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

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

***Abstract*—Services provided by mobile edge clouds offer low-latency responses for large-scale and real-time applications. Dynamic service management algorithms generate live service migration requests to support user mobility and ensure service latency in mobile edge clouds. To handle these migration requests, multiple migration planning and scheduling algorithms are necessary to calculate the migration order and optimize the performance and overhead of multiple migrations. However, current planning and scheduling algorithms in cloud data centers are not suitable for dynamic and large-scale scenarios in edge computing, as the network topology expands and the number of migration requests increases. Edge computing requires near real-time scheduling to handle user mobility-induced live migrations. To address this issue, this paper presents an efficient multiple migration planning and scheduling framework for edge computing. The framework includes a lifecycle management framework and innovative iterative Maximal Independent Set-based scheduling algorithms based on the resource dependency graph of multiple migrations. Our solution is shown to efficiently schedule live migrations at scale using real-world taxi traces and telecom base station coordinates. It can achieve significant processing speedups over existing migration planning algorithms in clouds, up to 3,000 times, while ensuring multiple and individual migration performance for time-critical services.**

***Index Terms*—Mobile-edge computing, quality of service, service migration, multiple migration scheduling.**

I. Introduction

EDGE computing [1] offers the opportunity to enhance the performance of user-oriented applications, such as Vehicle to Cloud (V2C), Vehicle to Vehicle (V2V), Virtual Reality (VR), Augmented Reality (AR), Artificial Intelligence (AI), and Internet of Things (IoT), among others, by bringing computation and intelligence closer to end-users. Due to their smaller memory footprint and quicker startup, microservices driven by container virtualization are becoming more suited for dynamic deployment on edge computing [2], [3]. By hosting containerized services in Edge Data Centers (EDCs) or Mobile Edge Clouds (MECs) [4], end-to-end (E2E) communication delays between end-users and services can be reliably guaranteed.

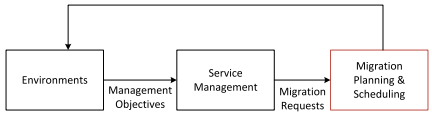


Fig. 1. General migration management framework.

Edge infrastructure and service providers, however, are faced with new challenges in dynamic resource management and user mobility [5]. Live migration is a solution to these challenges by offering non-application-specific management of compute and memory state. Live Virtual Machine (VM) migration [6] and Checkpoint/Restore in Userspace (CRIU)-based container migration [7], [8] aim to minimize disruption (downtime) to the running service during migration in edge computing. Companies such as IBM, RedHat, and Google have integrated live VM and container migration into their production systems [9], [10]. For example, Google has adopted live VM and container migration for both stateful and stateless services in its Borg cluster manager [10], [11] for tasks of higher priority, software updates (kernel and firmware), and reallocation for availability and performance. With a median downtime of only 50 ms, a minimum of one million migrations are performed monthly in their production fleet [11].

The procedure for managing services using live migration technologies is depicted in Fig. 1. Service management algorithms generate multiple migration requests based on various management objectives [5], [12], [13], [14], [15], [16]. Traffic flow of migration requests can be optimized through online SDN-enabled routing algorithms [17], [18] or proactively calculated during the generation of migrations in the service management process [19]. While live migration costs are taken into account in service management algorithms, performing multiple migration scheduling in a disorganized manner can result in service degradation [20], [21], [22], [23], [24]. This is particularly true in the edge computing environment, where network and computing resources are limited. In this paper, we focus on the challenging problem of multiple migration planning and scheduling in SDN-enabled Edge Computing. The migration requests generated by the service management algorithms serve as inputs for our solution.

Service Management in Edge: Hardware-based solutions for processing acceleration have been proposed recently [25], [26],

Manuscript received 28 June 2022; revised 11 September 2023; accepted 10 October 2023. Date of publication 23 October 2023; date of current version 7 May 2024. This work was partially supported in part by Australian Research Council (ARC) under Grants DP160102414 and DP230100081. Recommended for acceptance by A. Striegel. *(Corresponding author: TianZhang He.)*

TianZhang He and Rajkumar Buyya are with the CLOUDS Lab, School of Computing and Information Systems, University of Melbourne, Parkville, VIC 3052, 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).

This article has supplementary downloadable material available at https://doi.org/10.1109/TMC.2023.3326610, provided by the authors.

Digital Object Identifier 10.1109/TMC.2023.3326610

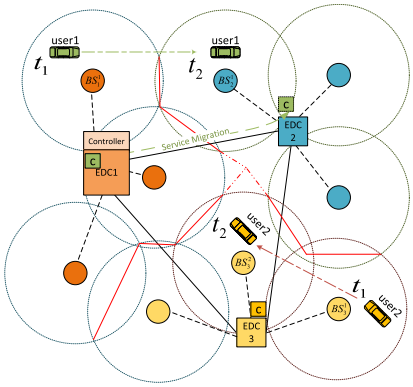


Fig. 2. User-mobility-induced service migration in edge computing generated stochastically.

[27]. However, ensuring end-to-end delay guarantees for services is an inherent problem in edge computing systems. To tackle the problem of user mobility, many service management algorithms have been proposed [5], [12], [13], [14], [15], [16]. These algorithms reactively or proactively generate service migrations to ensure Quality of Service (QoS). When users move from one base station to another, the network delay between the service and end-users can increase, potentially deteriorating the service quality. To avoid SLA violations, the service may need to be migrated to a nearby Edge Data Center (EDC) through live migration. For example, as shown in Fig. 2, user1's movement could trigger a migration of the service from the host of EDC1 to the host of EDC2 based on the nearest EDC first policy, or the migrations could be proactively generated to anticipate delay violations [16].

The placement of SDN controllers and computing orchestrators in regional edge data centers is determined based on the workload and latency requirements. A hierarchical architecture can be created by allocating multiple controllers in selected edge data centers. Data exchange between controllers is facilitated through west and east-bound interfaces, resulting in a logically centralized controller.

Multiple Migration Planning and Scheduling: Taking multiple migration requests with different source and destination pairs as inputs, the migration planner and scheduler determine the start time of each migration based on the status of migrating services and the availability of network and computing resources. Migration requests compete for computing and network resources due to co-location overheads and network routing issues [24]. With a complex behavior model of live migration that takes into account both computing and networking aspects [6], [23], the problem of multiple migration scheduling cannot be easily modeled as a maximum flow problem with network sharing [24].

The objective of migration planning and scheduling algorithms is to maximize the performance of multiple migrations in terms of total migration time, individual migration time, and downtime, in order to minimize the overhead and impact on other services [23], [24]. The "migration time" refers to the time interval from the initiation of the pre-migration process on the source host to the successful completion of the post-migration process on the destination host. On the other hand, "downtime" (also referred to as "freeze time") refers to the time interval during which the container or virtual machine (VM) is suspended, due to the final dump (stop-and-copy), commitment, and activation phases. When multiple migration requests are present, the "total migration time" is the interval from the start of the first migration to the completion of the last migration, which is considered the actual convergence time of service management optimization [23].

The elephant flows created during migration can lead to reduced availability of the network for other services in the system, negatively impacting the response time of both migrating and co-located services [23]. For time-critical services, a prolonged migration time increases the risk of violating migration deadlines and degrading the Quality of Service (QoS). Thus, determining the optimal sequence of multiple migration tasks is crucial for optimizing the performance of multiple migration scheduling.

Current algorithms for multiple migration planning and scheduling in cloud data centers are not suitable for edge computing due to differences in arrival patterns, processing times, and scales of services and edge networks: 1) The migration scheduling framework, which is periodically triggered by resource management policies with a long time interval, is not appropriate for mobility-induced migrations in edge computing, which have a stochastic arrival nature. 2) Mobility-induced migrations are generated based on real-time requirements for end-to-end delay. 3) The network topology and available network paths for migration flows are much more complex in edge environments compared to traditional data center networks. 4) The number of end-users and live migration requests can be ten thousand times higher in edge environments [14]. Without proper modeling, the problem complexity increases dramatically as the number of migration requests and network scale increases. Therefore, current algorithms for multiple migration planning and scheduling face serious challenges in terms of processing efficiency and scalability.

In this paper, we propose a solution to address the challenges in current multiple migration planning and scheduling algorithms for edge computing. Our proposed algorithms are focused on efficient large-scale live service migration planning and scheduling. They can also be applied to dynamic resource management at scale. The *key contributions* of this paper can be summarized as follows:

- A novel algorithm based on Iterative-Maximal Independent Sets (MISs) for efficient large-scale migration planning and scheduling in edge computing with real-time constraints.
- Lifecycle management for multiple migrations in edge computing.
- Introduction of a resource dependency graph to consider resource competition among migration requests and simplify the problem complexity.
- Modeling of the problem as finding the iterative Maximum Independent Sets of the remaining dependency graph and theoretical proof.
- Demonstration of QoS degradation without proper multiple migration planning and scheduling through realistic live migration simulations.

- Evaluation of the proposed framework and algorithms using a real-world large-scale Telecom dataset and taxi GPS traces in an event-driven simulator.

The rest of the paper is organized as follows. We review the related work in Section II and present the system architecture in Section III. Section IV analyzes and models the problem of multiple migration scheduling. Then, we propose two main methods of large-scale migration scheduling in Section V. Section VI shows the experimental design and evaluation with real-world datasets. Finally, we conclude the paper in Section VII. Appendices contains single migration model, proofs of the corresponding theorems, and performance analysis of proposed algorithms in graph processing.

II. Related Work

The live VM migration realization [6] and its application in cloud data centers, such as dynamic resource management, have matured in the last few years. Furthermore, live container migration technology has been studied recently. Checkpoint/Restore In Userspace (CRIU) [8] is a Linux software to migrate a container's in-memory state in userspace, which is integrated with LXC, Docker (runC), and OpenVZ. Nadgowda et al. proposed [28] a CRIU-based memory migration together with the data federation capabilities of union mounts to minimize migration downtime. Ma et al. [29] utilized the layered storage feature based on the AUFS storage drive to improve the performance of docker container migration.

Research on service management in edge computing through live migration technology is extensive (see survey [5]), with a focus on generating multiple migration requests reactively or proactively to meet network delay requirements with user mobility [12], [13], [14], [15], [16]. However, existing solutions for mobility-aware and delay-ware VNF, SFC, and service management neglect an essential phase, the performance of multiple migration scheduling, which could cause QoS degradation and SLA violations. Similar to [14], Ma et al. [16] proactively migrate services to several strategic locations to avoid migration time delay. The authors assume all migration requests for VNF replication are finished before each time slot.

With multiple migration requests generated by resource and service management algorithms, multiple live migration planning and scheduling algorithms are investigated to optimize the performance of migration scheduling [20], [21], [22]. In cloud data centers, Bari et al. [20] 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 online scheduler, the estimated start time of a live migration can lead to unacceptable migration performance and QoS degradations. Moreover, the work neglects the cost of individual migration by only comparing the migration group cost. Without considering the connectivity between VMs and the change of bandwidth, Wang et al. [21], [22] simplified the problem by maximizing the network transmission rate rather than minimizing the total migration time and proposed a polynomial-time approximation algorithm by omitting certain variables. However, the solution, Fully Polynomial-time Approximation Scheme (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 [24].

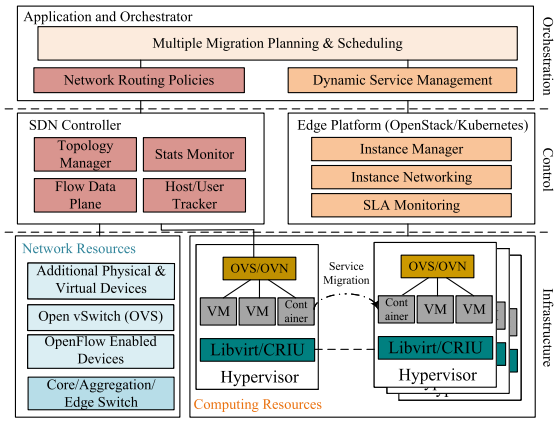


Fig. 3. System overview.

Table I shows that current algorithms can not meet the requirement of large scale in edge. The framework of migration scheduling [20] which is periodically triggered by resource management policies with a long time interval in clouds is not suitable for the mobility-induced migration scenario. By modeling and calculating every resource competition of migration directly, the problem complexity [20], [21], [22] increases along with the migration request number which is not suitable for the large-scale situation. For example, FPTAS [21] only considers 12 nodes with 19 links and 40 migration requests in experiments. The planning running time is also too large to schedule time-critical migrations. The algorithms [20], [21], [22] do not consider the deadline or urgency (priority) of migration. In addition, without an online 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.

III. System Architecture

The SDN controller plays a pivotal role in the proposed framework and algorithms by offering a range of essential capabilities. These include the provision of real-time network monitoring, such as end-to-end delay and remaining bandwidth. It facilitates the establishment of a finely segmented global network partition dedicated to migration activities. It also empowers the implementation of proactive networking routing and dynamic bandwidth allocation at the application level.

A. System Overview

This section outlines the details of the system architecture for migration management. The architecture of the SDN-enabled edge computing system consists of three layers: the orchestration layer, the controller layer, and the physical resource layer (as depicted in Fig. 3).

TABLE I
COMPARISONS OF MULTIPLE MIGRATION PLANNING AND SCHEDULING ALGORITHMS

research	real-time planning	large-scale	service correlations	user-mobility	deadline	SDN-enabled	Mig-Orient	Flow-Orient	Cloud	Edge
CQNCR [20]	–	–	✓	–	–	–	✓	–	✓	–
SLAMIG [24]	–	–	✓	–	✓	✓	✓	–	✓	–
FPTAS [21], [22]	–	–	–	–	–	✓	–	✓	✓	–
Our work	✓	✓	✓	✓	✓	✓	✓	–	✓	✓

The *infrastructure layer* utilizes the OpenFlow protocol and Open VSwitch (OVS) to provide flexible network resource management. Open Virtual Network (OVN), an OVS-based SDN solution, is widely used in industrial cloud networking. OVN provides network virtualization features with Kubernetes, such as subnetting, Quality of Service (QoS), static IP allocation, traffic mirroring, and OpenFlow-based network policy. Additionally, Libvirt and CRIU are employed to enable the live migration of VMs and containers across different servers.

The *control layer* consists of several controllers, including the networking manager (SDN controller), resource monitor engines, cloud controller (e.g., OpenStack), and container control plane (e.g., Kubernetes). The SDN controller manages the software-defined switches and flow entries, and collects device and flow statistics through its southbound interfaces. Networking applications use its northbound interfaces to perform topology discovery, network provisioning, and network routing policies. With the help of SDN, real-time monitoring of network topology and the status of virtual and physical links is possible. As a result, the service manager can continuously monitor the end-to-end delay between end users and edge services.

Unlike traditional setups in clouds, edge computing does not have a dedicated network for live migration [30]. By incorporating SDN into edge computing, the centralized controller can dynamically separate network resources from the service network [31] to build a virtual WAN network for live migrations. The bandwidth and network routing for migrations are dynamically allocated based on the reserved bandwidth of the service network, which reduces the overhead of live migrations on other services and ensures the performance of multiple live migrations. Additionally, the cloud and container control planes manage the lifecycle of individual live migrations and provide information related to live migration, such as dirty page rates and memory size.

In the *orchestration layer*, network routing engines such as QoS-oriented routing and network slicing are implemented. The service manager and multiple migration scheduler are facilitated by integrating the instance orchestrator and SDN controller. The service manager generates multiple migration tasks with different source and destination pairs and specific network routing based on the current network status. To achieve a detailed schedule, the planning and scheduling algorithm of multiple migrations interacts with the SDN controller and edge or cloud platform through northbound interfaces to control the start of each migration. The scheduling algorithm is automatically triggered at the beginning of each scheduling time interval.

B. Migration Lifecycle Management

The section presents the proposed lifecycle of a single migration request in edge computing (Fig. 4), including management

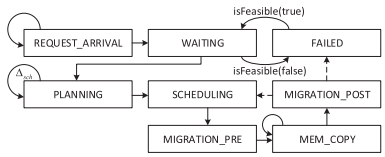


Fig. 4. Lifecycle for stochastic arrival live migrations.

states (*REQUEST_ARRIVAL*, *WAITING*, *FAILED*, *PLANNING*, *SCHEDULING*) and pre-copy live migration states (*MIGRATION_PRE*, *MEM_COPY*, and *MIGRATION_POST*).

Live migration with one state dump has limited migration time, which is solely dependent on the dump size and network bandwidth. However, the extended service downtime is not suitable for service migration in edge computing. Furthermore, downtime is highly sensitive to network bandwidth and cannot be guaranteed. Since pre-copy migration is the most commonly used technology to reduce migration downtime, it is considered the base model for live migration [8]. Compared to a simple live migration strategy, pre-copy migration with a downtime threshold can guarantee downtime with a longer migration time through iterative state synchronization [23].

Compared with periodically arrived multiple migrations in cloud data centers, the arrival of live migrations in edge computing is stochastic due to user mobility. To address this, we design a scheduling framework to handle the planning and scheduling of live migration in stochastic environments. When a migration request arrives, it enters the *WAITING* state and is buffered if it is feasible for scheduling, such as the container is not currently in migration. An infeasible request will be added to the *FAILED* waiting migration list for the corresponding container. The migration *PLANNING* event is triggered periodically at short intervals Δ_{sch} (e.g., 1 s) regarding the time-critical nature of live migration requests in edge. It generates a migration scheduling plan based on both *WAITING* and *SCHEDULING* migrations within each interval.

It is important to note that the scheduling interval plays a crucial role in the performance of multiple migrations. A large scheduling interval can lead to a negative impact on the efficiency and timeliness of service migration in edge computing due to the increase in the number of waiting migration requests. Conversely, a small interval can also have a detrimental effect on the performance of multiple migration scheduling. If the interval is significantly small, there may be only a limited amount of migration requests that can be scheduled per interval, leading to performance similar to that of First-Come-First-Serve (FCFS) scheduling and limiting the opportunities for optimization. It is important to note that initiating a migration immediately upon each request reception would not lead to any meaningful

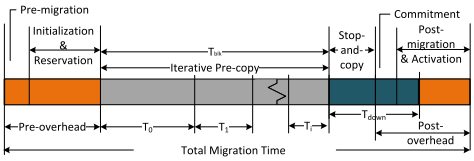


Fig. 5. Pre-copy live migration.

improvement or performance degradation in our scenario. By referring to the baseline algorithm (nosch) in Section VI, this point becomes clearer.

Based on the migration plan, the SDN-enabled online scheduler [24] initiates the migration process by allocating the bandwidth and routing. The pre-copy live migration then begins its pre-migration phase (*MIGRATION_PRE*) by extracting the procedure tree of the container. It traces the dirty memory in the userspace of the source server, and creates an empty container instance in the destination for state synchronization [8]. In *MEM_COPY*, the dirty memory is transferred iteratively to the instance in the destination for synchronization. In the post-migration phase (*MIGRATION_POST*), network communication is redirected to the new instance in the destination. The migrated container then recovers at the destination and triggers the start of subsequent resource-dependent migrations in the plan. Then, the feasibility flag of the first migration request of the same container in the *FAILED* migration waiting list is changed.

IV. Motivations and Problem Formulation

In this section, we first present the performance model for a single pre-copy live migration and then examine the challenges posed by multiple live migration scheduling in edge computing. These challenges include resource competition or dependency and real-time planning and scheduling. Finally, to address these challenges, we model the problem as an iterative process of generating the Maximal Independent Set (MIS) based on the resource dependency graph.

A. Single Migration Model

The performance of a single live migration, denoted by T_{mig}, can be divided into three components (see Fig. 5): pre-migration computing (pre-dump), memory-copy networking, and post-migration (restore) computing overheads, which are represented as $T_{mig} = T_{pre} + T_{mem} + T_{post}$. The details of single migration model in terms of memory transmission time T_{mem} (A.1) and stop-and-copy conditions, memory iteration copying rounds i (A.2), and downtime threshold T_{dthd} are introduced in Appendix A, available online.

Specifically, CRIU utilizes a pre-copy live migration technique that involves pre-dumping and incremental dumps. In small containers with 11 tasks, each pre-dump stage has been estimated to take around 0.5 seconds, while the restore time ranges from 0.7 to 1.9 seconds, depending on the size of the dump [8], [28]. In this paper, we set these values to 0.5 seconds and 1.0 seconds, respectively.

Every individual live migration task is non-preemptible, meaning that interruptions during the state synchronization process between the source and destination servers would render the migration process useless. As a result, in order to ensure a successful live migration, it is necessary to only initiate migrations when the available bandwidth is larger than the dirty page rate ($L > R$). This makes the problem of scheduling multiple live migrations not simply solvable by maximizing bandwidth utilization using a network flow optimization approach.

B. Concurrent and Sequential

For scheduling multiple migration tasks with various source and destination pairs, it is important to take concurrency into consideration to maximize performance. Two or more migrations that are dependent on shared resources should not be scheduled simultaneously, as demonstrated in [19], [23]. According to (A.1) and (A.2), available online, a reduced bandwidth allocation leads to an exponential increase in migration execution time and a potential increase in downtime due to the ratio σ. Thus, it is essential to optimize the networking resources in the WAN during multiple migration scheduling. For instance, when multiple migrations share network paths partially or entirely, scheduling them at the same time will result in a total migration time that is greater than the sum of their single execution time, as demonstrated in the experimental results [23]. Therefore, it is more efficient to schedule resource-dependent migrations sequentially to optimize migration performance, as suggested by [20], [23]. Meanwhile, migrations that are independent of resources can be scheduled concurrently to reduce the total migration time. To achieve optimal total and individual migration time, and downtime, it is important to exclusively allocate one network path to one migration until it is completed.

In addition, previous empirical experiments and models [6], [23] have shown that when the iterative memory copying overheads (dirty page rate and memory size) are small enough, scheduling multiple migrations with the same source and destination on the same network paths can reduce the total migration time with a slight increase in individual migration time. However, network bandwidth is often the limiting factor due to the large memory size and dirty page rate of VMs in the WAN. As indicated by (A.1), available online, and empirical results, both the total and individual migration time are increased when bandwidth is shared between live migrations. Thus, optimizing multiple migration scheduling by maximizing network flows would result in unacceptable migration time and downtime [24]. Additionally, live migrations that start with sufficient bandwidth converge faster than those that start with a smaller bandwidth, even if the average bandwidth is the same [24].

C. Real-Time Planning

There are several multiple migration planning and scheduling algorithms for live VM migration in cloud data centers [20], [21], [22]. However, the processing time of the scheduling sequence for multiple live migrations based on these algorithms is not suitable for the real-time requirements of mobility-induced migrations in edge computing. For instance, the processing time

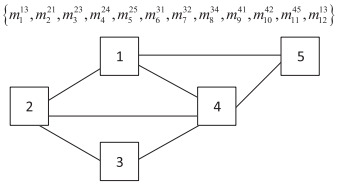


Fig. 6. Example live migration requests and the network topology with five compute hosts.

of FPTAS [21], [22] and CQNCR [20] 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, respectively, when generating the scheduling plan for 500 migrations. In traditional dynamic resource management, the algorithm is triggered every 30 minutes or 60 minutes, leaving enough processing time for algorithms to generate the optimal scheduling sequence. However, in the case of mobility-induced live migrations in edge computing environments, migration requests arrive at any time stochastically and most of them are time-critical. Therefore, the processing time of the planning and scheduling algorithm for mobility-induced migrations should be optimized to meet real-time requirements.

D. Resource Dependency Modeling

We utilize the capabilities of the SDN controller to generate a dynamic dependency graph of migration requests based on the flow routing algorithm, global network topology, available bandwidth, and host network interfaces. Fig. 6 illustrates an example with twelve live migration requests in an edge network topology of five server nodes. The migration request to migrate instance i from node s to node d is represented as m_i^{sd}. In this example, for simplicity, the network interfaces used for migration traffic are limited to one. This means that migration flows share the same interface when either the source or the destination server is the same. The network routing policy initially chooses the shortest network path with the largest available bandwidth for each migration. For instance, there are two network routes between node 1 and node 3, and since there is one migration from node 2 to node 3, the network path $\{1, 4, 3\}$ is chosen for $m_1^{1,3}$ and $m_{12}^{1,3}$. As shown in Algorithm 1, this can easily be extended to consider the set of network interfaces of source $\{s\}$ and destination servers $\{d\}$ and multiple migration network path choices $\{p\}$.

Resource-independent migrations from one concurrent scheduling group can be performed simultaneously. The planning algorithm should create a schedule plan that consists of several concurrent migration groups, with each group as large as possible. A larger concurrent group means that more migrations can be carried out at the same time, leading to improved performance in terms of total migration time and the quality of service of the migrating service. It's worth noting that migrations from different migration groups do not necessarily have any resource dependencies.

We model the interdependence of resources among migrations as an undirected graph (Algorithm 1), which is based on the source, destination, and network routing of migration requests according to the network topology provided by the SDN controller. Each node v_p^{sd} represents a sorted list of migrations with the same source s, destination interfaces d, and network paths p. The list is sorted based on the calculated migration time, as per Section IV-A. In other words, migrations within one node list form a complete graph as all migrations are interdependent on each other. For example, the migration list of node v^{13} is $\{m_1^{13}, m_{12}^{13}\}$. This significantly simplifies the problem complexity as the number of migration requests increases. The edge of the dependency graph signifies resource competition (network interfaces at the source or destination or bandwidth sharing along network routes) between migrations.

Algorithm 1: Dependency Graph Generation for Multiple Migrations Based on SDN Controller.

Input: $\{m_i^{sd}\}$
Result: $\{G_{dep}\}$
foreach $m_i \in M$ **do**
  $P_i \leftarrow$ GETPaths (m_i);
  ADDNode (G_{dep}, s, d, P_i);
foreach $v \in V(G_{dep})$ **do**
  foreach $u \in V(G_{dep})$ **do**
    if $v! = u$ **then**
      if IsDependent (u,v) **then**
        ADDEdge $(G_{dep}, (u, v))$;
return G_{dep} , $\{M(v_{sd})\}$

The proposed dependency graph is designed to be a generic model that is compatible with both existing network slicing [17] and routing algorithms [18]. For instance, it considers both physical and virtual paths as a path p in a uniform manner. The routing algorithm used in this paper is the k-shortest path routing with the largest available bandwidth, where k is equal to 2, representing the two network interfaces of a physical server. This means that each server can have two NICs allocated to migration flows. In Algorithm 1, two migrations $u(m_i^{s_i,d_i,\{p_i\}})$ and $v(m_j^{s_j,d_j,\{p_j\}})$ are considered resource-dependent only if either the network interface sets s_i and s_j are equal, or the network interface sets d_i and d_j are equal, or the following statement is true:

$$bw(\{p_i\} - \{p_i\} \cap \{p_j\}) < \min(bw(s_i), bw(d_i)) \vee$$
$$bw(\{p_j\} - \{p_j\} \cap \{p_i\}) < \min(bw(s_j), bw(d_j)). \quad (1)$$

Before discussing the Maximum Independent Set of the resource dependency graph, we first review some basic graph concepts [32]. Such concepts include clique, C, and independent set, I. A clique is a subset of vertices of an undirected graph G, where 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 graph G is the opposite of a clique in 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 represents a concurrent migration group, meaning that the migrations from

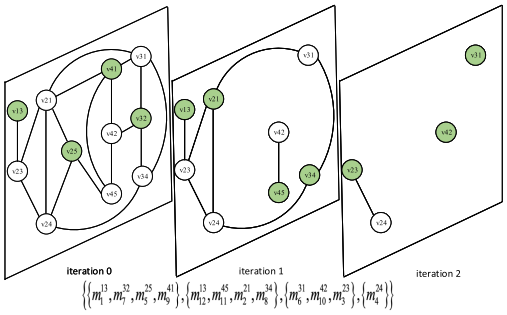


Fig. 7. Iterative maximal independent sets as concurrent migration groups based on the temporal graph of migration resource dependency for the illustrative example.

the nodes in the independent set I can be scheduled concurrently. Conversely, migrations from the nodes in a clique are resource-dependent and must be scheduled sequentially.

As shown in Fig. 7, a maximal concurrent scheduling group is a set of resource-independent migrations that is not a subnet of any other concurrent group. No additional migration can be included in the group to make all migrations perform simultaneously. As a result, the group is equivalent to a Maximal Independent Set (MIS) and the largest MIS is the maximum independent set. There are multiple combinations of migrations that can form a maximal scheduling group. To solve this problem, we can use the Maximum Independent Set algorithm, where in each iteration, the migrations with the shortest migration time from each node within the maximum independent set are selected and removed from the graph. The algorithm stops when there are no nodes left in the graph. For instance (Fig. 7), in the first iteration, one of the maximum groups could be $\{m_1^{13}, m_7^{32}, m_5^{25}, m_9^{41}\}$, while another maximal group could be $\{m_3^{23}, m_{11}^{45}, m_6^{31}\}$. The maximum group is a better choice. After selecting migrations from the nodes in the maximum independent set, we delete these migrations and update the dependency graph. A node is deleted from the graph when there are no remaining migrations in its node list. For instance, after the first iteration, nodes v^{25}, v^{41}, v^{32} are deleted, but node v^{13} still has one migration m_{12}^{13} left in its list. Therefore, it is crucial to choose migrations carefully to achieve the largest concurrent group size. In the end, the online migration scheduler schedules all migrations in the first group. Then, when one migration is finished, the scheduler starts all migrations blocked by the finished migration in the order of migration groups.

E. Problem Modeling

The planning and scheduling algorithm is triggered periodically at set intervals of Δ_{sch}. We use M_{arriv}^t to denote the set of arrival migration requests at planning time t, M_{wait}^t as the set of migration requests waiting for planning at time t, and M_{fail}^t as the set of infeasible migrations such as those involving containers that are currently in migration. Additionally, M_{plan}^t represents migrations that have been planned but not yet completed at time t, while M_{finish}^t denotes migrations that have finished at time t.

The input of migration requests at every migration planning time t is given by $M_{input}^t = M_{plan}^t \cup M_{wait}^t$. For each live migration m_j, we have the source and destination edge data center, as well as the 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 for each live migration can be represented as $r_j = f_j - a_j$. The slack time for migration scheduling, which represents the remaining scheduling window that one migration will not miss its deadline, is given by $\tau_j = a_j + D_j - T_j - t$. The objective of live 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 an acyclic undirected graph where $|G| = |V|$. Each node $u \in V$ represents the list of migrations $M(u)$ where migrations share the same source s, destination d, and network routing p. By sharing the same source, 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 obtain the maximal independent set of the remaining graph and then update the graph by deleting the corresponding migrations. Let $G_{i+1} = G_i[V(G_i) - S_i]$ represent the remaining graph by directly deleting the vertex from the set of nodes S_i. Let I_i denote the maximal independent set of graph G_i. Then, the remaining graph G_{i+1} in each iteration can be represented as

$$G_{i+1} = G_i\left[\left[V\left(G_i\right) - I_i\right]\right] = G_i\left[V\left(G_i\right) - S_i\right], \quad (2)$$

by deleting a 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 graphs

$$\max\left|I_i\right|, \forall I_i \in \left\{I_{iter}^i\right\}, \quad (3)$$

where $\{I_{iter}^i\} = \{I_1, I_2, \ldots, I_K\}$ is the total K iterative independent sets and there are 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 the maximum independent set of the 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

$$W\left(\hat{m}\right) \geq W(m), \forall \hat{m}, m \in M(u). \quad (4)$$

Then, the objective of multiple migration planning is

$$\max W\left(I_i\right), \forall I_i \in \left\{I_{iter}^i\right\}. \quad (5)$$

Algorithm 2: Iterative Approximation Grouping.

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

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 [33], [34]. Therefore, generating the iterative maximum independent set as the subset is also NP-hard.

F. 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 [32], [33], [35]. In other words, listing all maximal independent sets or finding the maximum independent set of a graph is equivalent to listing all maximal cliques or finding the maximum clique of its complement graph. Thus, in each iteration, we can find the maximum independent set by obtaining 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 [35]. In other words, each clique can be generated in polynomial time in the listing of all maximal cliques [33]. When we only consider vertices, the optimal algorithm for maximal clique listing is the maximal cliques listing algorithm (CLIQUES) [34], [35] based on Bron-Kerbosch [32]. 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 $3^{n/3}$ [36]. The best-known time complexity of finding one maximum independent set has been improved is $O(2^{n/4})$ [37]. 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 real-time live migration scheduling in edge computing, which exhibits an exponential time complexity.

V. Migration Planning and Scheduling

The section presents the proposed planning and scheduling algorithms for large-scale live migrations in edge computing. With the waiting live migration requests and planned unfinished live migrations as input, the migration planner needs to efficiently schedule arriving migrations while maintaining the QoS. Based on the problem modeling in Section IV-E, this problem is reduced to finding an MIS of the migration dependency graph iteratively. Therefore, we propose two approaches to generate the iterative MISs of the dependency graph: 1) Direct iterative MISs generation, including approximation (approx) and greedy iterative MISs algorithm (iter-GWIN), and 2) Maximum Cliques (MCs)-based MISs generation, including Iterative-rounds MCs (iter-MCs) and Single-Round MCs Algorithm (single-MCs). Appendix C, available online, presents a detailed computational performance evaluation and analysis for the problem of iterative MIS generation and the proposed algorithms.

A. Direct Iterative-Maximal Independent Sets

For the direct iterative MISs 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 are no vertices left $G_{dep} = \emptyset$.

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 2), we use the approximating maximum independent sets algorithm by excluding subgraph [38] 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 MISs of the remaining graph, there is at least one node removed from the graph G_i. Given total m live 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 [38].

Based on the newly generated scheduling plan $\{I_{iter}\}$, the SDN-enabled online 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.

2) Greedy Iterative MISs Algorithm: The greedy algorithm (iter-GWIN) generates the concurrent groups (MIS) of live migration iteratively. A greedy maximal independent set algorithm (GWIN) [39] based on the weight and the degree of a node is adapted to directly generate the MIS in each iteration. Let Δ_G denote the maximum degree and $\bar{d}_G$ is the average degree of G. The degree of node u in G is $d_G(u) = |N_G(u)|$. $N_G(u)$ is the set of neighbor nodes of vertex u and $N_G^+(u) = N_G(u) \cup \{u\}$.

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

$$W(u)/(d_{G_i}(u) + 1). \tag{6}$$

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

The problem model shows that 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 [20] after migration for non-time-critical migrations and the deadline or slack time for real-time

Algorithm 3: iter-GWIN.

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

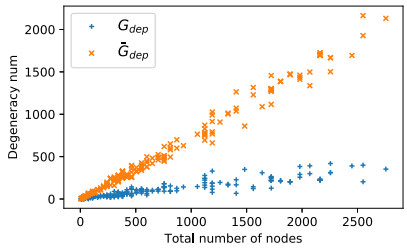


Fig. 8. Total number of nodes and degeneracy number of the resource dependency graph of WAN topologies [40] and its complement graph.

migrations scheduling. We consider the weight function regarding the slack time τ as follows:

$$W(m) = \begin{cases} 10 \cdot \beta/\tau & \tau > \beta \\ 100 \cdot |\tau|/\beta & \tau < -\beta \\ 100 & \text{other} \end{cases}, \tag{7}$$

where β is the slack time threshold. We set $\beta = 1$ in this paper. The weight of the 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 a situation where the priorities of all migrations are the same, we only need to consider the size of MIS. The node weight is set to $W(u) = 1$. In each iteration, the lower-bound of the maximum (weighted) independent set is $\sum_{u \in V} W(u)/(d_G(u) + 1)$ [39]. 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.

B. The Maximum Cliques-Based Heuristics

Based on the observation of the density property of the migration resource dependency graph, we propose the iterative Maximum Cliques (MCs)-based algorithm (iter-MCs). We first discuss the rationales of iter-MCs.

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 of the graph spareness. For an n-vertex graph with degeneracy d, by introducing the sequence ordering based on degeneracy, Bron-Kerbosch Degeneracy algorithm [41] can list all maximal cliques in time $O(dn3^{d/3})$. With a spare graph that $n \geq d + 3$, the upper bound of all maximal cliques number is $(n - d)3^{d/3}$. Fig. 8 illustrates the nodes and the density (degeneracy) of the resource dependency graph of WAN network topologies [40] and its complement. It shows that the degeneracy

Algorithm 4: Iterative Heuristic MCs.

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

of the complement graph $\bar{G}_{dep}$ is 4.34 times that of G_{dep}. For G_{dep} and its complement graph, the average ratio of dependency d to the total number of nodes n is 0.153 and 0.714, respectively. The resource dependency graph is considerably more sparse than its complement graph. Therefore, for G_{dep}, there are much fewer maximal cliques than the total MIS. According to the theoretical time complexity, the running time of listing all maximal cliques or the maximum clique of G_{dep} is much smaller than that of listing all maximal independent sets or the maximum independent set of 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 in the same independent set, the iterative maximum cliques serve as a heuristic pruning decider to speed up the algorithm.

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 Maximum Cliques (MCs)-based heuristic algorithm (Algorithm 4) follows two steps: 1) generating the maximum clique iteratively and 2) obtaining the MIS from the iterative maximum cliques.

As shown in Algorithm 4, we first create dependency graph G_{dep} as the input based on the source-destination of the given migrations and the network topology. From lines 1-6, the algorithm calculates the iterative maximum cliques of the dependency graph until there are no vertices left. In each iteration, it generates the maximum clique (Bron-Kerbosch Degeneracy algorithm) [41] of the remaining graph. It is proved that the algorithm is highly efficient in a sparse graph, such as the resource dependency graph [41]. 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

$$W(u)/\left(d_{G[C]}(u) + 1\right). \tag{8}$$

In the second step (lines 7–12), it generates maximal independent sets based on the iterative maximum cliques. In each round (lines 9–11), it selects the feasible node with the maximum score of each maximum clique $\hat{C}_i$ and adds the largest-weight migration from its list into the independent set. A node is

Algorithm 5: Single-Round Iterative MCs.

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

feasible when it can be included in the current independent set. If there are 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 $O(dn^2 3^{d/3})$.

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

Proof: see Appendix B, available online □

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 all maximal cliques of the initial dependency graph G_{dep}. The maximum clique size of each iteration is the same as the iter-MCs. We also prove the correctness of the proposed single-round iterative maximum cliques algorithm. The first step of the iter-MCs algorithm is replaced by Algorithm 5. The algorithm only generates the list of all maximal cliques $\{\bar{C}\}$ once by using the Bron-Kerbosch Degeneracy algorithm. Until there are 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.

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: see Appendix B, available online □

In conclusion, the first group of proposed algorithms, approx and iter-GWIN, directly calculate the MISs of the remaining graph in each iteration. iter-GWIN and single-MCs have the lowest time complexity. In the second group, heuristic algorithms, iter-MCs, and single-MCs calculate iterative MISs based on the Maximum Cliques of the original graph according to the property of the dependency graph. The approximation algorithm (approx) can guarantee an approximate ratio $O(n/(\log n)^2)$ in polynomial time. Compared with other algorithms for generating iterative MISs of a graph, single-MCs can largely reduce the processing time as it is only based on the initial result of Maximal Independent Sets at the first iteration. Furthermore, we prove the correctness of single-MCs that direct MIS calculation for the remaining graph in each iteration.

VI. Performance Evaluation

This section evaluates 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. We explain the placement of edge data centers, the network topology, and the regional coverage of each EDC. The event-driven simulator for software-defined network-enabled edge-cloud computing [42] is extended to emulate the user movement and live migration. It provides a network operating system based on 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.

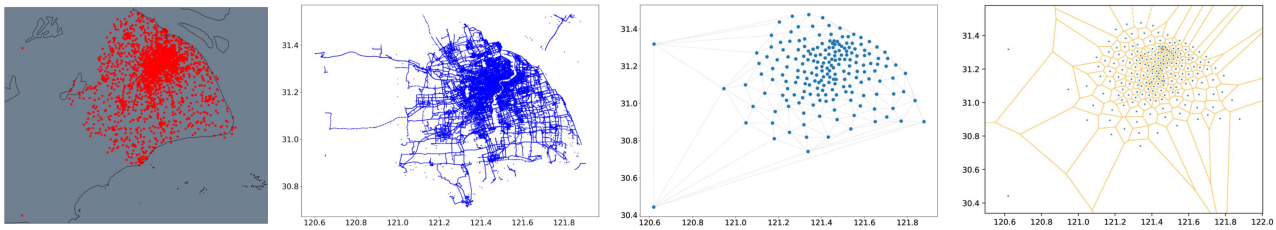
Based on the evaluation of graph processing performance in Appendix C, available online, we select iter-GWIN and single-MCs algorithms to compare with the state-of-art inter-cloud algorithm (FPTAS) [21], [22] and a baseline service management policy without planning and scheduling algorithm. We evaluate the performance of live migration planning and scheduling algorithms in processing time, migration time, downtime, transferred data, deadline violations, and network transmission time. The time interval of each scheduling window is set at one second.

A. Experimental Data

We use coordinates from the Shanghai Telecom dataset[1] and the Shanghai Qiangsheng taxi GPS trace dataset[2] (April 1, 2018). The longitude and latitude values are limited to the range of 30.40° N to 31.35° N and 120.51° E to 122.12° E. The Shanghai Telecom dataset contains 3,233 base station coordinates (Fig. 9(a)). Using the K-means algorithm [43], we generate 200 Edge Data Center (EDC) locations based on the base station coordinates. The taxi GPS trace dataset (Fig. 9(b)) contains timestamped GPS coordinates and additional data on taxi status and vehicle parameters (e.g., speed, direction, number of connected satellites). Base stations are clustered and connected to regional Edge Data Centers (EDCs) (Fig. 9(c)). We use Delaunay Triangulation [44] to generate links between the EDC gateways (Fig. 9(c)). For network routing within this topology, we use the shortest path that is no longer than $4\pi/3\sqrt{3}$ times the euclidean distance between source and destination. The EDC regions' boundaries (Fig. 9(d)) form a Voronoi diagram [44], where any point's euclidean distance 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 [13], [14], 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 migration in edge computing. Fig. 10 demonstrates an example that the request of live migration is induced when a taxi moves across the boundary between two clusters of EDCs. The deadline of each live migration is generated based on the average mobility speed of users in the

[1] [Online]. Available: http://sguangwang.com/TelecomDataset.html
[2] [Online]. Available: http://soda.shdataic.org.cn/download/31



(a) Shanghai Telecom base stations (b) The taxi GPS trace in the first hour (c) Delaunay triangulation network topology (d) Edge data center coverage with Voronoi cells

Fig. 9. Experimental dataset and configurations with longitude and latitude data as x and y axes.

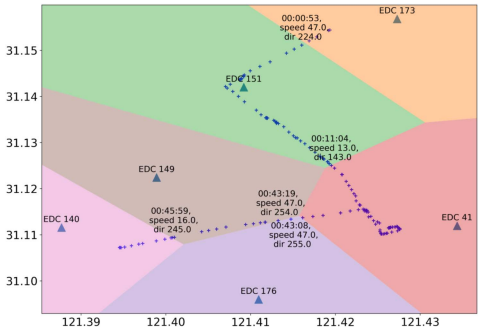


Fig. 10. Example of live migration request triggered by user movement with longitude and latitude data as x and y axes.

last 3 GPS records, travel direction, and the signal strength of base stations.

B. Experimental Setup

This sectiondescribes the details of the experiment setup. The end-to-end delay between the user and the service is the time interval from the user (taxi) sending workload to the container assigned in the EDC to the result 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 1,000 cycles per bit [14]. The resulting packet sent back from the container to the user is 128 bytes. The total CPU power frequency for each EDC with multiple CPUs is 25 GHz [45], [46]. The reserved virtual (application) bandwidth between a container server and the user client is set at 3 Mbps, which is sufficient for streaming applications in edge computing environments. For instance, Google streaming bandwidth requirements for 720p video bitrate ranges from 1,500 to 4,000 Kbps, 480p from 500 to 2,000 Kbps, and 360p from 400 to 1,000 Kbps, and a 0.6 Mbps configuration was also chosen in mobile edge environments [14]. Thus, it is reasonable and sufficient for video streaming and object detection in the edge (e.g., Yolo 608 and 416 resolution setup). To simulate the limited network resources for migrations, the bandwidth of physical network links between EDCs is set to 1 Gbps. The network delay between base stations and regional EDCs is fixed at 5 ms, while the delay between EDCs is randomly generated between 5 and 50 ms [14].

TABLE II
SERVICE MIGRATION SCENARIOS WITHIN 1 HOUR

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

According to the evaluation results of container memory and dirty memory size during live container migrations [29], we generate the container memory from 100 MB to 400 MB. The dirty page rate for each dirty memory transmission is from 2 MB/s to 8 MB/s and the data compression rate is 0.8 [47]. We configure the downtime threshold and the maximum iterations for live migration at 0.5 seconds and 30 times [23], respectively. Based on the SDN controller, the remaining network bandwidth between the source and destination hosts in 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.

We consider the GPS trace of randomly selected vehicles from 1,000 to 4,000 within one hour (S1 to S3 in Table II). 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 first policy is considered in our experiments to generate live migration requests to allocate service containers to the nearest edge data center. According to the user mobility, one live migration will be triggered when one vehicle exits the coverage area of its current edge data center. There are 9,933, 19,522, and 37,822 migration requests induced by these vehicles' movement, respectively. During the live migration, the dirty memory of migrating instances 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 assume that the container image as a universal service is already available in all EDCs or shared by the network storage. CloudSimSDN-NFV [42], an event-driven simulator, is extended with corresponding components to support pre-copy live migration and user mobility in edge computing.

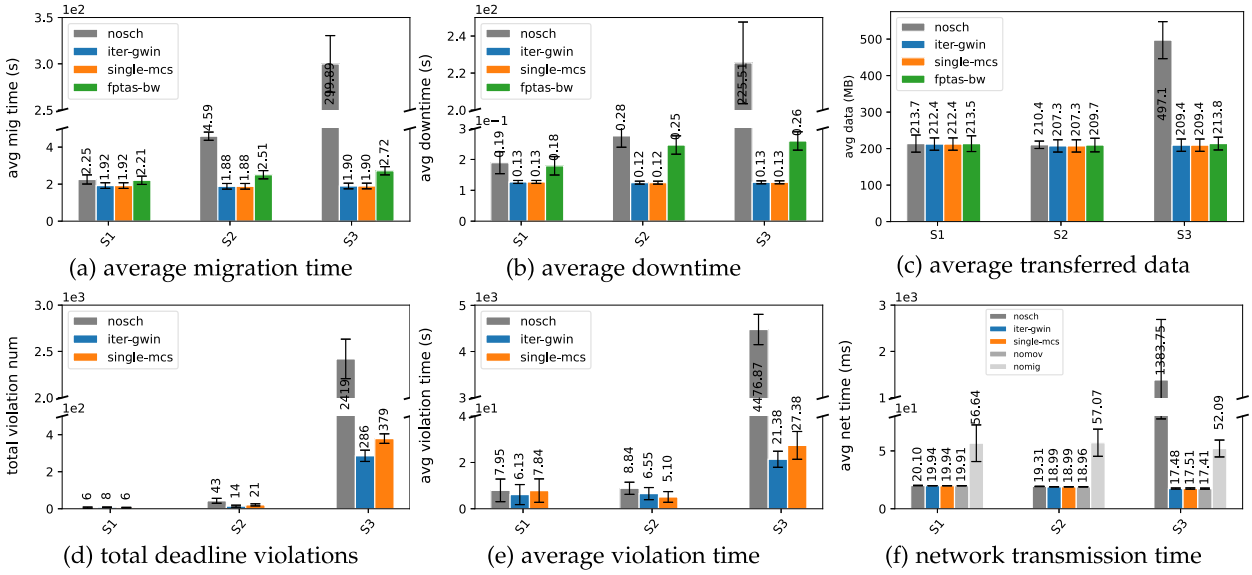


Fig. 11. Migration performance compared to no scheduler, iter-GWIN, and single-MCs under different scenarios with 0.95 CI.

TABLE III
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 [21], [22]	903597.39	1923036.81	4677990.57

C. Experimental Results and Analysis

We compare the experimental results between no migration scheduler, iter-GWIN, single-MCs, and the current state-of-the-art migration planning and scheduling for inter-clouds FPTAS [21], [22]. In FPTAS, to maximize the total bandwidth utilized by migrations, one migration can be started even though 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) as the migration start condition that the available bandwidth must be larger than the dirty page rate. As the vehicle number increases from S1 to S3, the density of live migration requests in certain areas increases dramatically. The resource competition or resource dependency among live migration requests will also increase. As a result, the complexity of the dependency graph may also increase. When the requirements of live migration requests exceed the resource capacity provided by edge computing, it is inevitable that some of the deadlines of some migration requests can not be satisfied.

Table III shows the total processing time of migration planning and scheduling algorithms within 1 h 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 graph algorithm evaluation in Appendix C, available online. Furthermore, compared to FPTAS [21], [22], the performance of our solution in terms of processing time has been improved by more than 3,000 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. As the results of FPTAS-BW in deadline violation exceed chart comparison limits, 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. 11(a)). Particularly, in S3, the allocated bandwidth may either be smaller than the dirty page rate and cause a large downtime for some migrations. Or, it causes a much longer migration time due to a large number of memory-copying iterations. As a result, the migrating service suffers a devastating consequence. Furthermore, for FPTAS-BW, by maximizing the total migration bandwidth rather than the resource competition, it suffers smaller average bandwidth per migration. Thus, as shown in Fig. 11 the performance of our purposed solution in terms of average migration time, average downtime, and total transferred data are increased by up to 30.24%, 51.56%, and 2.06%, respectively. Meanwhile, for the proposed planning and scheduling algorithms iter-GWIN and single-MCs, the performance of live migration can be guaranteed even with severe resource competition. Results (Fig. 11(a) and (b)) show that the average migration time and downtime are optimal at 1.9 and 0.13 seconds as there is no bandwidth sharing between resource-dependent migrations.

As shown in Section IV-A, the migration time of each individual migration is influenced by factors such as memory dump size, dirty page rate, downtime threshold, and bandwidth between the migration source and destination. Therefore, actual container and VM migration time can vary from a few

seconds to minutes [23], [29]. For instance, the experiment demonstrates Xonotic server migration across the world,[3] and the live migration of a Minecraft container from AWS to Azure in around 60 seconds.[4] To verify our model and results, we also evaluated live container migration in real hardware. We performed live migrations of the Xonotic server container with controlled bandwidth between servers. The Pre-dump size is around 400 M and the second-round dump size is around 20 M. Plus the processing time of pre-migration and post-migration (Section IV-A), the migration time shown in Fig. 11 is justifiable.

Furthermore, for all migrations that arrive within the 3,600 seconds 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 48878.65 seconds in S3. A shorter average migration means less possibility of QoS degradation and less occupation time on the network resource. Less downtime equals less disruption to 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. 11(c)). Because of the container's small memory footprint, the shared bandwidth can still satisfy the downtime threshold with relatively small memory-copying iterations. However, when the bandwidth becomes the bottleneck, a large number of memory-copying iterations are needed to meet the downtime threshold. Therefore, the total transferred data in S3 increased 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. 11(d)). However, the ratio for no migration scheduler is 3.07 times in S2 and 8.46 times compared to the best result from iter-GWIN. The ratio of total violation time to the service time of all containers in one hour is 0.00127% and 0.00148% in S2 and 0.0425% and 0.0720% in S3, respectively (Fig. 11(e)). In S3, although migration performance in migration time and downtime is optimized by the migration scheduler, the network resource is insufficient to schedule all 37,822 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. 11(f)). In the scenario that all vehicles stay at the start point 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 elephant migration flows and downtime and deadline violations have a considerable impact on the service network delay. Compared to the iter-GWIN result, without considering requests during migration downtime, network delay increases 6.62 times at 133.24 ms and 79.16 times at 1.38 seconds without and with timeout network requests, respectively (Fig. 11(f)). Taking direct downtime impacts into account, the average delay increases to 5.42 seconds with 8.32 confidence intervals due to the disproportionately large service downtime.

In summary, our proposed algorithms can efficiently plan and schedule large-scale mobility-induced live migrations in edge computing. Even in the case of a migration request surge, it guarantees the performance of live migrations and maintains the QoS of migrating services. It significantly reduces average migration time (up to 99.36%), average downtime (up to 99.94%), total deadline violations (up to 88.18%), and violation time (up to 99.94%).

VII. Conclusion and Future Work

This paper investigated the challenges of live migration scheduling in edge computing, including resource competitions or dependencies among live migrations and real-time migration planning and scheduling. To address these challenges, we proposed a framework for migration scheduling triggered by users or users' mobility that utilizes SDN to minimize the impact of migration flows on other edge services. Specifically, we modeled the relationship of resource dependencies among migrations as an undirected graph and formulated the scheduling problem as generating the maximum independent set of the dependency graph iteratively. To further improve the efficiency and scalability of the proposed framework, we developed two large-scale migration planning and scheduling algorithms based on iterative Maximal Independent Sets. We conducted computational experiments to evaluate the performance of the graph algorithms, and real-world data experiments further demonstrated their effectiveness in optimizing live migration performance and minimizing deadline violations in migration scheduling. Our work contributes to the development of fine-grained multiple migration planning and scheduling techniques for edge computing, which can maintain the QoS of migrating services while accommodating the mobility of users and resources in a complex network environment. As part of the future work, we intend to investigate base station clustering for EDC placement based on user mobility to reduce the number of live migrations.

[3][Online]. Available: http://www.redhat.com/en/blog/container-migration-around-world

[4][Online]. Available: http://www.infoq.com/articles/container-live-migration/

ACKNOWLEDGMENT

The authors thank the editor-in-chief, associate editors, and anonymous reviewers, 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, Jan. 2017.
[2] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 140–147, Feb. 2018.
[3] T. V. Doan et al., "Containers vs virtual machines: Choosing the right virtualization technology for mobile edge cloud," in *Proc. IEEE 2nd 5G World Forum*, 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] 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 Comput. Surv.*, vol. 52, no. 5, pp. 1–33, 2019.
[6] C. Clark et al., "Live migration of virtual machines," in *Proc. 2nd Conf. Symp. Netw. Syst. Des. Implementation*, 2005, pp. 273–286.
[7] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, "Containers checkpointing and live migration," in *Proc. Linux Symp.*, 2008, pp. 85–90.
[8] CRIU, "Live container migration," 2019. [Online]. Available: https://criu.org/Live_migration
[9] Container migration with podman on RHEL, Aug. 2019. Accessed: Jan. 22, 2020. [Online]. Available: https://www.redhat.com/en/blog/container-migration-podman-rhel
[10] V. Marmol and A. Tucker, "Task migration at scale using CRIU," Nov. 2018. Accessed: Feb. 22, 2022. [Online]. Available: https://lpc.events/event/2/contributions/69/
[11] A. Ruprecht et al., "VM live migration at scale," *ACM SIGPLAN Notices*, vol. 53, no. 3, pp. 45–56, 2018.
[12] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *Proc. IFIP Netw. Conf.*, 2015, pp. 1–9.
[13] 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 Trans. Netw.*, vol. 27, no. 3, pp. 1272–1288, Jun. 2019.
[14] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 939–951, Mar. 2021.
[15] Y.-T. Chen and W. Liao, "Mobility-aware service function chaining in 5G wireless networks with mobile edge computing," in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–6.
[16] Y. Ma, W. Liang, J. Li, X. Jia, and S. Guo, "Mobility-aware and delay-sensitive service provisioning in mobile edge-cloud networks," *IEEE Trans. Mobile Comput.*, vol. 21, no. 1, pp. 196–210, Jan. 2022.
[17] S. Vassilaras et al., "The algorithmic aspects of network slicing," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 112–119, Aug. 2017.
[18] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation," *IEEE Commun. Surv. Tut.*, vol. 20, no. 1, pp. 388–415, First Quarter 2018.
[19] T. He, A. N. Toosi, and R. Buyya, "CAMIG: Concurrency-aware live migration management of multiple virtual machines in SDN-enabled clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2318–2331, Oct. 2022.
[20] M. F. Bari, M. F. Zhani, Q. Zhang, R. Ahmed, and R. Boutaba, "CQNCR: Optimal VM migration planning in cloud data centers," in *Proc. IFIP Netw. Conf.*, 2014, pp. 1–9.
[21] H. Wang, Y. Li, Y. Zhang, and D. Jin, "Virtual machine migration planning in software-defined networks," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 487–495.
[22] 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, Fourth Quarter 2019.
[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.
[24] 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.
[25] A. Rodríguez et al., "FPGA-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The ARTICO3 framework," *Sensors*, vol. 18, no. 6, 2018, Art. no. 1877.
[26] P. Shantharama, A. S. Thyagaturu, and M. Reisslein, "Hardware-accelerated platforms and infrastructures for network functions: A survey of enabling technologies and research studies," *IEEE Access*, vol. 8, pp. 132021–132085, 2020.
[27] T. Belabed, M. G. F. Coutinho, M. A. C. Fernandes, V. Carlos, and C. Souani, "Low cost and low power stacked sparse autoencoder hardware acceleration for deep learning edge computing applications," in *Proc. Int. Conf. Adv. Technol. Signal Image Process.*, 2020, pp. 1–6.
[28] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete container state migration," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2137–2142.
[29] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 2020–2033, Sep. 2019.
[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] 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.
[32] C. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, 1973.
[33] 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.
[34] 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.
[35] 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.
[36] J. W. Moon and L. Moser, "On cliques in graphs," *Isr. J. Math.*, vol. 3, no. 1, pp. 23–28, 1965.
[37] J. M. Robson, "Finding a maximum independent set in time O (2n/4)," Tech. Rep. 1251–01, LaBRI, Université Bordeaux I., Talence, France, 2001.
[38] R. Boppana and M. M. Halldórsson, "Approximating maximum independent sets by excluding subgraphs," *BIT Numer. Math.*, vol. 32, no. 2, pp. 180–196, 1992.
[39] S. Sakai, M. Togasaki, and K. Yamazaki, "A note on greedy algorithms for the maximum weighted independent set problem," *Discrete Appl. Math.*, vol. 126, no. 2/3, pp. 313–322, 2003.
[40] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.
[41] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," in *Proc. Int. Symp. Algorithms Comput.*, Springer, 2010, pp. 403–414.
[42] 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. Exp.*, vol. 49, no. 12, pp. 1748–1764, 2019.
[43] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2866–2880, Oct. 2016.
[44] P. Virtanen et al., "Scipy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020.
[45] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.
[46] Y. Xiao and M. Krunz, "QoE and power efficiency tradeoff for fog computing networks with fog node cooperation," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[47] 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.