

CAMIG: Concurrency-Aware Live Migration Management of Multiple Virtual Machines in SDN-Enabled Clouds

TianZhang He¹, Adel N. Toosi², *Member, IEEE*, and Rajkumar Buyya¹, *Fellow, IEEE*

Abstract—By integrating Software-Defined Networking and cloud computing, virtualized networking and computing resources can be dynamically reallocated through live migration of Virtual Machines (VMs). Dynamic resource management such as load balancing and energy-saving policies can request multiple migrations when the algorithms are triggered periodically. There exist notable research efforts in dynamic resource management that alleviate single migration overheads, such as single migration time and co-location interference while selecting the potential VMs and migration destinations. However, by neglecting the resource dependency among potential migration requests, the existing solutions of dynamic resource management can result in the Quality of Service (QoS) degradation and Service Level Agreement (SLA) violations during the migration schedule. Therefore, it is essential to integrate both single and multiple migration overheads into VM reallocation planning. In this paper, we propose a concurrency-aware multiple migration selector that operates based on the maximal cliques and independent sets of the resource dependency graph of multiple migration requests. Our proposed method can be integrated with existing dynamic resource management policies. The experimental results demonstrate that our solution efficiently minimizes migration interference and shortens the convergence time of reallocation by maximizing the multiple migration performance while achieving the objective of dynamic resource management.

Index Terms—Live migration, dynamic resource management, migration scheduling, software-defined networking, cloud computing

1 INTRODUCTION

WITH the rapid adoption of cloud computing for hosting applications and always-on services, it is critical to provide Quality of Service (QoS) guarantees through the Service Level Agreements (SLAs) between cloud providers and users. In this direction, many research works have investigated various aspects of dynamic resource management, such as delay-aware Virtual Network Function (VNF) placement [1], load balancing [2], [3], [4], energy-saving [5], flow consolidation, scheduled maintenance, as well as emergency migration, in terms of accessibility, quality, efficiency, and robustness of cloud services. Virtual Machine (VM) is one of the major virtualization technologies to host computing and networking resources in cloud data centers. As a dynamic resource management tool, the live VM migration is used to realize the objectives in resource

management by relocating VMs between physical hosts without disrupting the accessibility of cloud services [6].

Cloud infrastructure and service providers, such as AWS, Azure, and Google, have been integrating live VM and container migration [7], [8], [9], [10], [11] for the purposes, such as higher priority task preemption, kernel and firmware software updates, hardware updates, and reallocation for performance and availability. For example, the Google cluster manager Borg controls all computing tasks and container clusters of up to tens of thousands of physical machines. In Google production fleets, a lower bound of 1,000,000 migrations can be performed monthly [11]. These show the critical importance of migration management techniques in dynamic resource reallocation.

Fig. 1 illustrates the general migration management workflow. Based on the various objectives, the resource management algorithms [2], [3], [4], [5], [12], [13], [14], [15], [16] find the optimal placement by generating multiple live migrations. With the generated multiple migration requests, the migration planning and scheduling algorithm [17], [18], [19], [20] optimizes the performance of multiple migrations, such as total and individual migration time and downtime, while minimizing the migration cost and overheads, such as migration impact on application QoS. On the other hand, the computing and networking resources are reallocated and affected by multiple migrations.

As a resource-intensive operation, live migration consumes both computing and networking resources when transmitting the memory dirty pages from the source to the destination host. It puts stress on both the migrating services and other services in the cloud data centers. Thus, it is

- TianZhang He and Rajkumar Buyya are with CLOUDS Lab, School of Computing and Information Systems, University of Melbourne, Parkville, VIC 3010, Australia. E-mail: tianzhangh@student.unimelb.edu.au, rbuyya@unimelb.edu.au.
- Adel N. Toosi is with the Department of Software Systems and Cybersecurity, Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia. E-mail: adel.n.toosi@monash.edu.

Manuscript received 22 Aug. 2020; revised 24 Aug. 2021; accepted 18 Dec. 2021. Date of publication 28 Dec. 2021; date of current version 7 Mar. 2022.

This work was supported in part by Australian Research Council (ARC) Discovery Project under Grant DP160102414 and in part by the China Scholarship Council - University of Melbourne PhD Scholarship.

(Corresponding author: TianZhang He.)

Recommended for acceptance by A. Orgerie.

Digital Object Identifier no. 10.1109/TPDS.2021.3139014

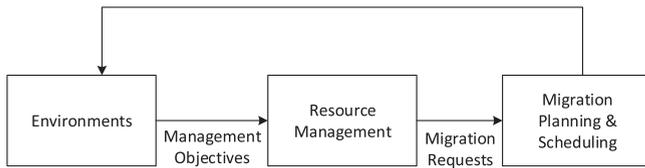


Fig. 1. A general migration management framework.

crucial to minimize migration interference during dynamic resource management. There are continuous efforts to take migration overheads into consideration during the dynamic resource management [2], [3], [5], [12]. Currently, most migration cost models consider overheads of single migration [21], [22], [23], such as migration time (single execution time), downtime, transferred data with respect to the size of memory, dirty page rate, data compression rate and available bandwidth while allowing multiple migrations in dynamic resource management. For the migration selection, existing resource management algorithms utilize the linear cost model of single migration to minimize the overheads. Then, with the migration requests generated as the input, multiple migration planning and scheduling algorithms [17], [18], [19], [20] decide the sequence of migration requests to achieve the maximal scheduling performance.

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

Resource dependency between two migrations, such as sharing source and destination hosts or network paths, can largely affect the performance of multiple migration scheduling. With the network as a bottleneck, two resource-dependent migrations can only be scheduled sequentially, while independent ones scheduled concurrently [18], [19], [23]. If large amount of resource dependencies among migrations are generated by dynamic resource management, the performance of multiple migration scheduling will suffer a significant degradation. Since single migration overheads are only related to one migration, it is critical to consider multiple migration overheads in order to generate migration requests with less resource dependencies.

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

- We propose and model the multiple migration selection problem to minimize interference due to resource dependency among multiple migrations

while achieving the objective of dynamic resource management.

- We introduce the resource dependency graph to model migration concurrency.
- We propose a flexible concurrency-aware migration selection strategy for dynamic resource management.
- We conduct extensive experiments in an event-driven simulation to show the performance improvement in terms of total migration time in correspondence with resource management objective.

The rest of the paper is organized as follows. Related works of migration cost management and multiple migration scheduling are reviewed in Section 2. The system framework and migration overheads are discussed in Section 3. The problem model is described in Section 4. Section 5 proposes the concurrency-aware migration selection algorithm. Section 6 compares our proposed algorithm with other dynamic resource management algorithms in load-balancing and energy-saving scenarios. Finally, Section 7 summarizes the paper.

2 RELATED WORK

Many dynamic resource management solutions utilize live migration as a tool to achieve objectives, such as load-balancing [2], [3], [4], [24], energy efficiency [13], [14], network delay [15], and communication cost [16]. Among these solutions, some resource management algorithms consider a linear model of the total migration overheads as the sum of individual migration overhead [2], [3], [4], [5], [12], [14], [24]. However, existing research only considers the objectives of resource management while neglecting the multiple migration overheads and migration scheduling performance. Generally, during the dynamic resource management, there are three steps to generate migration requests: source host selection; VM selection; and destination host selection. The overhead or interference model of single migration [21], [22], [23] is considered during the VM and destination selections.

For the VM and destination host selection, many dynamic resource management policies consider single migration overheads in terms of the memory size of migrating VM, single migration time, and the impact of one migration on other VMs located in the source or destination host, such as CPU, bandwidth of host network interface, and application bandwidth. In the load balancing scenario, Verma *et al.* [3] estimated the migration cost based on the deduction of application throughput. It selects the smallest memory size VMs from the over-utilized hosts and assigns them to the under-utilized hosts in the First Fit Decreasing (FFD) order. Singh *et al.* [24] proposed a multi-layer virtualization system HARMONY. It migrates VMs and data from hotspots on servers, network devices, and storage nodes. The load balancing algorithm is a variant of Toyoda multi-dimensional knapsack problem based on the evenness indicator Extended Vector Product (EVP). It considers the single live migration impact on application performance based on CPU congestion and network overheads. Wood *et al.* [2] proposed the load balancing algorithm Sandpiper that selects the smallest memory size VM from one of the most overloaded hosts to minimize the migration overheads. Mann *et al.* [4] focused on the VM and destination selection for the load balance of application network flows by considering

TABLE 1
Comparison of Approaches on Dynamic Resource Management Through Live Migration

algorithm	resource management	single migration overhead	dependency aware	migration performance	migration scheduling
FFD [3]	load/energy	memory size	-	sum of migration cost	-
HARMONY [24]	load	CPU, network	-	single exe. time	one-by-one
Sandpiper [2]	load	memory size	-	single exe. time, migration number	-
Xiao <i>et al.</i> [25]	load/energy	migration number	-	migration number	-
lrmmt [5]	load/energy	memory size	-	migration number	-
iAware [12]	flexible	single exe. time, computing	-	sum of normalized cost	one-by-one
Our work (CAMIG)	flexible	migration model, computing	computing, network sharing	total mig. time, downtime	multiple scheduling

the single migration cost model based on the dirty page rate, memory size, and available bandwidth.

In the energy-saving scenario, Xiao *et al.* [25] investigated dynamic resource allocation through live migration. The proposed algorithm avoids the over-subscription while satisfying the resource needs of all VMs based on exponentially weighted moving average to predict the future loads. It also minimizes the physical machines regarding the energy consumption. Similarly, LR-MMT [5] focused on energy saving with local regression based on history utilization to avoid over-subscription. It chooses the least memory size VM from the over-utilized host and the most-energy saving destination. Wu *et al.* [14] also studied the same problem of maximizing the power saving through VM consolidation by limiting individual migration costs. With the input of candidate VMs and destinations provided by other resource management algorithms, iAware [12] is a migration selector minimizing the single migration cost in terms of single migration execution time and host co-location interference. It considers dirty page rate, memory size, and available bandwidth for the single migration time. They argue that co-location interference from a single live migration on other VMs in the destination host in terms of performance degradation is linear to the number of VMs hosted by a physical machine in Xen. However, it only considers one-by-one migration scheduling.

Taking the migration task list generated by resource algorithms as input, migration scheduling algorithms focus on minimizing the migration time by efficiently scheduling them. To find a possible sequence of migrations, one-by-one scheduling [17] focused on avoiding the deadlock on the available resource of physical hosts. The multiple migration planning and scheduling algorithms [18], [19], [20] focused on the migration performance in terms of minimizing the total migration time by scheduling given migration tasks concurrently when necessary. Table 1 summarizes representative related works and the proposed generic solution for existing dynamic resource management algorithms in management target, migration overhead, interference, migration performance, and migration scheduling method.

3 LIVE MIGRATION IN RESOURCE MANAGEMENT

We first introduce the background of live migration management including system overview and single cost model. Then, we discuss the resource dependency problem.

3.1 System Overview

By integrating Software-Defined Networks (SDN) [26], the SDN-enabled cloud data centers have a centralized solution for the monitoring, planning, and scheduling of virtualized computing and networking resources [27]. Fig. 2 illustrates the migration framework in the orchestration layer. The dynamic resource manager integrated with migration selector and multiple migration scheduler based on both monitoring computing resource and network resources. VMs are hosted on physical machines to provide various cloud services. Computing resources are controlled by VM Manager (VMM), such as OpenStack Nova, while the networking resources (such as available bandwidth and routing) are managed by the SDN controller and VM Networking Service, such as OpenStack Neutron, in a centralized way. The SDN controller can dynamically manage the routing for migration elephant flows to avoid the congestion and alleviate the impact on cloud services. We can predict the cost of live migration by the available bandwidth between the source and destination hosts.

3.2 Single Migration Cost Model

To better understand the impact of multiple migrations on performance in dynamic resource management settings, we first introduce the mathematical model of a single live migration [23]. Live migration can be categorized into two types: post-copy and pre-copy migration. Since the pre-copy migration [6] is the most widely used approach in hypervisors (KVM, VMWare, Xen, etc), we consider it as the base model. During the pre-copy live migration for VMs or Containers, the hypervisor or the Checkpoint/Restore agent

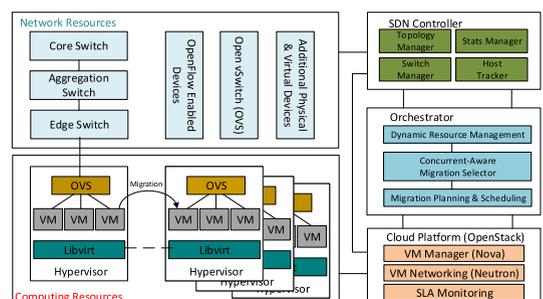


Fig. 2. System overview.

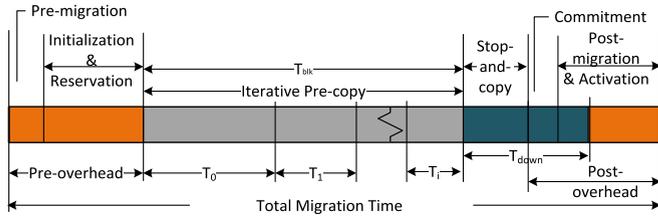


Fig. 3. Pre-copy live migration.

in the userspace (CRIU) [28] iteratively copies the dirty memory pages in the previous transmission interval from the source host to the destination host.

The most important aspect of single migration overheads is the migration time or the single migration execution time. According to the live migration process [6], the pre-copy live migration consists of eight phases (see Fig. 3): pre-migration, initialization, reservation, iterative memory copy, stop-and-copy, commitment, and post-migration. Thus, live migration consumes both computing resources (pre-/post-migration overheads) and networking resources (memory copy and dirty page transmission) [23]. The total single migration time T_{mig} can be categorized into three parts: pre-migration computing overheads, memory-copy networking overheads, and post-migration computing overheads:

$$T_{mig} = T_{pre} + T_{mem} + T_{post}. \quad (1)$$

Based on the iterative pre-copy illustrated in Fig. 3, the migration performance in terms of memory-copy can be represented as [23]:

$$T_{mem} = \frac{\rho \cdot Mem}{L} \cdot \frac{1 - \sigma^{i+1}}{1 - \sigma} \quad (2)$$

$$i = \min \left(\left\lceil \log_{\sigma} \frac{V_{thd}}{M} \right\rceil, \Theta \right), \quad (3)$$

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

3.3 Resource Dependency

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

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

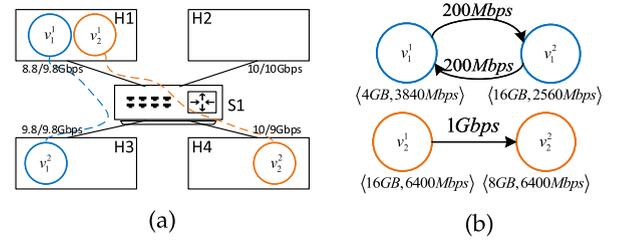


Fig. 4. Scenario of resource dependencies during migration selections: (a) Initial placement and (b) Virtual connections between VMs with memory size and dirty page rate.

to resource dependencies among migrations. If sharing the same source or destination hosts, or part of the network routing, two live migrations are resource-dependent.

Two resource-dependent migrations can not be scheduled at the same time [18], [23]. Because, according to Equation (2), larger bandwidth allocation means a smaller migration execution time and downtime. Thus, the networking resources are the bottlenecks which need to be optimized during the multiple migrations. For example, we have a number of migrations partially or entirely sharing network paths. Based on Equation (2), if scheduled at the same time, experimental results [23] show that the total migration time will be more than the sum of single execution time. Thus, sequential scheduling of dependent migrations is the most efficient way to optimize the migration performance [18], [23]. Meanwhile, migrations which are resource-independent can be scheduled concurrently to reduce the total migration time. Therefore, it is essential to exclusively allocate one network path to only one migration until it is finished to achieve the optimal total migration time, average execution time, and downtime.

3.4 Illustrative Example

Fig. 4a shows the initial VM placement of the illustrative example along with the resource dependency among possible migration selections. Fig. 4b illustrates the virtual connections between VMs and the memory size (GB) and dirty page rate (Mbps) for each. Moreover, the threshold of iteration rounds is 30 and downtime threshold is 0.5 seconds. The objective of the management policy is to reduce the communication cost by VM consolidation. There are several potential migration combinations which can fulfill the objective: M1: $v_1^1 : H1 \rightarrow H3$ and $v_2^1 : H1 \rightarrow H4$; M2: $v_1^1 : H1 \rightarrow H3$ and $v_2^2 : H4 \rightarrow H1$; M3: $v_1^1 : H3 \rightarrow H1$ and $v_2^1 : H1 \rightarrow H4$; and M4: $v_1^1 : H3 \rightarrow H1$ and $v_2^2 : H4 \rightarrow H1$. We can schedule two resource-independent migrations concurrently (M2 and M3). On the other hand, one migration can only be scheduled in sequence after the completion of another dependent migration (M1 and M4).

We use Mininet [29] to emulate the iterative network transmission of the live migration. The execution time for each potential migration of v_1^1 , v_2^1 , v_1^2 , and v_2^2 based on the available bandwidth is 6.2791, 15.0889, 29.1980, and 12.5143 seconds, respectively. The total migration time of combination M1-M4 is 34.8858, 12.4334, 28.4711, and 27.6032 seconds. Moreover, when the service network and migration (control) network are running separately [30], the available bandwidth for each live migration is the same (10 Gbps). Based on multiple migration planning and scheduling algorithms [18], [19], [20], the total migration time of four

different combinations M1-M4 is 28.1936, 12.1227, 22.6056, and 26.8893 seconds, respectively. Comparing M2 with M1 and M4, since there is no resource-dependent migration in M2, the total migration time is significantly shorter. Comparing M2 with M3, although there is no network resource sharing in both combinations, the single live migration overheads of M2 is smaller due to the memory size, dirty page rate, and available bandwidth. Summarily, although all the potential combinations can achieve the desired objective, the scheduling performance of multiple migrations varies considerably. Thus, it is essential to minimize both resource dependencies among migration requests and single live migration overheads during dynamic resource management.

4 PROBLEM MODELING

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

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

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

$$x_{(i,\bar{h}_i)} = 0 \quad \forall \bar{h}_i \in \bar{H}_i = H \setminus H_i, \quad (4)$$

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

The migration execution time t_i^h of $x_{(i,h)} = 1, h! = p_i$ can be calculated according to Equations (1), (2), and (3). Furthermore, we normalize the migration execution time based on the largest and smallest execution time among the different source and destination pairs for every VMs.

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

$$\sum_{h \in H} x_{(i,h)} = 1 \quad \forall i \in N. \quad (5)$$

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

$$x_{(i,h)} - y_{(i,h)} \leq 0 \quad \forall i, h \in N \times H. \quad (6)$$

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

$$1 \leq \sum_{h \in H} y_{(i,h)} \leq 2 \quad \forall i \in N, \quad (7)$$

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

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

$$z_{(i,j,h_1,h_2)} \in \{1, 0\} \quad \forall i, j \in N, h_1, h_2 \in H, \quad (8)$$

where $z_{(i,j,h_1,h_2)} = 1$, if $y_{(i,h_1)} = 1, y_{(j,h_2)} = 1$ and $p_i! = h_1, p_j! = h_2$. Otherwise, $z_{(i,j,h_1,h_2)} = 0$.

There is a resource dependency graph G_{dep} for all possible migrations. Let $v_{s,d}$ denote a migration with source host s and destination host d . If node v_{p_i,h_1} and v_{p_j,h_2} are connected in graph G_{dep} , then edge $e_{(i,j,h_1,h_2)} = 1$. This indicates that potential migrations of VM i from host p_i to h_1 and VM j from host p_j to h_2 are resource-dependent which can only be scheduled in a sequential manner. Thus, the resource dependency between two potential migrations can be represented as:

$$e_{(i,j,h_1,h_2)} \cdot z_{(i,j,h_1,h_2)}. \quad (9)$$

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

$$|O(x_{(i,h)}) - O_{tar}| \leq \varepsilon \quad \forall (i, h) \in N \times H. \quad (10)$$

In practice, we can replace (10) for a specific placement score function. For example, in load balancing policies, let w_i and w_j denote the load of VM i and j . We can represent the constraints of dynamic resource target for the final placement as:

$$\left| \sum_{i \in N} x_{(i,h_1)} \cdot w_i - \sum_{j \in N} x_{(i,h_2)} \cdot w_j \right| \leq \varepsilon' \quad (11)$$

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

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

$$\sum_{i \in N} x_{(i,h)} \cdot w_i \leq C_{(Work)}^h \quad \forall h \in H. \quad (12)$$

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

$$Inter_{single} = \sum_{i \in N} (y_{(i,p_i)} - x_{(i,p_i)}) \cdot t_i^h \quad (13)$$

$$Inter_{multi} = \sum_{i,j \in N, h_1, h_2 \in H} (t_i^{h_1} + t_j^{h_2}) \cdot e \cdot z \quad (14)$$

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

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

$$\min(Inter_{single} + Inter_{multi}), \quad (15)$$

subject to constraints (4), (5), (6), (8), (10), (11), and (12).

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

5 CONCURRENCY-AWARE SELECTION

Solving the MIP model in Equation (15) is NP-hard, it is not practical to use MIP solver to get the solution. In this section, we introduce the Concurrency-Aware Migration (CAMIG) selection algorithm for minimizing the resource dependencies and overheads among migrations during dynamic resource management. Based on the three selection steps of resource management policy, CAMIG has the flexibility to integrate with existing algorithms. Provided that VMs are selected by the policy, CAMIG selects migration destinations to minimize resource dependency. Moreover, if only the management objective and source host selection criteria are given, CAMIG selects both VMs and migration destinations.

The rationale behind CAMIG is to select the migration with the least resource dependency and single migration overhead in each round with the currently selected migrations and minimize the dependency for the future one based on maximal cliques and independent sets of the resource dependency graph. Graph theory concepts, such as maximal cliques and independent sets, are explained in Section 5.2. There are mainly three steps: (1) build the migration dependency graph; (2) get all maximal cliques and independent sets of a migration from the dependency graph; and (3) calculate the single migration interference and migration concurrency metric (MIGC) of candidate migrations.

5.1 Migration Dependency Graph Build

We first explain how to generate the resource dependency graph G_{dep} based on the potential migrating VMs and destinations. For the undirected graph $G_{dep} = (V, E)$, let v ($v \in V$) be the source-destination pair (src-dst) node or vertex representing one potential migration. Migrations with same src-dst node are categorized in list $M(v_{sd})$. Let $e(v, u) \in E$ be the dependency between two migrations with src-dst node v and u . As shown in

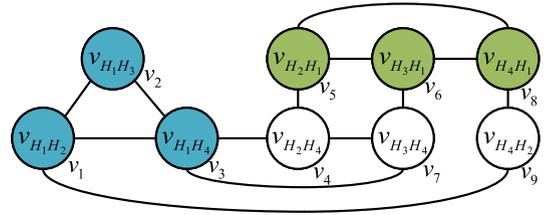


Fig. 5. A resource dependency graph with two of its maximal cliques marked by color.

Algorithm 1, with the input of potential migrating VMs and corresponding destination candidates H_i , we first add src-dst nodes and classify potential migrations into the corresponding node in $M(v_{sd})$. Then, we add edges into G_{dep} based on the source and destination of each node. Fig. 5 demonstrates an illustrative example of resource dependency graph based on a given list of potential migrations (v_1 to v_9) in a specific dynamic resource management which involves 9 src-dst pairs in the same physical network topology shown in Fig. 4a (four hosts connected through one switch). Each vertex v_{H_s, H_d} indicates the pair of source and destination host for a group of potential migrations. For the sake of conciseness, we use v_1 to v_9 to represent node v_{H_1, H_2} to v_{H_4, H_2} .

Algorithm 1. Create G_{dep} and v_{sd} Queues

Input: potential VM $i \in N$, Destinations $\{H_i\}$
Result: migration depGraph $G_{dep}, \{M(v_{sd})\}$

- 1 **foreach** $i \in N$ **do**
- 2 $s \leftarrow p_i$;
- 3 **foreach** $d \in H_i$ **do**
- 4 AddNode(G_{dep}, v_{sd});
- 5 $M(v_{sd}) \leftarrow M(v_{sd}) \cup i$;
- 6 **foreach** $v \in V(G_{dep})$ **do**
- 7 **foreach** $u \in V(G_{dep})$ **do**
- 8 **if** $v \neq u$ **then**
- 9 **if** ISDependent(u, v) **then**
- 10 AddEdge($G_{dep}, (u, v)$);
- 11 **return** $G_{dep}, \{M(v_{sd})\}$

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

Note that the dependency graph supports the multiple routing transmission and dynamic migration routing based on the current network status. In certain data center networks, multi-path transmission and multiple network

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

Fig. 6. All maximal cliques and MISs of G_{dep} in Fig. 5.

interfaces of physical hosts are supported. Thus, the vertex v_{sd}^P in G_{dep} can be extended to indicate the network paths P_{sd} for migrations from the specified network interfaces set s of source host to interfaces set d of destination host. Let $u(P)$ indicate the available bandwidth of network paths P . Given two pairs of src-dst interfaces set (s_j, d_j) and (s_k, d_k) and corresponding network paths P_j and P_k , two vertices v_j and v_k are resource-independent, when statement (16) are true and $s_k \cap s_j = \emptyset$ and $d_k \cap d_j = \emptyset$:

$$\begin{aligned} u(P_j) - u(P_j \cap P_k) &\geq \min(u(P_j), NC_s^j, NC_d^j) \wedge \\ u(P_k) - u(P_k \cap P_j) &\geq \min(u(P_k), NC_s^k, NC_d^k), \end{aligned} \quad (16)$$

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

5.2 Maximal Cliques and Independent Sets

Before discussing how to get maximal cliques and maximal independent sets (MISs) which include a certain node v , we first review some basic concepts, such as clique, independent set, and degeneracy. A clique is a subset of vertices of an undirected graph G such that every two distinct vertices in the subset are adjacent [31]. The maximal clique is a clique that cannot be extended by including one more adjacent vertex. An independent set of a graph G is the opposite of a clique that no two nodes in the set are adjacent. Fig. 6 shows all maximal cliques and MISs of the G_{dep} (Fig. 5). For example, $\{v_3, v_1, v_2\}$ is one of its maximal cliques and $\{v_2, v_7, v_5, v_9\}$ is one of its MISs. The problems of finding all maximal independent sets and cliques are complementary and NP-hard [31], [32]. Finding all maximal independent sets of a graph is equal to finding all maximal cliques of its complement graph [33]. As a robust metric to indicate graph density or sparseness, degeneracy of a graph G is the smallest value d such that every nonempty subgraph of G contains a vertex of degree at most d [34].

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

Algorithm 2. Get All MISs of Node v in G_{dep}

Input: G, \bar{G}, v , node neighbors in \bar{G} $\{\bar{G}_N(n)\}$, $n \in V$
Result: All MISs $\{I^v\}$ of node v , $v \in V(G_{dep})$

- 1 **Function** Cliques($\bar{G}, cand$)
- 2 $n \leftarrow \text{GetNodeMaxDegree}(\bar{G});$
- 3 **foreach** $m \in cand - \bar{G}_N(n)$ **do**
- 4 $del(m, cand);$
- 5 $I \leftarrow I \cup \{m\};$
- 6 **if** $\bar{G} \cap \bar{G}_N(m) = \emptyset$ **then**
- 7 $\{I^v\} \leftarrow \{I^v\} \cup I;$
- 8 **else**
- 9 **if** $cand \cap adj[m] \neq \emptyset$ **then**
- 10 $\text{Cliques}(\bar{G} \cap \bar{G}_N(m), cand \cap \bar{G}_N(m));$
- 11 $del(m, I);$
- 12 **End Function**
- 13 $I^v \leftarrow \emptyset; I \leftarrow \{v\};$
- 14 $cand \leftarrow cand - G_N(v) - \{v\};$
- 15 $V(G) \leftarrow V(G) - G_N(v) - \{v\};$
- 16 **return** Cliques($\bar{G}, cand$);

We propose an algorithm for listing $\{C^w\}$ and $\{I^v\}$ based on $\{C\}$ of dependency graph. For getting all maximal cliques $\{C\}$ of a graph, the general-purpose algorithms for listing all maximal cliques [33], [35] based on Bron-Kerbosch algorithm [31] take exponential time due to the maximum possible number of cliques. These general-purpose algorithms are not sensitive to the density of a graph. Therefore, parametrized by degeneracy, we use a variant algorithm Bron-Kerbosch Degeneracy [36] to generate all maximal cliques of the original resource-dependency graph without duplication. All maximal cliques are generated in the tree-like structure by employing the pruning methods with pivoting to allow quick backtrack during the search. Based on the Bron-Kerbosch algorithm with pivoting, the Bron-Kerbosch Degeneracy uses a degeneracy ordering to order the sequence of recursive calls without pivoting at the outer level of the original Bron-Kerbosch algorithm [36]. Applied to a n -vertex graph with d degeneracy, it lists all maximal cliques in time $O(dn3^{d/3})$.

As shown in the dependency graph property analysis (Appendix A, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TPDS.2021.3139014>) and the time analysis in the performance evaluation (Section 6.2), it is not practical to generate all maximal independent sets $\{I\}$ due to the density of the complement of G_{dep} . Thus, we propose a clique-based maximal independent set algorithm to calculate $\{I^v\}$. As shown in Algorithm 2, it first excludes all adjacent nodes of v in the resource dependency graph G . Then, it chooses node with maximum degree from each connected candidates of the remaining complement graph \bar{G} recursively in a branch-and-bound method until there is no vertex left. Algorithm 2 can achieve the worst-case optimal time complexity of finding all MISs of a node v as $O(3^{m/3})$ [35], where $m = |V(G) - G_N(v)| - 1$.

5.3 Concurrency for Migration Candidates

In this section, we introduce the migration concurrency metric (MIGC) to indicate the resource dependency level of a potential migration. It is based on the maximal cliques and independent sets of an src-dst pair node. Let

M_{mig}^x be the list of migrations have been selected currently. Let M^x be the list of src-dst pair nodes v_j of each migration $m_j \in M_{mig}^x$. For the first round $x = 0$, when the list of selected VM migration is empty, MIGC can be calculated as:

$$MIGC_v = \kappa \cdot \max(|C^v|) / \max(|I^v|) \quad (17)$$

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

$$MIGC_v^{M^x} = MIGCliq_v^{M^x} + 1 / MIGInd_v^{M^x} \quad (18)$$

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

$$MIGInd_v^{M^x} = \frac{\sum_{v_j \in M^x} \sum_{I^v \in \{I^v\}} |v_j \cap I^v|}{|\{I^v\}| \cdot |M^x|} \quad (19)$$

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

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

$$MIGCliq_v^{M^x} = \frac{\sum_{v_j \in M^x} \sum_{I^v \in \{C^v\}} |v_j \cap C^v|}{|\{C^v\}| \cdot |M^x|} \quad (20)$$

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

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

5.4 Concurrency-Aware Migration Selector

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

Algorithm 3. CAMIG

Input: Performance Objective $Score^*$, potential VMs i , source H_s , dst H_d

Result: Selected Migration List M_{mig}

- 1 Step 1. get node clique matrix
- 2 $G_{dep}, \{M(v_{sd})\} \leftarrow \text{CreatedepGraph}(H_s, H_d, k);$
- 3 $\{C\} \leftarrow \text{allCliques}(G_{dep});$
- 4 $x \leftarrow 0; M^x \leftarrow \emptyset; M_{mig}^x \leftarrow \emptyset;$
- 5 **do**
- 6 Step 2. get candidate VMs
- 7 $\text{UpdateMigInterference}(VM_i, H_d^i, L_{sd}^i);$
- 8 $\hat{Score}^{x+1}, \{v_{sd}^j\}, \{m_{sd}^j\} \leftarrow \text{GetMigCandidates}(p_{current}, \{w_i\}, \{H_d^i\}, Score^x, M_{mig}^x);$
- 9 Step 3. select the optimal migration
- 10 $\hat{v}_{sd}^j \leftarrow v_{sd}^0; \hat{m}^j \leftarrow m_{sd}^0;$
- 11 **if** $|\{v_{sd}^j\}| > 1$ **then**
- 12 **foreach** $v \in \{v_{sd}\}$ **then**
- 13 $C^v = \text{allCliques}(\{C\}, v);$
- 14 $I^v = \text{allIndepSet}(G_{dep}, \{C\}, v);$
- 15 **if** $Inter^{j,v} < Inter_{min}$ **then**
- 16 $Inter_{min} \leftarrow Inter^{j,v};$
- 17 $\hat{v}_{sd}^j \leftarrow v_{sd}^j; \hat{m}^j \leftarrow m_{sd}^j;$
- 18 $M^{x+1} \leftarrow M^x \cup \hat{v}_{sd}^j; M_{mig}^{x+1} \leftarrow M_{mig}^x \cup \hat{m}_{sd}^j;$
- 19 $\text{UpdatedepGraph}(G_{dep}, \{C\}, \hat{m}_{sd}^j, \hat{v}_{sd}^j);$
- 20 **while** $|\hat{Score}^{x+1} - Score^*| > \delta$ and $Score^{x+1} > \hat{Score}^x$ and $x + 1 < |\{m\}|$
- 21 **Return** M_{mig}

In *Step 1*, G_{dep} and $M(v_{sd})$ are generated according to Algorithm 1. In *line 3*, we find all maximal cliques $\{C\}$ of G_{dep} . From *line 5-18*, at each round x , we select the optimal migration from src-dst node \hat{v}_{sd}^j based on both $MIGC$ and single migration overhead $Inter_{single}$. As a result, it gets the overall minimal dependencies and single overheads of the total migrations to satisfy the objective of the dynamic resource management. For *Step 2*, in each optimal round, it first updates the single migration interference of each candidate VM for its potential destinations. According to the selected migrations of previous rounds M_{mig}^x and current placement, it gets the newest VM to Host mapping. Then, it obtains the candidate migrations $\{m_{sd}^j\}$ and corresponding pairs v_{sd}^j in this round with the same objective score \hat{score}^{x+1} . It can generate more potential migrations by enlarging the score tolerance of the optimal objective in each round. For *Step 3*, the optimal migration with the minimum total migration interference $Inter_{min}$ is selected. It first calculates $\{C^v\}$ based on all maximal cliques $\{C\}$ generated based on Bron-Kerbosch Degeneracy algorithm and $\{I^v\}$ according to Algorithm 2. Then, based on the pair list of already selected migrations M^x , the migration overhead of migration m_i with src-dst pair v can be calculated as:

$$Inter^{i,v} = \kappa_{mig} \cdot Inter_{single}^{i,v} + \kappa_{mig} \cdot Inter_{single}^{i,v} \cdot MIGC_v^{M^x}, \quad (21)$$

where κ_{mig} is the coefficient for the value normalization of single migration overheads. Then, the single migration overhead $Inter_{single}^{i,v}$ and $MIGC_v^{M^x}$ can be calculated based on Equation (1), (2), and (3), and (17), (18), and (19), respectively. In *line 17*, it adds the optimal migration of this round

\hat{m}_{sd}^j and its pair node \hat{v}_{sd}^j to the currently selected migration list M_{mig}^x and corresponding node list M^x .

In *line 19*, algorithm *UpdatedepGraph* updates the dependency graph and all maximal cliques according to the selected migration. Certain potential migrations related to the selected optimal migration are deleted from the pair list. For example, in Section 3.4, if we choose migration $v_1^1 : H1 \rightarrow H3$, then $v_1^2 : H3 \rightarrow H1$ is excluded for future selection. Note that we do not need to use Bron-Kerbosch Degeneracy to recalculate $\{C\}$ based on the new subgraph (Theorem 1). If the pair list is empty after update $M_{sd} = \emptyset$, the corresponding node v_{sd} will be removed from G_{dep} and $\{C\}$. If the updated clique size is 1 and the only one vertex left has connected edge, remove such clique. Duplicated cliques are also removed.

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

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

Proof. Bron-Kerbosch Degeneracy generates all and only maximal cliques $\{C\}$ of G [36]. (1) For $\forall C', \forall C'', |C'| = 1$ and $|C''| = 1$. Because the $V(G) \setminus \{v'\} = V(G')$. Thus, $\{C'\} = \{C''\}$. (2) For $\forall C', \forall C'', |C'| > 1$ and $|C''| > 1$. For the sake of prove, we assume that $\exists C', C' \notin \{C''\}$. Then, $\exists C_e, \exists C'_e, C_e = C' \cup \{v_e\} \cup \{v'_e\}$, where $C_e \in \{C\}$, $C'_e \in \{C'\}$, part of remaining vertices $\{v_e\} \subseteq V(G) \setminus \{v'\}$, part of removing vertices $\{v'_e\} \subseteq \{v'\}$. Then, we have $C'_e \cup \{v_e\} \in \{C''\}$. If $\{v_e\} \neq \emptyset$, because $\forall C', \exists C', C' \subseteq C$, then $C'_e \cup \{v_e\} \in \{C'\}$. We have a contradiction, as C'_e is a maximal clique of G' . If $\{v_e\} = \emptyset$ or $C_e = C'_e$, as the UpdatedepGraph removes all $v' \in \{v'\}$, we have a contradiction $C'_e \in \{C''\}$. Thus, $\forall C' \in \{C''\}$. Similarly, we can prove $\forall C'' \in \{C'\}$. Therefore, $\{C'\} = \{C''\}$. \square

The worst-case running time of Bron-Kerbosch Degeneracy is $O(dn3^{d/3})$ [36] with total n vertices and degeneracy d . The upper bound of all maximal cliques/independent sets of a Graph G is $(n-d)3^{d/3}$. Thus, given c maximal cliques, the time complexity of the algorithm for calculating MIGC is $O(cn)$. Then, the worst-case running time of CAMIG is $O((n-d)n2^{3d/3})$. We perform extensive computational evaluation on time complexity in Section 6.2. It demonstrates that algorithm CAMIG is very fast in practice.

6 PERFORMANCE EVALUATION

In this section, we evaluate the performance of our proposed concurrency-aware migration selection (CAMIG) algorithm for dynamic resource management with several parameters, such as total migration time, total migration number, and corresponding dynamic resource management performance in load balancing and energy-saving scenarios. We used both real-world workload trace from PlanetLab [37] and synthetic workloads for the evaluation. We also performed extensive

computational experiments for time analysis. The results show that the proposed algorithm can significantly improve the multiple migration performance [20] while achieving the target of resource management.

The scalability of Mininet is limited due to the limitation of its resource usage and the operating systems, which prevents the cloud-scale simulations. Furthermore, it can not simulate the computing resource for the dynamic resource management and multiple migration scheduling. Thus, we have implemented components for the multiple migration scheduling simulations [20] based on the CloudSimSDN [38]. The accuracy of network processing of CloudSimSDN compared to Mininet is validated in [39]. Based on the phases of pre-copy migration, the event-driven simulator¹ can evaluate the performance of multiple migrations in terms of the total migration time, migration execution time, total transferred data, and downtime.

6.1 Load Balancing Scenario

In this section, we evaluate the impact of migration concurrency during the dynamic resource management on the performance of multiple migration scheduling in load balancing scenarios. The target of the resource management policy in this experiment is to keep the total CPU utilization of each physical host to 50%. For other solutions besides the optimal, we set the target range of the total CPU utilization from 45% to 55%. We compare our algorithm CAMIG with the result of the optimal and other load-balancing algorithms: Sandpiper [2], FFD [3], and iAware [12]. We first evaluate algorithms on small-scale experiments with 8 physical hosts in a Fat Tree. Then, we extend the experimental scale for complex scenarios with more resource dependencies. In extensive experiments, by integrating the proposed concurrency-aware algorithm with existing dynamic resource management algorithms, we directly evaluate and illustrate the scheduling performance improvement in multiple migration planning and scheduling algorithm [20].

6.1.1 Experimental Setup

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

The actual location of physical hosts in Fat Tree topology with different resource utilization is generated randomly, which causes different source and destination selections and resource dependencies in each random setup. Without specific explanation, the result is the average value of 10 experiments. Causing utilization difference among hosts, the initial placement of VMs in each machine with different CPU utilization and memory size is shown in Fig. 7. To differentiate the migration value in management objective and migration schedule, we create VMs with different combinations of high, medium, and low value of resource utilization and memory size. The CPU utilization of each VM is from

1. CloudSimMig. <https://github.com/hetianzhang/CloudSimMig>

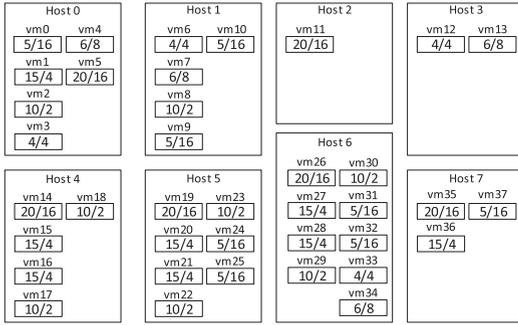


Fig. 7. Initial mapping for 8 different physical hosts with CPU utilization (%)/Requested Memory(GB).

4% to 20% of the total host CPU resource. As a result, the CPU utilization of each host is from 10% to 90%. The Memory size of each VM is from 2 GB to 16 GB, which can result in various migration overheads.

Other parameters of pre-copy migration are set as the same for each VM. The dirty page rate factor is 0.001 per second. For example, with a 0.001 per second dirty page rate factor, the dirty page rate of a VM with 16 GB memory is 128 Mbps. The data compression ratio is 0.8. The iteration and downtime threshold is 30 and 0.5 seconds, respectively. We create a k-8 FatTree Data Center Network (128 hosts) with 1 Gbps bandwidth between switches. For the purpose of irrelevant parameter exclusion in experiments, each physical host has 16 CPUs with 10000 MIPS each, 10GB RAM, 1 TB storage, and 1 Gbps network interface. Note that hosts are not required to be identical in the proposed algorithm.

Dual simplex (Gurobi optimizer 9.0² and Python-MIP 1.6.7³) were used to get the optimal solution of the MIP model. We also proposed a baseline algorithm called HostHits (hht). As shown in CAMIG selections, several potential destinations can achieve the same objective of dynamic resource management. It chooses the least selected/hit host as the destination of VM migration in each migration selection iteration.

For original Sandpiper, FFD and iAware without multiple migration scheduling, the sum of migration execution time is the actual total migration time of these algorithms, because they only consider one-by-one migration scheduling. However, given the multiple migration requests, we apply the multiple migration planning and scheduling algorithm [20] to all resource management algorithms in experiments and evaluate and show the results of corresponding performance in multiple migration scheduling.

The rationale is that Sandpiper chooses the largest volume/memory VM from one of the most overloaded physical host to minimize live migration overheads. The volume as the multi-dimensional loads indicator is defined as: $Volume = \frac{1}{(1-cpu)(1-net)(1-mem)}$ [2], where *cpu*, *net*, and *memory* are normalized utilizations of corresponding resources. FFD (First-Fit Decreasing) algorithm selects the smallest size VMs from over-utilized hosts and assigns them in the FFD ordering of the spare resources to under-utilized hosts. iAware considers both co-location VM interference and the single live migration overheads. The co-location VM interference is linear to the number of VMs one physical machine

2. Gurobi solver, <https://www.gurobi.com/>

3. Python-MIP. <https://github.com/coin-or/python-mip>

TABLE 2
Total Migration Time/Sum of Migration Execution Time Comparison in the Extending Mapping Scenarios

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

hosts in Xen. The migration selection in iAware is sequentially decided in each round of the greedy algorithm.

6.1.2 Scalability Evaluation

We extend the scale of experiments (multi2, multi3, and multi4) by multiplying the same mapping 2, 3, and 4 times. Total of N hosts are randomly placed among the first N number locations in the Fat Tree topology with 128 hosts. Each scenario has 16, 24, 32 candidate destination hosts with a total 76, 114, and 152 potential migration VMs, respectively. For example, the physical Host 16, Host 8 and Host 0 have the same VM initial allocation. However, for each scenario, the placement of each physical host in the FatTree is generated randomly. As the resource management algorithms do not have the prior knowledge of the initial placement, the combination of source, destination, and instances during migration selection is increased exponentially. As a result, with the experiment scale increasing, more random source and destination combinations of potential migrations are generated for each experiment. We conducted 10 experiments in each scenario and show the average results.

Tables 2 and 3 show the results of the optimal solution, CAMIG and the optimal solution with Sandpiper VM selection, Sandpiper, FFD, and iAware in total migration time with multiple migration schedule, total migration execution time (one-by-one schedule), the number of dependent migration tasks, multiple migration interference value, and the load-balancing performance (standard deviation of CPU utilization). The multiple migration interference value is the sum of normalized single overheads from dependent migrations. Although all physical hosts are arranged randomly, the optimal result should be the same as in scenario multi1.

Analysis. Tables 2 and 3 show that the MIP model achieves the optimal in all scenarios. With the source host selection from Sandpiper, comparing CAMIG with the optimal solution, as the problem scale increases, CAMIG can maintain the optimal performance in multiple migration scheduling as well as the number of resource-dependent migrations. In multi3 and multi4, CAMIG over-satisfies the requirement of load-balancing by losing the value of multiple migration interference. For the Sandpiper and iAware, as the the scale of the problem increases, the number of dependent migrations and the value of multiple migration interference increase dramatically,

TABLE 3
Comparison of Dependent Migrations/Multiple Migration Interference/Standard Deviation of CPU Utilization

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

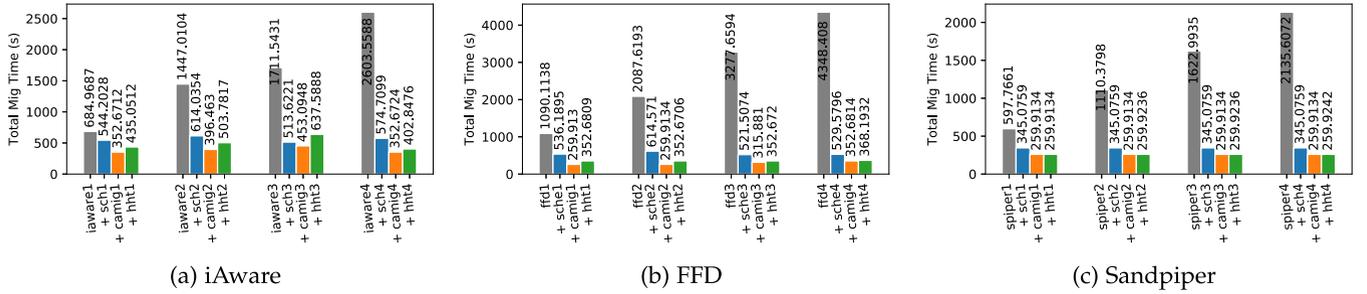


Fig. 8. Performance comparison with one-by-one, direct multiple scheduling, CAMIG and HostHits.

which leads to a larger total migration time in both multiple and one-by-one scheduling. FFD can not satisfy the requirement of load-balancing in the system.

The total migration time of Sandpiper is increased by 15.01% in multi3 and multi4. In Table 3, although FFD has the lowest total migration time and migration execution time, it cannot achieve the ideal load-balancing performance. The standard deviation of FFD is the largest among other algorithms. Moreover, the largest total migration is increased by 21.33% compared to the lowest. For iAware, the actual total migration time equals to the total migration execution time by only allowing one-by-one scheduling. With multiple migration scheduling, iAware has the worst performance in total migration time and load-balancing due to the trade-off between migration execution time and collocation interference. The total migration time varies largely in different scenarios, increasing at most 568.68%.

6.1.3 Extensive Evaluation

As every load-balancing policy has its own logic for VM selection, it is difficult to evaluate the improvement of multiple migration directly. Thus, in this section, we extended the experiments by integrating the HostHits and CAMIG algorithm with the existing policies: iAware, FFD, and Sandpiper. With the benefit of flexibility, CAMIG can be adapted to other existing dynamic resource management algorithms. We randomly

generated VM Memory Size from 8 to 14 GB with the same scenarios (Fig. 7). Each result is the average value of 10 experiments in each scenario. Fig. 8 illustrates the multiple migration performance in total migration time based on the migration requests of these policies with one-by-one scheduling and multiple migration scheduling (+sch), and multiple migration scheduling performance based on the migration requests of CAMIG (+camig) and HostHits (+hht) in 4 different scenarios.

Analysis. Fig. 8a indicates that iAware with CAMIG can achieve the best performance with multiple migration scheduler in all 4 scenarios. The performance is increased by 20.55%, 57.57%, 70.02%, and 77.93% when migration requests scheduled by the multiple migration scheduler, respectively. However, with CAMIG the performance is increased by 48.54%, 72.63%, 73.52%, and 86.48% compared to the original iAware and increased by 35.29%, 35.50%, 11.89%, and 38.68% compared to the performance of iAware with only multiple migration scheduler. Moreover, although iAware with HostHits generally has a better performance compared to iAware +scheduler, as shown in scenario multi3, it results in a worse total migration time due to creating a larger clique of the dependency graph. For FFD, CAMIG can increase the performance up to 91.90%, 57.82%, and 26.42% compared to FFD with one-by-one scheduler, multiple migration scheduler, and HostHits (Fig. 8b). Moreover, Fig. 8c shows that the performance of Sandpiper with CAMIG in total migration time is increased by up to

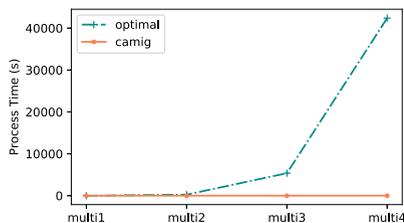


Fig. 9. Runtime comparison between optimal and CAMIG.

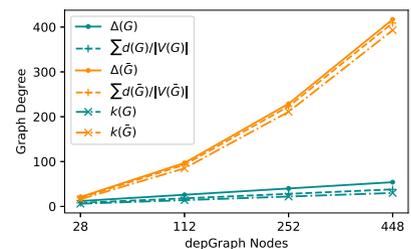


Fig. 10. Average and maximum degree and degeneracy.

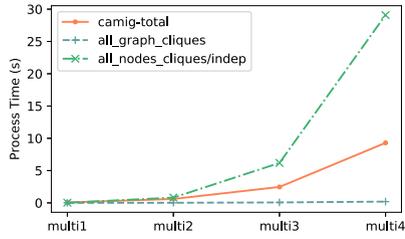


Fig. 11. Runtime of CAMIG, all maximal cliques, and all maximal cliques/independent sets of nodes.

87.87% and 24.68% than Sandpiper with one-by-one scheduler and multiple migration scheduler, respectively.

6.1.4 Summary

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

6.2 Processing Time Analysis

In this section, we analyze the time complexity of the proposed CAMIG algorithm. The experiments were run in the computer with i7-7500U CPU with 2.70 GHz, and 15.9 GB RAM in Windows 10 64-bit Operating System. Fig. 9 illustrates that the runtime of the optimal solution solved by MIP solver is increased exponentially against the linear growth of the problem size. The runtime of the optimal solution on average is 3.07s, 251.51s, 5373.35s, and 42388.0s in 4 scenarios, respectively. Thus, it is impractical to generate the optimal result when facing the problem in real life.

Fig. 10 illustrates the connectivity properties of dependency graph in terms of average degree $\sum d(G)/|V(G)|$, maximum degree $\Delta(G)$, and degeneracy of the dependency $k(G)$ and its complement \bar{G} . The number of maximal cliques is 12, 28, 42, 56 with the degeneracy (a measure of graph sparseness) of the dependency graph as 6, 14, 22, 30. Therefore, it is much easier to generate all maximal cliques with a small degeneracy. However, the degeneracy of the complement dependency graph increased dramatically as 16, 85, 211, 393. Thus, it is impractical to generate all maximal cliques of the complement graph as the problem size becomes significantly large. In other

words, Bron-Kerbosch Degeneracy algorithm can reach the worst-case runtime when the graph becomes considerably dense. As a result, it can only generate all 661 maximal independent sets in the smallest scale scenario (multi1). Fig. 11 shows the runtime comparison of CAMIG in total processing time, finding all maximal cliques, and generating all maximal cliques and independent sets for every node. As shown in Algorithm 2, we do not need to calculate all maximal cliques and independent sets of every node in the graph. The *all_nodes_cliques/indep* illustrates the upper-bound of runtime. The processing time of CAMIG is increased linearly against the total src-dst node in resource dependency and the average degree or the degeneracy of the complement of the dependency graph as shown in Fig. 10.

6.3 Long-term Energy Saving Scenario

To evaluate the proposed algorithm with the real-world long-term workloads [37], we compared CAMIG with LR-MMT [5] in the energy-saving scenario in terms of total migration time, migration numbers, downtime, total/average CPU serve time with and without the timeout workloads, and energy (power) cost of both hosts and switches.

6.3.1 Evaluation Configuration

For the long-term experiments, we created a k-16 FatTree topology (1024 hosts) with 1 Gbps physical links between switches to simulate the environment with limited network resources for live migrations. Each physical host has 8 CPUs with 4000 MIPS, 1024 GB Memory size, 1000 GB Storage, and 1 Gbps network interface. The real-world workload trace of CPU utilization from Planetlab [37] was used for the experiments running in 24 hours. There are 1052 CPU utilization files mapping to the same amount of VMs. We generated the workloads based on the MIPS requirement and the CPU utilization varied along the time. In order to illustrate the influence of multiple migration performance, there is no application traffic between different VMs other than the migration flows. There are 4 flavors of VM: 2 vCPUs, [2500, 2000, 1000, 1000] MIPS, [2, 4, 4, 2] GB RAM, 100 Mbps virtual bandwidth, and 4 GB Disk Size. The initial placement of VMs are allocated based on the optimization criteria defined by LR-MMT [5].

The LR-MMT algorithm utilizes the Local Regression (LR) method to predict overloading hosts in the upcoming monitor interval. Minimum Migration Time (MMT) policy is used for VM selection to minimize migration overheads. During each monitoring interval of dynamic resource management, CAMIG, as a flexible algorithm, utilizes the same local regression to detect over/under-utilized hosts. In LR-

TABLE 4
Performance Comparison Between LR-MMT, HostHits, CAMIG in Energy-Saving Scenario

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

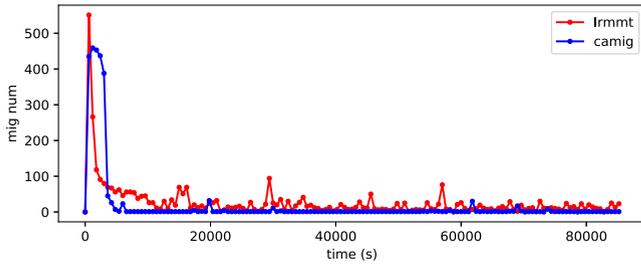


Fig. 12. Migration number within each interval.

MMT, though there are many equivalent optimal destinations, it only chooses the first fit. For the sake of fair comparison, destination candidates used in CAMIG are provided by the same energy-saving policy in LR-MMT.

6.3.2 Evaluation Results

As shown in Table 4, CAMIG algorithm outperforms both LR-MMT and HostHits. The total energy consumption under no dynamic resource management is 1733432.22 Wh. The LR-MMT algorithm saves 72.86% energy consumption. Comparing CAMIG with LR-MMT, the host and switch energy consumptions are 3.78% and 38.01% less, respectively. The total migration number is 32.26% less, the sum of total migration time of each monitoring interval is 73.42% less, the total downtime is 49.85% less than the LR-MMT algorithm. The performance improvements in total migration time also result in fewer workload timeouts and CPU resource shortages. For VM processing, the average CPU server time is 92.70% less when there is no timeout mechanism. With a timeout mechanism, CAMIG also reduces the workload timeout by 14.30% compared to the LR-MMT.

As the sum of total migration time and total migration time of each monitoring interval shown in Table 4 and Fig. 13, within the 24 hours experiment, the performance of CAMIG in multiple migration scheduling is largely better than the LR-MMT. A shorter total migration time during each monitoring interval means a quicker state convergence for minimizing the over-utilization period and maximizing the energy-saving through VM consolidation for under-utilizing hosts. In other words, minimizing the dependencies among multiple migrations is not only critical for the migration scheduling, but also for the dynamic resource management that provides the migration list.

During the experiments, we find out that there are relatively large equivalent destination candidates in terms of energy saving. Therefore, by exploring the concurrency score among these candidates, we can minimize the resource dependencies among the migrations. As shown in Fig. 12, there are more

migrations in CAMIG from 1200s to 3600s than LR-MMT. It is because in LR-MMT once the candidate is used it will be excluded from the remaining destinations. However, by choosing equivalent hosts during the destination selection, CAMIG algorithm enables more available destinations for VMs which need to be migrated from both under and over-utilized hosts. Thus, CAMIG algorithm actually produces fewer migrations in the remaining monitor intervals. It also illustrates that in some cases even the total migration number of CAMIG is larger, the total migration time is much smaller due to the minimum dependency among the migrations. Fig. 13 shows that, under certain circumstances (the peak migration time at 20000 second), even if there is a small number of migration tasks, the total migration time is still very large. Due to the nature of the consolidation algorithm, there are many migration tasks sharing the same destination or source hosts. Therefore, in traditional architectures, such as FatTree or even the dedicated migration network, it is inevitable that the convergence of multiple migrations is slower. As a result, the performance of multiple migration scheduling may be limited by this nature of resource competition among the consolidating VM migrations.

In summary, the evaluation demonstrates that, CAMIG can efficiently minimize the resource dependency among multiple migration tasks and achieve the objective of dynamic resource management in the long run. Thus, it also improves the performance of dynamic resource management algorithms in terms of QoS and energy consumption.

7 CONCLUSION

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

ACKNOWLEDGMENTS

The authors would like to thank the Editor-in-Chief, Associate Editor, anonymous reviewers, Redowan Mahmud, Linnan Ruan, Tawfiqul Islam, and Shashikant Ilager for their valuable comments and suggestions to help improve the paper.

REFERENCES

- [1] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vNF placement at the network edge," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 693–701.
- [2] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Sandpiper: Black-box and gray-box resource management for virtual machines," *Comput. Netw.*, vol. 53, no. 17, pp. 2923–2938, 2009.

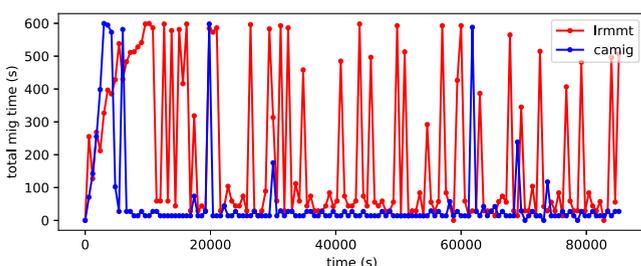


Fig. 13. Total migration time within each interval.

- [3] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," in *Proc. 9th ACM/IFIP/USENIX Int. Conf.*, 2008, pp. 243–264.
- [4] V. Mann et al., "Remedy: Network-aware steady state VM management for data centers," in *Proc. Int. Conf. Res. Netw.*, 2012, pp. 190–204.
- [5] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency Comput. Pract. Exp.*, vol. 24, no. 13, pp. 1397–1420, 2012.
- [6] C. Clark et al., "Live migration of virtual machines," in *Proc. 2nd Conf. Symp. Netw. Syst. Des. Implementation*, 2005, pp. 273–286.
- [7] "Container migration with Podman on RHEL," Accessed: Jan. 22, 2020. [Online]. Available: <https://www.redhat.com/en/blog/container-migration-podman-rhel>
- [8] "Dynamic resource management in E2 VMs," Accessed: Jun. 29, 2021. [Online]. Available: <https://cloud.google.com/blog/products/compute/understanding-dynamic-resource-management-in-e2-vms>
- [9] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, pp. 1–17.
- [10] V. Marmol and A. Tucker, "Task migration at scale using CRIU," in *Proc. Linux Plumbers Conf.*, 2018.
- [11] A. Ruprecht et al., "VM live migration at scale," *ACM SIGPLAN Notices*, vol. 53, no. 3, pp. 45–56, 2018.
- [12] F. Xu, F. Liu, L. Liu, H. Jin, B. Li, and B. Li, "iAware: Making live migration of virtual machines interference-aware in the cloud," *IEEE Trans. Comput.*, vol. 63, no. 12, pp. 3012–3025, Dec. 2014.
- [13] J. Bi et al., "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Trans. Sci. Eng.*, vol. 14, no. 2, pp. 1172–1184, Apr. 2017.
- [14] Q. Wu, F. Ishikawa, Q. Zhu, and Y. Xia, "Energy and migration cost-aware dynamic virtual machine consolidation in heterogeneous cloud datacenters," *IEEE Trans. Serv. Comput.*, vol. 12, no. 4, pp. 550–563, Jul./Aug. 2019.
- [15] Q. Peng et al., "Mobility-aware and migration-enabled online edge user allocation in mobile edge computing," in *Proc. IEEE Int. Conf. Web Serv.*, 2019, pp. 91–98.
- [16] Y. Jiang, J. Wang, J. Shi, J. Zhu, and L. Teng, "Network-aware virtual machine migration based on gene aggregation genetic algorithm," *Mobile Netw. Appl.*, vol. 25, pp. 1457–1468, 2020.
- [17] S. Ghorbani and M. Caesar, "Walk the line: Consistent network updates with bandwidth guarantees," in *Proc. 1st Workshop Hot Top. Softw. Defined Netw.*, 2012, pp. 67–72.
- [18] M. F. Bari, M. F. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "CQNCR: Optimal VM migration planning in cloud data centers," in *Proc. Netw. Conf.*, 2014, pp. 1–9.
- [19] H. Wang, Y. Li, Y. Zhang, and D. Jin, "Virtual machine migration planning in software-defined networks," *IEEE Trans. Cloud Comput.*, vol. 7, no. 4, pp. 1168–1182, Oct.–Dec. 2019.
- [20] T. He, A. N. Toosi, and R. Buyya, "SLA-aware multiple migration planning and scheduling in SDN-NFV-enabled clouds," *J. Syst. Softw.*, vol. 176, 2021, Art. no. 110943.
- [21] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the performance of virtual machine migration," in *Proc. IEEE Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst.*, 2010, pp. 37–46.
- [22] C. Jo, Y. Cho, and B. Egger, "A machine learning approach to live migration modeling," in *Proc. Symp. Cloud Comput.*, 2017, pp. 351–364.
- [23] T. He, A. N. Toosi, and R. Buyya, "Performance evaluation of live virtual machine migration in SDN-enabled cloud data centers," *J. Parallel Distrib. Comput.*, vol. 131, pp. 55–68, 2019, doi: [10.1145/3387514.3405885](https://doi.org/10.1145/3387514.3405885).
- [24] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers," in *Proc. ACM/IEEE Conf. Supercomput*, 2008, pp. 1–12.
- [25] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.
- [26] N. McKeown et al., "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.
- [27] J. Son and R. Buyya, "A taxonomy of software-defined networking (SDN)-enabled cloud computing," *ACM Comput. Surv.*, vol. 51, no. 3, pp. 59:1–59:36, May 2018.
- [28] "CRIU," Accessed: Feb. 22, 2020. [Online]. Available: https://criu.org/Iterative_migration
- [29] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Top. Netw.*, 2010, pp. 1–6.
- [30] K. Tsakalozos, V. Verroios, M. Roussopoulos, and A. Delis, "Live VM migration under time-constraints in share-nothing IaaS-clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 8, pp. 2285–2298, Aug. 2017.
- [31] C. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, 1973.
- [32] E. L. Lawler, J. K. Lenstra, and A. Rinnooy Kan, "Generating all maximal independent sets: NP-hardness and polynomial-time algorithms," *SIAM J. Comput.*, vol. 9, no. 3, pp. 558–565, 1980.
- [33] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theor. Comput. Sci.*, vol. 363, no. 1, pp. 28–42, 2006.
- [34] D. R. Lick and A. T. White, "k-degenerate graphs," *Can. J. Math.*, vol. 22, no. 5, pp. 1082–1096, 1970.
- [35] F. Cazals and C. Karande, "A note on the problem of reporting maximal cliques," *Theor. Comput. Sci.*, vol. 407, no. 1–3, pp. 564–568, 2008.
- [36] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," in *Proc. Int. Symp. Algorithms Comput.*, 2010, pp. 403–414.
- [37] K. Park and V. S. Pai, "CoMon: A mostly-scalable monitoring system for PlanetLab," *ACM SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 65–74, 2006.
- [38] J. Son, T. He, and R. Buyya, "CloudSimSDN-NFV: Modeling and simulation of network function virtualization and service function chaining in edge computing environments," *Softw. Pract. Experience*, vol. 49, no. 12, pp. 1748–1764, 2019.
- [39] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya, "CloudSimSDN: Modeling and simulation of software-defined cloud data centers," in *Proc. IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2015, pp. 475–484.



TianZhang He received the BS and MS degrees in computer science from Northeastern University, China and the PhD degree from the School of Computing and Information Systems, University of Melbourne, in 2014, 2017, and 2021, respectively. His research interests include software-defined networking, edge and cloud computing, and network function virtualization.



Adel N. Toosi (Member, IEEE) received the PhD degree from the School of Computing and Information Systems, University of Melbourne, in 2015. He is currently a lecturer (assistant professor) with the Department of Software Systems and Cybersecurity, Faculty of Information Technology, Monash University, Australia. From 2015 to 2018, he was a postdoctoral research fellow with the University of Melbourne. His research interests include cloud, fog, edge computing, software-defined networking, green computing, and energy efficiency.



Rajkumar Buyya (Fellow, IEEE) is currently a Redmond Barry distinguished professor and the director with the Cloud Computing and Distributed Systems Laboratory, University of Melbourne, Australia. He has authored more than 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 with (h-index=152, g=324, and 120,700+ citations).

▷ For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.