micro-benchmark to stress memory and CPU to pinpoint the impacts of parameters on migration performance.

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

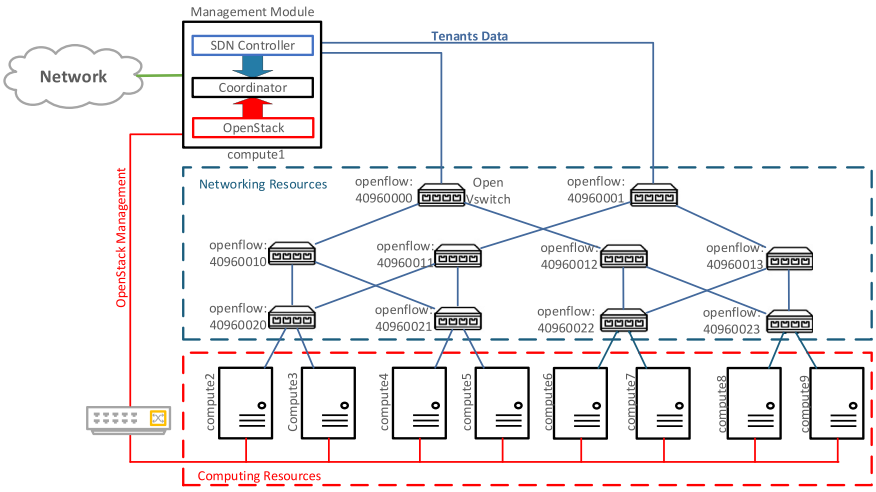


Fig. 4. SDN-enabled data center platform.

Table 1
Specifications of physical hosts in CLOUDS-Pi.

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

Table 2
Specifications of VM flavors in OpenStack.

No.	Name	vCPUs	RAM	Disk	No.	Name	vCPU	RAM	Disk
1	Nano	1	64MB	1GB	5	Medium	2	3.5GB	40GB
2	Tiny	1	512MB	1GB	6	Large	4	7GB	80GB
3	Micro	1	1GB	10GB	7	Xlarge	8	15.49GB	160GB
4	Small	1	2GB	20GB					

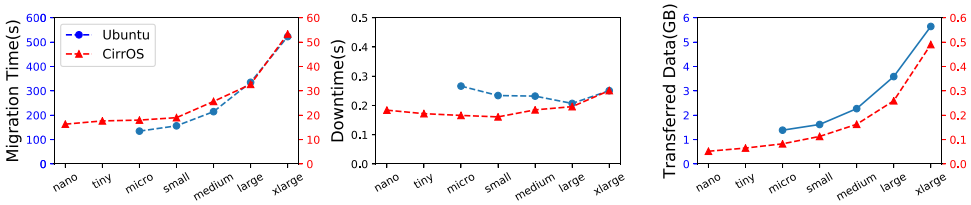
5.2. Primary parameters

First, we evaluate the fundamental parameters, such as flavor, memory and CPU loads, which affect the migration time, downtime and total transferred data of block live VM migration in OpenStack. As we measured, the amount of data from destination to source can be omitted because it only accounts for around 1.8% of total transferred data. The transferred data is measured by the SDN controller through OpenFlow protocol. We set 7 flavors in OpenStack, which are nano, tiny, micro, small, medium, large, xlarge (Table 2). Not only the RAM size but the ephemeral disk size can affect the migration time as well as the total transferred data (Eq. (8)). We evaluate these primary parameters by migrating instances from *compute2* to *compute3*. In the flavor experiment, we use two Linux images, CirrOS and Ubuntu-16.04, and the smallest flavor suitable for the Ubuntu image is micro. The image size of CirrOS is 12.65 MB, and Ubuntu is 248.38 MB. In memory stress experiment, we evaluate the migration performance of different memory-stressed Ubuntu-16.04 instance with micro flavor from 0% to 80%. In CPU stress memory experiment, we compare the migration performance with 0 to 100 stressed CPU between Ubuntu instance with 0 memory stress (mem0) and 40% memory-stressed (mem40) VMs.

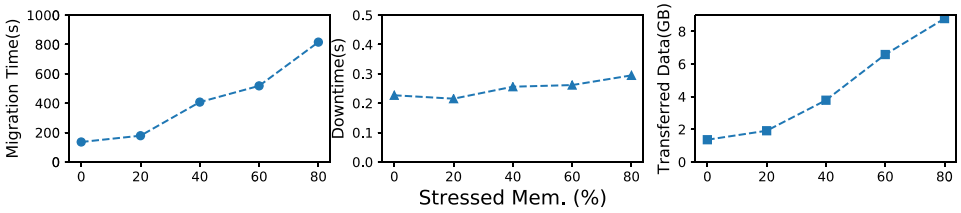
Flavor: Fig. 5(a) illustrates the migration performance (migration time, downtime, and total transferred data) of idle VMs with different flavors. Larger RAM and disk sizes lead to longer migration time and total transferred data. The VM block live migration cost with the same flavor could be a huge difference due to the system disk size and the required RAM of different OS instance. According to the downtime adjustment algorithm, a longer migration time can lead to a larger downtime. However, the difference of downtime is small compared to the significant difference of migration time. From flavor micro to xlarge, the transferred data is increased linearly. Furthermore, the transferred data vs. flavor figure illustrates that there is a constant data size difference between CirrOS and Ubuntu with the same flavor. With the same flavor, VM with a larger and more complex OS installed has a longer migration time and larger transferred data as the data size difference of the OS base image and dirty rate caused by OS processes.

Memory: The dirty page rate (and dirty block rate) directly affects the number of pages that are transferred in each pre-copy iteration. Fig. 5(b) shows that the performance of different memory-stressed Ubuntu instances from 0% to 80% on the migration time, downtime, total data transferred from source. As shown in Eqs. (8) and (9), the relationship between the dirty page rate and live migration performance is not linear due to the downtime adjustment algorithm. The downtimes of migrations may be constant with different dirty page rates because of the delay of every downtime adjustment, such as 0% and 20% memory-stressed VMs. With the downtime adjustment algorithm, the downtimes of live migrations with drastically different dirty page rate remain at a stable range.

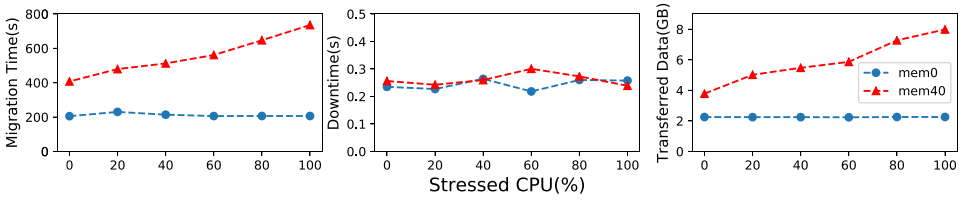
CPU: Higher CPU workloads can lead to a migration performance degradation because of the page copying operation overhead during the pre-copy iterations. Meanwhile, the high CPU workloads can also cause interference among memory-intensive tasks which leads to a large migration time. We examine the block live migrations based on various CPU loads from 0% to 100%. Fig. 5(c) shows that, without stressed memory, the CPU loads inside VMs are irrelevant to the downtime and duration of live migration with the minor copying overhead due to the pre-copy iterations. However, as the CPU usage of a 40-percent-stressed memory task is 100%, an extra CPU workload can lead to a larger amount of total transferred data and migration time. For idle VMs, the migration time and transferred data are constant under various range of CPU workload. For more busy VMs, extra



(a) Ubuntu and CirrOS VM with different flavors



(b) Live migration based on stressed memory



(c) Live migration based on CPU loads

Fig. 5. Primary VM parameters.

CPU workload leads to a linear increase in migration time and transferred data size.

5.3. Downtime configuration effectiveness

In OpenStack, the live VM migration time could shift dramatically based on different configuration tuples (*max_downtime, steps, delay*). Although only implemented in OpenStack, the downtime adjustment algorithm can also apply to other cloud computing platforms. In this experiment, the Ubuntu-16.04 instance with micro flavor is migrated between NOVA compute node *compute2* and *compute3*. We perform migrations based on the different step or delay settings and other two default values, i.e., (500, 4, 75) and (500, 10, 5), with 0% to 75% stressed memory VM.

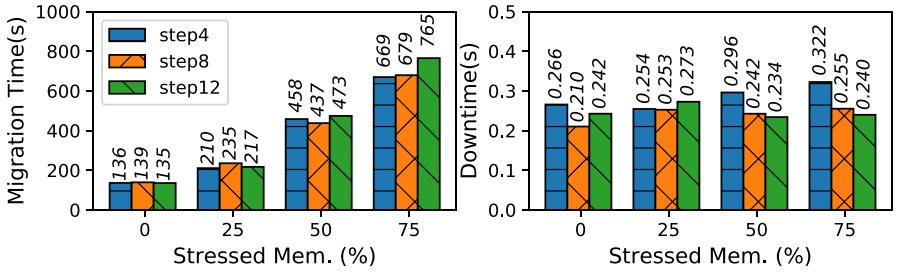
Fig. 6 indicates that for less memory stressed VMs (low dirty page rate), the static algorithm based on short delay could lead to a higher downtime with a slightly different migration time. However, for heavy memory stressed VMs (high dirty page rate), the adjustment of large delay setting, such as delay40, delay110, leads to an extremely long migration duration. The larger adjustment step setting leads to a larger migration time with a smaller downtime. However, step8 (500, 8, 75) leads to a better result in migration time compared to step12 and in downtime compared to step4 when VM memory is 75% stressed. We also notice that the setting (500, 10, 5) is a better choice when VM has high dirty page rate and (500, 4, 75) is better when the rate gets lower. When the dirty page rate is high, the migration time gets benefits from quickly raised downtime threshold while the downtime remains at a stable range. When it is low, the downtime gets benefits from smaller downtime threshold with slow adjustment. We should dynamically configure the optimal downtime setting tuple to improve both migration time and downtime based on the migration model for every live migration task.

5.4. Live VM migration in parallel

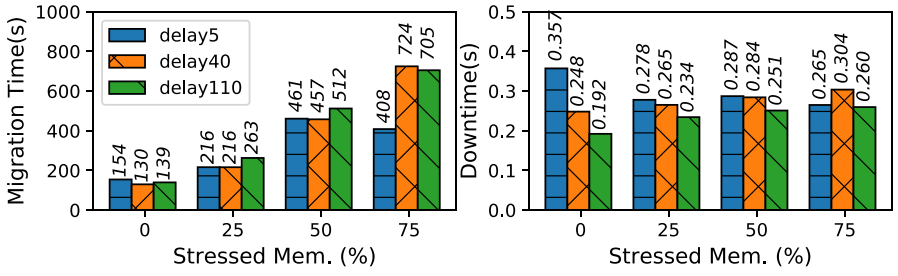
The default NOVA configuration of max allowed parallel migration is *max_concurrent_live_migrations=1*, which means only one live migration could be performed at the same time. In this experiment, we evaluate the migration duration of one compute host that needs to evacuate all VMs to another. First, we need to change the default max allowed parallel migration to perform maximum *m* live migrations in parallel. The CirrOS instances with tiny flavor are migrated between node *compute2* and *compute3*. All migration operations are released at the same time with different maximum parallel migration. We measure the response time of each migration task and the total evacuation time of 10 idle CirrOS VMs. We also examined the sequential live migration with several VMs from 2 to 10.

Fig. 7(a) indicates that the response time (rt), migration time (mt) and evacuation duration (dur) of sequential live migrations increase linearly with the number of VMs. Fig. 7(b) only demonstrates the rt and dur, as the mt equals to the rt in this parallel migration experiments. However, the parallel migrations could significantly reduce the total evacuation time and each migration time of 10 idle VMs. With the max allowed concurrent migration increasing from 1 to 10, the total live migration evacuation time decreases by 59.6%. Meanwhile, the migration time of each VM decreases up to 50%.

As shown in Eq. (19), (20), when $\Delta_{network} < \Delta_{workload}$, the pre- and post-migration overheads constitute a large portion of the total migration time, e.g., parallel migration of the tiny flavor CirrOS VMs with 100Mbps bandwidth (Fig. 7(b)). Therefore, several pre- and post-live-migration processes concurrently running on both hosts can reduce the total evacuation time (15) and average response time (18) compared to the sequential live migrations (12), (17). Therefore, when the multiple VM evacuation happens

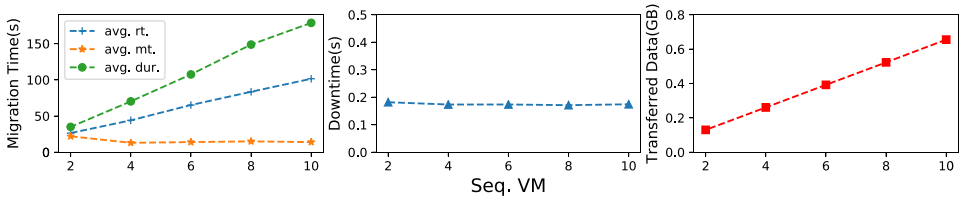


(a) *live_migration_downtime_steps*

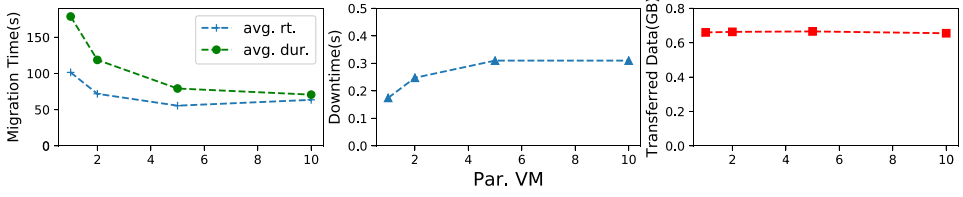


(b) *live_migration_downtime_delay*

Fig. 6. Live migrations based on different step and delay settings.



(a)



(b)

Fig. 7. (a) Sequential migrations with different number of VMs; and (b) Multiple live migrations of 10 VMs where *x*-axis indicates the max allowed concurrent migration.

in the same network path, we need to decide the sequential and parallel live migration based on both network and computing aspects to achieve a better total migration time (duration).

5.5. Network-aware live migration

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

TCP and UDP traffic: Block live Migration is highly relative to the network bandwidth as well as the background traffic on the links. The total migration time and downtime are negatively correlated with the network bandwidth. Therefore, we measure the migration performance under the default downtime configuration with various network traffic scenarios with different constant bandwidth rate (CBR) in TCP and burst transmission in UDP. UDP datagrams are sent in the same data size in every 10 s. The *iperf3* [1] is used to generate background traffic between live migration source and destination hosts through the same path in SDN-enabled data center network. The image of VM is Ubuntu-16.04 with micro flavor under no stressed memory. Fig. 8 indicates that, when the dirty page rate is 0, the transferred data is not linearly increased with the migration time. The migration time is increased linearly with the bandwidth decreasing.

Dynamic SDN flow scheduling: In this experiment, we pinpoint the impact of the flow scheduling algorithm update rate on block live migration in SDN-enabled cloud computing. When

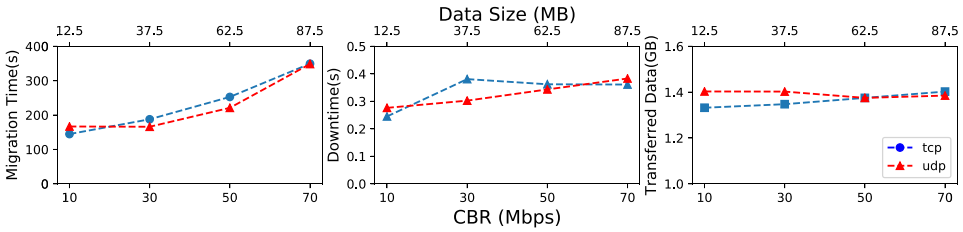
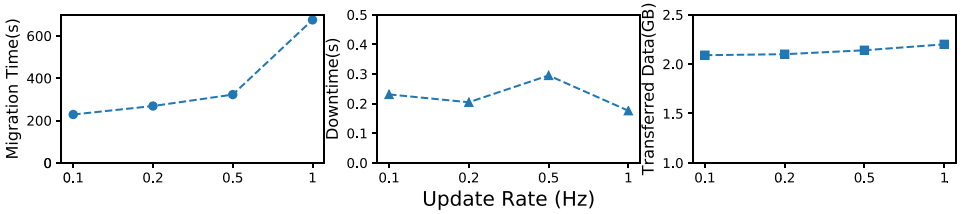
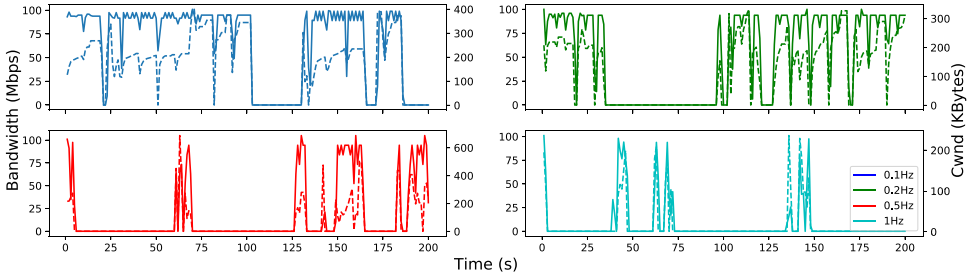


Fig. 8. Block live migrations with TCP and UDP background traffic.



(a) Comparison of migrations



(b) Comparison of Iperf throughput: solid line indicated the Bandwidth and dot line indicated the Cwnd

Fig. 9. Live migrations based on different SDN scheduling update rate.

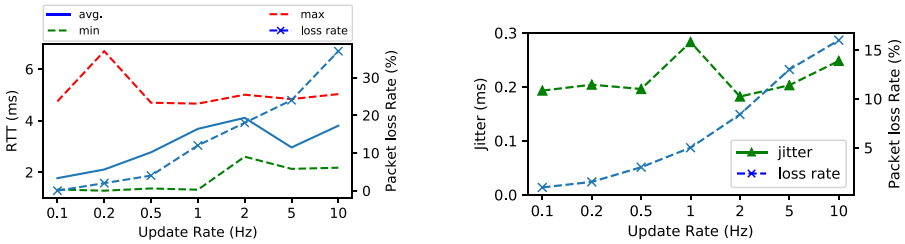
SDN controller is proactively scheduling the flows, latencies exist between controller and switches (PacketOut message send to the switches and PacketIn to the controller). Moreover, in the flow tables, latencies occur when installing, deleting flow entities. The scheduler based on SDN controller (OpenDayLight) REST APIs proactively pushes the end-to-end flow in a certain time period to dynamically set the best path. The idle Ubuntu-16.04 instance with micro flavor is migrated from *compute3* to *compute9*. As shown in Fig. 4, there are two shortest paths between *compute3* and *compute9* that each one contains 5 OpenFlow nodes (OpenFlow-enabled switches). A round-robin scheduler rescheduling the traffic of live migration periodically based on these paths. We also use *iperf3* to generate TCP and UDP traffic to evaluate the latency, TCP window size, and packet loss rate.

Fig. 9(a) shows that the migration time is positively correlated with the update rate while the transferred data is just slightly increased. As the dynamic scheduling update rate increases, the link bandwidth rapidly decreases which leads to a large migration time. Meanwhile, Fig. 9(b) indicates that the TCP throughput goes down more frequently with high flow update rate. The TCP congestion window size decreases to 1.41 KBytes when the bandwidth is 0 bits/s. Fig. 10 shows the TCP and UDP protocol performance with different update rates from 0.1 Hz to 10 Hz. The packet loss rate increases linearly with the update rate and the average maximum TCP latency (Round-Trip Time) is 2 times larger at 2 Hz than the minimum value at 0.1 Hz. When the TCP traffic suffers the bandwidth degradation, the UDP transmission rate is always around 90 Mbps regardless of the scheduling update rate.

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

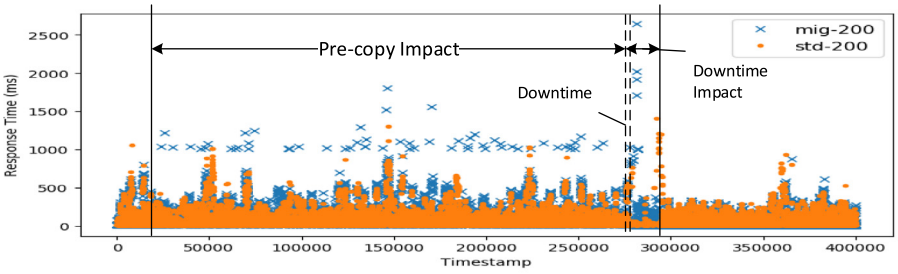
5.6. Impacts on multi-tier application response time

In this experiment, we evaluate the impact of VM live migration on the real web application, such as *MediaWiki*, using *WikiBench* [33]. It uses *MediaWiki* in the application server and real database dumps in the database server. In client VM, the *wikijector* as traffic injector controls the simulated client to reply the traces of real *Wikipedia* traffic. Regarding the scale of the testbed, we use 10% of *Wikipedia* trace to simulate the real traffic. The database and *MediaWiki* Apache servers are allocated in *compute3*, and one *WikiBench* injector as the client VM located

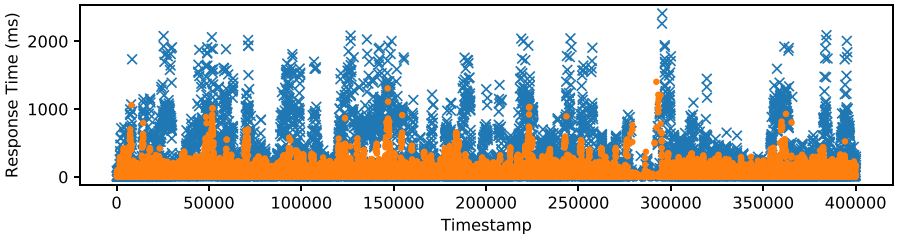


(a) TCP latency and packet loss rate (b) UDP jitter and packet loss rate

Fig. 10. Network performance with different SDN scheduling update rate.



(a) Client VM migration



(b) Service VM migration segment

Fig. 11. Response time of Wikipedia in 400 s.

Table 3
Request response time without VM migration.

Exp.	Duration(s)	RT(ms)	HTTP0	HTTP200	Total
Initial	1200	74.21	35	42 310	51 634
Initial	500	75.90	22	17 435	21 438
Initial	400	75.77	22	13 893	17 088
Scheduled	400	65.380	17	14 175	17 357

Table 4
Application performance in 400 s.

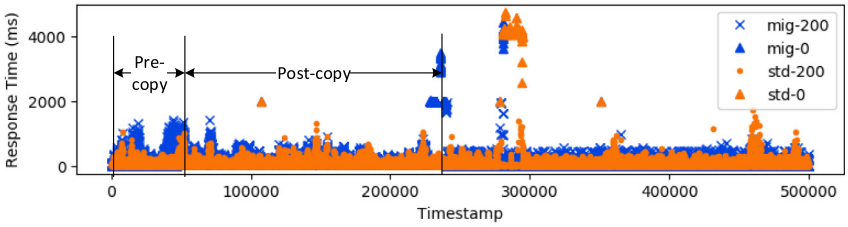
Exp.	MT(s)	RT(ms)	HTTP0	HTTP200	Total
c-93	248.34	84.201	20	14 166	17 348
s-39	N/A	192.273	109	13 410	16 558

in *compute9*. The client and server VMs are the Ubuntu instances with micro flavor and database server is with large flavor. The first scenario (c-93) is migrating the client VM to *compute3* to simulate the consolidation (scheduled) to reduce the latency. The second one (s-39) is migrating the application server to *compute9* in order to evaluate the effect of live migration on application response time.

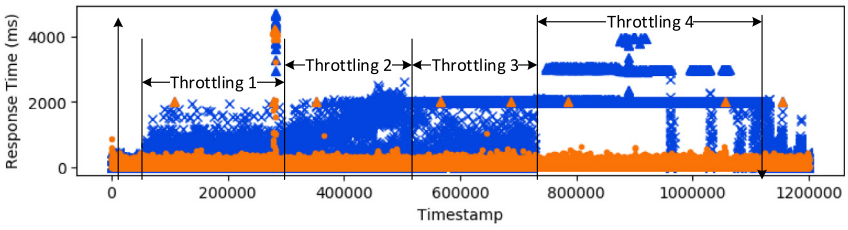
In the scenario c-93, the major application traffic is outbound traffic from the destination host. Therefore, the live migration traffic would just slightly affect the QoS of web service. Table 3 indicates that the application response time (RT) is improved after the VM consolidation (scheduled). Fig. 11(a) shows the initial response time (std-200) of the success requests (HTTP 200) and the response time of success requests during the client VM migration (mig-200). It indicates that the **response time** is increased during the migration and the **worst-case response time** occurs after the downtime of client's live VM migration because the application server needs to process extra requests and migration downtime postpones the response time of the requests which are sent before and during the downtime. On the other hand, if the injector and application server are located in the same host when the migration is performing, due to all requests happened inside the host, the live migration traffic will not affect the application response time.

However, in scenario s-39, i.e., the application traffic is sent to client VM (*compute9*), the pre-copy live migration traffic flow will contend for the shared bandwidth due to the same traffic direction. Therefore, the **worst case response time** may occur not only after downtime but during the migration time as shown in Fig. 11(b). Meanwhile, Table 4 shows that the average response time of requests is dramatically larger than the migration of client VM. The request timeout (HTTP 0) happens much often due to the server migration.

We notice that the server migration from *compute3* to *compute9* cannot finish in 20 min. For memory-intensive instances,



(a) Hybrid post-copy



(b) Auto-convergence

Fig. 12. Response time of successful server migrations.

Table 5
Server migration under different strategies.

Exp.	MT(s)	Duration(s)	RT(ms)	HTTP0	HTTP200	Total
s-39	N/A	400	192.27	109	13 410	16 558
AC	908	1200	245.33	6722	18 461	29 915
H-PC	237	500	156.73	190	16 906	20 912

like the Wikipedia server, there are two optional strategies to perform a successful live migration: **Hybrid post-copy (H-PC)** and **Auto-convergence (AC)**. Thus, we evaluate the migration performance and impacts on the response time of the hybrid post-copy and auto-convergence strategies for Wikipedia server in the scenario s-39. Table 3 shows the initial response time of 1200 s, 500 s and 400 s time intervals without any migration as well as the average migration time (Duration), response time (RT), and the number of success (HTTP200), timeout (HTTP0), and total requests.

Hybrid post-copy: With the start of pre-copy mode, the post-copy migration will be activated if the memory copy iteration does not make at least 10% increase over the last iteration. It will suspend the VM and process state on the source host. The VM will resume on the target host and fetch all missing pages as needed. However, the post-copy page fetching will slow down the VM which degrades the service performance and the VM will reboot if the network is unstable. The average **response time** of hybrid post-copy is better than the pre-copy migration as shown in Table 5. The **timeout requests** are slightly increased during the post-copy migration. Furthermore, Fig. 12(a) indicates response time of success and timeout requests without migration (std-200, std-0) and during the hybrid post-copy (mig-200, mig-0). It illustrates that under a stable network environment, the impacts of missing page fetching on application response time is less than pre-copy iteration traffic.

Auto-convergence: By throttling down the VM's virtual CPU, auto-convergence will only influence the workloads where the memory write speed is dependent on the CPU execution speed. As migration time flows it will continually increase the amount of CPU throttling until the dirty page rate is low enough for migration to finish. Fig. 12(b) indicates that the task of Wikipedia request has a worse **response time** under a larger throttling amount. The request tasks are highly related to the CPU execution speed. Therefore, the throttling down leads to a successful migration of the Wikipedia server. However, as the timeout threshold of a request is 2 s, the performance of the server is devastated under the last throttling down, i.e., most requests are timed out (mig-0). A larger timeout threshold for requests should be set according to the amount of throttling down. Although it can successfully perform the live server migration, the average response time is even larger than the pre-copy migration requests' (Table 5). Moreover, compare to the hybrid post-copy strategy, the auto-convergence leads to a much larger migration time.

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

6. Conclusions and future work

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

From the QoS perspective, we investigated the response time pattern of client and server live migrations with pre-copy,

hybrid post-copy, and auto-convergence strategies. For memory-intensive VM, as the pre-copy migration cannot finish in a reasonable time, we should choose hybrid post-copy to perform a successful migration if the network environment is stable. Otherwise, we could perform the auto-convergence feature during the pre-copy migration. However, the auto-convergence dramatically influences the application response time, i.e., requests are timed out because of the CPU slowdown. Moreover, for the pre-copy migration of server VM, as the migration and application traffic flows contend with each other, the worst-case response time will not just occur after the downtime but during the migration. Moreover, the models and parameters in our paper are compatible with other optimization technologies for single live VM migration [8,12,26,29] or algorithms of multiple migrations [7,9,11,30,35,36] because these work focus on different optimization factors. Therefore, the results in our paper still stand and can benefit other optimization methods and algorithms.

In the future, we plan to investigate the impact of these parameters' evaluation outcomes on the resource management in SDN-enabled cloud computing. In particular, we intend to investigate and develop: (a) the prediction model of live VM migration with static downtime adjustment algorithm and the optimal downtime adjustment configuration for different live migration tasks; (b) Deadline-aware multiple live VM migration planning by considering the parallel and sequential sequence in multiple and one network path; (c) SDN latency-aware traffic scheduling algorithm based on the trade-off between bandwidth increasing and rescheduling rate; and (d) QoS-aware resource scheduling strategy by considering application traffic pattern to minimize the influence of live migrations on application response time.

Acknowledgments

This work is partially supported by an Australian Research Council (ARC) Discovery Project and the China Scholarship Council (CSC).

Conflict of interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.jpdc.2019.04.014.

References

[1] Iperf3, 2016, https://iperf.fr/. (Accessed 11 February 2018).
[2] Open vSwitch, 2016, https://www.openvswitch.org/. (Accessed 15 January 2018).
[3] Libvirt Virtualization API, 2017, https://libvirt.org/. (Accessed 25 January 2018).
[4] Opendaylight Carbon release, 2017, https://docs.opendaylight.org/en/stable-carbon/index.html. (Accessed 25 January 2018).
[5] QEMu project, Live block operation documentation, 2017, https://kashyapc.fedorapeople.org/QEMU-Docs/_build/html/index.html. (Accessed 11 February 2018).
[6] S. Akoush, R. Sohan, A. Rice, A.W. Moore, A. Hopper, Predicting the performance of virtual machine migration, in: Proceedings of 2010 IEEE International Symposium on Modeling Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2010, pp. 37–46.
[7] M.F. Bari, M.F. Zhani, Q. Zhang, R. Ahmed, R. Boutaba, Cqncr: Optimal vm migration planning in cloud data centers, in: Proceedings of Networking Conference, 2014 IFIP, IEEE, 2014, pp. 1–9.
[8] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, 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.
[9] U. Deshpande, K. Keahey, Traffic-sensitive live migration of virtual machines, Future Gener. Comput. Syst. 72 (2017) 118–128.
[10] M. Forsman, A. Glad, L. Lundberg, D. Ilie, Algorithms for automated live migration of virtual machines, J. Syst. Softw. 101 (2015) 110–126.
[11] S. Ghorbani, M. Caesar, Walk the line: consistent network updates with bandwidth guarantees, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ACM, 2012, pp. 67–72.
[12] T. Guo, U. Sharma, P. Shenoy, T. Wood, S. Sahu, Cost-aware cloud bursting for enterprise applications, ACM, Trans. Internet Tech. 13 (3) (2014) 10.
[13] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L.E. Li, M. Thottan, Measuring control plane latency in sdn-enabled switches, in: Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, ACM, 2015, p. 25.
[14] J.J. Herne, Auto-convergence feature, 2015, https://wiki.qemu.org/Features/AutoconvergeLiveMigration. (Accessed 11 February 2018).
[15] M.R. Hines, U. Deshpande, K. Gopalan, Post-copy live migration of virtual machines, ACM SIGOPS, Oper. Syst. Rev. 43 (3) (2009) 14–26.
[16] W. Hu, A. Hicks, L. Zhang, E.M. Dow, V. Soni, H. Jiang, R. Bull, J.N. Matthews, A quantitative study of virtual machine live migration, in: Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference, ACM, 2013, p. 11.
[17] S. Kikuchi, Y. Matsumoto, Impact of live migration on multi-tier application performance in clouds, in: Proceedings of 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), IEEE, 2012, pp. 261–268.
[18] C. King, Stress-ng, 2018, http://kernel.ubuntu.com/cking/stress-ng/. (Accessed 24 February 2018).
[19] M. Kuźniar, P. Perešíni, D. Kostić, What you need to know about sdn flow tables, in: Proceedings of International Conference on Passive and Active Network Measurement, Springer, 2015, pp. 347–359.
[20] Z. Li, G. Wu, Optimizing vm live migration strategy based on migration time cost modeling, in: Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems, ACM, 2016, pp. 99–109.
[21] H. Liu, C.-Z. Xu, H. Jin, J. Gong, X. Liao, Performance and energy modeling for live migration of virtual machines, in: Proceedings of the 20th International Symposium on High Performance Distributed Computing, ACM, 2011, pp. 171–182.
[22] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, A. Iyer, Remedy: Network-aware steady state vm management for data centers, in: Proceedings of International Conference on Research in Networking, Springer, 2012, pp. 190–204.
[23] A. Mashtizadeh, E. Celebi, T. Garfinkel, M. Cai, et al., The design and evolution of live storage migration in vmware esx, in: Usenix Atc, Vol. 11, 2011, pp. 1–14.
[24] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM, Comput. Commun. Rev. 38 (2) (2008) 69–74.
[25] O. Sefraoui, M. Aissaoui, M. Eleuldj, Openstack: toward an open-source solution for cloud computing, Int. J. Comput. Appl. 55 (3) (2012) 38–42.
[26] A. Shribman, B. Hudzia, Pre-copy and post-copy vm live migration for memory intensive applications, in: Proceedings of European Conference on Parallel Processing, Springer, 2012, pp. 539–547.
[27] J. Son, R. Buyya, A taxonomy of software-defined networking (sdn)-enabled cloud computing, ACM Comput. Surv. 51 (3) (2018) 59:1–59:36, http://dx.doi.org/10.1145/3190617, http://doi.acm.org/101145/3190617.
[28] J. Son, A.V. Dastjerdi, R.N. Calheiros, R. Buyya, Sla-aware and energy-efficient dynamic overbooking in sdn-based cloud data centers, IEEE Trans. Sustain. Comput. 2 (2) (2017) 76–89.
[29] X. Song, J. Shi, R. Liu, J. Yang, H. Chen, Parallelizing live migration of virtual machines, ACM, SIGPLAN Not. 48 (7) (2013) 85–96.
[30] G. Sun, D. Liao, V. Anand, D. Zhao, H. Yu, A new technique for efficient live migration of multiple virtual machines, Future Gener. Comput. Syst. 55 (2016) 74–86.
[31] P. Svärd, B. Hudzia, J. Tordsson, E. Elmroth, Evaluation of delta compression techniques for efficient live migration of large virtual machines, ACM, SIGPLAN Not. 46 (7) (2011) 111–120.
[32] A.N. Toosi, J. Son, R. Buyya, Clouds-pi: A low-cost raspberry-pi based micro data center for software-defined cloud computing, IEEE Cloud Comput. 5 (5) (2018) 81–91.
[33] E.-J. Van Baaren, Wikibench: A distributed wikipedia based web application benchmark, Master's Thesis, VU University Amsterdam, 2009.
[34] W. Voorsluys, J. Broberg, S. Venugopal, R. Buyya, Cost of virtual machine live migration in clouds: A performance evaluation, in: Proceedings of IEEE International Conference on Cloud Computing, Springer, 2009, pp. 254–265.
[35] H. Wang, Y. Li, Y. Zhang, D. Jin, Virtual machine migration planning in software-defined networks, in: INFOCOM, IEEE, 2015, pp. 487–495, http://dblp.uni-trier.de/db/conf/infocom/infocom2015html#WangLZJ15.
[36] F. Xu, F. Liu, L. Liu, H. Jin, B. Li, B. Li, Iaware: Making live migration of virtual machines interference-aware in the cloud, IEEE Trans. Comput. 63 (12) (2014) 3012–3025.

TianZhang He received the B.Sc. degree in 2014 and the M.Sc. degree in 2017, both in computer science and technology from Northeastern University, China. He is working towards the Ph.D. degree at the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, the University of Melbourne, Australia. His research interests include resource scheduling and optimization in Software-Defined Networking (SDN) and Network Function Virtualization (NFV)-enabled cloud computing.

Adel Nadjaran Toosi Dastjerdi is a lecture in computer systems at Faculty of Information Technology, Monash University, Australia. He received his B.Sc. degree in 2003 and his M.Sc. degree in 2006 both in Computer Science and Software Engineering from Ferdowsi University of Mashhad, Iran and his Ph.D. degree in 2015 from the University of Melbourne. Adel's Ph.D. studies were supported by International Research Scholarship (MIRS) and Melbourne International Fee Remission Scholarship (MIFRS). His Ph.D. thesis was nominated for CORE John Makepeace Bennett Award for the Australasian Distinguished Doctoral Dissertation and John Melvin Memorial Scholarship for the Best Ph.D. thesis in Engineering. His research interests include scheduling and resource provisioning mechanisms for distributed systems. Currently, he is working on resource management in Software-Defined Networks (SDN)-enabled Cloud Computing.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He served as a Future Fellow of the Australian Research Council during 2012–2016. He has authored over 625 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index = 114, g-index = 245, 66,900+ citations). Dr. Buyya is recognized as a "Web of Science Highly Cited Researcher" in 2016 and 2017 by Thomson Reuters, a Fellow of IEEE, and Scopus Researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier for his outstanding contributions to Cloud computing. He served as the founding Editor-in-Chief of the IEEE Transactions on Cloud Computing. He is currently serving as Co-Editor-in-Chief of Journal of Software: Practice and Experience, which was established over 45 years ago. For further information on Dr. Buyya, please visit his cyberhome: www.buyya.com