

Efficient Large-Scale Multiple Migration Planning and Scheduling in SDN-enabled Edge Computing

TianZhang He, Adel N. Toosi, *Member, IEEE*, Rajkumar Buyya, *Fellow, IEEE*

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



1 INTRODUCTION

The introduction of edge computing [1] brings opportunities to improve the performance of the emerging user-oriented applications by pushing computation and intelligence to end-users, including Vehicle to Cloud (V2C), Vehicle to Vehicle (V2V), Virtual Reality (VR), Augmented Reality (AR), Artificial Intelligent (AI), or Internet of Things (IoT) applications and so forth. Driven by container virtualization, microservices are more suitable for dynamic deployment on edge computing [2], [3] due to smaller memory footprint and faster startup. By allocating the containerized services in the Edge Data Centers (EDCs) or Mobile Edge Clouds (MECs) [4], strict end-to-end (E2E) communication delays between end-users and services can be guaranteed.

From the centralized cloud computing framework to decentralized edge computing, surveys [5], [6] investigated the challenges faced by the infrastructure and service providers regarding dynamic resource management and user mobility. By providing non-application-specific compute and memory state management, live migration is the solution to these challenges. Live migration of VM [7] and container [8] through the open-source Checkpoint/Restore in Userspace (CRIU) software [9], which had kernel support since Linux 3.11, aims to provide little or no disruption to the running service during migrating in the edge computing. It iteratively copies unfinished computation tasks with intermediate computation states in the memory from source to destination until the memory difference between two

synchronizing instances is small enough for the stop-and-copy phase. In addition, for the container image, if the image does not exist in the destination, it can be transferred from the previous EDC or remote clouds or accessed through shared storage. Thus, live migration performance highly relies on the available bandwidth of the network routing connecting source to destination.

Industrial infrastructure and service providers, such as IBM, RedHat, Google, etc, have been integrating live container migration into their productions [10], [11]. Google has adopted live VM and container migration with CRIU into Borg cluster manager [11], [12], [13] for reasons, such as, higher priority task preemption, software updates, such as kernel and firmware, or reallocating for availability or performance. It manages all compute tasks and runs on numerous container clusters each with up to tens of thousands of machines. A lower bound of 1,000,000 migrations monthly in the production fleet have been performed with 50ms median blackout [13]. Live container migration provides technical simplicity without handling state management and application-specific evictions. However, it is also identified that writing and reading to remote storage through network dominates the checkpoint/restore process and the scheduling delay is the large source of delay regarding the performance of multiple live migrations.

Recently, some works have focused on user or service mobility in mobile edge computing through live container migration [5], [6], [14], [15], [16], [17], [18]. With the limited coverage range of each EDC, when a user moves from one base station to another, the network latency could deteriorate after the network handover. To guarantee the Quality of Service (QoS), the service may need to be migrated from the previous EDC to the proximal one through live container migration. Figure 1 illustrates an example scenario where an

- T.Z. He and R. Buyya are with the CLOUDS Lab, School of Computing and Information Systems, University of Melbourne, Australia. (E-mail: tianzhangh@student.unimelb.edu.au; rbuyya@unimelb.edu.au)
- A. N. Toosi is with the Department of Software Systems and Cybersecurity, Faculty of Information Technology, Monash University, Australia. (E-mail: adel.n.toosi@monash.edu)

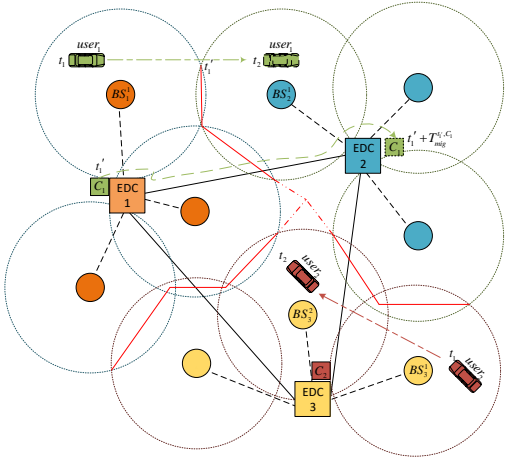


Fig. 1: User-mobility induced live container migration in edge computing environments

autonomous vehicle sends workload to the corresponding stateful service in the EDC for real-time object detection. There is a total of 9 base stations assigned to 3 different EDCs. Two autonomous vehicles move to a new position from time t_1 to t_2 . For vehicle user1, when leaving the range of base station BS_1^1 and entering the range of base station BS_2^1 at time t_1' , there is a live container migration from $EDC1$ to $EDC2$ induced by the user1's movement. Meanwhile, as user2 crosses the range boundary of the base station BS_3^1 to BS_2^2 , since the two base stations both belong to the same edge data center $EDC3$, the E2E delays of the service can be guaranteed. Therefore, there is no live migration induced by user2's movement.

In cloud computing environments, dynamic resource management policies triggers live migration requests periodically to optimize the resource usage or to maintain QoS of applications [19]. However, in the edge environment, service migration/mobility is highly relative to user mobility [6], [14], [15], [17], [18]. Migration requests of containers or VMs from services and users may share and compete for the computing and network resources, such as migration source, destination, and network routings [20]. It brings more challenges for the multiple live migration planning and scheduling. However, most research on live service migration in edge and cloud computing neglects the actual live migration cost regarding the iterative dirty memory transmission [7] and resource competition among migrations in both computing and network resources. As a result, performing multiple live migrations in arbitrary order can lead to service degradation [21], [22].

Few works focus on multiple VM migration planning and scheduling in cloud data centers [21], [22]. The framework of migration scheduling periodically triggered by resource management policies with a long time interval is not suitable for stochastic scenarios of mobility-induced migration in edge computing. Furthermore, the network scale, the numbers of end-users and live container migration requests increase ten thousand times in edge environments. Without proper modeling, the problem complexity will increase dramatically as the number of migration requests

and network scale increase. As a result, the complexity and processing time of current algorithms do not meet the real-time requirement of the live migration at scale in edge environments.

Moreover, to manage a highly distributed and dynamic network environment, Software-Defined Networking (SDN) [23] is introduced in edge computing, which allows dynamic configuration and operations of computer networks through centralized software controllers. Facilitated by OpenFlow protocol [24] and Open vSwitch (OVS) [25], network manager based on SDN controllers can perform network slicing or create a separate network [26] to minimize the influence of migration flows on other edge services. As a result, the migration planning and scheduling algorithm can fine-grained control the network resources for the migration competition, including the network routing and available bandwidth.

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

- We introduce the resource dependency graph of the source-destination pair for resource competition among migration requests to reduce the problem complexity.
- We model the problem as finding the Maximum Independent Set of the Dependency Graph iteratively.
- We propose iterative-Maximal Independent Set (MIS)-based algorithms for efficient large-scale migration scheduling and prove the corresponding Theros.
- We implement an event-driven simulator to evaluate the user mobility and live container migrations. The experiments are conducted with real-world dataset and traces.

The rest paper is organized as follows. We review the related work in Section 2 and define the system architecture in Section 3. In Section 4, we analyze and model the problem of multiple container migration scheduling. Then we propose two main methods of large-scale migration scheduling in Section 5. Section 6 shows the performance analysis of proposed algorithms and Section 7 shows the experimental design and evaluation with real-world dataset. Finally, we conclude the paper in Section 8.

2 RELATED WORK

The live VM migration realization [27] and its application in cloud data centers, such as dynamic resource management [28], [29], have been matured last few years. The research on live container migration in edge computing is an active field [5], [6]. Clark et al. [7] proposed the live VM migrations and discussed the details of pre-copy or iterative live migration. He et al. [30] evaluated the performance of live VM migrations and its overheads on the migration services in SDN-enabled cloud data centers. On the other hand, the research on live container migration is trending and becomes more

TABLE 1: Comparisons of multiple migration planning and scheduling works

research	real-time planning	large-scale	service correlations	user-mobility	deadline	SDN-enabled	Cloud DC	Edge computing
CQNCR [21]	–	–	✓	–	–	–	✓	–
FPTAS [22]	–	–	✓	–	–	–	✓	–
Our work	✓	✓	✓	✓	✓	✓	✓	✓

mature in recent years. Mirkin et al. [8] represented the checkpointing and restart features for the OpenVZ container. The checkpointing function is also used for live migration. Checkpoint/Restore In Userspace (CRIU) [9] is a Linux software to migrate container’s in-memory state in userspace. It is currently integrated with LXC, Docker (runC), and OpenVZ to achieve the live container migration. Nadgowda et al. also proposed [31] a CRIU-based memory migration together with the data federation capabilities of union mounts to minimize migration downtime. Similarly, Ma et al. [15], [16] utilized the layered storage feature based on AUFS storage drive and implemented a prototype system to improve the performance of docker container migration. Furthermore, several works studied the performance difference between container and VM live migration [2], [3]. Results show that the live container migration is much faster than the live VM migration due to its much smaller memory footprint and fast startup features.

More research recently focuses on dynamic resource scheduling in fog and edge computing environments based on the live container migration (details in survey [5], [6]). In [14], [17], the authors modeled the sequential decision making problem of generating service migration requests using the distance-based Markov Decision Process (MDP) framework. By reducing the state space, they proposed a distance-based MDP to get the approximated results. The research [18] also investigated the same problem by using the MDP framework. The authors proposed a reinforcement learning-based online microservice coordination algorithm to learn the optimal strategy for live migration requests to maintain the QoS in end-to-end delay.

Few studies focus on the optimization of the multiple live VM migration planning in cloud data centers [21], [22]. Bari et al. [21] investigated the multiple VM migration planning in one data center environment by considering the available bandwidth and the migration effects on network traffic reallocation. The authors proposed a heuristic migration grouping algorithm (CQNCR) by setting the group start time only based on the prediction model. Without an on-line scheduler, the estimated start time of a live migration can lead to an unacceptable migration performance and QoS degradations. Moreover, the work neglects the cost of the individual migration by only comparing the migration group cost. Without considering the connectivity between VMs and the change of bandwidth, Wang et al. [22] simplified the problem by maximizing the net transmission rate rather than minimizing the total migration time and proposed a polynomial-time approximation algorithm by omitting certain variables. However, the solution (FPTAS) can create migration competing on the same network path which degrades the migration performance in both average individual migration time and the total migration time.

However, as shown in Table 1, current algorithms can not meet the requirement of live container migrations in

edge computing. The framework of migration scheduling [21] which are periodically triggered by resource management policies with a long time interval is not suitable for the mobility-induced migration scenario. Furthermore, by modeling and calculating every resource competition of migration directly, the problem complexity [21], [22] increases along with the migration request number which is not suitable for large-scale situation. The running time of migration planning is also too large to schedule time-critical live container migrations. The algorithms [21], [22] do not consider the deadline or urgency (priority) of migration. In addition, without an on-line scheduler, the start time of a migration schedule is only based on the estimated migration time which can lead to migration performance and QoS degradation.

3 SYSTEM ARCHITECTURE

In the edge computing, there is no dedicated network for the live migration to support the user mobility compared with the traditional setups in cloud data centers [32]. By integrating the Software-Defined Networking (SDN) into edge computing, the centralized SDN controller can dynamically separate network resources from the service network [23] to build a virtual WAN network for live migrations. The available bandwidth and network routing are dynamically allocated based on the reserved bandwidth of the service network. This solution alleviates the overheads of live migration on other services and guarantees the performance of multiple live migrations. To achieve a fine-grained live migration scheduling, the migration scheduling service is integrated with the SDN controller [30], such as OpenDayLight (ODL), Open Network Operating System (ONOS) and Ryu, and container management and orchestration module, such as Kubernetes and Docker Swarm, to control both network and computing resources during each migration lifecycle.

3.1 Migration Lifecycle

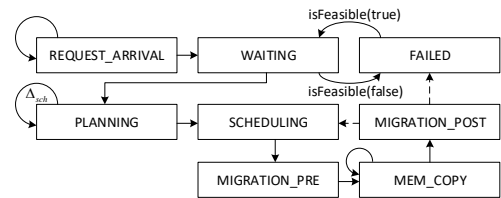


Fig. 2: Lifecycle for live container migrations

In this section, we introduce the framework of migration scheduling in edge computing. Compared with periodically arrived multiple live migrations in cloud data centers, the arrival of live container migration induced by user mobility

is stochastic. Therefore, we design the scheduling framework for the planning and scheduling of live container migration in edge computing with stochastic environments. As shown in Fig. 2, when a migration request arrives, it enters the WAITING state if it is feasible for scheduling, which means the container is not in migration. Otherwise, it will enter into the FAILED waiting migration list of the corresponding container. The migration planning event is triggered periodically within a short interval (such as every 1 second). It will generate the migration scheduling plan according to both waiting and running migrations. Based on the migration plan, the SDN-enabled on-line scheduler starts the migration with the allocated bandwidth and routing. Then, the live container migration will start the pre-migration phase to extract the container procedure tree. This will trace the dirty memory in the userspace of the source server and create an empty container instance in the destination for state synchronization [9]. In MEM_COPY, the dirty memory is transferred iteratively to the synchronizing instance in the destination. In the post-migration phase, the network communication of the migrated service will be redirected to the new instance in the destination. Then, the migrated container will recover at the destination. It will also trigger the start of subsequent resource-dependent migrations in the plan and change the feasibility flag of the first migration request of the same container in the FAILED migration waiting list.

4 MOTIVATIONS AND PROBLEM FORMULATION

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

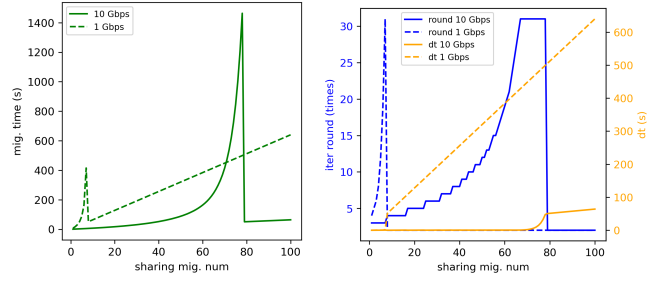
4.1 Single Migration Model

The performance of single live migration T_{mig} can be categorized into three parts: pre-migration computing, memory-copy networking, and post-migration computing overheads, i.e., $T_{mig} = T_{pre} + T_{mem} + T_{post}$. Due to the smaller footprint and fast start up of containers compared to VMs, the pre-migration and post-migration is much shorter [31]. Based on the iterative pre-copy container migration implemented in CRIU [9], the migration performance in terms of memory-copy can be represented as [30]:

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

where the ratio $\sigma = \rho \cdot R/L$, ρ is the data compression rate of dirty memory, Mem is amount of kernel memory the container uses, L is allocated available bandwidth, R is dirty page rate which is the memory difference in pagemap per second compared to the previous copy iteration, i is the total migration round.

We consider three conditions to enter the stop-and-copy phase: (1) reach the threshold of memory copy iteration; (2) the transmission time of remained memory difference is less than the downtime threshold; and (3) the allocated



(a) Average migration time (b) Iterations and downtime(dt)

Fig. 3: Migration performance against the number of migration sharing network bandwidth

bandwidth is less than the dirty page rate. The overhead of disk transmission for the container data and image can be ignored when shared network storage is available. The total iteration rounds of memory copy can be represented as:

$$i = \min \left(\left\lceil \log_{\sigma} \frac{V_{thd}}{Mem} \right\rceil, \Theta \right) \quad (2)$$

where Θ denotes the maximum allowed number of iteration rounds. The $\Theta = 0$ when the dirty page rate is larger than the allocated bandwidth at the start of migration. $V_{thd} = T_{dthd} \cdot L$ is the remaining dirty pages need to be transferred in the stop-and-copy phase, and T_{dthd} is the configured downtime threshold.

4.2 Resource Competition

We first explain the network sharing competition overheads in multiple migration scheduling. Two migrations may share the same source, destination, or part of network routings. Therefore, performing multiple live migrations in arbitrary order can lead to service degradation and unacceptable migration performance [21], [22]. A smaller bandwidth during the live migration means a longer migration time and more dirty pages need to transfer in order to limit the state difference between two instances for the last stop-and-copy phases which contributes as the downtime. Thus, the sum of the individual migration time of several live migrations is less than the total live migration time [30]. For example, based on the live migration model, Fig. 3a and 3b show the situation when several identical migrations sharing the same network path. The container's initial memory size is 1 GB with a 20 MB/s dirty page rate. The downtime and iteration threshold is configured at 0.5 seconds and 30 times, respectively. In this example, for the sake of a clear comparison between the sum of individual migration time and the total migration time, we start all migrations at the same time. In this case, the average migration time as shown also equals the total migration time.

The average execution time of live migrations scheduled sequentially with 10 Gbps and 1 Gbps is 0.8482 and 7.5241 seconds. The average downtime is 0.0082 and 0.1048 seconds. The iteration rounds are 3 and 4, respectively. However, the average migration time or total multiple migration time of 5 live migrations sharing 10 Gbps and 1 Gbps is 3.604 and 88.43 seconds. The average downtime is 0.2048 and 0.3689 seconds with 3 and 12 iterations, respectively. As

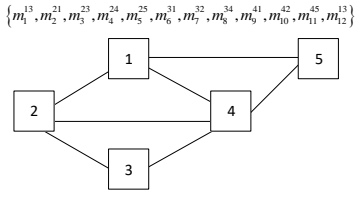


Fig. 4: Example live migration requests and the network topology with 5 edge data centers

the number of migrations increases (Fig. 3a), the allocated bandwidth decreases linearly. However, to achieve the required migration downtime, the average migration time will increase exponentially. At 7 and 80 migrations sharing of 1 Gbps and 10 Gbps respectively, the iteration rounds reach the threshold as 30 (Fig. 3b). Then, with more bandwidth-sharing migrations, the dirty page rate is larger than the allocated migration bandwidth. The downtime exceeds the 0.5 seconds threshold and increases significantly from 0.78 to 64.0 seconds and from 1.85 to 640 seconds. For the time-critical live migrations, a longer migration time will increase the possibility of migration deadline violation and QoS degradation. Therefore, it is optimal to sequentially schedule the resource-dependent migrations while concurrently schedule the independent ones. If there is a set of independent migrations and no other resource-dependent migrations are running, we can start all migration in such a concurrent scheduling group. The objective of migration scheduling is to maximize the number of migrations that can be scheduled concurrently.

Figure 4 shows an illustrative example with twelve live migration requests on the edge network topology of 5 total EDCs. Let m_i^{sd} denote the migration request that migrating container i from EDC s to EDC d . For the sake of a concise example, we limit the network interfaces used by the migration traffic. In other words, migration traffics share the same interfaces when the source or the destination is the same. It can be easily extended to the set of network interfaces in source $\{s\}$ and destination servers $\{d\}$ and the corresponding network paths $\{p\}$. The network routing policy considers the shortest network path with the minimal number of migration flows. For example, there are two network routes between EDC1 and EDC3. As there is one migration from EDC2 to EDC3, it chooses network path $\{EDC1, EDC4, EDC3\}$ in this case.

Resource-independent migrations from one concurrent scheduling group can be scheduled at the same time. The planning algorithm needs to generate a scheduling plan consists of several concurrent migration groups that each group size is as large as possible. A larger concurrent scheduling group indicates that there are more migrations could be performed at the same time. As a result, the better performance of multiple migrations in total migration time and the QoS of migrating service can be guaranteed. Note that two migrations from different migration groups are not necessarily resource-dependent. As shown in Fig. 5, based on the network topology provided by the SDN controller, we create an undirected graph of resource de-

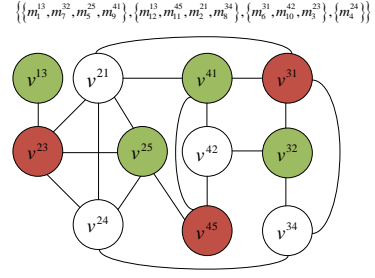


Fig. 5: The resource dependency graph of example migrations, iterative maximal independent set as concurrent migration groups, and two colored possible maximal independent sets for the first iteration, and one possible concurrent migration groups

pendency among migrations based on the source, destination, and network routing of migration requests. Each node v_p^{sd} represents a list of migrations sharing the same source s , destination d , and network path p . In other words, migrations in one node list form a complete graph as all migrations are resource-dependent to all others in the list. For example, the migration list of node v^{13} is $\{m_1^{13}, m_{12}^{13}\}$. It significantly limits the problem complexity as the number of migration requests increases. The edge of the dependency graph indicates resource competition (network interfaces at source or destination, or bandwidth sharing along network routes) between migrations. A concurrent group equals to an independent set of the resource dependency graph. A maximal concurrent scheduling group is a set of resource-independent migrations that is not a subnet of any other concurrent group. In other words, there is no other migration outside the concurrent group can be added to it so that all migrations can be performed at the same time. Therefore, it equals a maximal independent set (MIS). The largest size MIS is a maximum independent set. As shown in Fig. 5, there are several combinations of migrations for a maximal concurrent scheduling group. In the first iteration, one of the maximum group is $\{m_1^{13}, m_7^{32}, m_5^{25}, m_9^{41}\}$ and one of the maximal group is $\{m_3^{23}, m_{11}^{45}, m_6^{31}\}$. Thus, the maximum group is a better choice. After selecting the migrations from the nodes of the maximal independent set, we delete these migrations and update the dependency graph. One node is deleted from the graph when there is no migrations left in its migration list. For example, after the first iteration, we only delete nodes v^{25}, v^{41}, v^{32} , because there is still one migration m_{12}^{13} left in node v^{13} list. Thus, it is essential to select migrations carefully to achieve the maximum size of the concurrent groups. At the end, the on-line scheduler schedules all migration in the first group. Then, when there is one migration finishes, the scheduler starts all migrations blocked by the finished migration following the order of migration groups.

Before discussing how to get the Maximum Independent Set, the largest Maximal Independent Set (MIS), of the resource dependency graph, we first review some basic graph concepts [33], such as clique C and independent set I . A clique is a subset of vertices of an undirected graph G such that every two distinct vertices in the subset are adjacent.

The maximal clique is a clique that cannot be extended by including one more adjacent vertex. On the other hand, an independent set of a graph G is the opposite of a clique that no two nodes in the set are adjacent. The maximum clique or independent set is the maximal clique or independent set with the largest size. $\alpha(G)$ denotes the size of the largest MIS of graph G . Therefore, an independent set of the resource dependency graph equals a concurrent migration group. The migrations from the nodes in an independent set I can be scheduled concurrently. Meanwhile, migrations from the nodes in a clique are resource-dependent which need to be scheduled sequentially.

4.3 Real-Time Planning

There are few multiple migration planning and scheduling algorithms for live VM migration in cloud data centers [21], [22]. However, the processing time of the scheduling sequence of multiple live migrations based on the algorithms in cloud data centers is not suitable for the real-time requirement of mobility-induced migrations in edge computing. For example, the processing time of FPTAS [22] and CQNCr [21] for migration planning is about 5 and 10 seconds for 100 migrations. The processing time increases to 44.56 and 968.46 seconds for FPTAS and CQNCr to generate the scheduling plan of 500 migrations. For traditional dynamic resource management, the algorithm triggered every 10 minutes or 30 minutes. This leaves enough time budget for algorithms to generate the optimal scheduling sequence. However, in the edge computing environment for mobility-induced live migrations, the live migration requests arrive at any time stochastically. Most of the migration requests are also time-critical. Thus, the processing time of the planning and scheduling algorithm for mobility-induced migrations should be adapted to suit the real-time scenario.

4.4 Problem Modeling

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

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

At every planning and scheduling time t , the resource dependency graph $G = (V, E)$ denotes the acyclic undirected graph where $|G| = |V|$. Each node $u \in |V|$ represents

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

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

$$G_{i+1} = G_i [V(G_i) - I_i] = G_i [V(G_i) - S_i] \quad (3)$$

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

$$\max |I_i|, \forall I_i \in \{I_{iter}^i\} \quad (4)$$

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

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

$$W(\hat{m}) \geq W(m), \forall \hat{m}, m \in M(u) \quad (5)$$

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

$$\max W(I_i), \forall I_i \in \{I_{iter}^i\} \quad (6)$$

The weight of node $W(u) = 1$ when there is no need to differentiate migrations in different nodes. Generating the maximum (weighted) independent set of an undirected acyclic graph is a well known NP-hard problem [34], [35]. Therefore, generating the iterative maximum independent set as the subset is also NP-hard.

4.5 Complexity Analysis

Because an independent set of G is a clique in the complement graph of G and vice versa, the independent set problem and the clique problem are complementary [33], [34], [36]. In other words, listing all maximal independent sets or finding the maximum independent set of a graph equals listing all maximal cliques or finding the maximum clique of its complement graph. Thus, in each iteration, we can equivalently find the maximum independent set by getting the maximum clique C_i of the complement graph $C_i(\bar{G}_i) = I_i(G_i)$.

It is known that all maximal cliques can be calculated in a total time proportional to the maximum number of cliques in an n -vertex graph [36]. In other words, each clique is generated in a polynomial time in all maximal cliques listing [34]. When we only consider vertex, the maximal cliques listing algorithm (CLIQUES) [35], [36] based on Bron-Kerbosch [33] is the optimal algorithm. The worst-case running time of CLIQUES is $O(3^{n/3})$. The upper bound of all maximal cliques or independent sets of a graph is $3^{n/3}$ [37]. For the problem of finding one maximum independent set, the time complexity is improved from $O(2^{n/3})$ in [38] to $O(2^{0.276n})$ [39]. Based on the work [39], the best-known time complexity is $O(2^{n/4})$ [40]. Therefore, it is computationally impossible to solve the exact problem of listing all maximal cliques (maximum clique) of its complement graph \bar{G}_{dep} or all maximal independent sets (maximum independent set) of G_{dep} for the real-time live container migration scheduling in edge computing which exhibits an exponential time complexity.

5 MIGRATION PLANNING AND SCHEDULING

In this section, we present the proposed planning and scheduling algorithms for large-scale live container migrations in edge computing. With the waiting live container migration requests and planned unfinished live migrations as the input, the migration planner needs to efficiently schedule arriving migrations while maintaining the QoS. Based on the problem modeling in Section 4.4, this problem is reduced to finding an MIS of the migration dependency graph iteratively. Therefore, we propose two major approaches to generate the iterative MISs of the dependency graph: (1) Direct iterative MIS generation and (2) Maximum Cliques (MCs)-based MIS generation.

5.1 Direct iterative-Maximal Independent Sets

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

5.1.1 The Approximation

For the approximation algorithm (approx) of creating the iterative maximum independent set, the procedure is as follows: In the approximation algorithm (Algorithm 1), we use the approximating maximum independent sets algorithm by excluding subgraph [41] to generate MIS in each iteration. Note that we skip the MIS generation and remove the migrations directly if the node size of G_{dep} is unchanged in the current iteration. In other words, if we need to recalculate the MIS of the remaining graph, there is at least one node removed from the graph G_i . Given total m live container migrations, we create the corresponding dependency graph with n vertices. Therefore, regardless of the total number of migration requests, the upper bound of the complexity of planning multiple migrations scheduling

is limited by the involved source, destination, and network routing. In the worst case, the planning algorithm only needs at most n iteration rounds to calculate the concurrent migration group. In each iteration, it guarantees $O(n/(\log n)^2)$ approximate maximum independent set in polynomial time [41].

Algorithm 1: Iterative approximation grouping

Input: $\{G_{dep}\}$
Result: migGroups $\{I_{iter}\}$

- 1 $i \leftarrow 0; G_i \leftarrow G_{dep}; \{I_{iter}\} \leftarrow \emptyset;$
- 2 **while** $V(G_i) \neq \emptyset$ **do**
- 3 $I_i \leftarrow \text{APPROX_MIS}(G_i);$
- 4 $G_{i+1} \leftarrow G_i \llbracket V(G_i) - I_i \rrbracket;$
- 5 $\{I_{iter}\} \leftarrow \{I_{iter}\} \cup I_i;$
- 6 $i \leftarrow i + 1;$

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

5.1.2 Greedy MIS Algorithm

The greedy algorithm (iter-GWIN) generates the concurrent groups (MIS) of live migration iteratively. A greedy maximal independent set algorithm (GWIN) [42] based on the weight and the degree of a node is adapted to directly generate the MIS in each iteration. Let Δ_G denote the maximum degree and \bar{d}_G is the average degree of G . The degree of node u in G is $d_G(u) = |N_G(u)|$. $N_G(u)$ is the set of neighbor nodes of vertex u and $N_G^+(u) = N_G(u) \cup \{u\}$.

Algorithm 2: iter-GWIN

Input: $\{G_{dep}\}$
Result: migGroups $\{I_{iter}\}$

- 1 $i \leftarrow 0; G_i \leftarrow G_{dep}; \{I_{iter}\} \leftarrow \emptyset;$
- 2 **while** $V(G_i) \neq \emptyset$ **do**
- 3 $I_i \leftarrow \emptyset; G_j \leftarrow G_i; j \leftarrow 0;$
- 4 **while** $V(G_j) \neq \emptyset$ **do**
- 5 select node \hat{u} in G_j ;
- 6 $I_i \leftarrow I_i \cup \{\hat{u}\};$
- 7 $G_{j+1} \leftarrow G_j \llbracket V(G_j) - N_G^+(\hat{u}) \rrbracket;$
- 8 $j \leftarrow j + 1;$
- 9 $G_{i+1} \leftarrow G_i \llbracket V(G_i) - I_i \rrbracket;$
- 10 $\{I_{iter}\} \leftarrow \{I_{iter}\} \cup I_i;$
- 11 $i \leftarrow i + 1;$

As shown in Algorithm 2, from line 3-8, it selects the node with largest score regarding the minimal degree and maximal weight:

$$W(u)/(d_{G_i}(u) + 1) \quad (7)$$

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

As mentioned in the problem modeling, the weighted node equals the maximum weight of migrations from its list. The migration weight could be arrival time, estimated migration time, or correlation network influence [21] after migration for non-time-critical migrations and the deadline or slack time for real-time migrations scheduling. In this paper, we consider the weight function regarding the slack time τ as follows:

$$W(m) = \begin{cases} 10 \cdot \beta / \tau & \tau > \beta \\ 100 \cdot |\tau| / \beta & \tau < -\beta \\ 100 & \text{other} \end{cases} \quad (8)$$

where β is the slack time threshold. We set $\beta = 1$ in this paper. The weight of node is $W(u) = \gamma \cdot W(m)$, where γ is the coefficient regulator for the urgency of the scheduling migration. We set $\gamma = 1$. Moreover, in the situation that the priorities of all migrations are the same, we only need to consider the size of MIS. The node weight is set to 1 $W(u) = 1$. In each iteration, the lower-bound of the maximum (weighted) independent set is $\sum_{u \in V} W(u) / (d_G(u) + 1)$ [42]. As iteration is n in the worst case, the time complexity of iter-GWIN is $O(n^2 \log n)$ for weighted graph and $O(n^2)$ for unweighted graph.

5.2 The Maximum Cliques-based Heuristics

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

The degeneracy of a graph G is the smallest number d such that every subgraph of G contains a vertex of degree at most d . It is a measure for the graph sparseness. For an n -vertex graph with degeneracy d , by introducing the sequence ordering based on degeneracy, Bron-Kerbosch Degeneracy algorithm [43] can list all maximal cliques in time $O(dn3^{d/3})$. With a sparse graph that $n \geq d+3$, the upper bound of all maximal cliques number is $(n-d)3^{d/3}$. Figure 6 illustrates the nodes and the density (degeneracy) of the resource dependency graph of WAN network topologies [44] and its complement. It shows that the degeneracy of the complement graph \tilde{G}_{dep} is 4.34 times that of G_{dep} . For G_{dep} and its complement graph, the average ratio of dependency d to the total number of nodes n is 0.153 and 0.714, respectively. The resource dependency graph is considerably more sparse than its complement graph. Therefore, for G_{dep} , there are much fewer maximal cliques than the total MIS. As a result, according to the theoretical time complexity, the running time of listing all maximal cliques or maximum clique of G_{dep} is much smaller than that of listing all maximal independent sets or maximum independent set of \tilde{G}_{dep} . Therefore, the iterative Maximum Cliques (MCs)-based heuristics algorithm has two steps: (1) calculate the list of iterative maximum cliques and (2) generate the iterative maximal independent set based on the list. As nodes from one maximal clique can not be included into the same independent set, the iterative maximum cliques serve as a heuristic pruning decider to speed up the algorithm.

5.2.1 Iterative-rounds MCs algorithm

Let \hat{C}_i denote the maximum clique and $\{\bar{C}_i\}$ denote the maximal cliques list of round i graph. The iterative-rounds

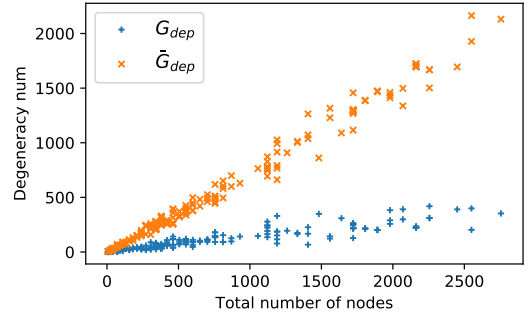


Fig. 6: Total number of nodes and degeneracy number of resource dependency graph of WAN topologies [44] and its complement graph

Algorithm 3: Iterative heuristic of migration grouping

Input: $\{G_{dep}\}$
Result: migGroups $\{I_{iter}\}$

- 1 $\{C_{iter}\} \leftarrow \emptyset; \{I_{iter}\} \leftarrow \emptyset;$
- 2 **while** $|G_{dep}| \neq 0$ **do**
- 3 {Iterative creating Maximum Cliques}
- 4 $\hat{C}_i \leftarrow \text{MAXIMUM_CLIQUE}(G_{dep});$
- 5 $G_{dep} \leftarrow G_{dep} \setminus [V(G_{dep}) - \hat{C}_i];$
- 6 $\{C_{iter}\} \leftarrow \{C_{iter}\} \cup \hat{C}_i;$
- 7 **while** $\{C_{iter}\} \neq \emptyset$
- 8 $I \leftarrow \emptyset$
- 9 **foreach** \hat{C}_i **in** $\{C_{iter}\}$ **do**
- 10 $m \leftarrow \text{ADDINDEP}(I, \hat{C}_i);$
- 11 $\text{DELNODE}(\hat{C}_i, m);$
- 12 $\{I_{iter}\} \leftarrow \{I_{iter}\} \cup I;$
- 13 **return** $\{I_{iter}\}$

Maximum Cliques (MCs)-based heuristic algorithm (Algorithm 3) follows two steps: (1) generating the maximum clique iteratively and (2) obtaining the MIS from the iterative maximum cliques.

As shown in Algorithm 3, we first create dependency graph G_{dep} as the input based on the source-destination of the given migrations and the network topology. From line 1-6, the algorithm calculates the iterative maximum cliques of the dependency graph until there is no vertices left. In each iteration, it generates the maximum clique (Bron-Kerbosch Degeneracy algorithm) [43] of the remaining graph. It is proved that the algorithm is highly efficient in a sparse graph, such as the resource dependency graph [43]. Then, it updates the remaining graph by deleting the nodes of the maximum clique from G_{dep} . Let $d_{G[C]}(u) = |N_{G[C]}(u)|$ denote the degree of node u to the remaining graph which excludes all nodes in the clique. The node score can be represented as:

$$W(u) / (d_{G[C]}(u) + 1) \quad (9)$$

In the second step (line 7-12), it generates maximal independent sets based on the iterative maximum cliques. In each round (line 9-11), it selects the feasible node with

maximum score of each maximum clique \hat{C}_i and adds largest-weight migration from its list into the independent set. A node is feasible when it can be included in the current independent set. If there is no migrations left in the migration list of the selected node $M(u)$, the selected node is removed from the clique. As the largest possible number of maximal cliques in an n -vertex graph with degeneracy d is $(n-d)3^{d/3}$. Therefore, according to the iter-MCs algorithm, the upper bound of the size of the iterative maximum independent set of each iteration is also $(n-d)3^{d/3}$. In the worst case, the time complexity of iter-MCs is $O(dn^23^{d/3})$.

Theorem 1 (Correctness of MIS from Maximal Cliques). *The Independent Sets generated from maximal cliques are the maximal independent sets of the graph.*

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

5.2.2 Single-Round MCs Algorithm

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

Algorithm 4: Single-round iterative maximum cliques

Input: $\{G_{dep}\}$
Result: migGroups $\{C_{iter}\}$
 SINGLE-ITER($\{G_{dep}\}$):
 1 $\{C_{iter}\} \leftarrow \emptyset;$
 2 $\{\bar{C}\} \leftarrow \text{FINDCLIQUES}(G_{dep});$
 3 **while** $\{\bar{C}\} \neq \emptyset$ **do**
 4 $\hat{C}_i \leftarrow \text{MAX}(\{\bar{C}\});$
 5 $\{C_{iter}\} \leftarrow \{C_{iter}\} \cup \hat{C}_i;$
 6 $\{\bar{C}\} \leftarrow \text{DELNODES}(\hat{C}_i, \{\bar{C}\});$
 7 **return** $\{C_{iter}\}$

The first step of the iter-MCs algorithm is replaced by Algorithm 4. The algorithm only generates the list of all maximal cliques $\{\bar{C}\}$ once by using the Bron-Kerbosch Degeneracy algorithm. Until there is no vertices left in the clique list, it selects the maximum clique (largest maximal clique) \hat{C}_i from the list and deletes the nodes of the selected maximum clique from all maximal cliques.

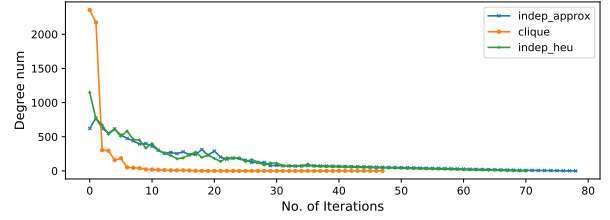


Fig. 7: Degree of iteration cliques/independent set to the remaining nodes

Theorem 2 (Correctness of the algorithm single-MCs). *Given a graph $G = (V, E) V \neq \emptyset$, the single iteration algorithm SINGLE-MCs generates all and only iteration maximum cliques.*

Proof. It is proven that the Bron-Kerbosch Degeneracy algorithm generates all and only maximal cliques without duplications [43]. Then, we only need to prove the results of iterative maximum cliques are the same in iter-MCs and single-MCs, i.e., one can get all the iterative maximum cliques based on the maximal cliques of the original graph by deleting the vertices from the maximum clique in the last round.

Let $C(G) = \{C_0, C_1, \dots, C_d\}$ denote all maximal cliques of the original graph G , where $|C_i| \geq |C_{i+1}|, \forall C_i, C_j \in C(G)$, that $C_i \neq C_j, C_i \not\subset C_j$. The next iteration graph is $G \setminus C_0 = G[V(G) - C_0]$. Then, $C(G \setminus C_0) = \{C_1, C'_2, \dots, C'_e\}$. The output of first round of single-MCs is $C(G) \setminus C_0 = \{C''_1, C''_2, \dots, C''_e\}$.

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

6 GRAPH ALGORITHM PERFORMANCE AND ANALYSIS

In this section, we evaluate proposed migration planning algorithms for the problem of iterative MIS generation: (1) iter-MCs (2) single-MCs; (3) approximation; and (4) iter-GWIN, in processing time, maximal independent set size, and iteration rounds. Based on more than two hundred real network WAN topologies [44], we consider a set of live migration requests with each source and destination combination. Each migration request corresponds to one combination with the network routing of the shortest path. We run the computational experiments in Python 3.6.3 and NetworkX package [45] version 2.4 as the graph library with source code.

The iterative maximum clique generation of migration dependency graph is faster than that of iterative MIS in three aspects: (1) As the analysis of dependency graph in section 5.2, dependency graph G_{dep} is more sparse than its complement \bar{G}_{dep} . Therefore, getting the maximum clique of G_{dep} in one iteration is faster than that of \bar{G}_{dep} ; (2) The

TABLE 2: Performance comparison with first, second, and third quartile of processing time, total MIS number (iteration), maximum, mean, 95th, and 99th quartile of the independent set size in each result of the total 202 WAN topologies

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

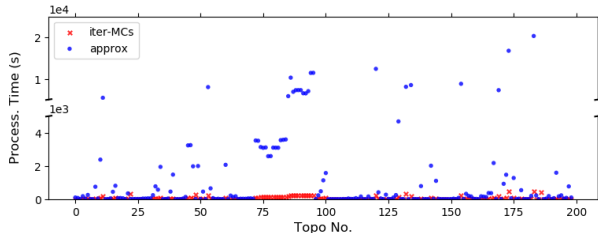


Fig. 8: Processing time comparison between iter-MCs and approximation

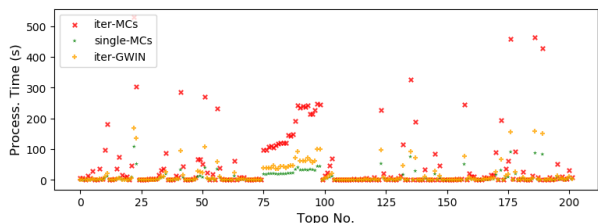


Fig. 9: Processing time comparison between iter-GWIN, iter-MCs, and single-MCs

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

Figure 7 demonstrates an illustrative result of one of the network topology (Australia’s Academic and Research Network, AARNet). The dependency graph consists of a total of 342 nodes and 11754 edges. It shows the degree of the maximum clique and independent set in each iteration to the remaining nodes. In other words, it is the edges of the removed nodes in each iteration excluding the edges between nodes from the maximum clique. Note that there is no edges (degree is zero) between nodes in one independent set. With the degree in the maximum clique and the degree to the remaining graph, the complexity of G_{dep} is dropped dramatically in the first three iterations. On contrary, by removing the maximum independent set, the complexity of the graph remains at a high level and declines steadily. Furthermore, the number of total iterative maximum cliques and iterative MISs is 47 and 70, respectively. Comparing the result of the approximation (indep_approx) with iter-MCs (indep_heu), the heuristic iterative MCs-based algorithm achieves better performance in the size of the maximum clique in each iteration and the total iteration rounds.

Since the processing time varies greatly, we use two separated figures to represent the results of processing time. Figure 8 shows the performance comparison between the

approximation (approx) and iter-MCs in processing time. Figure 9 shows the comparison between iter-MCs, single-MCs and iter-GWIN. The results of computational experiments indicate that the approximation algorithm has the worst performance in processing time. From approx to iter-MCs (Fig. 8), the average processing time of all topologies decreases by 91.32%. From iter-MCs to iter-GWIN and iter-GWIN to single-MCs, the average processing time decreases by 57.40% and 20.84%. Table 2 also illustrates the third (Q3), second (mean), and first quartile (Q1) of the average processing time. For the dependency graph with every source and destination combinations of a relative small size network, the single-MCs and iter-GWIN can both generate the scheduling plan in around 0.15 seconds. However, the performance difference in processing time increases with the size of the network topology. For mean and the Q3 of all processing time results, the average processing time of single-MCs decreased by 32.64% and 38.29% from iter-GWIN, respectively. In summary, the single-MCs algorithm has the best performance in processing time.

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

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

7 SIMULATION AND PERFORMANCE EVALUATION

In this section, we evaluate proposed solutions using real-world traces on an event driven simulator. We first describe

the real-world telecom base station dataset and taxi GPS traces used in the experiments. We explain the placement of edge data centers and the network topology and region coverage of each EDC. The event-driven simulator for software-defined network-enabled edge-cloud computing CloudSimSDN [46] is extended to emulate the user movement and the live container migration in edge computing. It provides a network operating system based on the software-defined networking for dynamic service and network resource monitoring and allocation. Compared to the simulation results driven by mathematical models, this can generate more realistic results without following the strong assumption encoded in the proposed mathematical modeling.

We compare and evaluate the performance of live container migration planning and scheduling algorithm (iter-GWIN and single-MCs) against a policy with no planning scheduling and the state-of-art live VM migration cloud algorithm FPTAS [22] in processing time, migration time, downtime, transferred data, deadline violations, and network transmission time.

7.1 Experimental Data

In this section, we describe the base stations coordinates provided by Shanghai Telecom dataset¹ and Shanghai Qiangsheng taxi GPS trace dataset (April 1, 2018)² used in our experiments. The given data is preprocessed by limiting the range of the longitude and latitude from 30.40° N to 31.35° N and 120.51° E to 122.12° E as there are some taxis travel to nearby cities. The Shanghai Telecom dataset contains the longitude and latitude coordinates of a total of 3233 base stations as shown in Fig. 10a. We use K-means algorithms [47] to generate the location of a total of 200 Edge Data Centers (EDCs) based on the longitude and latitude of the given base stations. Figure 10b illustrate the taxi GPS trace in the first hour. Besides the GPS coordinates and timestamp, each data record of the taxi dataset also includes the taxi id, service status, such as alarm, occupation, taxi light, road type, and breaking, as well as vehicle speed, direction, and the number of connected satellites.

Figure 10c illustrates base stations are clustered and connected to one of the regional Edge Data Centers. There is no information on the physical network topology and connectivity between EDCs. As shown in Fig. 10c, for the geometric spanner, we choose Delaunay Triangulation [48] to generate links between the gateway of each EDC. For the network routing within the generated network topology, we consider the shortest path, which is no longer than $4\pi/3\sqrt{3}$ times the Euclidean distance between source and destination. As a result, the boundary of EDC regions (Fig. 10d) is a Voronoi diagram [48] where the Euclidean distance of any point to its corresponding EDC region is less than or equal to its distance to any other EDC.

Similar to other research regarding the generation of mobility-induced live migrations in edge computing [17], [18], we combine these two datasets to simulate the scenarios where the user needs to connect to the services and maintains the low end-to-end latency through live

container migration in edge computing environments. Figure 11 demonstrates an example that the request of live container migration is induced when a taxi moves across the boundary between two clusters of EDCs. The deadline of each live container migration is generated based on the average mobility speed of users in the last 3 GPS records, travel direction, and the signal strength of base stations.

7.2 Experimental Setup

TABLE 3: Multiple container migration scenarios

Scenario	S1	S2	S3
vehicles	1000	2000	4000
migrations	9933	19522	37822

In this section, we describe details of the experiment setup. The end-to-end delay between the user and the service is the time interval from the user (taxi) sent workload to the container assigned in the EDC to the result is received by the user. To generalize computer vision use case workloads, the service task generated during the experiments follows the Poisson distribution with a mean of 24 per second (24 FPS). In each task, the network packet size sent from a user is 16384 bytes ($128 * 128$ bytes). The processing workloads in the container are randomly generated from 500 to 1000 cycles per bit [18]. The result packet sent back from the container to the user is 128 bytes. The sum CPU power frequency for each EDC with multiple CPUs is 25 GHz [49], [50]. To simulate the limited network resources for migrations in the edge, we consider that the reserved network bandwidth between a container and its user is 3 Mbps. The physical network bandwidth is 1 Gbps. The network delay between any based station to its regional EDC is 5 ms and the delay between two EDCs is randomly generated in the range 5 to 50 ms [18].

According to the evaluation results of container memory and dirty memory size during live container migrations [16], we generate the container memory from 100 MB to 400 MB. The dirty page rate for each dirty memory transmission is from 2MB/s to 8MB/s and the data compression rate is 0.8 [51]. We configure the downtime threshold and the maximum iterations for live container migration at 0.5 seconds and 30 times [30], respectively. Based on the SDN controller, the remaining network bandwidth between the source and destination EDC which is not utilized by services is allocated to the live container migration traffic. If several live migrations are sharing part of their routings, the bandwidth will be allocated evenly to each of the network flows.

From the experimental scenario S1 to S3 (Table 3), 1000, 2000 and 4000 vehicles are selected randomly. We consider the GPS trace of selected vehicles within 1 hour. For the initial placement at the start of the experiment, we allocate corresponding containers for each vehicle at the same edge data center according to its GPS coordinates. The nearest edge data center first policy is considered for our experiment to generate the live container migration requests. According to the user mobility, one live container migration will be triggered when one vehicle exits the coverage area of its current edge data center. There are 9933, 19522, and 37822 migration requests induced by these vehicles' movement,

1. <http://sguangwang.com/TelecomDataset.html>

2. <http://soda.shdataic.org.cn/download/31>

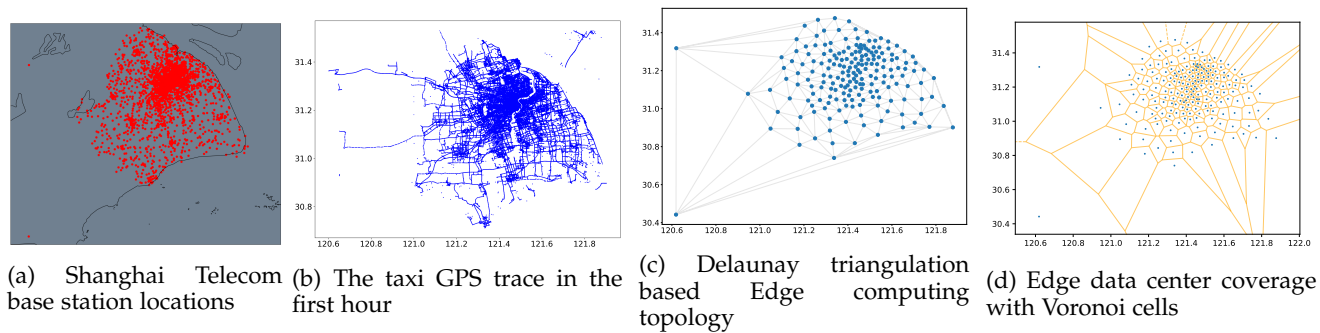


Fig. 10: Experimental dataset and configurations of longitude and latitude

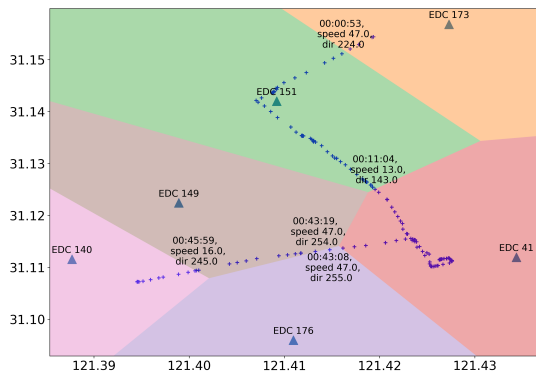


Fig. 11: Example of live migration request triggered by user movement of longitude and latitude

TABLE 4: Total processing time comparison in milliseconds

algorithm	S1	S2	S3
iter-GWIN	306.9607	762.3356	3997.8309
single-MCs	332.5682	583.3951	1544.6291
FPTAS [22]	903597.39	1923036.81	4677990.57

respectively. During the live container migration, the dirty memory of migrating container will be copied iteratively from the source edge data center to the nearest edge data center through the shortest network path. For the evaluation sensitivity, the results of each scenario are an average of 10 individual experiments. In this experiment, we consider that the container image as a universal service is already available in all edge data centers or shared by the network storage between EDCs. CloudSimSDN-NFV [46], an event-driven simulator, is extended with corresponding components to support the live container migration and user mobility in edge computing environments.

7.3 Experimental Results and Analysis

In Section 6, we compare the algorithm performance in terms of the size of iterative MISs and processing time. Thus, we only evaluate the two best algorithms in this section. We compare the experimental results between no migration scheduler, iter-GWIN, single-MCs, and the current state-of-the-art migration planning and scheduling in clouds FPTAS [22]. In FPTAS, to maximize the total bandwidth utilized by migrations, one migration can be started even there is considerably limited bandwidth which is much lower than

the dirty page rate per second. The solution can cause devastating migration performance. Thus, we improve FPTAS by adding a bandwidth threshold (FPTAS-BW) that the available bandwidth must larger than the dirty page rate as the migration start condition. As the vehicle number increases from scenario 1 (S1) to scenario 3 (S3), the density of live migration requests in certain areas increases dramatically. The resource competition or resource dependency among live container migration requests will also increase. As a result, the complexity of the dependency graph may also increase. When the requirements of live container migration requests exceed the resource capacity provided by the edge computing, it is inevitable that some of the deadlines of some migration requests can not be satisfied.

Table 4 shows the total processing time of migration planning and scheduling algorithms within 1 hour in milliseconds. From S1 to S3, the average processing time of single-MCs for each migration planning is 0.1175, 0.1936, and 0.4904 milliseconds. Compared to iter-GWIN, the processing time of single-MCs decreased by 61.36% in scenario S3. The results are consistent with the algorithm evaluation in Section 6. Furthermore, compared to FPTAS [22], the performance of our solution in terms of processing time has been improved by more than 3000 times. In S3, the processing time of FPTAS is about 78 minutes. As a result, even with any weight modification in the algorithm, the migration deadline in seconds will be missed. Therefore, as the results of FPTAS-BW in deadline violation are off the limit of chart comparison, we only compare it in the migration performance.

From S1 to S3, without migration planning and scheduling, more live migrations compete with each other on the network routing and the available bandwidth. As a result, the average migration time increases dramatically from 2.25 and 4.59 seconds to 299.89 seconds (Fig. 12a). Particularly, in S3, the allocated bandwidth may either be smaller than the dirty page rate and cause a large downtime for some migrations. Or, it causes a much longer migration time due to a large number of memory-copying iterations. As a result, the migrating service suffers a devastating consequence. Furthermore, for FPTAS-BW, by maximizing the total migration bandwidth rather than the resource competitions, it suffers smaller average bandwidth per migration. Thus, as shown in Fig. 12 the performance of our purposed solution in terms of average migration time, average downtime, and total transferred data are increased by up to 30.24%, 51.56%, and

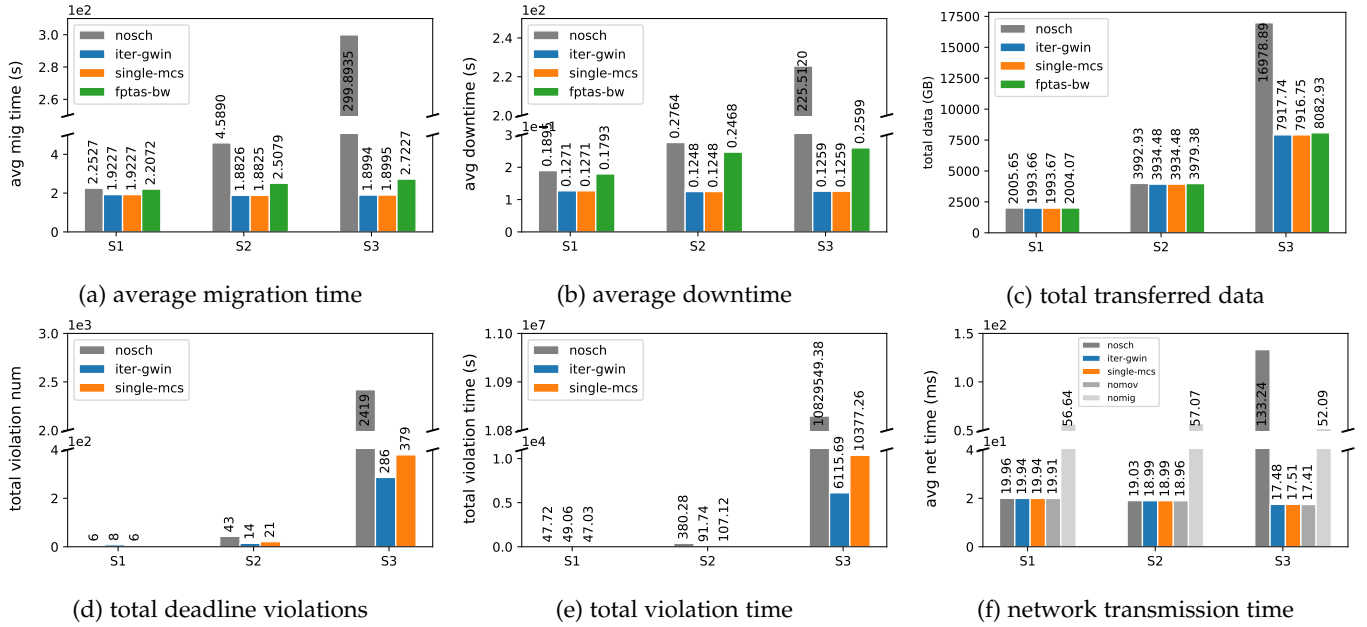


Fig. 12: Migration performance comparison with no scheduler, iter-GWIN, and single-MCs under different scenarios.

2.06%, respectively. Meanwhile, for the proposed planning and scheduling algorithms iter-GWIN and single-MCs, the performance of live migration can be guaranteed even with severe resource competitions. Results (Fig. 12a, 12b) show that the average migration time and downtime are optimal at 1.9 sand 0.13 seconds as there is no bandwidth sharing between resource-dependent migrations. Furthermore, for all the migrations that arrive within the 3600 seconds time interval in S3, iter-GWIN and single-MCs can finish the scheduling of all migrations in 3603.91 and 3601.43 seconds. However, the total migration time of no scheduler is 3603.43 seconds in S2 and 48878.65 seconds in S3. A shorter average migration means less possibility of QoS degradation and less occupation time on the network resource. A smaller downtime equals fewer disruptions on the migrating services.

Another critical migration performance is the transferred data of the live migration. It is also highly related to network energy efficiency. In S1 and S2, although average migration time and downtime increase due to less allocated bandwidth, there is no surge in the transferred data for the no migration scheduling situation (Fig. 12c). Because of the container's small memory footprint, the shared bandwidth can still satisfy the downtime threshold with relatively small memory-copying iterations. However, when the bandwidth becomes the bottleneck, a large number of memory-copying iteration needed to meet the downtime threshold. Therefore, the total transferred data in S3 increase by 114.47% compared with the optimal result from single-MCs.

The deadline of a live migration request is highly related to the QoS and SLA requirement of the real-time migrating service. For iter-GWIN and single-MCs, the ratio of migration violation numbers to the total migration number is 0.071% and 0.107% in S2 and 0.756% and 1.002% in S3 (Fig. 12d). However, the ratio for no migration scheduler is 3.07 times in S2 and 8.46 times compared to the best result from iter-GWIN. The ratio of total violation time to

the service time of all containers in one hour is 0.00127% and 0.00148% in S2 and 0.0425% and 0.0720% in S3, respectively (Fig. 12e). In S3, although migration performance in terms of migration time and downtime is optimized by the migration scheduler, the network resource is insufficient to schedule all 37822 migration requests on time with the live migration competitions. It is inevitable to violate the deadline of certain migrations with lower priority to satisfy the deadline for others. As a solution, one needs to increase the network resource by providing duplicate EDCs and additional network routing or available bandwidth in the hot spot to alleviate the deadline violation of real-time migrations.

The end-to-end delay for the migrating edge service is affected by the migration downtime and the duration of deadline violation. For the network transmission time, we compare the results of no user movement and no migration, no migration requests with user movement, no scheduler, iter-GWIN and single-MCs (Fig. 12f). In the scenario that all vehicles stay at the s and do not move during the experiment time (nomov), the average network transmission time to the service or the end-user is from 17.4 to 19.9 milliseconds from S3 to S1. Without the live migration requests (nomig), the end-to-end delay can be not guaranteed due to the network delay between the EDC and the end-user. Specifically, the average network transmission time is around 56 milliseconds. The live migration planning and scheduling algorithm (iter-GWIN and single-MCs) can guarantee the average service network transmission time. In S3, without a migration scheduler, the downtime and deadline violation have a considerable impact on the service network delay. The network delay increases by 6.62 times compared to the result of iter-GWIN.

In summary, our proposed algorithms can efficiently plan and schedule large-scale mobility-induced live container migrations in edge computing. Even in the case of a migration request surge, it guarantees the performance of

live container migrations and maintains the QoS of migrating services. It significantly reduces average migration time (up to 99.36%), average down time (up to 99.94%), total deadline violations (up to 88.18%) and violation time (up to 99.94%).

8 CONCLUSIONS AND FUTURE WORK

In this paper, we investigated the challenges of live container migration scheduling in edge computing environments including (1) resource competition or dependency among live migrations and (2) real-time migration planning and scheduling. We modeled the relationship of resource dependency among migrations as an undirected graph and the scheduling problem as generating the maximum independent set of the dependency graph iteratively. We proposed a framework for user-triggered or mobility-induced migration scheduling which is different from the traditional scheduling for live VM migrations in cloud data centers. The SDN is introduced to separate the computer network to minimize the impact of migration flows on other edge services. Based on the dynamic computing resources, network resources and topology provided by the container/VM orchestration engine and SDN controllers, the migration management service can plan and schedule multiple migration requests in a fine-grained manner. We proposed two methods for large-scale migration planning and scheduling algorithms based on iterative Maximal Independent Sets. Computational experiments were conducted to evaluate the algorithms' performance. Furthermore, the results of experiments based on real-world data indicate that proposed algorithms can efficiently plan and schedule large-scale mobility-induced live container migrations in a complex network environment in a timely manner, while maintaining the QoS of migrating services. It can optimize the live migration performance and minimize the deadline violation in migration scheduling. As part of the future work, we intend to investigate base station clustering for EDC placement based on user mobility information to reduce the number of live migrations.

ACKNOWLEDGMENT

The authors thank Shashikant Ilager and Mohammad Goudarzi for their valuable comments and suggestions.

REFERENCES

- [1] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
- [2] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 140–147, 2017.
- [3] T. V. Doan, G. T. Nguyen, H. Salah, S. Pandi, M. Jarschel, R. Pries, and F. H. Fitzek, "Containers vs virtual machines: Choosing the right virtualization technology for mobile edge cloud," in *2019 IEEE 2nd 5G World Forum (5GWF)*. IEEE, 2019, pp. 46–52.
- [4] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.
- [5] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23 511–23 528, 2018.
- [6] Z. Rejiba, X. Masip-Bruin, and E. Marín-Tordera, "A survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–33, 2019.
- [7] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation—Volume 2*. USENIX Association, 2005, pp. 273–286.
- [8] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, "Containers checkpointing and live migration," in *Proceedings of the Linux Symposium*, vol. 2, 2008, pp. 85–90.
- [9] CRIO, "Live migration," 2019. [Online]. Available: https://criu.org/Live_migration
- [10] "Container migration with Podman on RHEL," accessed 22 Jan 2020. [Online]. Available: <https://www.redhat.com/en/blog/container-migration-podman-rhel>
- [11] V. Marmol and A. Tucker, "Task migration at scale using criu," in *Linux Plumbers Conference*, 2018.
- [12] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*, 2015, pp. 1–17.
- [13] A. Ruprecht, D. Jones, D. Shiraev, G. Harmon, M. Spivak, M. Krebs, M. Baker-Harvey, and T. Sanderson, "Vm live migration at scale," *ACM SIGPLAN Notices*, vol. 53, no. 3, pp. 45–56, 2018.
- [14] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *2015 IFIP Networking Conference (IFIP Networking)*. IEEE, 2015, pp. 1–9.
- [15] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.
- [16] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2020–2033, 2018.
- [17] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1272–1288, 2019.
- [18] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. S. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Transactions on Mobile Computing*, 2019.
- [19] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.
- [20] T. He, A. N. Toosi, and R. Buyya, "Sla-aware multiple migration planning and scheduling in sdn-nfv-enabled clouds," *Journal of Systems and Software*, vol. 176, p. 110943, 2021.
- [21] M. F. Bari, M. F. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "Cqncr: Optimal vm migration planning in cloud data centers," in *Proceedings of the Networking Conference, 2014 IFIP*. IEEE, 2014, pp. 1–9.
- [22] H. Wang, Y. Li, Y. Zhang, and D. Jin, "Virtual machine migration planning in software-defined networks," *IEEE Transactions on Cloud Computing*, 2017.
- [23] 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.
- [24] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
- [25] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 117–130.
- [26] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, 2017.
- [27] V. Medina and J. M. García, "A survey of migration mechanisms of virtual machines," *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, pp. 1–33, 2014.

- [28] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo, "Dynamic resource management using virtual machine migrations," *IEEE Communications Magazine*, vol. 50, no. 9, pp. 34–40, 2012.
- [29] M. Xu, W. Tian, and R. Buyya, "A survey on load balancing algorithms for virtual machines placement in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12, p. e4123, 2017.
- [30] T. He, A. N. Toosi, and R. Buyya, "Performance evaluation of live virtual machine migration in sdn-enabled cloud data centers," *Journal of Parallel and Distributed Computing*, vol. 131, pp. 55–68, 2019.
- [31] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete container state migration," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2137–2142.
- [32] K. Tsakalozos, V. Verroios, M. Roussopoulos, and A. Delis, "Live vm migration under time-constraints in share-nothing iaas-clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2285–2298, 2017.
- [33] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.
- [34] E. L. Lawler, J. K. Lenstra, and A. Rinnooy Kan, "Generating all maximal independent sets: Np-hardness and polynomial-time algorithms," *SIAM Journal on Computing*, vol. 9, no. 3, pp. 558–565, 1980.
- [35] F. Cazals and C. Karande, "A note on the problem of reporting maximal cliques," *Theoretical Computer Science*, vol. 407, no. 1-3, pp. 564–568, 2008.
- [36] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical computer science*, vol. 363, no. 1, pp. 28–42, 2006.
- [37] J. W. Moon and L. Moser, "On cliques in graphs," *Israel journal of Mathematics*, vol. 3, no. 1, pp. 23–28, 1965.
- [38] R. E. Tarjan and A. E. Trojanowski, "Finding a maximum independent set," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 537–546, 1977.
- [39] J. M. Robson, "Algorithms for maximum independent sets," *Journal of Algorithms*, vol. 7, no. 3, pp. 425–440, 1986.
- [40] —, "Finding a maximum independent set in time $o(2n/4)$," Technical Report 1251-01, LaBRI, Université Bordeaux I, Tech. Rep., 2001.
- [41] R. Boppana and M. M. Halldórsson, "Approximating maximum independent sets by excluding subgraphs," *BIT Numerical Mathematics*, vol. 32, no. 2, pp. 180–196, 1992.
- [42] S. Sakai, M. Togasaki, and K. Yamazaki, "A note on greedy algorithms for the maximum weighted independent set problem," *Discrete applied mathematics*, vol. 126, no. 2-3, pp. 313–322, 2003.
- [43] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," in *International Symposium on Algorithms and Computation*. Springer, 2010, pp. 403–414.
- [44] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.
- [45] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.
- [46] J. Son, T. He, and R. Buyya, "Cloudsim-sdn-nfv: Modeling and simulation of network function virtualization and service function chaining in edge computing environments," *Software: Practice and Experience*, vol. 49, no. 12, pp. 1748–1764, 2019.
- [47] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2866–2880, 2015.
- [48] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, "Scipy 1.0: fundamental algorithms for scientific computing in python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.
- [49] Y. Sun, S. Zhou, and J. Xu, "Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2637–2646, 2017.
- [50] Y. Xiao and M. Krunz, "Qoe and power efficiency tradeoff for fog computing networks with fog node cooperation," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.
- [51] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," *ACM Sigplan Notices*, vol. 46, no. 7, pp. 111–120, 2011.