On Service Migrations in the Cloud for Mobile Accesses: A Distributed Approach

YANG WANG, Shenzhen Institutes of Advanced Technology
BHARADWAJ VEERAVALLI and CHEN-KHONG THAM, National University of Singapore
SHUIBING HE, Wuhan University
CHENGZHONG XU, Shenzhen Institutes of Advanced Technology

We study the problem of dynamically migrating a service in the cloud to satisfy an online sequence of mobile batch-request demands in a cost-effective way. The service may have single or multiple replicas, each running on a virtual machine. As the origin of mobile accesses frequently changes over time, this problem is particularly important for time-bounded services to achieve enhanced Quality of Service and cost effectiveness. Moving the service closer to the client locations not only reduces the service access latency but also minimizes the network costs for service providers. However, these benefits are not free. The migration comes at a cost of bulk-data transfer and service disruption, and hence, increasing the overall service costs. To gain the benefits of service migration while minimizing the caused monetary costs, we propose an efficient search-based algorithm *Dmig* to migrate a single server, and then extend it as a scalable algorithm, called *mDmig*, to the multi-server situation, a more general case in the cloud. Both algorithms are fully distributed, symmetric, and characterized by the effective use of historical access information to conduct virtual migration so that the limitations of local search in the cost reduction can be overcome. To evaluate the algorithms, we compared them with some existing algorithms and an off-line algorithm. Our simulation results showed that the proposed algorithms exhibit better performance in service migration by adapting to the changes of mobile access patterns in a cost-effective way.

CCS Concepts: • **Networks** → *Cloud computing*; • **Computing methodologies** → **Distributed algorithms;**

Additional Key Words and Phrases: Cloud computing, dynamic service migration, mobile access, dynamic virtual machine placement, virtual migration

ACM Reference Format:
Yang Wang, Bharadwaj Veeravalli, Chen-Khong Tham, Shuibing He, and Chengzhong Xu. 2017. On service migrations in the cloud for mobile accesses: A distributed approach. ACM Trans. Auton. Adapt. Syst. 12, 2, Article 6 (May 2017), 25 pages.
DOI: http://dx.doi.org/10.1145/3050438

This work was supported in part by NSFC under grant No. 61672513, No. 61572377, and No. 61603376, Science and Technology Planning Project of Guangdong Province (No. 2015B010129011 and No. 2016A030313183), Shenzhen Science and Technology Innovation (JSGG20160229200957727), and National Key Research and Development Program of China (2016YFB1000204).
Authors' addresses: Y. Wang and C. Xu, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, No. 1068 Xueyuan Avenue, Shenzhen University Town, Shenzhen, China, 518055; B. Veeravalli and C.-K. Tham, Department of Electrical and Computer Engineering, Block E4, Level 5, Room 45, 4 Engineering Drive 3, University of Singapore, Singapore 117583; S. He (corresponding author), Rm B604, Computer Building, School of Computer, Wuhan University, China 430072.

DOI: http://dx.doi.org/10.1145/3050438

1. INTRODUCTION

Cloud computing, as an emerging driving force of the next innovation wave, has been forming the basis for many novel applications across a wide range of fields. Among these fields, the mobile computing, by exploiting the virtue of cloud platforms, is enabling a new generation of services for mobile users. These services are, in general, hard, if not impossible, to achieve using traditional technologies due to the intrinsic characteristics of large-scale mobile accesses, which are typically sensitive to access latency, and moreover, always changing with respect to the time and locations of the mobile users. As such, providing network services without considering these factors may substantially increase access delays, and much worse, impose a large amount of network traffic, which may in turn cause service disruption. As a result, traditional technologies, such as those used by *Content Distribution Network* (CDN) to fix services in a set of carefully selected locations, are no longer cost-effective.

To mitigate this problem, migrating the service to some vantage locations in the network that are close to the users could be an effective way in terms of the minimization of access latency and, at the same time, the reduction of network costs for service providers. A typical example to illustrate the migration benefits is a multi-player mobile game (Figure 1), where a game server may migrate from `Site A` at 7:30am to `Site B` at 9:30am, and finally to `Site C` at 11:15am (here sites refer to physical machines located in the same or different data centers), depending on the changing locations of the dominant access loads at different time frames (from 7:30am to 11:15am). Traditionally, there is no effective solution available to achieve such benefits. Fortunately, by the virtue of virtualization technologies in the cloud, encapsulating the service in a set of virtual machines and migrating them on-demand (aka *Live Migration*) in the same, or across different, data centers is a promising way to deploy such a service with the aforementioned benefits.

Although the wide-area live virtual machine (VM) migration, including server memory image and associated file systems, remains expensive to use because of the bandwidth bottleneck, it is still feasible with some advanced technologies to minimize the migration overhead [Bradford et al. 2007; Liu et al. 2009; Al-Kiswany et al. 2011; Riteau et al. 2013; Lai et al. 2013]. For example, R. Bradford et al. [2007] showed 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 minimal disruption—3s in LAN and 68s over WAN. This impact was further reduced in follow-up studies by exploiting different features of the migration [Liu et al. 2009; Al-Kiswany et al. 2011; Riteau et al. 2013; Lai et al. 2013]. With these technologies, several preliminary results have demonstrated the benefits of service migration over virtual networks and autonomic networks [Bienkowski et al. 2010; Oikonomou and Stavrakakis 2010]. However, the trade-off between the benefits and the costs (from the monetary cost point of view) of the service migration in the cloud to facilitate mobile accesses has not been thoroughly studied. Given the characteristics and prevalence of cloud computing, this trade-off is particularly important for cloud service providers (CSPs) to maximize their profits when deploying services in the cloud. In this article, we investigate this problem and propose dynamic migration algorithms based on local search techniques. The proposed algorithms can adapt to user access patterns by migrating the service (hosted by virtual servers) in a cost-effective way, not only satisfying the service demands but also minimizing the access costs.

To this end, we first propose an efficient distributed algorithm, called *Dmig*, where a single virtual server that hosts the service is considered. The algorithm is fully symmetric to allow network nodes to effectively collaborate with each other to migrate the service in an efficient way. The efficiency of the algorithm is guaranteed by the virtue of the fact that, except for some pathological cases (where $O(n)$ messages are needed for n-node

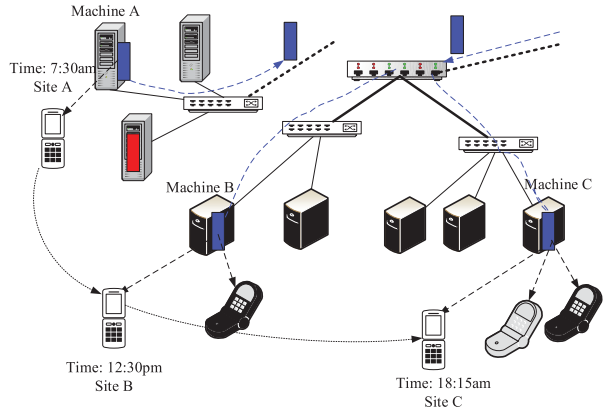


Fig. 1. A mobile game server is migrated at different time frames from Site A (7:30am) to Site B (9:30am), and finally to Site C (11:15am) to adapt to the mobile access patterns.

network), *Dmig* does not incur any global operation in the migration decision. Moreover, it leverages a concept of *virtual migration* to overcome the limitation of one-hop local search in the selection of migration targets. Compared with existing approaches, the novelty of the proposed algorithm is to allow the service to move to the optimal target node directly along the multi-hop path found by the virtual migration. We show how the algorithm eventually migrates a virtual server to the optimal service node and remains the server there as long as the access pattern is not significantly changed. Although the single-server case is simple, it is not uncommon in practice and has been studied in literature [Oikonomou and Stavrakakis 2010; Bienkowski et al. 2010; Arora et al. 2011b; Pantazopoulos et al. 2011] as a baseline before exploring more general cases.

We then extend the single-server results to a more general scenario, where the number of server replicas κ is greater than one. For this case, we propose a scalable multi-server *Dmig* algorithm, denoted by *mDmig*, which deliberately allows each virtual server to perform the *Dmig* algorithm independently. *mDmig* improves the algorithm in Wang et al. [2013, 2015] where only a very limited number of servers are allowed to migrate under a centralized control.

We validated our findings by conducting extensive simulation studies, where the proposed algorithms, together with their variants, are compared with some existing online and off-line algorithms under various access patterns and different network topologies. Our simulation results showed that compared with the existing algorithms, the proposed algorithms are more cost effectively adaptive to the changes of mobile access patterns and exhibit better performance in service migration.

The remainder of this article is organized as follows. We review some related work in Section 2 and formulate the service migration problem in Section 3. Then, we introduce the proposed dynamic migration algorithm *Dmig* for single-server migration followed by its extension *mDmig* to multi-server case in Section 4. We show the simulation results in Section 5, and conclude the article in Section 6.

2. RELATED WORK

Service migration in different forms has been studied in a number of related areas, each with similar or different goals [Chroboak et al. 1991; Oikonomou and Stavrakakis 2010; Bienkowski et al. 2010; Pantazopoulos et al. 2011; Phan et al. 2012; Bienkowski et al. 2014]. A typical related problem is the *service location problem* (SLP) in transportation and supply networks, which has inspired a broad range of research works. In this problem, some vantage locations of service facilities need to be found in the network

so the (off-line) demands made at a subset of the nodes can be served with minimum total distance cost [Farahani and Hekmatfar 2009].

There are two most common forms of SLP, namely *κ-median problem* [Jain and Vazirani 2001] and *facility location problem* [Swamy and Kumar 2002], depending on whether the number of facilities (i.e., κ) is given or the open cost for each facility is pre-defined. Both problems are NP-hard [Farahani and Hekmatfar 2009] and have a large number of variants in practice [Friggstad and Salavatipour 2011; Farahani et al. 2010; Görtz and Klose 2012], which are typically addressed by different approximation or heuristic algorithms.

Among these variants and highly related to our problem is the *mobile facility location problem* [Friggstad and Salavatipour 2011], where a rendezvous node for each client (or demand) and some facility need to be selected in the network to conduct the service so the total distances traveled by both the clients and the facilities are minimized. Although they considered the movements of the servers (facilities) and achieved a better 8-approximation algorithm, the problem is still static in nature, since the clients are still pre-defined, each with a fixed start node. Our results hence can be viewed as an extension to this problem, where the locations of the servers in a network can be changed to adapt to time-varying demand patterns.

As for its dynamic nature, our problem is related to the classic *κ-server problem* that allows κ mobile servers at some nodes of a graph to move on-site to serve request sequence in an online fashion so the total moving distance of these servers is minimized [Manasse et al. 1988; Chroboak et al. 1991]. Our problem is a variant of the κ-server problem in that the mobile servers cannot only be accessed remotely, but they can also be moved simultaneously for a batch request. In this sense, it is also related to the *dynamic servers problem* [Charikar et al. 1998], first proposed and studied by Charikar et al. [1998]. In this problem, the number of servers is not fixed beforehand, instead, the algorithm has freedom to increase and decrease the number of servers at will for the optimality. In contrast, the number of servers κ is pre-determined in our problem, and on the other hand, we consider the remote services to online batch requests, which is also different from the dynamic servers problem.

Service migration has been studied in the context of *virtual networks* (VNets) [Yu et al. 2008; Chowdhury and Boutaba 2010] to minimize service access latency. Bienkowski et al. [2010] presented a randomized online algorithm for a single-server migration in an n-node network in a centralized way and advocated the competitive analysis on the worst case of the algorithm. Their results, together with other extensions and findings, were later summarized in Bienkowski et al. [2014]. Unlike their research, our work concentrates on the distributed approach and its general performance in cloud environments.

As in VNets, service migration is also studied in *autonomic network environments* [Dobson et al. 2006] as a self-managing mechanism to overcome the rapidly growing complexity of networks. Oikonomou and Stavrakakis [2010] proposed a scalable algorithm for service migration in autonomic networks. By observing the differential demand traffic on each link between the node hosting the service and its opposite neighbors, the algorithm performs a number of local searches to repeatedly find the next one-hop migration target along the shortest-path tree to the optimal location. Although this algorithm has certain merits for service migration, it suffers from slow convergence due to the inefficient one-hop migration. Pantazopoulos et al. [2011] overcome this downside in their most recent centrality-driven migration algorithm, named *cDSMA*. However, this algorithm only targets a single server and, moreover, lacks the notion of migration cost.

Although our algorithm for the single-server migration is still based on local search techniques, it takes into account both the access costs and the migration costs, thereby accelerating the migration process. *Dmig* attains this by exploiting a novel concept of

virtual migration that could identify the target node by mimicking the service to the incoming requests. On the other hand, unlike cDSMA, *Dmig* is fully symmetric with a low message complexity of $\Theta(n)$.

In contrast to the above studies, which are not directly conducted in the cloud, Phan et al. [2012] proposed a framework, called *Green Monster*, to leverage dynamic service migration across *internet data centers* (IDCs) for energy efficiency in the cloud computing. A similar effort was made by Wang et al. [2013], who presented a decentralized approach to virtual machine migration inside data centers for energy saving while maintaining the quality of service. The major difference between our results with these two studies is that we focus on the service migration strategies for total monetary cost minimization for CSPs, instead of developing a new migration mechanism. Although our problem and theirs are orthogonal at the first sight, they could be connected inherently as the monetary cost could also be used to measure the energy consumption as well. However, this extension is still an open problem to our algorithm.

Wang et al. [2015] once studied the co-migration service problem in the cloud where κ virtual servers are coordinately migrated together via a centralized control to minimize the total service cost. However, the co-migration algorithm is complicated and also suffers from some limitation on the number of servers κ (i.e., $\kappa \leq O(\frac{\log n}{\log(1+\Delta)})$, where Δ is the maximum node degree). Our proposed algorithm is fully distributed, addressing all these issues by taking simplicity and scalability as a goal.

A more comprehensive study on virtual machine migration in cloud computing environments with respect to the benefits, challenges, and approaches can be found in a recent work by Boutaba et al. [2014].

3. SERVICE MIGRATION MODEL

We consider an arbitrary n-node network $G(V, E)$ as a service infrastructure, where the provided service has $\kappa \geq 1$ replicas, each running on a VM that is hosted by a compute node. The set of virtual machines consist of a *configuration*, denoted by $\mathcal{L}$, which is accessed by a sequence of batch requests $\sigma = \sigma_1\sigma_2 \cdots \sigma_m$ issued from a set of external machines (i.e., mobile terminals). The requests arrive in an online fashion and are served by triggering the migration of the virtual servers in $\mathcal{L}$. As a result, the subset of the server locations in the network could be frequently changed over time. We denote $\mathcal{L}$ at time t_i as $\mathcal{L}_i$.

Charging Model. A request is routed to the service over a wireless link, first to enter the network via an *access point* and then based on some (overlay) routing algorithm (e.g., the shortest-path-based routing) to reach the service. In this model, we assume the (wireless) connection cost is μ, and the transfer cost between a pair of nodes u and v is C_{uv}. According to the charging model adopted by most current cloud infrastructure services, it is reasonable to assume that both types of the costs are available from the infrastructure service providers (ISPs) to the overlaying CSPs.

Access Cost. In our model, a batch request σ_i is denoted by a set $\sigma_i = \cup_j \sigma_{ij}$, where σ_{ij} is a sub-request of σ_i sent to the network via access point a_j at time t_i. Clearly, to satisfy σ_i, each sub-request $r \in \sigma_i$ will be eventually routed to a certain $h \in \mathcal{L}_i$. This is typically achieved by the underlying *routing function* determined by ISPs. As a consequence, the total access cost of batch request σ_i can be simply written as

$$Cost_{acc}(\mathcal{L}, \sigma_i) = |\sigma_i|\mu + \sum_{r \in \sigma_i} C_{a_r \phi(r)}, \phi(r) \in \mathcal{L}, \tag{1}$$

where a_r is request r's nearest access point and $\phi(r)$ is r's service node determined by the (overlay) routing function $\phi(.)$, given configuration $\mathcal{L}$. In this definition, $|\sigma_i|\mu$ is the total cost of σ_i to connect its nearby access points and $\sum_{r \in \sigma_i} C_{a_r \phi(r)}$ define the cost for

σ_i to access the service replicas from the access points along the paths determined by $\phi(.)$. Note that in this formulation, we implicitly assume that the size of requests is so small that the network bandwidth is never the bottleneck, rather, the link latency is the issue.

Migration Cost. In contrast to the requests, which are rather light-weight, the traffic volume of migrating servers usually cannot be neglected due to the large size of server states. Unlike the access cost, which is dominated by the access latency, the migration cost $Cost_{mig}$ of a virtual server depends a great deal upon the server size and the available bandwidth on the migration path. Therefore, we equip each node v with a migration cost set $\beta_v = \{\beta_{vu} | u \in \mathcal{N}(v)\}$ ($\beta_{vv} = 0$) to reflect the server migration overhead from v to a corresponding target u, $u \in \mathcal{N}(v)$, where $\mathcal{N}(v)$ represents the neighbor set of v. Again, β_v is also given by ISPs in advance as local knowledge for each node v in the network.

To minimize the migration cost, we need to identify a *matching function* π that can figure out the migration target node in $\mathcal{L}_i$ for each server in $\mathcal{L}_{i-1}$. To this end, we denote $\mathcal{L}_{i-1} = \{u_1, u_2, \ldots, u_\kappa\}$ and $\mathcal{L}_i = \{v_1, v_2, \ldots, v_\kappa\}$ and have

$$Cost_{mig}(\mathcal{L}_{i-1}, \mathcal{L}_i) = \min_{\pi} \left\{ \sum_{u_j \in \mathcal{L}_{i-1}} \beta_{u_j v_{\pi(j)}} \right\}, \tag{2}$$

where π is a permutation of $\{1, 2, \ldots, \kappa\}$, while $Cost_{mig}(\mathcal{L}_{i-1}, \mathcal{L}_i)$ is the minimum cost to transfer $\mathcal{L}_{i-1}$ to $\mathcal{L}_i$.

Migration Goal. Given $Cost(\mathcal{L}_0) = 0$, for a sequence of batch requests $\sigma = \sigma_1 \sigma_2 \cdots \sigma_m$, the goal of the service migration is to determine $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_t$ to minimize the total service cost defined as

$$Cost(\mathcal{L}_i) = Cost(\mathcal{L}_{i-1}) + Cost_{acc}(\mathcal{L}_{i-1}, \sigma_i) + Cost_{mig}(\mathcal{L}_{i-1}, \mathcal{L}_i). \tag{3}$$

Note that in this model, to emphasize the network cost and simplify the discussion, we do not consider the workloads of the target hosts as well as their runtime cost for the service. Instead, we assume the target hosts are always available and have equal runtime cost rate. These factors, although important to determine the service migration, do not change the nature of the problem and can be easily processed without impact on our proposed algorithms.

4. DYNAMIC MIGRATION

In this section, we present our dynamic migration algorithms to service an online request sequence. We first consider the single-server migration and then extend it to the multi-server case. The frequently used symbols are summarized in Table I.

4.1. Single Server Migration

We concern ourselves with a single-server migration (i.e., $\kappa = 1$) and introduce a distributed migration algorithm, called *Dmig*. We first overview *Dmig* by defining some concepts and data structures, then introducing the basic idea, and finally describing the algorithm in a more formal way.

4.1.1. Some Concepts and Definitions. Given a batch-request σ_i and a server node v (the server is located at node v) at time t_i, the concepts used in the algorithm are defined in the following descriptions.

Table I. Notation Frequently Used in the Model and Algorithm

Symbol	Meaning
n	the number of nodes
δ	the minimum node degree
Δ	the maximum node degree
ω	$\omega = \Delta/\delta$
$Diam(G)$	the diameter of graph G
m	the number of batch requests
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
α	migration parameter ≥ 1
κ	the number of server replicas
σ	the given request sequence $\sigma = \sigma_1\sigma_2 \cdots \sigma_m$.
$\sigma_{\mathcal{E}}$	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
$w(v)$	access counter at node v
$d(v)$	profile recorder at node v
$q(v)$	the set of visited nodes in *Dmig* at node v
$c(a)$	the number of requests (not batch) accepted by access point a, which is recorded in $d(.)$
$\phi(r)$	r's service node determined by ϕ, the routing function

Local Knowledge. For the sake of simple presentation, we assume the following information is available to node v as its local knowledge[1]: (1) *Local Space* $\mathcal{N}(v)$, which is defined as node v's one-hop neighbors, (2) *Access Cost* $C_v = \{C_{uv} \vert \forall u \in V$ as an access point$\}$, and (3) *Migration Cost* $\beta_v = \{\beta_{vu} \vert u \in \mathcal{N}(v)\}$. Both C_v and β_v are pre-defined and provided by ISPs or computed implicitly by the algorithm. For example, if each hop has a constant cost, C_v can be computed by exploiting the IP source route information from v to the access point (or vice versa) for each request.

Epoch & Phase. The algorithm operates on a per-epoch basis along the time-line. The time-line is further divided into a sequence of phases. Each *phase* (except for the initial one) defines a migration followed by a period of time, called *time stage*, within which no migration is triggered to serve a sub-sequence of requests. As such, a phase is identified by the server node called *pivot node*, denoted by *p-node*, which is created at the beginning of the phase. The *epoch*, which is composed of one or multiple phases, is

[1]This information may depend on the parameters, such as the employed routing strategy, available bandwidth, and so on, which may not always be available from ISPs. However, in this article, we assume it is still desired for the ISPs to consider to disclose some infrastructure information in a certain extent as a trade-off to implement some new profitable application.

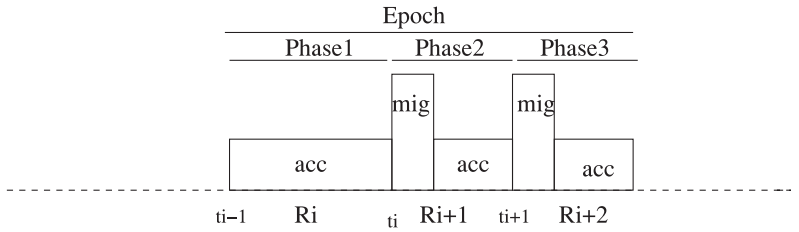


Fig. 2. The relationships between epochs, phases, and time stages in our algorithms.

delimited at certain time instance, when some properties are held by the neighbors of the p-node (discussed later). An example of the relationships between epochs, phases, and time stages is shown in Figure 2, where an epoch consists of three phases, *Phase1* spans across time stage $[t_{i-1}, t_i]$ without migration (the initial phase in an epoch). Whereas in *Phase2* and *Phase3* a migration is followed by a time stage.

Data Structure. For node v, the algorithm maintains two main data structures for each node in $\mathcal{N}(v)$ on a per-epoch basis. One is an *Access Counter* (AC), a scalar used to monitor and accumulate the access costs in an epoch. The other is a *Profile Recorder* (PR), which is a vector indexed by the request's access point to record the number of requests from each access point in the same epoch. During the service, the algorithm progressively accumulates and records (or samples) for the p-node v the access costs and the request profiles in the AC and PR at p-node, called *p-counter* and *p-recorder*, respectively.

4.1.2. Basic Idea. The essence of the algorithm is to leverage the *rent-or-buy paradigm* in the *ski rental problem*[2] to determine service migration and take advantage of a short historical access information to prune the local space for efficiently finding the migration target with a reduced service cost.

Movement Cost. The *movement cost* (i.e., the buy cost in the ski rental problem) is defined as the product of a migration parameter $\alpha \geq 1$ and the *maximum* migration cost from the source p-node v to a target neighbor in $\mathcal{N}(v)$, that is,

$$Cost_{mov}(v) = \alpha \cdot \max_{u \in \mathcal{N}(v)} \{\beta_{vu}\}. \tag{4}$$

The rationale behind parameter α is that the service movement cost should not merely rely upon the maximum migration cost from v but also weight on its location in the network. For example, given the same maximum migration cost, the service located at a hub node (high-degree node) should be more resilient against migration than those non-hub nodes. Therefore, we need a metric to measure the quality of nodes that can host the service. To this end, we select the *degree centrality* of node as migration parameter α to control the movement cost.[3] As a side effect, this parameter can also improve the stableness of the algorithm, which could prevent the algorithm from making *ping-pang* movements.

In contrast, the historical access information is gathered by the current phase during the service and used in the subsequent one or more phases to facilitate the server migrations with the total service cost reduction as a goal.

[2]Rent-or-buy paradigm is the name given to a class of problems in which there is a choice between continuing to pay a repeating cost or paying a one-time cost, which eliminates or reduces the repeating cost.
[3]Intuitively, the *closeness centrality* is better than the degree centrality to define the parameter, since it is closest to all others in terms of shortest paths. However, the node degree, as local knowledge, is much easier to obtain, compared to the closeness centrality.

How Algorithm Works. In each epoch the algorithm at p-node v (called the pivot algorithm for short) first leverages the rent-or-buy paradigm to determine the migration by comparing the access cost in its p-counter and the computed movement cost. If the access cost is less than the movement cost, then the p-node will be fixed at v to continually service the incoming requests until the value is greater than the movement cost. Otherwise, the pivot algorithm broadcasts its PR (received from the last pivot) to all its neighbor nodes in $\mathcal{N}(v)$. Each neighbor node will overwrite its own PR and mimic the service to the requests in its PR by accumulating the costs in its AC with its local information. After that, each neighbor node sends back its AC to node v, where the pivot algorithm compares the value of each neighbor's AC with that of the p-counter. Depending on the outcomes, two cases are distinguished to migrate the server, either virtually or physically.

(a) Virtual Migration: If there are some neighbors with AC values less than that of the p-counter, then the algorithm at pivot will randomly select a migration target from those with the minimum AC as a new p-node for the next phase and relay the p-recorder PR to the new phase. We call such a migration a *virtual migration*, since in this case we can repeat the same algorithm at the new target for the next, more preferable, location, rendering the new target to be only a temporary stop without needing to migrate the server physically.

(b) Physical Migration: Otherwise, if all the neighbors of the virtual p-node have the AC values greater than the virtual p-counter, the algorithm marks the end of the current epoch. In this case, the server will be physically moved to the current virtual node directly. In this situation, all the data structures will be reset for a new epoch, and the algorithm will be run from scratch as well. Additionally, when the value of the (virtual) p-counter of a node u is less than its movement cost $Cost_{mov}(u)$, it signifies the completion of a sequence of virtual phases. At this time, the algorithm will open up a new physical phase by informing the original p-node v to physically migrate the server to the new virtual p-node u in a direct way. Note that in this case, the data structures at u are not reset.

4.1.3. Dmig*: A Distributed Migration Algorithm.* The *Dmig* algorithm is fully symmetric, allowing each node to run the same algorithm. However, depending on the role of each node, either hosting the server (i.e., p-node) or not, the node will be performing different tasks in the algorithm. For each node v, we use $w(v)$ and $d(v)$ to represent its AC and PR, respectively. To facilitate the computation, we also define another variable $q(v)$ to collect the visited nodes in the algorithm. With these data structures and the local knowledge defined in Section 4.1.1, we design the *Dmig* algorithm in Figure 3. Since the p-node address can be piggybacked by each message, we assume it is readily available to each node, not showing in each message.

Given the understanding of its basic idea, it is not difficult to follow the algorithm, which runs in an endless loop and continually accepts seven types of messages: **Start**, **Request**, **VirtReq**, **AC**, **VirtMig**, **PhyMig**, and **Server**, each corresponding to a processing state. Therefore, the algorithm can be viewed as a finite-state machine.

Dmig initializes $w(v)$, $d(v)$, and $q(v)$ of each node v (Start) on a per-epoch basis in favor of migration as the p-counter would be increased quickly to exceed the movement cost. Of course, an alternative way is to reset the algorithm on a per-phase basis. However, this strategy bias toward a stationary server since compare to the on-per-epoch-basis strategy, it becomes harder for p-counter to accumulate sufficient access costs for server migration due to the high-frequency reset to the algorithm.

Each node v can receive either an incoming request σ_i or a batch of virtual requests that have been processed at source node. In the former case (**Request**), σ_i is gathered in $d(v)$ by increasing the counter values indexed by a_r for each $r \in \sigma_i$ (i.e., the semantics

Algorithm 4.1: MIGRATION()

local $C_{uv}, \mathcal{N}(v), \beta_{vu}$
Initialization :
$\begin{cases} w(v) \leftarrow 0,\ d(v) \leftarrow \varnothing,\ q(v) \leftarrow \varnothing,\ \alpha \leftarrow \deg(v) \\ Cost_{mov}(v) = \alpha \cdot \max_{u \in \mathcal{N}(v)}\{\beta_{vu}\} \end{cases}$
Message_Processing :
while $(msg \leftarrow \textbf{receive}(u))$
do
- **Start** :
  $w(v) \leftarrow 0,\ d(v) \leftarrow \varnothing,\ q(v) \leftarrow \varnothing$
- **Request**(σ_i) :
  $\begin{cases} \sigma_i \leftarrow msg.getReq() \\ w(v) \leftarrow w(v) + Cost_{acc}(v, \sigma_i) \\ d(v) \leftarrow d(v) \oplus \sigma_i \end{cases}$
  if $w(v) \geq Cost_{mov}(v)$
  then
  $\begin{cases} \textbf{for each } u \in (\mathcal{N}(v) \wedge v \notin q(v)) \\ \quad \textbf{do send}(u, VirtReq(d(v))) \end{cases}$
- **VirtReq**(**d**(**u**)) :
  $\begin{cases} d(v) \leftarrow msg.getD(d(u)),\ w(v) \leftarrow 0 \\ \textbf{for each } a \in d(v) \\ \quad \textbf{do } w(v) \leftarrow w(v) + c(a) \cdot C_{av} \\ u \leftarrow msg.getSrc(u) \\ \textbf{send}(u, AC(w(v))) \end{cases}$
  comment: gathers all the returned ACs
- **AC**(**w**(**u**)) :
  $\begin{cases} AC_u \leftarrow msg.getW() \\ q(v) \leftarrow q(v) \cup \{u\} \end{cases}$
  if $w(v) \geq \min_{u \in \mathcal{N}(v)}\{AC_u\}$
  then comment: Ties are broken randomly
  $\begin{cases} u \leftarrow \arg\min_{w \in \mathcal{N}(v)}\{AC_w\} \\ \textbf{send}(u, VirtMig(q(v))) \end{cases}$
  else comment: Epoch term.
  $\begin{cases} \textbf{send}(pivot, PhyMig(v)) \\ \textbf{goto Start} \end{cases}$
- **VirtMig**(**q**(**u**)) :
  if $w(v) < Cost_{mov}(v)$
  then
  $\begin{cases} \textbf{comment:} \text{ Phase term.} \\ \textbf{send}(pivot, PhyMig(v)) \end{cases}$
  else
  $\begin{cases} q(v) \leftarrow msg.getQ(q(u)) \\ \textbf{for each } u \in (\mathcal{N}(v) \wedge v \notin q(v)) \\ \quad \textbf{do send}(u, VirtReq(d(v))) \end{cases}$
- **PhyMig**(**u**) :
  $u \leftarrow msg.getTarget()$
  Send$(u, Server)$
  comment: Phy. mov. to target u
  SendSrv(u, ser)
- **Server** :
  comment: set v as the pivot
  $v.setpivot()$
  comment: Open up a new phase or epoch
  ReceiveSrv(ser, u)

Fig. 3. *Dmig* running on node v.

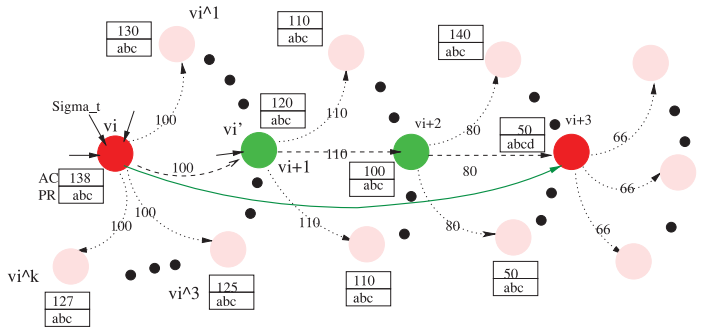


Fig. 4. A working example of *Dmig*. For each node, its AC and PR are also marked. The algorithm makes three virtual migrations, and the last one is turned into a physical migration because of a phase termination.

of $\oplus$ operation) and the access cost of σ_i is accumulated to $w(v)$. This process is repeated to serve the subsequent requests until $w(v) \geq Cost_{mov}(v)$ when according to the rent-or-buy paradigm, the server s at v should be migrated. Then, the algorithm tries to find a virtual target from its neighbor nodes by sending message `VirtReq(d(v))` to all the nodes $u \in \mathcal{N}(v) \wedge v \notin q(v)$ to gather their `AC` values in processing $d(v)$.

In contrast, the later case (**VirtReq**) is to process the received virtual sequence by aggregating the weighted access cost $c(a) \cdot C_{av}$ for each $a \in d(v)$ when server s is not at v, and the resulted `AC` is returned to the source node as the reply to `send(u,VirtReq)` in State `Request`.

Consequently, node v tries to gather all the `ACs` from its neighbor nodes (**AC**) and remember all the newly visited nodes. After that, it computes the node set with the minimum `AC` and randomly selects one of them as the next virtual target to move if its own $w(v) \geq \min_{u \in \mathcal{N}(v)}\{AC_u\}$. Finally, it notifies the next target by sending a `VirtMig` message, which includes all the so-far visited nodes. Otherwise, for node v if *all* its neighbors have `ACs` greater than its $w(v)$, the virtual migration process hits a wall to end up the current epoch. In this situation, a physical migration should be performed by sending a `PhyMig` message and a new epoch should be started.

When node v receives message `VirtMig`, it indicates that it has been selected by the source node as a virtual target (**VirtMig**). Therefore, its $w(v)$ and $d(v)$ have been computed in the last round. If $w(v) < Cost_{mov}(v)$, then the migration condition is not satisfied, and the current phase should be terminated by requiring a physical migration via message `PhyMig` from the p-node. Otherwise, the algorithm further considers those unvisited neighbor nodes of the virtual target by sending message `VirtReq`. Thus, this process `VirtReq`→`AC`→`VirtMig` can be repeated along the subsequent nodes until a phase or an epoch is terminated when a physical server migration is performed directly from the p-node.

Only p-node can react to message `PhyMig` as it hosts the physical sever (**PhyMig**). It gets the target node first and then sends a `Server` message to prepare it for receiving the server. The server is physically sent by `SendSrv`.

The process of `Server` message is relatively simple (**Server**). The node is first set itself as the pivot node and then receives the server by `ReceiveSrv`.

4.1.4. A Working Example of Dmig. A working example of the *Dmig* algorithm is shown in Figure 4, where the current phase is marked by the pivot configuration v_i. The algorithm serves σ_i and then makes migration a decision. In the example, the algorithm finds the value of the pivot counter (138 after serving `a`, `b`, and `c`) is greater than the movement cost (100) and in the meantime v_i' has the minimum access cost (120) to the requests (`abc`), which is less than the pivot counter (138). Then a virtual migration is

made from v_i to v_i', a new virtual phase starts marked by v_{i+1} (i.e., v_i'). As the virtual pivot recorder of v_i has been sent to v_{i+1} (i.e., abc), the same virtual migration process can be repeated at v_{i+1} (as $120 > 110$) to steer toward the next virtual target v_{i+2} with the minimum access cost (100) from which to the final virtual target v_{i+3}, where this virtual migration process cannot make any progress due to either the phase termination or the epoch termination (in our case, it's phase termination as $50 < 66$). In this situation, the algorithm at v_{i+3} turns the virtual phase into an *actual* one by directly migrating the server from v_i to v_{i+3}, thereby serving the next request d (PR=abcd). From this example, one can see that the *Dmig* algorithm can adapt to access patterns in a cost-effective way by allowing servers to be directly moved to a preferable location rather than the stepwise movements as in Oikonomou and Stavrakakis [2010].

4.1.5. Remark on Dmig.

Complexities and Properties. Since for each node v, $d(v)$ is indexed by the access points, we have $|d(v)| \leq n$. The message and time complexities of the algorithm thus largely depend on the *length* of the virtual migration path. For the message complexity, we have the following results:

THEOREM 4.1. *Given a static n-node network, the message complexity of* Dmig *is* $\Theta(n)$.

PROOF. As *Dmig* is crafted to allow each node connected to the virtual migration path to be only visited once for computing its AC, no circle could be constructed by the message links (i.e., the links with message transmission on them) in this process. Let's consider an ideal case that $\beta = 0$, and each node has at least one neighbor for the next moving step as well. We can observe that the total message links *at most* construct a *spanning tree* of the network, and the length of the virtual migration path is the height of the tree. Suppose the height of the tree has a length m, according to *Dmig*, there are exactly three messages (VirtReq, AC, and VirtMig) on each link of the path and two message on each of the other links (VirtReq, AC). The total number of the messages is thus at most $2(n + m) = O(n)$, which clearly also bounds the actual number of the messages.

Now, we consider a special case that all $n + 1$ nodes in the network are arranged in order on a line, and the server is located at node n, while all the requests are constantly made at the other end node 0. Given $\mu = 1$ and C_{uv} = the number of links between u and v. If β_{uv} is defined as

$$\begin{cases} \beta_{n(n-1)} = n^2 & i = 0; \\ \beta_{(n-i)(n-i-1)} < n(n-i) & 0 < i < n, \end{cases} \tag{5}$$

according to the algorithm, then it is not difficult to show that the server will be eventually moved from one end of the line to the other end along the virtual migration path if the migration cost is not a barrier. Therefore, the length of this path is at least n. Overall, we have the message complexity of $\Theta(n)$. □

Based on Theorem 4.1, the time complexity of the *Dmig* algorithm can be expressed by the following theorem,

THEOREM 4.2. *Given a static network with fixed rates of* μ *and* β_{uv} *for any neighbor node u and v, the time complexity of* Dmig *in migration decision is* $O(\omega n)$*, here* $\omega = \frac{\Delta}{\delta}$*,* Δ *and* δ *represent the maximum and minimum node-degrees of network graph* $G(V, E)$*, respectively.*

PROOF. With Lemma 4.1, we can easily show that the time complexity of the algorithm is only determined by computing the virtual migration path whose length is bounded by $Diam(G)$, and each node on the path will conduct at most $O(\Delta)$ operations to its

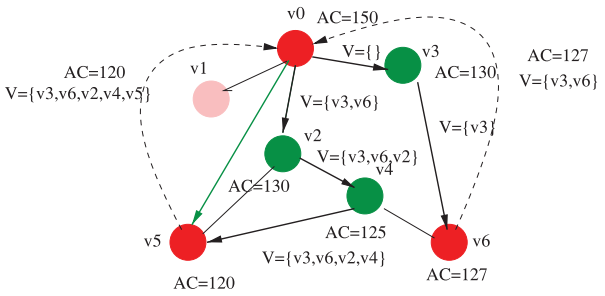


Fig. 5. *Dmig* refinement, searching all paths from the eligible neighbor nodes. Red nodes are terminal nodes (phase or epoch), green nodes are eligible nodes for the next migration step, and pink nodes are ineligible.

neighbors.[4] Therefore, the total complexity is at most $O(\Delta \cdot Diam(G))$. According to Moon's results [Moon 1965], $Diam(G) \leq \frac{3(n-1)}{\delta-1} - 3$. We have the conclusion that the time complexity is upper bounded by $O(\frac{\Delta}{\delta}n) = O(\omega n)$. □

Additionally, we also have the following properties regarding the algorithm.

LEMMA 4.3. *Given a tree-structured static network or a network with a unique shortest path tree, Dmig will eventually migrate the server along the optimal virtual path to the optimal location and remain there as long as the access pattern is not changed significantly.*

PROOF. According to the algorithm, when a new phase is created at time t by changing p-node v, there must exist a neighbor node in $\mathcal{N}(v)$ that has less AC value than node v. In other words, *Dmig* will always migrate the server to the particular neighbor of the p-node with less AC than is toward the optimal target node. This implies that the algorithm will never be stuck by AC for creating new epochs unless there is no further minimal neighbors (assuming there is no migration barrier due to the movement cost). Since there is a unique shortest path tree and the AC values of the nodes along the migration path are monontically decreased, the server will be eventually moved to the optimal node and remain there as long as the access pattern is not changed significantly. □

Although *Dmig* will eventually reach this optimality, it does not necessarily mean that migration paths of this algorithm are optimal, since the conducted virtual path might not be the optimal path to the optimal location.

A Variant of Dmig. In the algorithm, if multiple neighbor nodes were eligible for the next moving step, *Dmig* breaks the ties randomly, which might not eventually find a better target node for physical migration. An alternative way is to *recursively* search for the target node along *all* the possible paths starting from the eligible neighbor nodes. We call this refined algorithm recursive *Dmig*(*rDmig*).

Figure 5 is an illustrative example where two neighbor nodes v_2 and v_3 of p-node v_0 have equal AC(130). *rDmig* first searches from v_3 to v_6 (AC=127), gets back to v_0, and then continues to search from v_2 with the information gathered from the previous paths till v_5, where the search ends and returns back to v_0. After gathering all the candidate targets, the p-node can make a migration decision. In our case, v_5 has less AC and is better than v_6 to be the next target.

[4]Note that checking if an element is in a set can be done within $O(1)$ time.

By observing this example, we can also show that, in theory, the message and time complexity of this optimization is not increased if the algorithm can skip those visited nodes in the search process as before. However, there are still some practical messages overhead to trade off these benefits. We will evaluate the performance improvements of this refinement in Section 5.

4.2. Multiple Server Migration

With the understanding of the single-server migration, we can extend this problem to a more general case with κ sever replicas. There are two basic design choices, depending on whether the inter-server coordination is considered or not. If the inter-server coordination is considered to optimize the whole migration, then the co-migration algorithm would limit the scalability, because there could be a large amount of communication among the servers to jointly make a migration decision. In contrast, the algorithm that allows each server to perform the single-server algorithm independently would be very simple and highly scalable, although some optimality is scarified. However, given the service cost reduction of each independent server migration, we can still derive that the multi-servers as a whole could reduce the total service cost after the migration.

With these considerations, we propose a fully distributed algorithm called *mDmig* (short for multi-server *Dmig*), which simply allows the *Dmig* algorithm on each node to independently handle its server migration. This algorithm not only overcomes the limitation of the algorithm in Wang et al. [2015], where only a very limited number κ of servers are allowed to migrate under a centralized control, but also ensures the total cost reduction by the following theorem.

THEOREM 4.4. *In a static network, given a sequence of batch requests $\sigma = \sigma_1\sigma_2\cdots\sigma_m$, if mDmig changes configuration $\mathcal{L}_{i-1}$ to $\mathcal{L}_i$ by server migrations, then $Cost_{acc}(\mathcal{L}_{i-1}, \sigma) > Cost_{acc}(\mathcal{L}_i, \sigma)$.*

PROOF. Given a sequence of batch requests $\sigma = \sigma_1\sigma_2\cdots\sigma_m$, we consider its total access costs of $\mathcal{L}_{i-1}$ and $\mathcal{L}_i$, where $\mathcal{L}_i$ is derived from $\mathcal{L}_{i-1}$ by *mDmig* with server migrations. If we denote the requests served by $s_u \in \mathcal{L}_{i-1}$ as R_u and the requests served by the same server in $\mathcal{L}_i$ as R'_u, $1 \le u \le \kappa$, then for each server u, we have $R'_u = R_u \setminus A_u \cup B_u$, where A_u is those requests that are originally in R_u but now satisfied by other servers due to the server migrations from $\mathcal{L}_{i-1}$ to $\mathcal{L}_i$. Similarly, B_u is those requests that originally are not in R_u servers, but now are served by s_u. With these notions, we further have

$$\begin{aligned}\sum_{u=1}^{\kappa} Cost_{acc}(R'_u) &= \sum_{u=1}^{\kappa} Cost_{acc}(R_u \setminus A_u \cup B_u)\\ &= \sum_{u=1}^{\kappa} Cost_{acc}(R_u) - \sum_{u=1}^{\kappa}(Cost_{acc}(A_u) - Cost_{acc}(B_u)).\end{aligned} \tag{6}$$

Since A_u will be served by other servers with less costs than by s_u, and these costs are added by serving B_u, we have $\sum_{u=1}^{\kappa}(Cost_{acc}(A_u) - Cost_{acc}(B_u)) > 0$ due to $\cup_{u=1}^{\kappa} A_u = \cup_{u=1}^{\kappa} B_u$. Then $\sum_{u=1}^{\kappa} Cost_{acc}(\mathcal{L}_{i-1}, \sigma) > \sum_{u=1}^{\kappa} Cost_{acc}(\mathcal{L}_{i-1}, \sigma)$ when considering the service of σ. □

Theorem 4.4 indicates that *mDmig* will reduce the overall access costs of the request sequence by migrating the server replicas independently, as long as the access pattern is not significantly changed.

Table II. Profile of Different Network Topologies with 100 to 1000 Nodes Used in the Experiments

Size	Prof.	BA(3,3)	Lattice	ER(0.1)	Tree
100	Dist	2.6	6.67	2.25	6.29
	Deg	5.44	3.6	9.74	1.98
200	Dist	2.84	10.0	2.04	7.59
	Deg	5.68	3.7	19.2	2.0
400	Dist	3.13	13.33	1.92	9.2
	Deg	5.8	3.8	39.72	2.0
600	Dist	3.26	16.67	1.9	9.92
	Deg	5.86	3.84	60.56	2.0
800	Dist	3.88	20.0	1.9	10.52
	Deg	5.88	3.86	80.24	2.0
1000	Dist	3.44	23.33	1.9	10.92
	Deg	5.9	3.86	100.38	2.0

5. SIMULATION RESULTS

We evaluate the proposed algorithms through extensive simulation-based studies. To this end, we developed a simulator in Java to create network topologies, generate access patterns, and implement the migration algorithms in this article. The purposes of our evaluation are twofold: (1) to study the behaviour of the algorithms with respect to various impact factors, and (2) to show the cost effectiveness of our algorithms in service migrations compared to the selected reference algorithms.

5.1. Experimental Setup

5.1.1. Network Topology. We use networks with four typical topologies—*Tree(n)*, *Lattice(w,h)*, Erodös-Rényi (*ER(n,p)*) random graph [Erdös and Rényi 1959], and Barabási-Albert (*BA(n,e)*) graph [Barabási and Albert 1999]—to conduct simulation studies on the performance of the algorithms, each network connecting 100 to 1000 nodes and exhibiting different structural properties to represent a spectrum of communication networks [Oikonomou and Stavrakakis 2010; Li et al. 1999; Al-Fares et al. 2008; Pantazopoulos et al. 2011].

Both Tree and Lattice have strictly regular structures, allowing us to observe the behaviours of the algorithms under some extreme yet predictable conditions. 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.

The network profiles in these studies are shown in Table II, which summarizes the average distances between pairs of nodes and the average degrees for each type of the networks with different sizes.

As monetary cost is our primary concern, we do not explicitly model some properties and features of the networks, such as the network background traffic, bandwidth capacity, link latency, or CPU power. Instead, we assume these properties can be manifested themselves in the charging models. Therefore, we can focus squarely on modeling the request workloads and their access patterns.

5.1.2. Access Pattern. An access pattern is characterized by a sequence of online batch requests distributed across the network along time axis, each being specified by a time instance, a batch size, as well as distribution of access points and associated weights. In our experiments, we refine three types of access patterns that are often studied in literature [Pantazopoulos et al. 2011; Oikonomou and Stavrakakis 2010], each with different merits to evaluate the migration algorithms.

Uniform(p, q): To isolate the impact of the network topology, we first generate batch requests at a particular time instance by following uniform distribution for both batch size and request weight in the ranges of $[1, p]$ and $[1, q]$, respectively.

Zipf(p, ν, θ): To reflect the skewness among nodes (spatial skewness), we assume a uniform batch size on $[1, p]$ and a Zipf-like distribution among the nodes, characterized by a parameter $\theta \geq 1.0$, to capture the amount of weight skew of each requesting node, given the total number of requests ν.

Zone(p, ν, θ, ϰ): To model mobile access dynamics, we deliberately partition the network graphs into different $\varkappa$ zones by clustering the nodes according to nodeIDs. The nodes in different zones will make requests at the different time segments to mimic mobile accesses. In each zone, we also follow the Zipf-like distribution as above. Therefore, this pattern reflects both spatial skewness and temporal dynamics of the requests.

5.1.3. Charging Model. The charging model is defined and provided by underlying ISPs, and exploited by ICPs to optimize their service provisioning. In our particular case, the charging model includes the access model and the migration model. We assume that each link of a pair of nodes has an equal cost rate, η, then the access cost between a pair of nodes u and v can be defined as $C_{uv} = \eta D(u, v)$, where $D(u, v)$ is the hop-based length of the shortest path between u and v.

Although this model is not perfectly consistent with some existing models, such as those adopted by Amazon where C_{uv} is a constant, it is still feasible and practical in reality, especially for the services deployed in VPN-based private clouds. More importantly, the model can reflect some Quality of Service requirements, for example, the cost minimization always implies the reduction of request latency. On the other hand, the charging model for the cloud service is still an active research area [Ruiz-Agundez et al. 2011; Ma and Huang 2012; Woitaszek and Tufo 2010], which may likely experience changing and adjusting over time.

As discussed in the migration model, the migration cost between any pair of neighbor nodes u and v is given in advance by β_{uv}. However, for any pair of non-neighbor nodes u and v, any migration cost is feasible only if it is less than the total sum of the stepwise costs. In our experiments, we assume it is determined by the *maximum* of the migration costs along the path $D(u, v)$, that is, $\beta_{uv} = \max_{i \in [u,\ldots,v-1]}\{\beta_{i(i+1)}\}$, since the most expensive one is always a pragmatic concern in practice and considered in studies for service migration [Bienkowski et al. 2010; Arora et al. 2011a].

In all experiments, we fix the link cost $\eta = 2$ and $\mu = 5$, and we assume that the migration cost between a pair of neighbor nodes is uniformly distributed.

5.1.4. Reference Algorithms. To fully evaluate the proposed algorithms, we list some existing algorithms for comparison. The first five algorithms in Table III are compared for single-server migration. *Dmig* and its variant *Dmig′* are different, depending on whether or not the virtual migration (VM) is employed, so are *Dmig*(α) and its variant *Dmig′*(α). Thus, by comparing these algorithms, we can measure the benefits of the virtual migration. In both *Dmig* and *Dmig′*, we select node degree as α, but also conduct a parameter sweep study on α for *Dmig*(α) and compare it with *Dmig*. Note that given $\alpha = 0$, *Dmig′*(α) can be simply viewed as an improved *Migration Policy S* in Oikonomou and Stavrakakis [2010], where consecutive movements are performed by only considering the one-hop neighbors each time. *rDmig* is the recursive *Dmig*, the value of this optimization is also measured by comparing with *Dmig* and other reference algorithms. In contrast, *Migk* is an extension of the algorithm in Bienkowski et al. [2010] to handle the heterogeneous migration cost in a single server case, while *Migk′* is a variant of *Migk* to use β instead of β' to control the migration [Bienkowski et al. 2010], where $\beta = \max_{(u,v)\in E}\{\beta_{uv}\}$ and $\beta' = \min_{(u,v)\in E}\{\beta_{uv}\}$.

Table III. Compared Algorithms in the Experiments. The Standard Forms of the Proposed Algorithms are Marked

Algorithm	Specification
Dmig*	the standard *Dmig* in Section 4.1 where the virtual migration (VM) is employed.
Dmig'	a variant of *Dmig* without the VM.
Dmig(α)	the parameterized *Dmig* where $\alpha \in \mathbb{N}$.
Dmig'(α)	the parameterized *Dmig* without the VM.
rDmig	the recursive *Dmig* with the VM.
Migk	an extended algorithm in Bienkowski et al. [2010] where $\beta' = \min_{(u,v)\in E}\{\beta_{uv}\}$ is used to control migration.
Migk'	a variant of Migk using β instead of β' to control migration where $\beta = \max_{(u,v)\in E}\{\beta_{uv}\}$.
mDmig*	the standard multi-server *Dmig* with VM.
DP	the optimal off-line alg. in Wang et al. [2015].
SDP	the sampling DP alg. in Wang et al. [2015].

For multiple server migration, due to the lack of well-accepted reference algorithms in the literature, we measure the relative performance of *mDmig* with respect to some off-line algorithms in terms of cost ratio, which are shown in the last two algorithms, *DP* and *SDP*, proposed in Wang et al. [2015].

5.2. Results

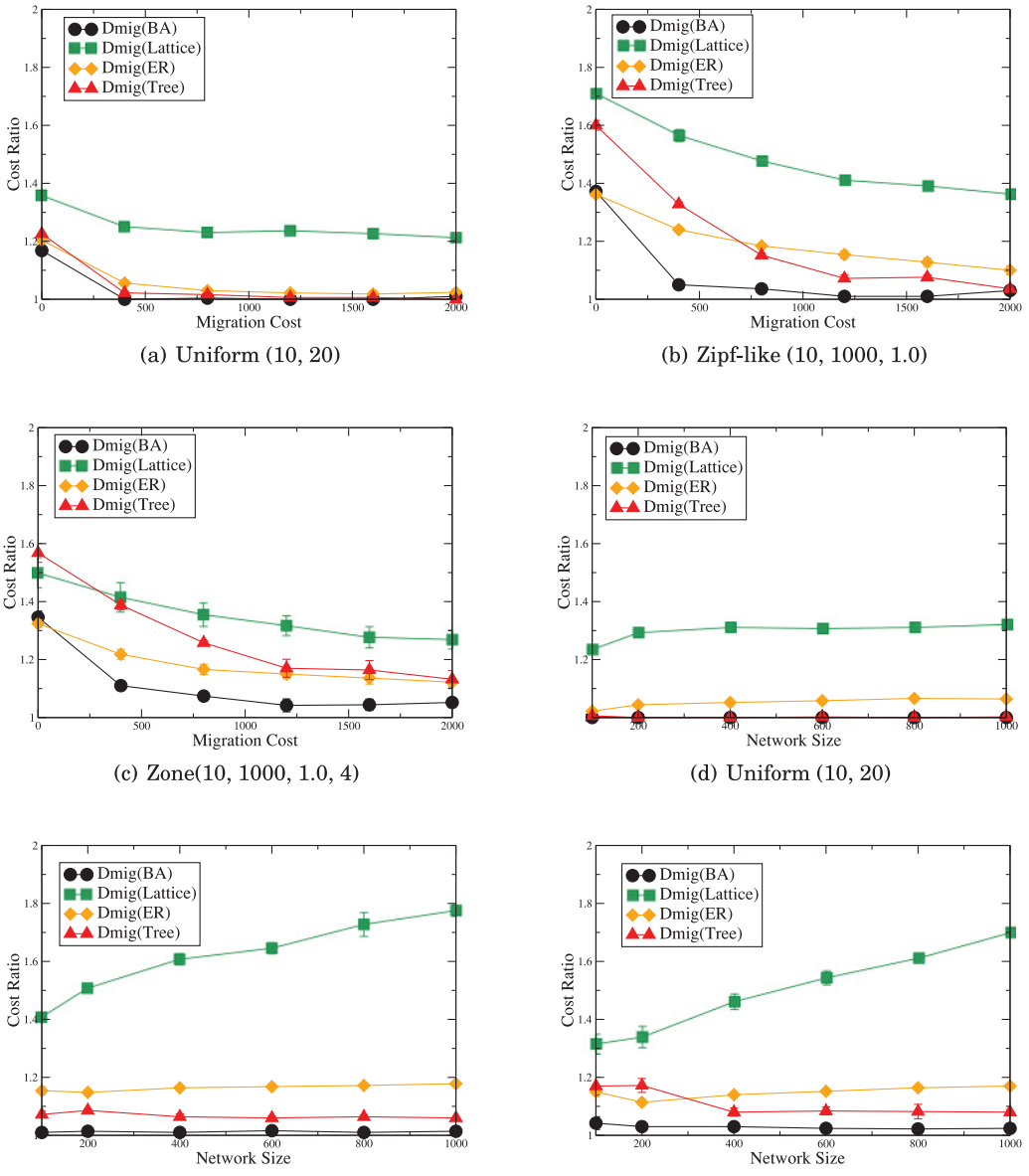
In this section, we show, in various cases, how our proposed algorithms behave and outperform the reference algorithms in service cost reduction. For the single-server migration, the major performance metric is defined as the ratio of the total service cost achieved by the compared algorithms over the cost achieved by the optimal off-line dynamic programming (DP) algorithm. However, for the multiple-server migration, the optimal solution is not feasible due to the configuration complexity. To address this problem, we implement the *SDP* algorithm and adopt it as the yardstick to measure the multi-server migration algorithms. The relative performance of *SDP* to the optimal DP is measured based on a set of small-size networks. To facilitate the understanding of the experimental results, other metrics are also defined in due course.

In the experiments, each data point in the graphs is averaged over five runs by changing the random number seed in the simulator, and its standard deviation is also computed.

5.2.1. Single-Server Migration. We study the behavior of *Dmig* for migrating a single server with respect to the changes of network topology, size, as well as access pattern. In addition, we also investigate the impact of migration parameter α on the service cost ratios.

Network Profile. The first set of experiments is to study how the network topology and size affect the performance of the *Dmig* algorithm. Figures 6(a)–6(c) show the impact of the network topology on the cost ratio of the algorithm when the network size is fixed as 100 nodes and the migration cost is uniformly changed from 0 to 2000. For all the studied topologies, the performance of the algorithm on BA shows the best, while on Lattice it is the worst. The performance on ER and Tree sits between. These observations are consistent across all the examined access patterns.

The BA graph exhibits the power-law degree distribution and short average inter-node distance (Table II). Thus, the server could migrate to a hub node (v) within a very limited number of moves. On the other hand, the migration control threshold of the node v, (i.e., $deg(v) \cdot \max\{\beta_{vu}\}$) is relatively large. The server is thus highly resilient against



(a) Uniform (10, 20)

(b) Zipf-like (10, 1000, 1.0)

(c) Zone(10, 1000, 1.0, 4)

(d) Uniform (10, 20)

(e) Zipf-like (10, 1000, 1.0)

(f) Zone(10, 1000, 1.0, 4)

Fig. 6. Impact of network topology, size, and access pattern on the total service costs of *Dmig*.

the migration. We can verify this by observing the *migration ratio*s of *Dmig* (i.e., # of online moves/# of off-line moves) in Table IV. Since the optimal off-line algorithm only incurs one move. The cost ratio of *Dmig* approaching to 1 exhibits a better performance. Compared to BA, ER also has short average inter-node distances but does not exhibit the power laws for the node degrees. Therefore, the migration ratio for the ER graph is higher than that of the BA network, although its value is still low.

Unlike BA and ER, both Lattice and Tree have relatively large average inter-node distances and low average node degrees (Table II). Although they have these similarities,

Table IV. Migration Ratios of *Dmig* with Respect to the Optimal Off-line Algorithm (Uniform Pattern)

	Migration Cost					
Topology	0	400	800	1200	1600	2000
BA(3,3)	0.99±0	0±0	0±0	0±0	0±0	0±0
Lattice	1.0±0	2.79±0.17	5.48±1.25	15.84±3.16	1.62±0.20	1.29±0.34
ER(0.1)	0.99±0	0.14±0.03	0.02±0.01	0.02±0.01	0.05±0.04	0.11±0.03
Tree	0.99±0	2.22±0.45	0.72±0.46	0.05±0.03	1.9±0.92	0.09±0.05

their cost ratios are different (Figure 6). For Lattice, due to its relatively high node degrees and long inter-node distances, the algorithm thus has more opportunities to select from a large target space, which could incur certain inferior migrations (Table IV), leading to the worst performance as well. In contrast, for Tree, these migration opportunities are reduced due to its unique path between any pair of nodes and thus result in a much better performance.

With each network size increasing from 100 to 1000, Figures 6(d)–6(f) show how the performance of the algorithm changes under different access patterns when the migration cost is fixed at 1200. Except for Lattice, the performance of the algorithm on other networks is nearly constant, independent of the network sizes. These observations are not surprising. As shown in Table II, for all the networks other than Lattice, the average inter-node distances are relatively stable. In contrast, the distance is steadily increased for the lattice network. Since we define the migration cost for *Dmig* as the maximum one-hop migration cost along the path, the migration cost between a pair of nodes will increase as length of the path increases.

Access Pattern. Figure 6 also compares the impact of the access pattern on the cost ratios of *Dmig* across all the examined networks. For the uniform pattern, the benefits of migration diminishes as the request weights are uniformly distributed among all the uniformly selected nodes. Except for the lattice network, we can see from Table IV that the ratios of server moves are quite small, allowing their performance curves to asymptotically approach to the optimal off-line results. The lattice network is a little bit difficult for the algorithm to achieve good results as we explained in the last paragraph.

Unlike the uniform pattern, both the Zipf and Zone-based patterns show some degrees of skewness in distribution of the request weights, rendering the migration to be beneficial in minimizing the service costs. As the request nodes for each batch are uniformly selected from the network, the Zipf pattern requires the algorithm to globally select vantage nodes for the migrations, which is usually difficult. In contrast, the Zone-based pattern only requires us to select the migration target from the current active zone unless the access pattern is changed to activate a different zone. As a result, *Dmig* has a slightly better performance for the Zone-based access pattern than that for the Zipf pattern, which is generally expected, in reality, as both the spacial skewness and temporal dynamics are commonplace. However, we should note that the performance changes for both patterns are not always consistent across all the studied networks. For example, the performance for the Zipf pattern on the tree network is slightly better than that for the Zone-based pattern. We attribute this phenomenon to the mismatch between the tree structure and the partition method used.

Migration Parameter (α). The results in this subset of experiments show how the migration parameter α affects the overall performance of the *Dmig* algorithm and thereby demonstrates the value of using the node degrees as the parameter. To this end, we conduct a parameter sweep study by first changing α from 1 to 10, which cover the average degree of each network graph, and then comparing *Dmig* with *Dmig*(α). Figure 7 shows the comparison results when β_{uv} is uniformly distributed in [1, 1200].

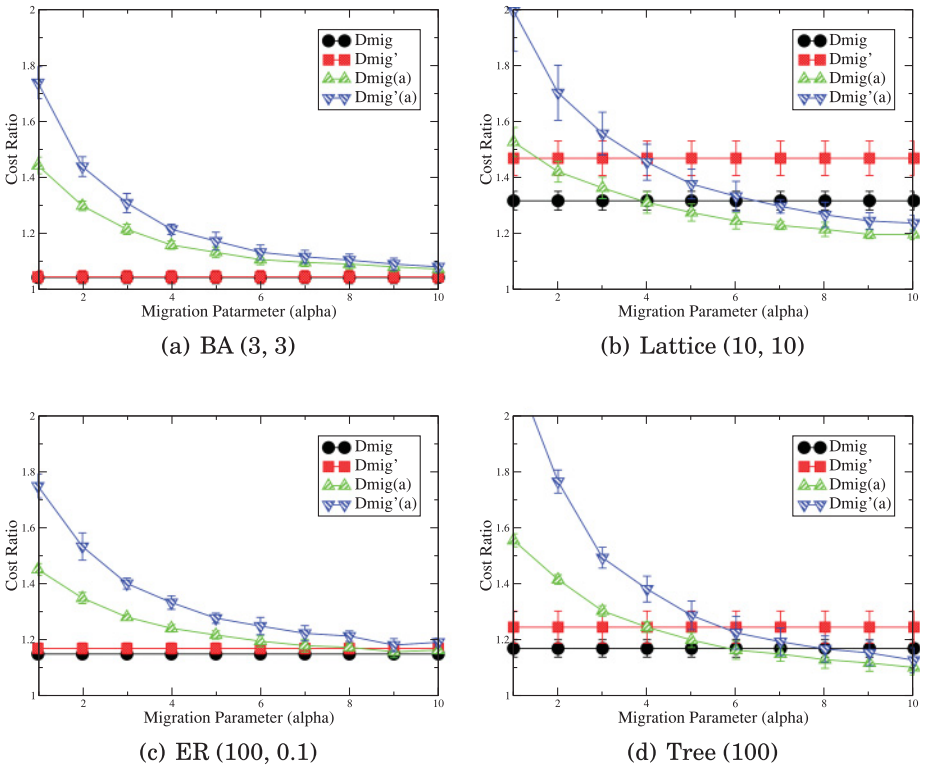


Fig. 7. Impact of migration parameter α on the performance of Dmig, Dmig(α), Dmig$'$, and Dmig$'$(α) for Zone(10, 1000, 1.0, 4) access pattern, respectively (β_{uv} is uniformly distributed in [0, 1200]).

From this figure, we can see that in most cases across all the network graphs and access patterns (only the Zipf pattern is shown), *Dmig* exhibits the best performance, indicating using the node degree as the parameter is reasonable for the algorithm to achieve good performance.

Performance Comparison. The performance comparison of *Dmig* with other selected reference algorithms is shown in Figure 8. The purposes of this comparison are threefold. First, by comparing *Dmig* with *Dmig*$'$(0) and *Migk*, we can evaluate its performance relative to some existing algorithms [Oikonomou and Stavrakakis 2010; Bienkowski et al. 2010]. Second, by comparing with *Dmig*$'$, we can measure the benefits of the virtual migration in service cost reduction. Finally, by comparing with *rDmig*, we can assess if the recursive search presented in Section 4.1 is worthwhile to optimize the performance.

From Figure 8, *Dmig* significantly outperforms *Dmig*(α), an optimized version of the *Migration Policy S* in Oikonomou and Stavrakakis [2010], when $\alpha = 0$, for all the studied networks, as the migration cost is not considered in the reference algorithm, it could result in a large number of expensive migrations, especially when the migration cost is high. In contrast to *Dmig*(α), the relative performance of *Dmig* to *Migk*, and its variant *Migk*$'$, is not consistent across the different networks. The figure shows that *Dmig* is better than or competitive with *Migk* or *Migk*$'$, whichever is the best for all examined networks and access patterns (only the Zipf pattern is shown). These results again demonstrate the advantages of *Dmig* over the reference algorithms to migrate servers in cloud environments.

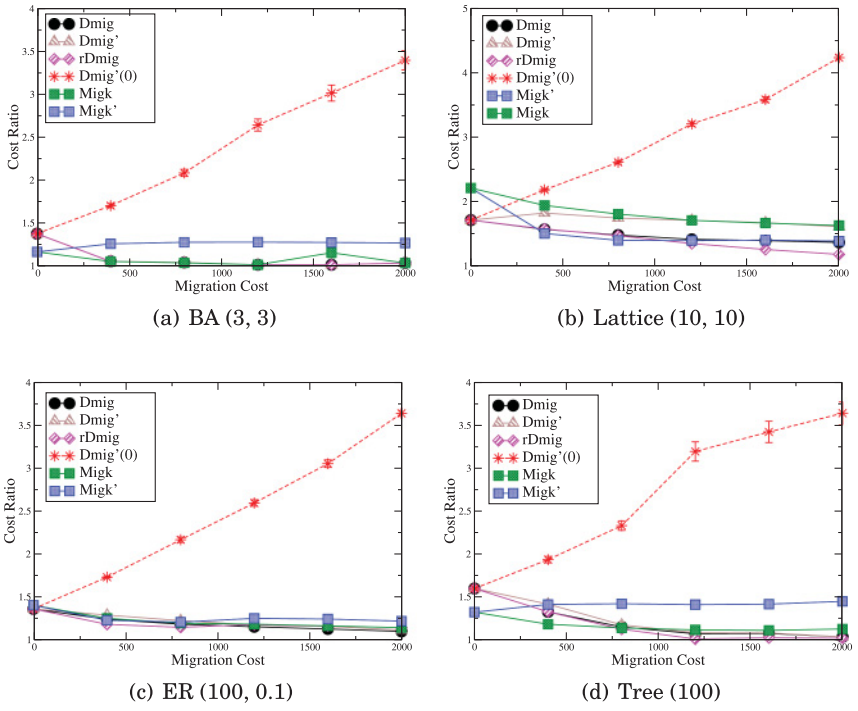
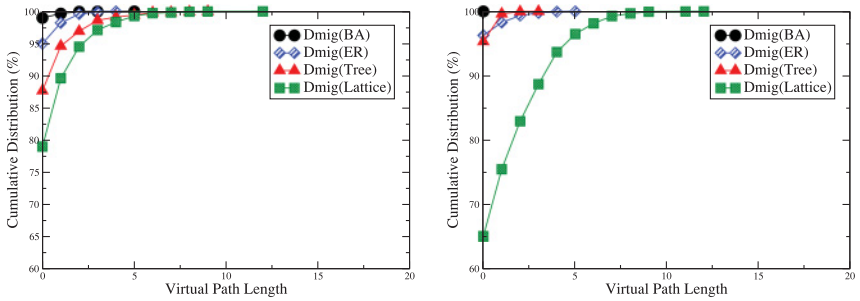


Fig. 8. Performance comparison of *Dmig* with other reference algorithms for Zipf(10, 1000, 1.0) access pattern.

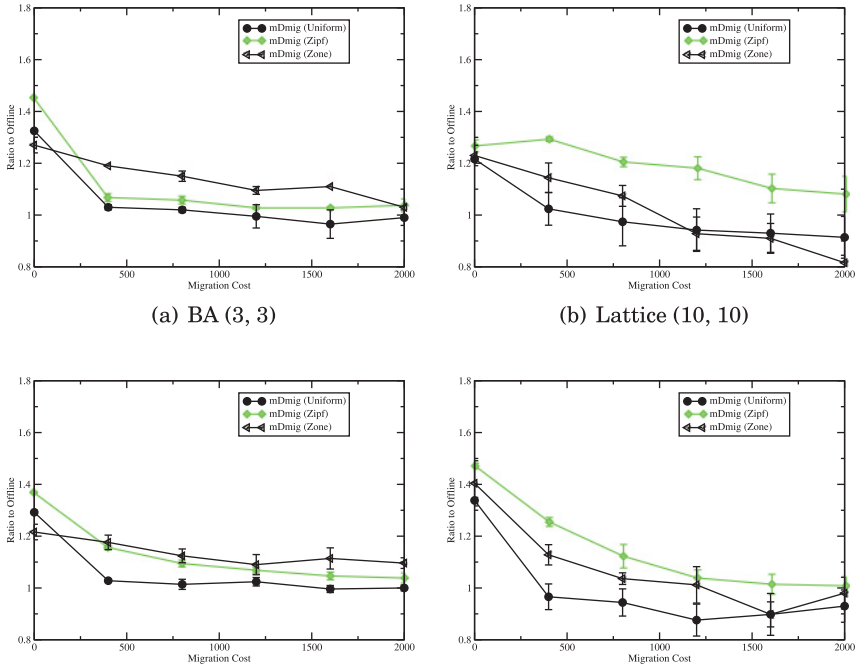
Another interesting observation is the performance comparison when the migration cost is zero. According to *Migk* (also *Migk′*), the epoch is reset after every batch request. As a result, the server will be fixed without any migrations. In contrast, *Dmig* (also *Dmig*(α)) will lose its phase control, and the server can be freely migrated. The performance results of these two extreme cases show that depending on the network topology, either migratable server or fixed server can achieve relatively better performance, no one is constantly better than the other. For example, it would be much better to fix the server at certain hub node in the BA network than to move it around to minimize the service cost. On the contrary, due to the large average inter-node distances, it is much preferable to migrate the server in the lattice networks, rather than to fix it, to achieve better performance.

The benefits of the virtual migration are also shown in Figure 8 (amplified in Figure 7), with respect to the growth of migration parameter α (1 to 10) and migration cost (0 to 2000), respectively. In the figure, *Dmig* and *Dmig*(α), together with their non-virtual migration versions (i.e., *Dmig′* and *Dmig′*(α)) are compared. One can easily observe the performance gaps between whether or not the virtual migrations are used for both *Dmig* and *Dmig*(α). Depending on the network topology and migration cost, such a gap can be as large as up to 10.20% for Lattice (Figure 7(b)) and 28.44% for Tree (Figure 7(d)), respectively, illustrating the value of the virtual migrations. These benefits are further demonstrated by the cumulative distribution of the virtual path lengths for the Zipf and Zone-based access patterns, which are shown in Figure 9. For both cases, Lattice exhibits the longest average virtual path, while BA has the shortest one, which is consistent with and further explains our previous performance results.



(a) Zone(10,1000,1.0,4) (b) Zipf(10,1000,1.0)

Fig. 9. Distributions of virtual path (vpath) lengths for different network topologies (100 nodes), vpath=0 means the server is fixed without migrations.

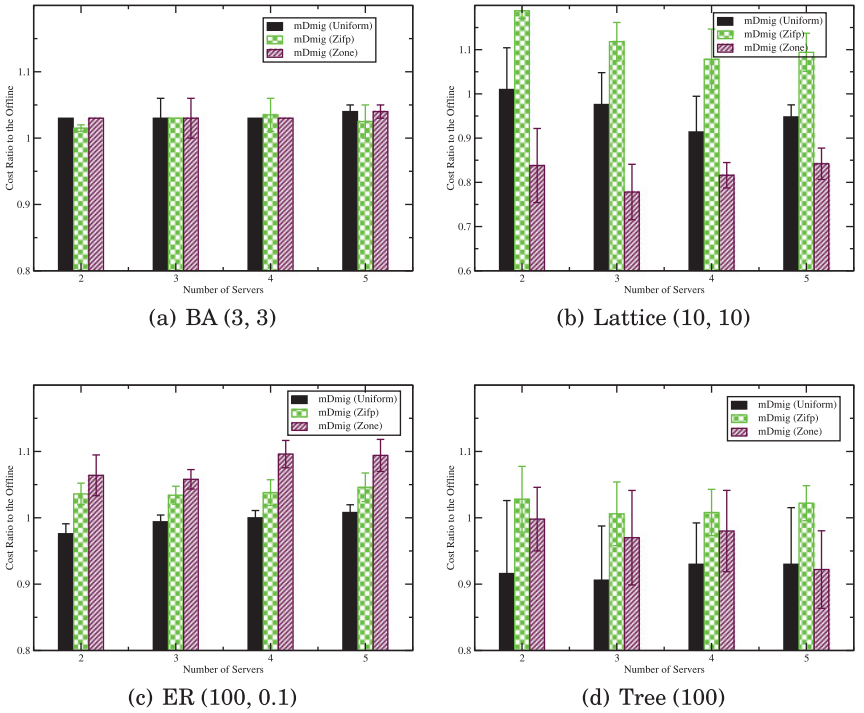


(a) BA (3, 3) (b) Lattice (10, 10)

(c) ER (100, 0.1) (d) Tree (100)

Fig. 10. Performance changes of *mDmig* with respect to the migration cost when the number of servers is fixed as four.

Unlike the virtual migration, which allows the server to bypass the intermediate sub-optimal nodes along the migration paths and quickly converge to a vantage (or optimal) node, the value of recursive search for finding the migration target is marginal, even though it is slightly better than the random selection. We validate it by comparing the cost ratios of *Dmig* and *rDmig* in Figure 8. The observation is reasonable as the recursive process is always to find an alternative path toward the vantage node. However, the quality of each such path should not be very different if the access pattern is not dramatically changed. Consequently, the marginal added-value of *rDmig* renders it to be not worthwhile to deploy in practice.



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

(c) ER (100, 0.1) (d) Tree (100)

Fig. 11. Performance changes of *mDmig* with respect to the the number of servers when the migration costs are uniformly distributed in [0, 2000].

5.2.2. Multiple Server Migration. In this section, we evaluate the algorithms for migrating multiple servers by comparing the cost ratio with the baseline *SDP*, which is on average within ≈1.20% of the optimal DP results in our small-scale test cases [Wang et al. 2015].

Figure 10 shows the results of *mDmig* with respect to the changes of migration costs for the selected access patterns on the studied networks when the number of servers is fixed as four. From this figure, one can easily observe that *mDmig* exhibits better performance as the migration cost increases for all types of access patterns across the different networks. This observation is understandable; as the migration cost increases, the migration opportunities for *mDmig* are reduced accordingly (i.e., the number of migrations that can be exploited to optimize the goal is decreased).

In addition to the migration costs, we also studied how *mDmig* behaves with respect to the number of virtual servers. Given the size of networks as 100 nodes, we studied the algorithm by varying the number of the servers from 2 to 5. The results are shown in Figure 11 when the migration cost is uniformly distributed between 0 and 2000. We can see that in all the cases, the cost ratio between *mDmig* and *SDP* is at most 1.2, showing the performance advantage of *mDmig*. Note that in some cases (e.g., uniform on Tree), it could be possible for *mDmig* to outperform the off-line algorithm, since the baseline algorithm is sup-optimal.

6. CONCLUSIONS

In this article, we formulated and studied the service migration problem in the cloud platforms. To this end, we first developed an efficient distributed algorithm *Dmig* for a single-server migration, which is fully symmetric, scalable, and easily deployed in practice. The algorithm is distinct from existing ones by its effective use of historical

access information to conduct virtual migration to minimize the total service costs. The essence of the virtual migration is to quickly find an optimal target node by mimicing the service to the requests at each temporary node on the multi-hop path. Given its distributed nature, we then extended *Dmig* to a multi-server situation and proposed a fully distributed algorithm called *mDmig*, where each server performs the same algorithm independently to achieve high scalability. Our extensive simulation results showed that, compared with several existing algorithms, the proposed algorithms can significantly reduce the overall service cost for all the studied access patterns made on the given representative networks in the cloud.

REFERENCES

Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM'08)*. 63–74.

Samer Al-Kiswany, Dinesh Subhraveti, Prasenjit Sarkar, and Matei Ripeanu. 2011. VMFlock: Virtual machine co-migration for the cloud. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. 159–170.

Dushyant Arora, Marcin Bienkowski, Anja Feldmann, Gregor Schaffrath, and Stefan Schmid. 2011a. 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 (IPTcomm'11)*. 10:1–10:11.

Dushyant Arora, Anja Feldmann, Gregor Schaffrath, and Stefan Schmid. 2011b. 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*. 2–2.

A. L. Barabási and R. Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439 (Oct. 1999), 509–512.

Marcin Bienkowski, Anja Feldmann, Johannes Grassler, Gregor Schaffrath, and Stefan Schmid. 2014. The wide-area virtual service migration problem: A competitive analysis approach. *IEEE/ACM Trans. Netw.* 22, 1 (Feb. 2014), 165–178.

Marcin Bienkowski, Anja Feldmann, Dan Jurca, Wolfang Kellerer, Gregor Schaffrath, Stefan Schmid, and Joerg Widmer. 2010. Competitive analysis for service migration in VNets. In *Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures (VISA'10)*. ACM, New York, 17–24.

Raouf Boutaba, Qi Zhang, and Mohamed Faten Zhani. 2014. Virtual machine migration in cloud computing environments: Benefits, challenges, and approaches. In *Communication Infrastructures for Cloud Computing*, Hussein T. Mouftah and Burak Kantarci (Eds.). IGI Global, Hershey, PA, 383–408.

Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. 2007. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE'07)*. 169–179.

Moses Charikar, Dan Halperin, and Rajeev Motwani. 1998. The dynamic servers problem. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'98)*. 410–419.

N. M. Mosharaf Kabir Chowdhury and Raouf Boutaba. 2010. A survey of network virtualization. *Comput. Netw.* 54, 5 (April 2010), 862–876.

M. Chroboak, H. Karloff, T. H. Payne, and S. Vishwanathan. 1991. New results on server problems. *SIAM J. Discr. Math.* 4 (1991), 172–181.

Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. 2006. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.* 1, 2 (Dec. 2006), 223–259.

P. Erdös and A. Rényi. 1959. On random graphs I. *Publicat. Mathemat.* 6 (1959), 290–297.

Reza Zanjirani Farahani and Masoud Hekmatfar. 2009. *Facility Location: Concepts, Models, Algorithms and Case Studies (Contributions to Management Science)*. Physica, 1st ed. 557 pages.

Reza Zanjirani Farahani, Maryam SteadieSeifi, and Nasrin Asgari. 2010. Multiple criteria facility location problems: A survey. *Appl. Math. Model.* 34, 7 (2010), 1689–1709.

Zachary Friggstad and Mohammad R. Salavatipour. 2011. Minimizing movement in mobile facility location problems. *ACM Trans. Algor.* 7, 3 (July 2011), 28:1–28:22.

Simon Görtz and Andreas Klose. 2012. A simple but usually fast branch-and-bound algorithm for the capacitated facility location problem. *INFORMS J. Comput.* 24, 4 (2012), 597–610.

Kamal Jain and Vijay V. Vazirani. 2001. Approximation algorithms for metric facility location and k-Median problems using the primal-dual schema and Lagrangian relaxation. *J. ACM* 48 (March 2001), 274–296. Issue 2.

Hsu-Fang Lai, Yu-Sung Wu, and Yu-Jui Cheng. 2013. Exploiting neigborhood similarity for virtual machine migration over wide-area network. In *Proceedings of the IEEE 7th International Conference on Software Security and Reliability (SERE'13)*. 149–158.

Bo Li, M. J. Golin, G. F. Italiano, Xin Deng, and K. Sohraby. 1999. On the optimal placement of web proxies in the internet. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, Vol. 3. 1282–1290.

Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu. 2009. 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 (HPDC'09)*. 101–110.

Dan Ma and Jianhui Huang. 2012. The pricing model of cloud computing services. In *Proceedings of the 14th Annual International Conference on Electronic Commerce (ICEC'12)*. 263–269.

M. S. Manasse, L. A. McGeoch, and D. D. Sleator. 1988. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*. 322–333.

J. W. Moon. 1965. On the diameter of a graph. *Mich. Math. J.* 12, 3 (1965), 349–351.

Konstantinos Oikonomou and Ioannis Stavrakakis. 2010. Scalable service migration in autonomic network environments. *IEEE J. Sel. A. Commun.* 28, 1 (Jan. 2010), 84–94.

Panagiotis Pantazopoulos, Merkourios Karaliopoulos, and Ioannis Stavrakakis. 2011. Centrality-driven scalable service migration. In *Proceedings of the 23rd International Teletraffic Congress (ITC'11)*. 127–134.

Dung H. Phan, Junichi Suzuki, Raymond Carroll, Sasitharan Balasubramaniam, William Donnelly, and Dmitri Botvich. 2012. Evolutionary multiobjective optimization for green clouds. In *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation*. 19–26.

Pierre Riteau, Chritine Morin, and Thierry Priol. 2013. Shrinker: Efficient live migration of virtual clusters over wide area networks. *Concur. Comput.: Pract. Exp.* 25, 4 (2013), 541–555.

Igor Ruiz-Agundez, Yoseba K. Penya, and Pablo G. Bringas. 2011. A flexible accounting model for cloud computing. In *Proceedings of the 2011 Annual SRII Global Conference (SRII'11)*. 277–284.

Chaitanya Swamy and Amit Kumar. 2002. Primal-dual algorithms for connected facility location problems. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*. 256–270.

Xiaoying Wang, Xiaojing Liu, Lihua Fan, and Xuhan Jia. 2013. A distributed virtual machine migration approach of data centers for cloud computing. *Math. Problems Eng.* 2013 (2013), 10.

Yang Wang, Wei Shi, and Menglan Hu. 2015. Virtual servers co-migration for mobile accesses: Online versus off-line. *IEEE Trans. Mobile Comput.* 14, 12 (Dec 2015), 2576–2589.

Yang Wang, Wei Shi, and Lingfang Zeng. 2013. Adaptive search-based service migration with virtual moves in clouds for mobile accesses. In *Proceedings of the Conference on Utility and Cloud Computing (UCC'13)*.

Matthew Woitaszek and Henry M. Tufo. 2010. Developing a cloud computing charging model for high-performance computing resources. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology (CIT'10)*. 210–217.

Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. 2008. Rethinking virtual network embedding: Substrate support for path splitting and migration. *SIGCOMM Comput. Commun. Rev.* 38, 2 (2008), 17–29.

Received September 2016; revised December 2016; accepted January 2017