Virtual Servers Co-Migration for Mobile Accesses: Online vs. Off-line

Yang Wang, Wei Shi *IEEE Member*, and Menglan Hu

Abstract—In this paper, we study the problem of co-migrating a set of service replicas residing on one or more redundant virtual servers in clouds in order to satisfy a sequence of mobile batch-request demands in a cost effective way. With such a migration, we can not only reduce the service access latency for end users but also minimize the network costs for service providers. The co-migration can be achieved at the cost of bulk-data transfer and increases the overall monetary costs for the service providers. To gain the benefits of service migration while minimizing the overall costs, we propose a co-migration algorithm *Migk* for multiple servers, each hosting a service replicas. *Migk* is a randomized algorithm with a competitive cost of $O(\frac{\gamma \log n}{\min\{\frac{1}{\kappa}, \frac{\mu}{\lambda+\mu}\}})$ to migrate κ services in a static n-node network where γ is the maximal ratio of the migration costs between any pair of neighbor nodes in the network, and where λ and μ represent the maximum wired transmission cost and the wireless link cost respectively. For comparison, we also study this problem in its static off-line form by proposing a parallel dynamic programming (hereafter DP) based algorithm that integrates the branch&bound strategy with sampling techniques in order to approximate the optimal DP results. We validate the advantage of the proposed algorithms via extensive simulation studies using various requests patterns and cloud network topologies. Our simulation results show that the proposed algorithms can effectively adapt to mobile access patterns to satisfy the service request sequences in a cost-effective way.

Index Terms—VM co-migration; service migration; mobile access; branch and bound algorithm

1 Introduction

CLOUD-based services for the mobile world leverage the advantages of both cloud platforms and mobile devices in order to fulfill user requests with enhanced quality of service (QoS) regardless of time and location. As a direct consequence of this advantageous integration, the cloud-based mobile service market presents a continuous growth rate over the past years. For example, as of the end of 2013, the increased use of small-scale clouds and mobile devices has led more than 88% of enterprises to participate in *bring your own device* (BYOD) programs [1]. That is, most businesses have employees accessing their clouds using mobile devices while working at remote and/or off-site locations.

Cloud platforms allow the cloud service providers (CSPs) to utilize their provisioned computing resources in a cost-effective way while mobile devices offer convenient and powerful tools to communicate with services over wireless networks. Although this combination is promising, the design of mobile services is still a great challenge when considering the characteristics of mobile access patterns and the use of clouds. For example, batches of requests stemming from mobile devices require access to services residing in clouds. But the positional origin of such requests frequently changes over time. This complicates the delivery of the requested services (especially when aiming to achieve enhanced QoS and cost effectiveness), which is particularly true in the case of real-time services, that is, services with constraints on their response time. Consequently, a mobile user's dynamic behavior patterns may be associated with temporal access patterns [2]. In contrast, cloud resources are provisioned on demand and are usually charged based on a *pay-as-you-go* model. As such, providing cloud services without considering these characteristics may significantly decrease the service quality and compromise the user experience. Moreover, an inappropriate solution may also impose a large amount of access traffic on the network, which is very likely to entail

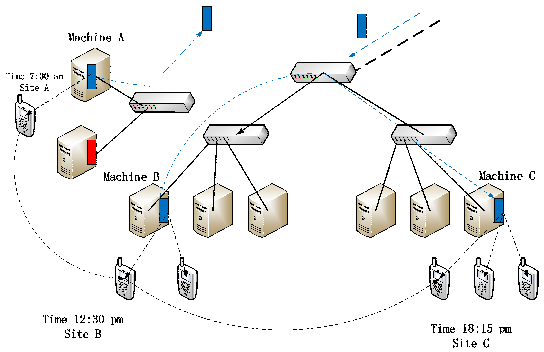


Fig. 1. An example of stream service accesses and migration. The service has two replicas hosted by respective virtual server. One of the servers is migrated from Site A to Site C via Site B to adapt to the changing locations of the dominant access loads at different time frames from 7:30am to 18:15pm.

- *Y. Wang is with the Faculty of Computer Science, University of New Brunswick, Fredericton, Canada, E3B 5A3. E-mail: ywang8@unb.ca*
- *W. Shi is with the Faculty of Business and I.T., University of Ontario Institute of Technology, Ontario, Canada, H1L 7K4. E-mail: wei.shi@uoit.ca*
- *M. Hu is with the department of Electronics and Information Engineering at Huazhong University of Science and Technology, Wuhan, China, 430074 E-mail:humenglan@gmail.com*

budget loss for the service providers. To address this problem, migrating the service on-demand to some vantage locations in the network that are close to the users could be a viable solution if the cost of migration is less than the reaped benefits to both end users and CSPs after the migration. A highly visible example to illustrate the migration benefits is a mobile stream service (see Fig. 1) which could be accessed by a crowd of devoted people on their way to work in the morning, in leisure spaces at lunch time, and as well in certain preferred sites after getting off work. Thus, depending on the changing locations of the dominant access loads, the stream server may migrate from `Site A` to `Site B` and finally to `Site C` (may be in different datacenters) at different time frames (from 7:30am to 18:15pm). This kind of service has been used as a test case in [3] based on their network virutalization prototype, and thus could be the next disruptive innovation for some entertainment media companies like Netflix [4] and Lions Gate [5].

Traditionally, achieving such benefits usually requires human intervention to manually move the residing server(s) of the requested services over a wide-area network. This is very hard and error prone, if not impossible. Conversely, virtualization technologies in the cloud make it possible to encapsulate a service in a virtual machine (VM) as a *virtual appliance*[1] and move it around in the same or across different datacenters. The movement allows a VM to be dynamically transferred across different physical servers at runtime without needing the hosted services to perceive the environment changes. We refer such a procedure to *live* or *hot* migration [7], [8], as opposed to the costly *stop-and-copy* or *cold* migration.[2] Such migration is achieved at the cost of bulk-data transfer (e.g., VM memory image and associated data files), which could incur service disruption and higher network traffic. Fortunately, advances in virtualization technologies make it feasible to migrate one or more virtual server(s) (including relevant requested services) over a wide-area network (WAN) with minimum overhead [9]–[11]. Some well-designed VM migration systems (e.g., VMFlock [12], CloudNet [13], and Shrinker [14]) have demonstrated the feasibility even efficiency of such an approach in practice. Thus, from a technical viewpoint, co-migration over a WAN is never a *prohibitively* expensive operation. This enables the provision of cloud-based services that can adapt to the relevant mobile access patterns in a timely fashion.

In this paper, our focus is not on technical solutions to efficiently move a collection of VMs across datacenters. Instead, we concentrate on the *monetary trade-off* that exists between the benefits and the costs of such a service migration in the cloud environments. More specifically, we consider migrating a service that is realized with up to κ redundant servers, each hosting a service replica, in clouds to satisfy a sequence of online or off-line batch-request demands with minimum service costs. Multiple redundant servers render the service to be not only fault tolerant but also load sharing or balancing. Given these features as the basis, the co-migration problem is particularly important for end users to enhance the user experience and cloud service providers (CSPs) to maximize the profits as well. However, to the best of our knowledge, this problem is barely studied in the literature. We notice that the most recent work in this area is [3] where the virtual service migration in a wide-area network is studied in a competitive analysis approach. However, the work is only a comprehensive extension to the authors' previous results with the single server migration as the focus. They showed that for the service migration problem (single server), there does not exist any online algorithm whose competitive ratio is smaller than $\Omega(\log n/\log\log n)$ [3], [15]. But they did not further improve the bound for multiple server co-migration in their previous results in [16].

Our goal in this paper is to propose co-migration algorithms for multiple servers that adapt to the user demands with reduced total service cost (under the assumption that VM migration over a WAN is already available at minimum cost). These algorithms work in a centralized manner to co-migrate service replicas residing on a group of virtual servers (VMs) between datacenters. As such, they can be easily integrated with any cloud service management systems [17], especially well-fitted with the infrastructure of the software defined network (SDN) [18].

We propose both online and off-line algorithms based respectively on local search and on sampling-based DP techniques. Our studies differ from previous ones (e.g., [3], [19]–[21]) that deal with single server migration using distributed approaches. In particular, because we rely on centralized control, not only can we decide to migrate (or not) the requested service (encapsulated in one or more virtual servers), but also we can offer fast system health diagnostics based on the easy tracking of the location of each server that is under migration.

For our online algorithm, we let the migration of each set of requested severs be triggered by each single mobile batch request. We tailor the network model presented in [19] to define the neighbor set for κ virtual servers and propose a randomized online co-migration algorithm *Migk* with a competitive cost of $O(\frac{\gamma\log n}{\min\{\frac{1}{\kappa},\frac{\mu}{\lambda+\mu}\}})$ under the oblivious adversary model for a n-node network. Here, λ is the maximum inter-node (wired) cost, μ is the wireless link cost, and γ is the maximal ratio of the migration costs between any pair of neighbor nodes in the network. The algorithm is efficient in the sense that it can make a migration (for serving a batch request) in linear time on average with respect to κ and n (i.e., $O(\kappa n)$). This result in certain cases improves the one introduced in [16], which is $O(\gamma\kappa^2\log n)$-competitive with a $O(\binom{n}{\kappa})$ configuration time on average.

To thoroughly study this problem and for comparison purposes, we also develop an off-line co-migration algorithm. The fundamental assumption of this algorithm is that the sequences of requests are a priori known. Our off-line algorithm is based on dynamic programming techniques where the *state* of the system is defined to be the configuration of the κ virtual

1. A virtual appliance (VA) is "a virtual machine (VM) image file consisting of a pre-configured operating system (OS) environment and a single application." quoted from [6].

2. In a *live* migration, a target VM is first stopped, then the relevant memory pages and file system blocks are copied to the destination. Finally the new VM containing the requested service(s) starts.

servers in network $G(V, E)$ where $\kappa \leq O(\frac{\log n}{\log(1+\Delta_{max})})$. In order to maintain a low configuration time, we adopt multi-round sampling techniques in the algorithm to exploit a certain subset of configurations whereby a parallelized branch&bound algorithm is designed to consistently decrease the total service time cost, while progressively updating the corresponding migration decision policies at the same time.

We validate our findings by conducting extensive simulations whose empirical results show that the proposed algorithms can adapt well to changes in mobile access patterns in order to efficiently satisfy service requests in a cost-effective way.

The remaining paper is organized as follows: we first introduce some background knowledge regarding the service migration (VM live migration in particular) before we introduce the related work in Section 2. We then describe our service migration model, together with the assumptions in Section 3. Based on this model, we propose the online co-migration algorithm in Section 4 and the off-line algorithm in Section 5, respectively. We present our empirical studies involving extensive simulations in Section 6, and and finally conclude the paper with a brief summary of our findings in the last section.

2 Background and Related Work

Live service migration, due to its merits and potentials in a variety of application contexts, has been arousing significant interest in both industry and academia since the seminal work of Clark *et al.* [7] in 2005. The research ranges across a wide spectrum, from practical viewpoint to minimize the migration overhead [7], [8], [11], [12], [14], [22]–[27] to the theoretical perspective that leverages the migration mechanism as a substrate to develop various migration strategies for different research goals [19]–[21], [28]–[32].

Most early efforts and only some recent ones focus on the VM live migration within a local network and exploit it as a typical technology to maintain the quality of cloud-based services by optimizing certain system-centric criteria such as load balancing [30], [33], fault tolerance [28], [29], resource utilization [32], [34], [35], high availability and good performance [36], and so on. Given the simplicity of the implementation, the VM live migration within a local network is relatively mature, and thus it has been supported by almost all the mainstream VMs (e.g., Xen [7] and KVM [37]) and also integrated into some major cloud management frameworks including OpenStack [38], CloudStack [39] and Eucalyptus [40], to quote a few.

However, the research landscape is quite different from that of the VM live migration over WANs where the overhead of the long-distance migration including the file-system data is the subject of the major concern as it is a prerequisite for building up advanced strategies that leverage the live migration to achieve different goals. As a result, most of current works concentrate on improving the migration efficiency over the long distance by developing some advanced technologies to take advantages of the features inherent to the migration.

R. Bradford *et al.* [9] show that when combining a block-level solution with *pre-copying* and *write throttling* strategies, an entire running web server, including its local file system, can be migrated with a minimal disruption of 3 seconds in a LAN and 68 seconds in a WAN. Later, H. Liu *et al.* [10] report that by adopting *checkpointing/recovery* and *trace/replay* technologies, a transparent VM migration for both LAN and WAN environments can be achieved in a very efficient way where the downtime for a variety of workloads (including static and dynamic web applications) could be reduced to only 200 ms (with a small standard deviation), Additionally, there are also some prototyped systems developed to further demonstrate the feasibility of migrating a single or multiple VMs in practice in Clouds [12]–[14], [41], each of them can be used as a substrate to support our migration algorithms.

A variety of migration models for diverse network contexts have been proposed in literature [19], [20], [42]. An early work from the model point of view is to consider using the service migration[3] in *autonomic network environments* [43] as a self-managing mechanism to overcome the rapidly growing complexity of the networks. Oikomomou *et al.* [20] propose a scalable algorithm to service migration in autonomic networks by observing the differential demand traffics on each link between the node hosting the service and its opposite neighbor. Although this algorithm has certain merits in service migration, it suffers from the slow convergence due to its inefficient one-hop migration per step. Pantazopoulos *et al.* [21] overcome this downside in their most recent centrality-driven migration algorithm, named *cDSMA*. However, this algorithm only targets at a single server, and lacks notion of the migration cost.

In contrast, Bienkowski *et al.* [19] consider the migration problem in the context of *virtual networks* (VNet) [44] to minimize the access latency. To this end, they present a randomized online algorithm to migrate a single server in an n-node network. and advocate the competitive analysis on the worst case of the algorithm instead of its general performance, which is also our pragmatic concern. A follow-up research to this result is from Arora *et al.* [45] who remove the randomness from the algorithm and propose a deterministic online algorithm with the same competitive ratio of $O(\log n)$ for a single server migration. In the meantime, by extending the techniques used in [19], they also obtain an $O(\gamma\kappa^2 \log n)$-competitive algorithm at the cost of a combinatorial complexity of $O(\binom{n}{\kappa})$ to migrate κ servers for a batch request, where γ follows the previous definition [16]. Most recently, they summarized their previous results into [3], but fail to further improve the competitive bound with respect to the multi-server migration.

In this paper, we adopt the model in [19] and fit it to the cloud environments whereby the co-migration problem is studied. We extend the results in [19] and achieve an efficient $O(\frac{\gamma \log n}{\min\{\frac{1}{\kappa}, \frac{\mu}{\lambda+\mu}\}})$-competitive randomized algorithm within $O(\kappa n)$ time to migrate κ servers to serve a batch request. Our algorithm is an improvement to the result in [16] which suffers from the configuration complexity of $O(\binom{n}{\kappa})$ in

3. In the following discussion we use the term *service migration* to refer to the VM live migration together with its hosted service at abstract design level instead of the underneath implementation technologies.

average time, rendering our algorithm to be more practical.

Unlike the aforementioned studies which are not directly conducted in the context of Clouds, Goudarzi and Pedram [46] implement load balancing across geographically distributed datacernters for online service applications in order to meet the service level agreements (SLAs) or service deadlines for VMs/tasks and in the meantime to decrease the operational cost of the cloud system as well. In spite of leveraging the same idea, we address a different problem with a different targeted goal from a monetary point of review, which is unique in our research.

By applying the same idea of the service migration across the Internet datacenters (IDCs), Phan *et al.* [31] develop a framework, called *Green Monster*, to address the energy efficiency issues in Cloud computing. The service migration is determined by an evolutionary multiobjective optimization algorithm which is able to strike a balance between the conflicting optimization objectives. In contrast, our problem has a single objective to minimize the total monetary cost with the service migration. Although both problems are orthogonal at the first sight, they could be connected inherently as the monetary cost could also be used to model the energy consumption as well. However, this extension is still an open problem to our algorithm.

3 Service Migration Model

We consider an arbitrary n-node network $G(V, E)$ as a service infrastructure to provide mobile services. A service has $\kappa \geq 1$ replicas, each running on a virtual machine (VM) (also called *virtual server*). The set of hosting (physical) machines (referred to as servers here after) has a *configuration*, denoted by $\mathcal{L}$, which is the specifications of this set of physical machines that are running the VMs. This set of machines are accessed by a sequence of batch requests $\sigma = \sigma_1\sigma_2...\sigma_m$ issued from a set of external devices (i.e., mobile terminals). In the online migration scenario, the requests arrive one at a time. Each of such request is satisfied by triggering the migration of a set of servers in $\mathcal{L}$ into an ideal location. On the other hand, in the off-line migration scenario a migration is not triggered until a sequence of requests are received, namely, the whole access pattern has been detected and known in advance. It not only benefits the scenarios to proactively schedule the service to facilitate the mobile accesses but also provides a metric to measure the performance of its online counterpart.

During a migration, the service moves through a subset of the virtual servers over time in a live fashion to maximize the efficiency [7], [8]. The required resources for the migration on the target machines are always assumed to be available. This can be achieve by reservation [47] or pre-configuration of the machines with sufficient resources. We do not distinguish the active and inactive servers like in [16] and all the servers in our model are active. We denote the configuration of a subset of servers at t_i as $\mathcal{L}_i$. Some frequently used symbols in this and subsequent sections are listed in Table 1 for quick references.

Each specific request is routed to the service over a wireless link first to connect the network via a *connect point* and then based on some algorithms or metrics to reach a service. We denote the connection cost (monetary) as μ and the transmission cost (monetary) between any pair of nodes u and v as C_{uv}. According to the charge models of the most current cloud infrastructure services, it is reasonable to assume that both these two types of costs are available from the infrastructure service providers (ISPs) priory to the overlying cloud service providers (CSPs). Therefore, a batch request σ_i at time t_i can be represented by $\sigma_i = \cup_j\{(a_{ij}, \sigma_{ij})\}$, here a_{ij} and σ_{ij} is a sub-request of σ_i sent to the network via connect point a_{ij}. In other words, $\cup_j\sigma_{ij}$ represents the entire set of requests of σ_i. Fig. 2 is an example of this model where batch request σ_i contains four sub-requests $\sigma_i = \{(a_{i1}, R_{i1}), ..., (a_{i4}, R_{i4})\}$. There are two server replicas located at r and u, and one of them moves from u to v during t_{i-1} and t_i to satisfy σ_i for total cost reduction, that is $\mathcal{L}_{i-1} = \{r, u\}$ and $\mathcal{L}_i = \{r, v\}$.

TABLE 1
Notation frequently used in model and algorithm descriptions

Symbol	Meaning
n	the number of nodes in the network
Δ_{max}	the maximum node degree
m	the length of a requests sequence
C_{uv}	transmission cost between node u and v
C_v	$C_v = \{C_{uv} \vert \forall u \in V \text{ as a connect point}\}$
λ	$\lambda = \max_{u,v\in V}\{C_{uv}\}$
β_{uv}	migration cost between node u and v
β_v	$\beta_v = \{\beta_{vu} \vert u \in \mathcal{N}(v)\}$
β	$\beta = \max_{u,v\in V}\{\beta_{uv}\}$
β'	$\beta' = \min_{u,v\in V}\{\beta_{uv}\}$
γ	$\gamma = \beta/\beta'$
μ	wireless link cost
κ	the number of service replicas
σ	the request sequence $\sigma = \sigma_1\sigma_2...\sigma_m$.
$\sigma_{\mathcal{E}}$	the total served sequence in epoch $\mathcal{E}$
σ_i	the ith request $\sigma_i = \cup_j\{(a_{ij}, \sigma_{ij})\}$
a_{ij}	the access point of the jth request in σ_i
a_r	the access point of request r
σ_{ij}	the jth request in σ_i
$\mathcal{L}_i$	the configuration at time i
$\mathcal{N}(v)$	neighbor nodes of node v
$\phi(r)$	r's service node determined by routing func ϕ
s_i^v	service s_i at node v
$w(s_i^v)$	access counter of s_i at node v
$d(s_i^v)$	profile recorder of s_i at node v

Obviously, in order to satisfy σ_i, each request $r \in \sigma_i$ will be eventually routed to certain $h \in \mathcal{L}_i$ via the closest $a_r \in V$. This is typically achieved by the underlying *routing function* determined by ISPs. As a result, the total monetary cost of access (hereafter *access cost*) of batch request σ_i can be simply calculated as the total connection costs plus transmission costs of all the requests in σ_i along the routing path, which can be expressed as $Cost_{acc}(\mathcal{L}, \sigma_i) = |\sigma_i|\mu + \sum_{r\in\sigma_i} C_{a_r\phi(r)}, \phi(r) \in \mathcal{L}$, where a_r is request r's nearest connect point and $\phi(r)$ is r's service node determined by the routing function $\phi(.)$. Note that in this equation we implicitly assume that the sizes of requests are so small that the network bandwidth is never the bottleneck for their transmissions, rather, the link latency is the issue. Clearly, this cost is affected by the level of network latency.

In contrast to the requests, which are rather light-weight, the

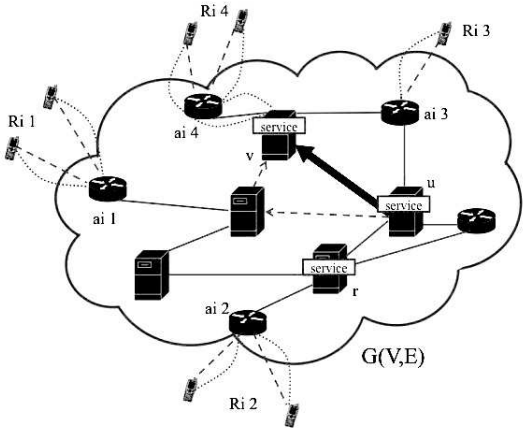


Fig. 2. An example of service access and migration model where two service replicas are located at server u and r, respectively, and a_is represent the access points at time t_i. In the figure, a server is migrated from u to v to adapt to the requests at t_i (dashed lines: wireless Connections; dotted lines: Service requests at t_i, represented by R_is; arrowed lines: planned server migration route; arrowed dashed lines: actual service migration route).

traffic volume of migrating services is usually not negligible due to the large size of service states. Unlike the access cost which is dominated by the access latency, the migration cost (monetary) $Cost_{mig}$ of the service depends greatly on the service size and available bandwidth on the migration path. Therefore, we assign node v with a migration cost set $\beta_v = \{\beta_{vu} | u \in \mathcal{N}(v)\}$ (clearly, $\beta_{vv} = 0$) to reflect the server migration costs from v to the corresponding target $u, u \in \mathcal{N}(v)$ where $\mathcal{N}(v)$ represents the neighbor set of v. Particularly, for any u and v in G we denote $\beta = \max_{(u,v)\in E}\{\beta_{uv}\}$, $\beta' = \min_{(u,v)\in E}\{\beta_{uv}\}$, and $\gamma = \beta/\beta'$ for later discussion. Again, β_v is also given by the ISPs in advance for each node v in the network.

To minimize the access cost, we need to identify a *matching function* π that can figure out the migration target server in $\mathcal{L}_i$ for each server in $\mathcal{L}_{i-1}$. To this end, we denote $\mathcal{L}_{i-1} = \{u_1, u_2, ..., u_\kappa\}$ and $\mathcal{L}_i = \{v_1, v_2, ..., v_\kappa\}$ and have $Cost_{mig}(\mathcal{L}_{i-1}, \mathcal{L}_i) = \min_{\pi}\{\sum \beta_{u_j v_{\pi(j)}}\}$ where π is the permutation of $\{1, 2, ..., \kappa\}$. $Cost_{mig}(\mathcal{L}_{i-1}, \mathcal{L}_i)$ represents the minimum migration cost from $\mathcal{L}_{i-1}$ to $\mathcal{L}_i$.

Therefore, for a sequence of batch requests $\sigma = \sigma_1\sigma_2...\sigma_m$, the goal of the service migration is to determine $\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_t$ (t is to be determined) to minimize the total service cost defined as $Cost(\mathcal{L}_i) = Cost(\mathcal{L}_{i-1}) + Cost_{acc}(\mathcal{L}_{i-1}, \sigma_i) + Cost_{mig}(\mathcal{L}_{i-1}, \mathcal{L}_i)$, given $Cost(\mathcal{L}_0) = 0$. This recurrence indicates that the total cost to satisfy σ_i is equal to the total cost of satisfying the first $i-1$ requests plus the access cost in configuration $\mathcal{L}_{i-1}$ and then the migration cost from $\mathcal{L}_{i-1}$ to $\mathcal{L}_i$.

Although this model is more amenable to inter-datacenter migration, it could be also applied to the intra-datacenter migration even if the intra-datacenter network latency is low and the bandwidth is usually high. Note that we do not model the workloads of the target servers and the runtime cost of the service as these factors, though important in determining the service migration, can be neglected in the model without having impact on our proposed algorithms. Furthermore, with advance of the technologies in live migration as we showed (e.g., [10]), the service disruption (aka service downtime) can be also ignored in our model.

4 Online Co-Migration Algorithm

In this section, we present our online co-migration algorithm which dynamically migrate κ server replicas as a whole to adapt to the incoming requests in a time sequence order without any knowledge of the complete access patterns. We first present the basic idea of the algorithm and then detail it in a more formal way.

4.1 Basic Idea

The algorithm divides the time into epochs, and compare the cost of our algorithm with that of the optimal algorithm on a per-epoch basis. An epoch consists of one or multiple phases between which *Migk* migrates at least one server.[4] An epoch ends up with the condition that no servers can be migrated. In that case a new epoch starts by resetting related data structures.

The algorithm handles each server individually to service each incoming request by gathering the request and accumulating its access cost in related data structures. When the accumulated access cost reaches the migration cost β', an migration decision is made by selecting the target node. Instead of searching migration target in one-hop neighborhoods, we adopt the strategy in [19] where the target node is selected randomly from those nodes that have not been selected before and whose total access cost to the collected requests gathered at the source server node is less than β'. If such a node exists, the server will be migrated to that node. Otherwise, the server will be kept stationary. Clearly, for a κ-server configuration, the corresponding selected targets all together define a local neighbor of the source configuration for the next co-migration step. The rationale behind this definition is twofold. On one hand, the definition is easy for competitive analysis to ensure the performance of the algorithm has an upper bound in the worst case. On the other hand, for multiple servers, this definition can help jump out from the local optimal configurations in the course of the search. When no server can be migrated, the current epoch is terminated and the algorithm is reset to start a new one again.

4.2 *Migk*: An Online Co-Migration Algorithm

Suppose the κ-server set $S = \{s_1, s_2, ..., s_\kappa\}$ is initially located at $\mathcal{L}$, and let $X - x = X \setminus \{x\}$ and $X + x = X \cup \{x\}$, we have the algorithm working as follows:

4. A batch request could trigger multiple servers to co-migrate but no server is migrated more than once between consecutive phases. The migration is only determined after each batch request has been served (at most κ servers are migrated).

1) In each epoch $\mathcal{E}$, for each request $r \in \sigma$, the algorithm first computes its service server $\phi(r) = s_i^v$ (i.e., server s_i at node $v \in \mathcal{L}$, $1 \leq i \leq \kappa$) as well as the corresponding access cost $C_r(s_i^v) = \mu + C_{a_r s_i^v}$ and then adds it to a counter of s_i^v denoted as $w(s_i^v)$ which is initialized to zero. In addition, the algorithm also records the request set for s_i^v denoted as $d(s_i^v)$, which is initially empty. Therefore, $w(s_i^v) = \sum_{r \in d(s_i^v)} C_r(s_i^v)$. In general, for each server s_j, $1 \leq j \leq \kappa$, we have a pair of $w(s_j^v)$ and $d(s_j^v)$ that satisfies the aforementioned conditions for each $v \in \mathcal{L}$. We call them *access counter* and *profile recorder*, respectively.
2) The κ servers are fixed to serve request r that is routed to $s_i^v \in S$ with the following updates:

$$\begin{cases} w(s_i^v) & = w(s_i^v) + C_r(s_i^v) \\ d(s_i^v) & = d(s_i^v) + r \end{cases} \quad (1)$$

until there is a non-empty co-migration subset $M \subseteq \mathcal{L}$ where $M = \{s_i^v | v \in \mathcal{L}, w(s_i^v) \geq \beta'\}$. At this time t, a new phase starts, the algorithm selects *all* the server $s_i^v \in M$ to move for serving the next batch request.
3) For each $s_i^v \in M$, the algorithm does the following:
   a) compute $w(s_i^{v'})$ for each $v' \in V \setminus \mathcal{L}$ that has not been visited by s_i, which is the cost to serve the requests in $d(s_i^v)$ at the corresponding target node v'[5];
   b) prune away those nodes $v' \in V \setminus \mathcal{L}$ whose $w(s_i^{v'}) \geq \beta'$ by marking them visited by s_i;
   c) identify a node u chosen uniformly among the remaining nodes with the property $w(s_i^u) < \beta'$;
   d) if u exists, migrate the server at v to u, mark v as being visited, update $\mathcal{L} = \mathcal{L} - v + u$ and set $d(s_i^u) = d(s_i^v)$ (note that $w(s_i^u) < \beta'$). Otherwise, if no such u can be found, *Migk* does not migrate server s_i^v (s_i stops at v) and keeps its $w(s_i^v)$.

   This co-migration procedure is repeated until all servers in M have been processed, then resets $M = \varnothing$ and goes to Step 4 for checking the epoch termination.
4) If $\sum_{v \in \mathcal{L}} w(s_i^v) \geq \kappa\beta'$, the epoch $\mathcal{E}$ ends in that round (every server stops moving) and goes to Step 5. Otherwise, goes to Step 2 for the next new phase.
5) The next epoch starts in the next round from the current stop configuration and the counters $w(s_i^v)$ and $d(s_i^v), v \in V, 1 \leq i \leq \kappa$, are reset to zero and empty, respectively. The entire request sequence served in epoch $\mathcal{E}$ is denoted as $\sigma_{\mathcal{E}}$.

Remarks: Performance anomaly could happen when $\beta' = 0$, i.e., the migration cost is free as in this case no node in each migration can be pruned away and thus the effective epoch cannot be created. On the other hand, although we describe the algorithm in a centralized manner, it is not difficult to transform the algorithm to work in distributed fashion. Therefore, our algorithm is feasible enough.

5. here we assume to move server s_i from v to v' to continue gathering the requests.

4.3 Analysis

Lemma 4.1: The expected number of phases within one epoch is at most κH_n where H_n is the n-th harmonic number where $n = |V|$.

Proof: Based on the results in [19], we can easily conclude the lemma by knowing that the expected number of migrations of each server $s_i \in S$ to serve $d(s_i^v)$ within one epoch is at most H_n where v is the stop location of s_i at the end of the epoch. □

Theorem 4.2: Migk is $O(\frac{\gamma \log n}{\min\{\frac{1}{\kappa}, \frac{\mu}{\lambda+\mu}\}})$-competitive.

Proof: First, we consider the cost of the optimal off-line algorithm. If *Opt* co-migrates in an epoch $\mathcal{E}$, it has cost β' (at least one server is migrated). Otherwise, we try to find a lower bound for the access cost of *Opt* in $\mathcal{E}$ where κ servers are fixed at $\mathcal{L}$. Obviously, according to *Migk*, we have *Opt*($\mathcal{E}$)$\geq |\sigma_{\mathcal{E}}|\mu$. Suppose at the end of $\mathcal{E}$, κ servers stop at T, we have $\sum_{v \in T} w(s_i^v) = \sum_{v \in T} \sum_{r \in d(s_i^v)} C_r(s_i^v) \geq \kappa\beta'$, then we can estimate $|\sigma_{\mathcal{E}}| = \sum_{v \in T} |d(s_i^v)|$ by lower bounding it as:

$$\begin{aligned} & \sum_{v \in T} \sum_{r \in d(s_i^v)} (\mu + C_{a_r \phi(r)}) \geq \kappa\beta' \\ \Rightarrow & \sum_{v \in T} \sum_{r \in d(s_i^v)} (\mu + \lambda) \geq \kappa\beta' \\ \Rightarrow & \sum_{v \in T} (\mu + \lambda) |d(s_i^v)| \geq \kappa\beta' \\ \Rightarrow & (\mu + \lambda) \sum_{v \in T} |d(s_i^v)| \geq \kappa\beta' \\ \Rightarrow & |\sigma_{\mathcal{E}}| \geq \frac{\kappa}{\mu + \lambda} \beta' \end{aligned} \quad (2)$$

So we have *Opt*($\mathcal{E}$)$\geq \frac{\kappa\mu}{\lambda+\mu}\beta'$. Overall, *Opt*($\mathcal{E}$) $\geq \min\{\beta', \frac{\kappa\mu}{\mu+\lambda}\beta'\}$.

For *Migk*, based on Lemma 4.1, we have *Migk*($\mathcal{E}$)$\leq \beta \sum_{v \in T} \#mig(s_i^v, d(s_i^v)) \leq \kappa\beta \cdot O(\log n)$ where $\#mig(s_i^v, d(s_i^v))$ is the number of migrations for server s_i^v within the period of $\mathcal{E}$ to satisfy $d(s_i^v)$.[6] Finally we conclude

$$Migk(\mathcal{E}) \leq O(\frac{\gamma \log n}{\min\{\frac{1}{\kappa}, \frac{\mu}{\lambda+\mu}\}}) Opt(\mathcal{E}). \quad (3)$$

□

Theorem 4.2 shows how the competitive ratio of the algorithm relies on the charging model of the ISPs. Given $\kappa \geq 1 + \frac{\lambda}{\mu}$, *Migk* could be $O(\gamma\kappa \log n)$ competitive, a κ times better than the result in [16]. However, the result is not constant better than that in [16]. In particular, when $\kappa < 1 + \frac{\lambda}{\mu}$, the result can be simplified as $O(\gamma(1 + \frac{\lambda}{\mu}) \log n)$, which, depending on the ratio of λ/μ, could be better or not than $O(\gamma\kappa^2 \log n)$. For example, when $\lambda = \mu$, there is only a single migrated server (i.e., $\kappa = 1$) in this case, we thus have the competitive ratio with the same order of [16] (i.e., $O(\gamma \log n)$).

More importantly, for all the cases, our algorithm is much more efficient, and thus practical in reality as we remove the configuration complexity for migration search space in [16]. We achieve these merits by considering the migration of each

6. The cost order is not changed if we also consider the access cost.

server independently and coordinating them as a whole via a global phase control.

Theorem 4.3: The expected time complexity of *Migk*($\mathcal{E}$) for epoch $\mathcal{E}$ is at most $O(\kappa n|\sigma_{\mathcal{E}}|)$ where $n = |V|$.

Proof: According to Lemma 4.1, each server is expected to move at most H_n times within one epoch, and for each movement, four kinds of costs are incurred from 3(a) to 3(d). According to the algorithm, the counter value of each server is monotonically increased with respect to each visited node. Thus, for a particular server, each node in G is visited only once to see if the counter value associated with the visiting server at that node is not less than β', and at the end of H_n phases (i.e., one epoch), all n nodes have been visited, and for each visit, at most $|\sigma_{\mathcal{E}}|$ requests are used to compute the counter value at that visited node. Then the cost of 3(a) for each server in one epoch is at most $n|\sigma_{\mathcal{E}}|$. 3(b) can be computed in conjunction with 3(a) at constant cost, and 3(c) is assumed to be a constant cost operation. Finally, for configuration updates in 3(d), we can also achieve a constant cost operation depending on the data structure for organizing the configuration (e.g., hashtable). Overall, for at most κ servers in M, the total cost in Step 3 is in the order of $O(\kappa n|\sigma_{\mathcal{E}}|)$. Obviously, given the precomputed all pairs of shortest paths, the cost for servicing each request in Step 2 is $O(1)$. In Step 5, the reset operation takes $O(\kappa)$ time. Therefore the total time complexity of *Migk* is $O(1 + \kappa + \kappa n|\sigma_{\mathcal{E}}|) = O(\kappa n|\sigma_{\mathcal{E}}|)$. In other words, on average the algorithm takes $O(\kappa n)$ time to serve a request. □

5 Off-line Co-Migration Algorithm

In this section, we study the co-migration problem in its off-line form with the assumption that the whole access pattern has been detected and known in advance. Therefore, the migration can be figured out with the complete knowledge of the access requests, which would exhibit some monetary advantages (less monetary cost) over its online counterpart. The off-line study not only benefits the scenarios when the access patterns could be predicted but also provides a metric to measure the performance of the online algorithm. We first briefly discuss the optimal algorithm based on dynamic programming (DP) technique, and then propose a multithreaded branch&bound algorithm, which is built on the sampling method, to simplify and speed up the DP computation.

5.1 Optimal DP Algorithm

OPT exploits the optimal sub-structure property of this problem to construct the DP recurrence in Equation (4) where the minimum-cost solution $w_i(\mathcal{L}_i^j)$ at t_i can be obtained from the minimum-cost solutions to *all* the sub-problems of $\mathcal{L}' \in \mathbb{C}_{i-1}$ at t_{i-1} given sever configuration $\mathcal{L}_i^j$ (there could be $\binom{n}{\kappa}$ possible configurations) for $1 \le j \le \binom{n}{\kappa}$,

$$w_i(\mathcal{L}_i^j) = \min_{\mathcal{L}' \in \mathbb{C}_{i-1}} \{w_{i-1}(\mathcal{L}') + Cost_{acc}(\mathcal{L}', \sigma_i) + Cost_{mig}(\mathcal{L}', \mathcal{L}_i^j)\} \tag{4}$$

where $\mathbb{C}_{i-1} = \{\mathcal{L}_{i-1}^1, ..., \mathcal{L}_{i-1}^{\binom{n}{\kappa}}\}$ is the set of all the possible server configurations at t_{i-1} and σ_i is the batch request at t_i. Initially, $\mathcal{L}_0^1 = \{v_1, v_2, ..., v_\kappa\}$ and $w_0(\mathcal{L}_0^1) = 0$. For any other configuration $\mathcal{L}_0^j$, $w_0(\mathcal{L}_0^1, \mathcal{L}_0^j) = Cost_{mig}(\mathcal{L}_0^1, \mathcal{L}_0^j), 2 \le j \le \binom{n}{\kappa}$, and the optimal solution is then given by,

$$w^*(\mathcal{L}^*) = \min_{1 \le j \le \binom{n}{\kappa}} \{w_m(\mathcal{L}_m^j)\} \tag{5}$$

the optimal migration strategies can be reconstructed from $\mathcal{L}_n$ by recursively backtracking the optimal configuration $\mathcal{L}'$ at t_{i-1} which steers to the optimal configuration $\mathcal{L}_i^j$ at t_i.

5.2 Multithreaded Branch&Bound Algorithm

It is easy to see that handling the large configuration space and minimizing $w_n(\mathcal{L}_n)$ in the DP algorithm to find the optimal migrations is an overly complex task. However, the DP algorithm is still valuable as its recurrence matrix contains a lot of viable, and some are relatively good solutions although not optimal. Therefore, it is worthwhile to find or approximate them in an efficient way. In the sequel, we adopt a sampling method with local search on the DP matrix whereby designing a multithreaded branch&bound algorithm to overcome the complexity and solve this problem under a certain condition.

5.2.1 Sampling Method with Local Search

This method is based on DP Recurrence (4) to handle the configuration complexity by repeatedly sampling the configuration space $\mathbb{C}$ to obtain the initial κ *random* configurations, and then updating them with local search according to a revised recurrence of (4). Specifically, suppose the selected $\kappa + 1$ configurations are $\mathcal{L}_0^0, \mathcal{L}_0^1, ..., \mathcal{L}_0^\kappa$, and the κ servers are initially located at $\mathcal{L}_0^0$, then for a request sequence $\sigma = \sigma_1\sigma_2...\sigma_m$, the DP matrix in (4) can be simplified as

$$\begin{pmatrix} w_0(\mathcal{L}_0^0) & w_0(\mathcal{L}_0^1) & ... & w_0(\mathcal{L}_0^\kappa) \\ w_1(\mathcal{L}_1^0) & w_1(\mathcal{L}_1^1) & ... & w_1(\mathcal{L}_1^\kappa) \\ ... & ... & ... & \\ w_m(\mathcal{L}_m^0) & w_m(\mathcal{L}_m^1) & ... & w_m(\mathcal{L}_m^\kappa) \end{pmatrix} \tag{6}$$

where $w_0(\mathcal{L}_0^0) = 0$, and for $1 \le l \le \kappa$, we have $w_0(\mathcal{L}_0^l) = Cost_{mig}(\mathcal{L}_0^0, \mathcal{L}_0^l), l \ge 1$ and for $1 \le i \le m, 0 \le l \le \kappa$,

$$w_i(\mathcal{L}_i^l) = \min_{\substack{0 \le j \le \kappa \\ X_{i-1}^j \subset \mathcal{L}_{i-1}^j \\ Z_i^l = \mathcal{N}(X_{i-1}^j)}} \{w_{i-1}(\mathcal{L}_{i-1}^j) + Cost_{acc}(\mathcal{L}_i^j, \sigma_i) + Cost_{mig}(X_{i-1}^j, Z_i^l)\} \tag{7}$$

among which $\mathcal{L}_i^l$ is constructed by some $\mathcal{L}_{i-1}^j, 0 \le j \le \kappa$ who migrates exactly l servers to reach $\mathcal{L}_i^l$ while minimizing (7), i.e., $\mathcal{L}_i^l = \mathcal{L}_{i-1}^j - X_{i-1}^j + Z_i^l$, $|X_{i-1}^j| = |Z_i^l| = l$. This recurrence indicates that $w_i(\mathcal{L}_i^l)$ is computed from the minimum configuration of $\mathcal{L}_{i-1}^0, \mathcal{L}_{i-1}^1, ..., \mathcal{L}_{i-1}^\kappa$ when each moves exactly l severs. We develop this strategy to consider some neighborhoods of each selected configuration via local search so that the sampling can cover more DP state space.

Similar to (5), the final suboptimal result of this algorithm can be found at

$$w^*(\mathcal{L}^*) = \min_{0 \le j \le \kappa} \{w_m(\mathcal{L}_m^j)\} \tag{8}$$

Note that with local search, for a particular column the configuration of each row in the DP matrix (6) is varied

with respect to the selection of the minimum configuration in the last step, which is different from the standard DP matrix determined by Recurrence (4) where the configuration of each row for a particular column is fixed. We thus call this algorithm *Sampling-based DP* algorithm (SDP).

The migration scheme can be reconstructed by starting from $\mathcal{L}^*$ to repeatedly backtrack matrix (6) to $t = 0$. For example, if $w_5(\mathcal{L}_5^\kappa)$ is the minimal result for $\sigma = \sigma_1, ..., \sigma_5$, we can backtrack which one at $t = 4$ leads to it, say $w_4(\mathcal{L}_4^3)$. This indicates that migrating κ servers from $\mathcal{L}_4^3$ to reach the final result. Similarly, from $\mathcal{L}_4^3$, we can see who migrate 3 servers to result in it. To enable this backtracking procedure to work, we have to keep the related information in the matrix.

For the time complexity of this algorithm, we have the following result,

Theorem 5.1: Given $\kappa \leq \frac{\log n}{\log(1+\Delta_{max})}$, the algorithm has the time complexity of $O(m(\kappa n)^2)$ to serve the request sequence σ with a length of m.

Proof: Since in each row i, $w_i(\mathcal{L}_i^l)$ is computed by selecting the minimum out of the costs to migrate exactly l servers, each from $\mathcal{L}_{i-1}^0, \mathcal{L}_{i-1}^1, ..., \mathcal{L}_{i-1}^\kappa$, the time complexity to compute a row can be written as

$$O(\kappa \sum_{l=0}^{\kappa} \binom{\kappa}{l} \Delta_{max}^l (l + \kappa|\sigma_i|) \leq O(\kappa^2 (1+\Delta_{max})^\kappa |\sigma_i|) = O(n\kappa^2|\sigma_i|) \quad (9)$$

given $\kappa \leq \frac{\log n}{\log(1+\Delta_{max})}$. As in general $|\sigma_i| \leq n$, the complexity can be simplified as $O(n^2\kappa^2)$. Therefore, the total time to serve the request sequence σ with a length of m is at most $O(m(\kappa n)^2)$. □

As for the worst case of the algorithm, its performance can be upper bounded by the following results, regardless of the local search.

Lemma 5.2: The off-line algorithm serves batch request $\sigma_i \in \sigma$ by migrating at most $|\sigma_i|$ servers from $\mathcal{L}_{i-1}$ to $\mathcal{L}_i$ if $|\sigma_i| \leq \kappa$.

Proof: According to the algorithm's Equation (7), after serving a request, the server's configuration is updated by proactively migrating some servers to some nodes for the next request. As the storage cost is not countered, this effect can be equivalently achieved by delaying the proactive migration until the request is actually made to this server at some time step afterwards. Therefore, given $|\sigma_i| \leq \kappa$, there are at most $|\sigma_i|$ server migrations. □

Theorem 5.3: The off-line algorithm in its worst case is within $1 + \frac{\lambda+\beta}{\mu}$ times of the optimal result.

Proof: Clearly, for any $\sigma_i \in \sigma, 1 \leq i \leq m$, we have $OPT(\sigma_i) \geq \mu|\sigma_i|$. On the other hand, the cost of the SDP algorithm is less than or equal to $(\lambda + \mu)|\sigma_i| + l_i\beta$ where l_i is the number of servers that are triggered by σ_i to migrate. Obviously, if $|\sigma_i| \leq \kappa$, due to Lemma 5.2, there are at most $|\sigma_i|$ servers that migrate. Otherwise, there are at most κ servers to migrate. In either case $l_i \leq |\sigma_i|$, we have

$$\frac{SDP}{OPT} \leq \frac{(\lambda+\mu)\sum_{i=1}^m |\sigma_i| + \sum_{i=1}^m l_i\beta}{\mu \sum_{i=1}^m |\sigma_i|} \leq \frac{(\lambda+\mu+\beta)\sum_{i=1}^m |\sigma_i|}{\mu \sum_{i=1}^m |\sigma_i|} = 1 + \frac{\lambda+\beta}{\mu}. \quad (10)$$

Then, $SDP \leq (1 + \frac{\lambda+\beta}{\mu})OPT$. □

Remarks: Unlike the online scenario, this ratio is independent of the number of servers and the network size. In particular, this theorem shows that the performance of the non-migration algorithm is away from the optimal migration version at most $1 + \frac{\lambda}{\mu}$ times.

Algorithm 1 Multithreaded Branch&Bound algorithm (BnB)

```
1: procedure BRANCH&BOUND(N_s, N_t, κ, n, σ)
2:     ▷ N_s: # of samples, N_t: # of threads, σ: request seq.
3:     Bound : B ← ∞
4:     Number : N_l = N_s/N_t
5:     while N_s ≥ 0 do
6:         Do PARALLEL
7:             ▷ k ∈ [1...N_l], # of generated samples in SDP
8:                 ▷ there are ⌈N_l/k⌉ rounds for each SDP
9:         N_s ← N_s − k
10:        call SDP(&B, k, κ, n, σ)
11:        END PARALLEL
12:    end while
13:    output(B)                ▷ B stores the final cost value
14: end procedure
```

5.2.2 Algorithm Description

We can repeat the SDP algorithm by multiple rounds, each with κ initial configurations plus $\mathcal{L}_0^0$ to progressively improve the quality of the results. To this end, we design a branch&bound (BnB) algorithm which is more effectively than the simple multi-round approach to reach a good result. Since the SDP algorithm can select the samples (i.e., κ configurations) independently, we can allow multiple SDP instances to execute in parallel on a multicore platform via multithreading.

Intuitively, based on the given number of the samples, the BnB algorithm can generate all the samples at once and store them in a central queue. The SDP threads organized in a thread pool then pick up the samples from the queue, one at a time to compute the total service cost of that selected sample. The so far best (lowest) upper bound of the cost is stored in a global variable, which is used to cut off those branches whose lower bounds will be greater than the upper bound. The stored upper bound is updated whenever a new lower upper bound is found in some SDP instance. Although this algorithm is simple and can achieve load balancing to a great extent, it is not that effective as a large number of nodes in the BnB tree will be generated and examined.

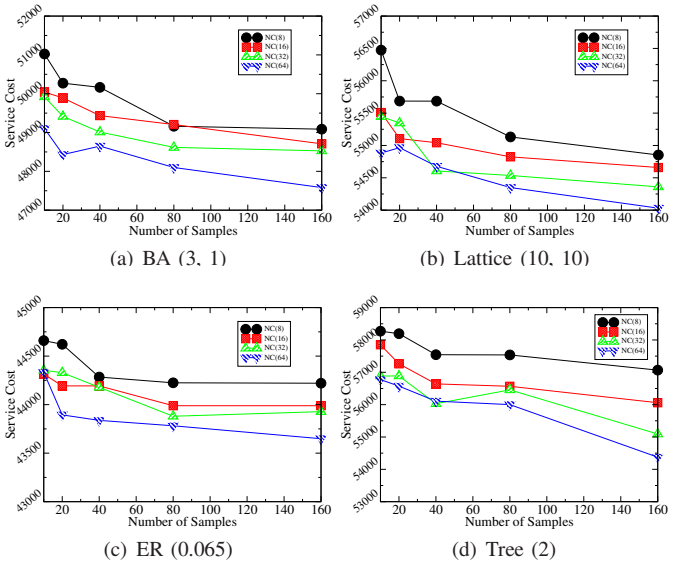
To address this problem, we allow each SDP thread in the pool to be assigned equally $N_l = N_s/N_t$ samples in

Algorithm 2 Sampling-based DP algorithm (SDP)

1: **procedure** SDP(B, k, κ, n, σ)
2: $PriorityQ : Q$
3: $Sample : u$
4: ▷ Generate k samples and put into Q, $\sigma = \sigma_1, ...\sigma_m$
5: **for** $j \in [1, ..., k]$ **do**
6: $Sample : s^j \leftarrow \mathbf{RandSample}(\kappa, n)$
7: $s^j.c \leftarrow \mu \sum_{i=1}^{|\sigma|} |\sigma_i|$ ▷ the lower bound of cost
8: $s^j.t \leftarrow 1$ ▷ σ_t is the next req to be served
9: $Q.enq(s^j)$
10: **end for**
11: **while** $Q \neq \varnothing$ **do**
12: $u \leftarrow Q.deq()$
13: **if** $u.t = |\sigma| + 1$ **then**
14: ▷ a new upper bound found
15: **if** $u.c < B$ **then**
16: $B \leftarrow u.c$ ▷ update the best bound
17: **end if**
18: **else** ▷ σ_t is served in u
19: ▷ w_t^u is from (7) for u
20: ▷ $u.c$ is a lower bound of u
21: $u.c \leftarrow \min_{0 \le j \le \kappa}\{w_t^u(\mathcal{L}_t^j)\} + \mu \sum_{i=t+1}^{|\sigma|} |\sigma_i|$
22: $u.t \leftarrow t + 1$
23: **if** $u.c < B$ **then**
24: $Q.enq(u)$ ▷ put back u
25: **end if**
26: **end if**
27: **end while**
28: **end procedure**

TABLE 2
Average inter-node distances and node degrees of different networks with 100 nodes used in the experiments

Prof.	BA(3,1)	Lattice(10,10)	ER(0.065)	Tree(2)
Dist	2.6	6.67	2.25	6.29
Deg	5.44	3.6	9.74	1.98



(a) BA (3, 1) (b) Lattice (10, 10)

(c) ER (0.065) (d) Tree (2)

Fig. 3. Performance changes of *BnB* with respect to the number of samples and the number of configurations (NC) in each sample. The migration costs of four server replicas are uniformly distributed in [0, 2000].

$\lceil N_l/k \rceil$ rounds, each having k samples instead of one at a time, here, N_s and N_t are the numbers of the samples and threads respectively. The proposed BnB algorithm is shown in Algorithm 1.

Algorithm 2 illustrates the SDP algorithm. As described, there are k samples that are locally generated in each SDP thread. Due to the large sample space, the probability of any two samples (in the same or different threads) that are not the same is as high as $1 - \kappa!/\binom{n}{\kappa}^{\kappa}$. Therefore, the repeated computation in the algorithm can be ignored.

In the algorithm, each SDP thread will handle k samples, each having κ randomly selected configurations plus $\mathcal{L}_0$. All the k samples, each being initialized with its lower bound of service cost to serve σ (Line 7) and the next request in σ to be served (Line 8), are organized in a priority queue with the minimum at the top (Line 5-10). Each time, the one at the top is dequeued and used to serve a request in σ (Line 11-12).

For a particular sample u, the algorithm first checks if $u.t = |\sigma|+1$. If so, the whole sequence σ has been served by u, and a new upper bound is found. If this value is smaller than B, B will be updated with the new upper bound (Line 13-17).

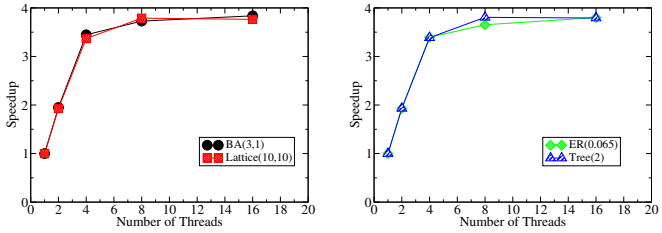
Otherwise, if $u.t < |\sigma| + 1$, its priority value is computed as a lower bound of its total service cost after satisfying σ_t:

$$u.c = \min_{0 \le j \le \kappa}\{w_t^u(\mathcal{L}_t^j)\} + \mu \sum_{i=t+1}^{|\sigma|} |\sigma_i| \quad (11)$$

here, the first part of Equation (11) is the minimum cost to serve the request sequence up to time t (i.e., σ_t) while the second part is a lower bound of the cost to serve the remaining requests[7] (Line 18-22). The priority of each sample is updated whenever a request is served. The updated priority will be compared with the globally best bound B. If it is lower than B, u as a promising sample will be put back to Q again for future examination with respect to the remaining requests in σ. Otherwise, u is discarded and the next one is dequeued (Line 23-26).

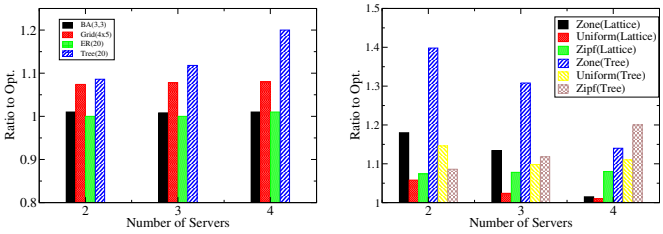
Arguably, the artifact of selecting the samples (the number and configurations) could be further improved according to the access patterns. However, we adopt the random sampling because this strategy is independent of the access distribution, and can eventually adapt to the distribution when the sample space selected by the multiple rounds is significantly large.

7. $\min_{0 \le j \le \kappa}\{w_t^u(\mathcal{L}_t^j)\}$ is computed by Matrix (6) of the sample u.



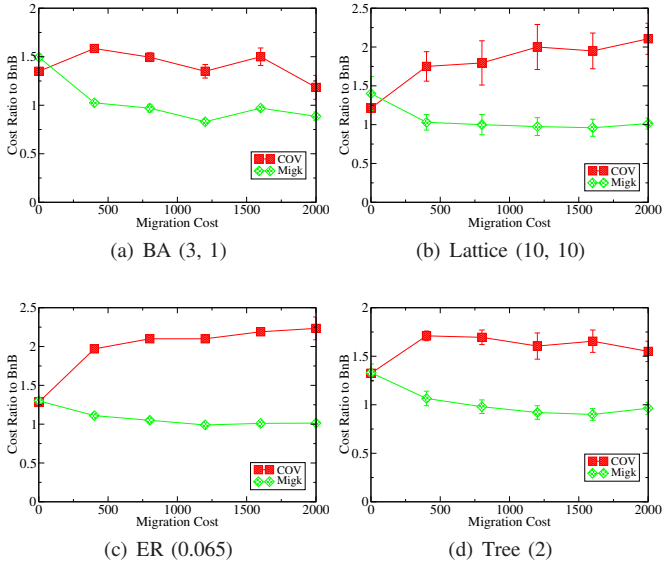
(a) BA(3,3) and Lattice(10,10) (b) ER(0.065) and Tree(2)

Fig. 4. Speedup of *Migk* when the number of threads is varied from 1 to 16 for Zipf-like access pattern on all studied networks (Samples=160, configurations=32).



(a) Zipf-like Access Pattern in Network (Size≤20) (b) All Studied Access Patterns in Network (Size≤100)

Fig. 5. Performance ratios of BnB over the optimal DP algorithm when both the network sizes and the number of servers are limited.



(a) BA (3, 1) (b) Lattice (10, 10)

(c) ER (0.065) (d) Tree (2)

Fig. 6. Performance comparison between *COV* and *Migk* with respect to the changes of the migration costs when the number of servers is fixed as four (Uniform Pattern).

6 Performance Evaluation

We evaluate the proposed algorithms through extensive simulation-based studies. To this end, we developed a simulator in Java to create network topologies, generate the access patterns and execute the migration algorithms, online and off-line, to move the servers to satisfy each generated batch request according to a certain distribution in a time order. Each request is served by its closest server in terms of the shortest path distance. Ties-are-broken arbitrarily.

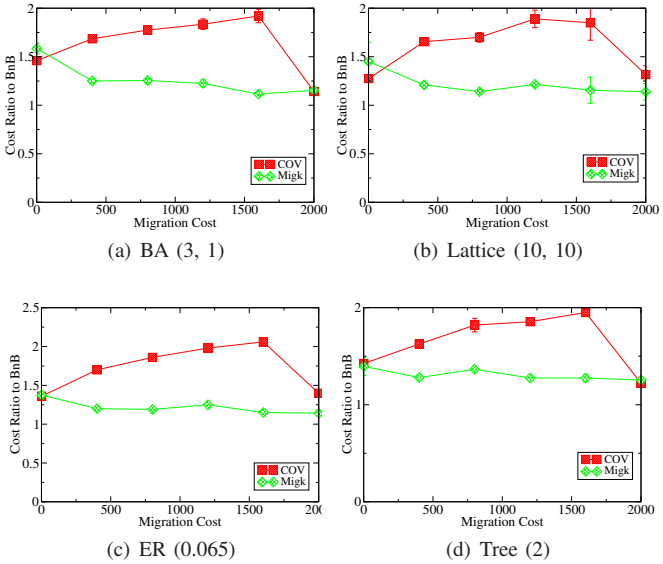
6.1 Experimental Setup

We choose networks with four typical topologies: tree, lattice, Erodös-Rényi (ER) random graph [48] and Barabási-Albert (BA) graph [49], each of which consists 100 nodes. These topologies exhibit different structural properties to represent a spectrum of communication networks in previous studies [20], [21], [50]–[52].

Both tree and lattice network topologies have typical topological characteristics that allow us to observe the behaviours of the algorithms under some extreme yet predictable conditions. In particular, the tree structure, characterized by average node degree, is always the intensively studied structure in different contexts including Clouds [50]–[53]. The lattice structure, characterized by its width and height, represents planar networks that have been observed in a surprising number of graphs in the Internet topology zoo [54]. In contrast, ER and BA graphs are random graphs without enforcing any regular structure. Both graphs are considered here as a complement to model general inter-networks that could be used in inter-cloud connections. In ER, each edge is included in the graph with probability p independent from every other edge whereas the BA graph is determined by the number of initial nodes and edges to be added at each time step. In our experiments, we always fix the network size as 100 since we expect that in reality the number of machines that can host the movable service is not that large. The average inter-node distances and node degrees of the four above-mentioned of network topologies are listed in Table 2.

In our model, access pattern is characterized by a sequence of online batch requests distributed across the network along the time axis. Each batch request is specified by its arrival time instance, batch size as well as distribution of the connect points together with the associated weights. In our experiments, we adopt three types of patterns that are often used in network studies [20], [21], each with different merits to evaluate the migration algorithms: First, to isolate the impact of the network topologies, we use uniform distribution, denoted as *Uniform(p,q)*, for both the batch size (i.e., the number of requesting nodes) in [1,p] and request weight in [1,q]. Then, in order to reflect the skewness among nodes, we assume a uniform batch size in [1,p] and a Zipf-like distribution, characterized by a parameter $\alpha \geq 1$, to capture the amount of access weight skew of each requesting node. We denote it as *Zipf(p,α)*. Finally, to model the mobile access dynamics, we deliberately partition the network graphs into different zones (ω) by clustering nodes based on their locations. The nodes in different zones will make requests at the different time segments to mimic the mobile accesses. In each zone, we also follow the Zipf-like distribution (*Zipf(p,α)*). Therefore, this pattern reflects both spatial skewness and temporal dynamics, denoted as *Zone(p,α,ω)*.

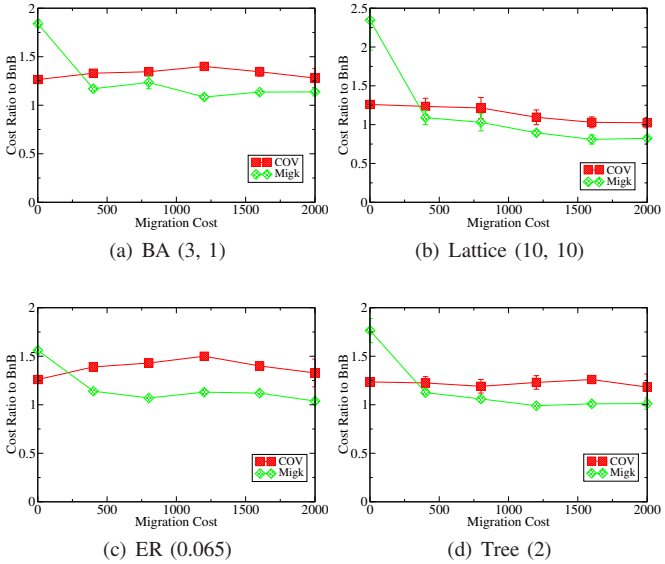
The major performance metric is defined as the ratio of the total service cost achieved by the proposed algorithm over the one achieved by a sub-optimal off-line algorithm, an effective sampling-based approximation. We chose this metric as it can



(a) BA (3, 1) (b) Lattice (10, 10)

(c) ER (0.065) (d) Tree (2)

Fig. 7. Performance comparison between *COV* and *Migk* with respect to the changes of the migration costs when the number of servers is fixed as four (Zipf-like Pattern).



(a) BA (3, 1) (b) Lattice (10, 10)

(c) ER (0.065) (d) Tree (2)

Fig. 8. Performance comparison between *COV* and *Migk* with respect to the changes of the migration costs when the number of servers is fixed as four (Zone Pattern).

not only measure the relative performance to the potential optimal results but also demonstrate the performance changes of the algorithms between different configurations.

6.2 Results

In this section, we evaluate the performance of the proposed algorithms in various cases. Since the compared online algorithms are evaluated relative to the off-line algorithm, we first measure the performance of the BnB algorithm in terms of its ratio over the optimal DP algorithm whereby we use it as a baseline to compare the online algorithms. In the experiments, each data point in the graphs is averaged over five runs by changing the random number seed in the simulator. To measure the distribution of the values for each data point, we also compute the standard deviation of each data point. However, for clarity of presentation, we omitted standard deviation bars on the graphs if most of them are less than 10% of the data point's values.

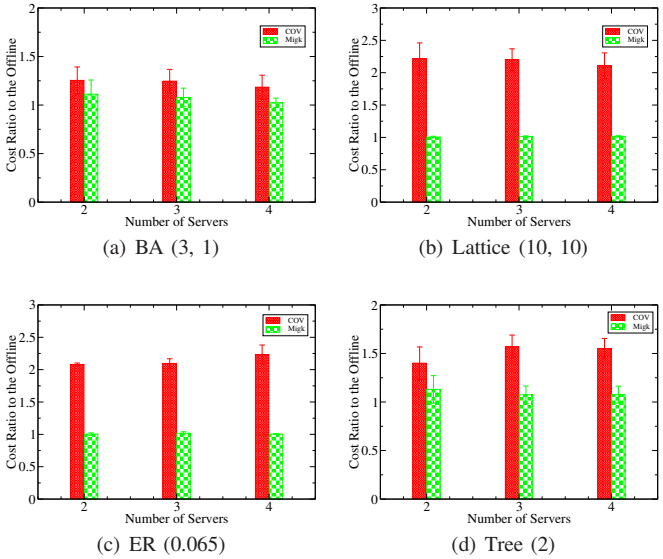
6.2.1 Performance Studies on the BnB Algorithm

Performance Behaviours: We first study the performance behaviours of *BnB* with respect to the number of samples. In theory, as the number of samples increases, the performance of the algorithm should also increase accordingly. Fig. 3 verifies this by showing how the service costs decrease along the x-axis (the number of samples) of all the sub-figures when the number of threads is four. Since each sample consists of a set of randomly selected configurations, the performance of the algorithm is not only related to the sample size but also reliant on the number of configurations in each sample. To evaluate this impact, we also compare the performance of the algorithm when the numbers of configurations in each sample are varied. As shown in Fig. 3, the service costs of the algorithm decrease as the number of configuration increases, which is not beyond our expectation. Although the decreasing trends are obvious with respect to the changes of samples, the values of the decreasing, in various cases, are not that significant, which allows us to use a small-scale sample spaces (including the number of samples and the number of randomly selected configurations in each sample) to evaluate the algorithm without sacrificing too much performance while maintaining the efficiency.

By fixing the sample size as 160 and the number of configurations as 32, we also measure the speedups of *BnB* for Zipf-like access pattern in Fig. 4 where the experiments are performed under Ubuntu 12.04 running on a hardware configuration of 8GB memory and a 3392.183 MHz quadcore processor, with total of 8 threads, each with 8192K cache. From this figure, one can easily observe that the algorithm achieves a linear speedup up to 4 threads and maximizes at 8 threads across all the examined cases. This observation is understandable as the processor is quadcore with total of 8 physical threads. These results demonstrate the scalability of our algorithm.

Performance Evaluations: In these experiments, we leverage the conclusions in the last sub-section and evaluate *BnB* by using a relative small-scale sampling strategy (100 samples, each with 16 configurations). To compare the performance in different cases (e.g., different network topologies and access patterns), we normalize the service costs of *BnB* with those of the optimal DP algorithm.

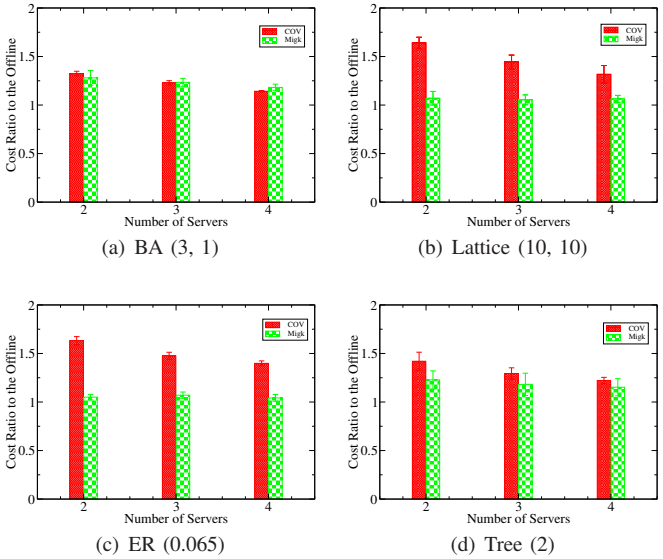
Fig. 5 shows the results when both the network sizes and the number of servers are limited to small scales for overcoming the configuration complexity. For the sake of clarity, we only plot the ratios for Zip-like access pattern on all the studied networks with size$\leq$20 (Fig. 5 (a)), and the ratios for all the access patters on the lattice and tree networks, each having 100 nodes since the results of these cases are worse and more varied than those of the other two topologies (Fig. 5 (b)). In



(a) BA (3, 1) (b) Lattice (10, 10)

(c) ER (0.065) (d) Tree (2)

Fig. 9. Performance comparison between *COV* and *Migk* with respect to the changes of the number of servers when the migration costs are uniformly distributed in [0, 2000] (Uniform Pattern).



(a) BA (3, 1) (b) Lattice (10, 10)

(c) ER (0.065) (d) Tree (2)

Fig. 10. Performance comparison between *COV* and *Migk* with respect to the changes of the number of servers when the migration costs are uniformly distributed in [0, 2000] (Zipf-like Pattern).

both cases, there are at most 4 servers. From this figure, one can easily see that, except for the tree with less than 4 servers, the performance of *BnB* is less than 20% of the optimal results across all the studied cases. These results demonstrate that the performance variances of *BnB* are relatively stable with respect to the changes of network sizes, topologies, and access patterns.

Although these results cannot fully reflect the actual performance of *BnB*, they at least exhibit some supportive evidences that *BnB* is a relatively good sub-optimal off-line algorithm for comparison. Note that BnB requires $\kappa \leq \frac{\log n}{\log(1+\Delta)}$, but we think this constraint is practical in reality as the number of servers is usually much less than the total number of network nodes. By combing the results in Fig. 3 and Fig. 5, we can see that the *BnB* algorithm with the examined configurations can effectively approximate the optimal results while reducing the runtime overhead (again, using small sampling space without sacrificing the performance). Therefore, in the following experiments, we will use it as a baseline to measure the performance of compared online algorithms in various cases.

6.2.2 Performance Evaluation on the Migk Algorithm

To evaluate the online algorithm *Migk*, in addition to normalizing it with *BnB* to assess its performance relative to the off-line results, we also compare it with a reference algorithm from our previous studies [55], called *Cooperative Migration with Virtual Moves* algorithm (*COV*) *COV* is squarely designed for the service migration in clouds and has shown some advantages over some existing algorithms [16], [20]. For instance, COV overcomes the inefficient one-hop migration of the service for each migration (i.e., the local-search limitation) in [20] and in the meanwhile removing the configuration complexity for search space reduction in [16] as well with a proposed concept of *virtual move*. However, it does not assure the worst case performance. By comparing with *COV*, we can show how *Migk* behaves better than the existing algorithm in practice and also guarantees the performance bound in the worst case as well.

By fixing the number of the servers as four, we show from Fig. 6 to Fig. 8 the compared results of both algorithms with respect to the changes of the migration costs for the three selected access patterns on the different networks. From these figures, one can easily observe that *Migk* exhibits better overall performance than *COV* across all the studied cases, except for the scenario when migration cost is free. These results are consistent with our original expectations as *Migk* is designed to be sensitive to the changes of the access pattern and adaptive to them with service migration. It always allows each server to independently select its own migration target along a cost-decreasing path, rather than taking all the server replicas in a configuration as as whole to make migration decision as *COV* does. As a consequence, the computation of *Migk* can be steered toward a more cost efficient way than *COV* which, compared with *Migk*, is biased against the migration to serve the demand sequence. This also explains why the performance of *Migk* is inferior to that of *COV* in most comparisons since in this case, as we discussed in Section 4.2, the effective epochs of *Migk* cannot be created when $\beta' = 0$, thus *Migk* would incur too many migrations to adapt to the access patterns. Admittedly, this is a shortcoming of *Migk*. Fortunately, migration for free is not a reasonable practice in cloud-based services.

In addition to the performance merits, *Migk* also exhibits another advantage over *COV*, that is, compared with *COV*, the performance of *Migk* is relatively stable when the migration costs are varied. This is a nice feature for the service providers that allows the service costs are highly predictable.

In addition to the migration costs, we also studied how the compared algorithms behave with respect to the number of

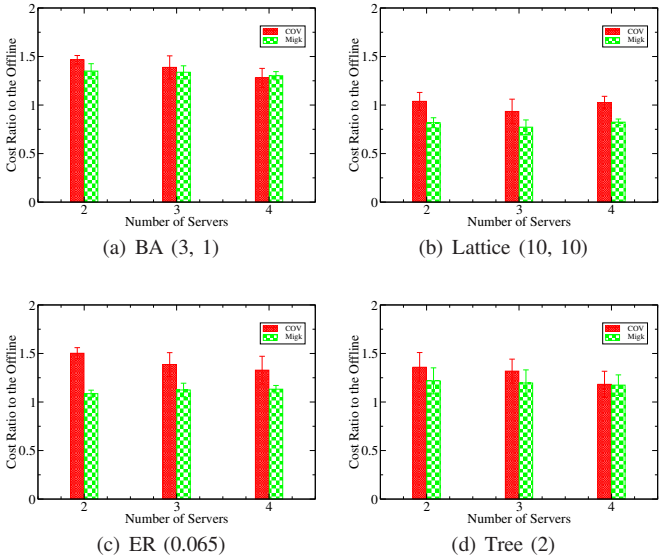


Fig. 11. Performance comparison between *COV* and *Migk* with respect to the changes of the number of servers when the migration costs are uniformly distributed in [0, 2000] (Zone Pattern).

servers. Given the size of networks as 100 nodes, we studied the algorithms by varying the number of the servers from 2 to 4. The results for different access patterns are shown in Fig. 9, Fig. 10 and Fig. 11, respectively, when the migration cost is uniformly distributed between 0 and 2000. As understood from the previous explanations, *Migk* is consistently better than *COV* for all the access patterns on the studied networks. These results are also consistent with those in our previous experiments. Moreover, as the number of server replicas is varied, the performance of *Migk* is also stable, which is again within our expectations. Note that in some cases (e.g., Zone pattern on Lattice), it could be possible for *Migk* to outperform the off-line algorithm since the baseline algorithm is sup-optimal.

In summary, all these experimental results demonstrate the performance advantages of *Migk* over *COV*, rendering *Migk* to be more practical in reality. On the other hand, the performance of *Migk* in its worst case is guaranteed as we showed in Theorem 4.2, which is different from *COV* whose performance is unbounded.

7 Conclusions

In this paper, we formulated and studied the service migration problem in cloud platforms from both online and off-line perspectives. For the online case, we proposed *Migk*, an online randomized algorithm with a mean competitive ratio of $O(\frac{\gamma \log n}{\min\{\frac{1}{\kappa}, \frac{\mu}{\lambda+\mu}\}})$ to co-migrate κ servers in a static n-node network within $O(\kappa n)$ time, which is better than the previous results in terms of both the competitive ratio and time complexity [16].

For the off-line case, we proposed a multithreaded branch&bound algorithm, named *BnB*, which is based on DP techniques and leverages sampling methods to efficiently approximate the optimal results when the number of servers is upper-bounded by $O(\frac{\log n}{\log(1+\Delta)})$, a practical case in reality. Our simulation results showed that the results of *BnB* on average are consistently within 120% of the optimal result across all our experimental cases.

Using *BnB* as the baseline, we further compared the proposed algorithm with *COV*, an optimized co-migration algorithm that exhibits some performance advantages over some existing algorithms. After analyzing the experimental results, we conclude that *Migk* achieves a better cost reduction than *COV* under the designed experimental setup.

References

[1] "Good technology's 2nd annual state of BYOD report," 2013, http://media.www1.good.com/documents/Good-BYOD-Report-2013.pdf.
[2] S.-C. Lee, E. Lee, W. Choi, and U.-M. Kim, "Extracting temporal behavior patterns of mobile user," in *Networked Computing and Advanced Information Management, 2008. NCM '08. Fourth International Conference on*, vol. 2, 2008, pp. 455–462.
[3] M. Bienkowski, A. Feldmann, J. Grassler, G. Schaffrath, and S. Schmid, "The wide-area virtual service migration problem: A competitive analysis approach," *IEEE/ACM Trans. Netw.*, vol. 22, no. 1, pp. 165–178, Feb. 2014.
[4] "Netflix," 2014, http://www.netflix.com/.
[5] "Lions gate," 2014, http://www.lionsgate.com/.
[6] "Virtual appliance," 2014, http://searchservervirtualization.techtarget.com/definition/virtual-appliance.
[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*, ser. NSDI'05, 2005, pp. 273–286.
[8] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, pp. 14–26, Jul. 2009.
[9] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *Proceedings of the 3rd international conference on Virtual execution environments*, ser. VEE '07, 2007, pp. 169–179.
[10] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, "Live migration of virtual machine based on full system trace and replay," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*, ser. HPDC '09, 2009, pp. 101–110.
[11] H.-F. Lai, Y.-S. Wu, and Y.-J. Cheng, "Exploiting neigborhood similarity for virtual machine migration over wide-area network," in *Software Security and Reliability (SERE), 2013 IEEE 7th International Conference on*, 2013, pp. 149–158.
[12] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "Vmflock: virtual machine co-migration for the cloud," in *Proceedings of the 20th international symposium on High performance distributed computing*, 2011, pp. 159–170.
[13] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe, "Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE '11, 2011, pp. 121–132.
[14] P. Riteau, C. Morin, and T. Priol, "Shrinker: efficient live migration of virtual clusters over wide area networks," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 541–555, 2013.
[15] J. Schneider and S. Schmid, "Optimal bounds for online page migration with generalized migration costs," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 1905–1913.
[16] D. Arora, A. Feldmann, G. Schaffrath, and S. Schmid, "On the benefit of virtualization: strategies for flexible server allocation," in *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, ser. Hot-ICE'11, 2011, pp. 2–2.
[17] J. Lee, H. Yu, and J. Gil, "A virtual machine migration management for multiple datacenters-based cloud environments," in *Advanced Computer Science and Information Technology*, ser. Communications in Computer and Information Science, T.-h. Kim, H. Adeli, R. Robles, and M. Balitanas, Eds. Springer Berlin Heidelberg, 2011, vol. 195, pp. 198–205.
[18] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *Queue*, vol. 11, no. 12, pp. 20:20–20:40, Dec. 2013.
[19] M. Bienkowski, A. Feldmann, D. Jurca, W. Kellerer, G. Schaffrath, S. Schmid, and J. Widmer, "Competitive analysis for service migration in vnets," in *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, ser. VISA '10. New York, NY, USA: ACM, 2010, pp. 17–24.
[20] K. Oikonomou and I. Stavrakakis, "Scalable service migration in autonomic network environments," *IEEE J.Sel. A. Commun.*, vol. 28, no. 1, pp. 84–94, Jan. 2010.

[21] P. Pantazopoulos, M. Karaliopoulos, and I. Stavrakakis, "Centrality-driven scalable service migration," in *Proceedings of the 23rd International Teletraffic Congress*, ser. ITC '11, 2011, pp. 127–134.
[22] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive, memory compression," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, Aug 2009, pp. 1–10.
[23] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan, "Snowflock: Rapid virtual machine cloning for cloud computing," in *Proceedings of the 4th ACM European Conference on Computer Systems*, ser. EuroSys '09. New York, NY, USA: ACM, 2009, pp. 1–12.
[24] A. Shribman and B. Hudzia, "Pre-copy and post-copy vm live migration for memory intensive applications," in *Euro-Par 2012: Parallel Processing Workshops*, ser. Lecture Notes in Computer Science, I. Caragiannis, M. Alexander, R. Badia, M. Cannataro, A. Costan, M. Danelutto, F. Desprez, B. Krammer, J. Sahuquillo, S. Scott, and J. Weidendorfer, Eds. Springer Berlin Heidelberg, 2013, vol. 7640, pp. 539–547.
[25] F. Ma, F. Liu, and Z. Liu, "Live virtual machine migration based on improved pre-copy approach," in *Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on*, July 2010, pp. 230–233.
[26] M. Gahlawat and P. Sharma, "Reducing the cost of virtual machine migration in federated cloud environment using component based vm," *Journal of Information Systems & Communication*, vol. 3, no. 1, pp. 288–290, 2012.
[27] G. Soni and M. Kalra, "Article: Comparative study of live virtual machine migration techniques in cloud," *International Journal of Computer Applications*, vol. 84, no. 14, pp. 19–25, December 2013, published by Foundation of Computer Science, New York, USA.
[28] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for hpc with xen virtualization," in *Proceedings of the 21st Annual International Conference on Supercomputing*, ser. ICS '07. New York, NY, USA: ACM, 2007, pp. 23–32.
[29] C. Engelmann, G. Vallee, T. Naughton, and S. Scott, "Proactive fault tolerance using preemptive migration," in *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, Feb 2009, pp. 252–257.
[30] Y. Zhao and W. Huang, "Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud," in *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, ser. NCM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 170–175.
[31] D. H. Phan, J. Suzuki, R. Carroll, S. Balasubramaniam, W. Donnelly, and D. Botvich, "Evolutionary multiobjective optimization for green clouds," in *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation*, 2012, pp. 19–26.
[32] N. Bila, E. de Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan, "Jettison: Efficient idle desktop consolidation with partial vm migration," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 211–224.
[33] V. Patil and G. Patil, "Migrating process and virtual machine in the cloud: Load balancing and security perspectives," *International Journal of Advanced Computer Science and Information Technology*, vol. 2, no. 2, 2012.
[34] C. Isci, J. Liu, B. Abali, J. Kephart, and J. Kouloheris, "Improving server utilization using fast virtual machine migration," *IBM Journal of Research and Development*, vol. 55, no. 6, pp. 4:1–4:12, Nov 2011.
[35] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2009, pp. 41–50.
[36] F. Hermenier, J. Lawall, and G. Muller, "Btrplace: A flexible consolidation manager for highly available applications," *Dependable and Secure Computing, IEEE Transactions on*, vol. 10, no. 5, pp. 273–286, Sept 2013.
[37] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, Ottawa, Ontario, Canada, Jun. 2007, pp. 225–230.
[38] "Openstack," 2014, http://www.openstack.org/.
[39] "Apache cloudstack," 2014, http://cloudstack.apache.org/.
[40] "Eucalyptus," 2014, https://www.eucalyptus.com/.
[41] K. Ye, X. Jiang, R. Ma, and F. Yan, "VC-Migration: live migration of virtual clusters in the cloud," in *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*. IEEE, 2012, pp. 209–218.
[42] C. Avin, O. Dunay, and S. Schmid, "Strategies for traffic-aware vm migration," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2013, pp. 305–306.
[43] R. Mortier and E. Kiciman, "Autonomic network management: some pragmatic considerations," in *Proceedings of the 2006 SIGCOMM Workshop on Internet Network Management*, 2006, pp. 89–93.
[44] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, Apr. 2010.
[45] D. Arora, M. Bienkowski, A. Feldmann, G. Schaffrath, and S. Schmid, "Online strategies for intra and inter provider service migration in virtual networks," in *Proceedings of the 5th International Conference on Principles, Systems and Applications of IP Telecommunications*, ser. IPTcomm '11, 2011, pp. 10:1–10:11.
[46] H. Goudarzi and M. Pedram, "Geographical load balancing for online service applications in distributed datacenters," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, ser. CLOUD '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 351–358.
[47] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang, "Live migration of multiple virtual machines with resource reservation in cloud computing environments," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 267–274.
[48] P. Erdös and A. Rényi, "On random graphs I." *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.
[49] A. L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.
[50] B. Li, M. Golin, G. Italiano, X. Deng, and K. Sohraby, "On the optimal placement of web proxies in the internet," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 1999, pp. 1282–1290.
[51] H. Gupta and B. Tang, "Data caching under number constraint," in *the IEEE International Conference on Communications*, 2006.
[52] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08, 2008, pp. 63–74.
[53] K. Oikonomou, I. Stavrakakis, and A. Xydias, "Scalable service migration in general topologies," in *World of Wireless, Mobile and Multimedia Networks, 2008. WoWMoM 2008. 2008 International Symposium on a*, june 2008, pp. 1 –6.
[54] R. Bowden, H. X. Nguyen, N. Falkner, S. Knight, and M. Roughan, "Planarity of data networks," in *Proceedings of the 23rd International Teletraffic Congress*, ser. ITC '11, 2011, pp. 254–261.
[55] Y. Wang, W. Shi, and L. Zeng, "Adaptive search-based service migration with virtual moves in clouds for mobile accesses," in *UCC2013*, Dec. 2013.

Yang Wang received the BSc degree in applied mathematics from Ocean University of China, and the MSc and Ph.D degrees in computer science from Carleton University (2001) and University of Alberta, Canada (2008), respectively. He is currently working with IBM Center for Advanced Studies (CAS Atlantic), University of New Brunswick, Fredericton, Canada. His research interests include Cloud computing, big data analytics, and Java Virtual Machine on multicores. He is an Alberta Industry R&D Associate (2009-2011), and a Canadian Fulbright Fellow (2014-2015).

Wei Shi is an assistant professor at the University of Ontario Institute of Technology. She is also an adjunct professor at Carleton University. She holds a Bachelor of Computer Engineering from Harbin Institute of Technology in China and received her Ph.D. of Computer Science degree from Carleton University in Ottawa, Canada. Prior to her academic career, as a software developer and project manager, she was closely involved in the design and development of a large-scale Electronic Information system for the distribution of welfare benefits in China, as well as of a World Wide Web Information Filtering System for Chinas National Information Security Centre.

Menglan Hu received the B.E. degree in Electronic and Information Engineering from Huazhong University of Science and Technology, China (2007), and the Ph.D. degree in Electrical and Computer Engineering from the National University of Singapore, Singapore (2012). He joined the faculty of the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China (2014), where he is currently an Associate Professor. His research interests includes cloud computing, parallel and distributed systems, scheduling and resource management, as well as wireless networking.