2016 IEEE 22nd International Conference on Parallel and Distributed Systems

On Autonomous Service Migrations in the Cloud for Mobile Accesses

Yang Wang[†], Shuibing He[‡], Fuji Ren[§] Lujia Wang[†], and Chengzhong Xu[†♯]
[†]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, China
[‡]School of Computer, Wuhan University, China
[♯]Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202.
[§]Faculty of Engineering, University of Tokushima, 2-1 Minami-Josanjima, Tokushima 770-8506, Japan
{yang.wang1,cz.xu}@siat.ac.cn, heshuibing@whu.edu.cn, ren@is.tokushima-u.ac.jp

***Abstract*—We study the problem of autonomous service migration in the cloud to satisfy an online sequence of mobile batch-request demands in a cost-effective way. As the origins of the mobile accesses frequently change over time, this problem is particularly important for time-bounded services to achieve enhanced QoS and cost effectiveness. Moving the service closer to its client locations not only reduces the service access latency but also minimizes the network costs for service providers. However, the migration comes at costs of bulk-data transfer and service disruption, as a result, increasing the overall service costs. To gain the benefits of service migration while minimizing the service costs, we propose an efficient search-based algorithm *Dmig* the service migration in an autonomous way. Compared with existing algorithms, the proposed algorithm is fully distributed, symmetric, and characterized by the effective use of historical access information to perform virtual migration that overcomes the limitation of traditional local search in cost reduction. To evaluate the algorithm, we compared it with some existing algorithms, and show that the proposed algorithm exhibits better performance by adapting to the changes of mobile access patterns in a cost effective way.**

***Index Terms*—cloud computing; dynamic service migration; mobile access; dynamic virtual machine placement; virtual migration**

I. Introduction

Cloud computing for mobile world is becoming a well-accepted technique that enables 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. For example, the mobile service requests are typically sensitive to access latency. Moreover, they are always changing with respect to time and locations of mobile users. As such, providing network services without considering these factors may significantly increase access delays and, much worse, impose a large amount of network traffic which may cause service disruption and performance degradation. As a result, traditional technologies, such as those used in the *Content Distribution Network* (CDN) to fix the service in a set of carefully selected locations [1]–[3], 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 to minimize the access latency and reduce the network costs for service providers. A typical example to illustrate the migration benefits is a multi-player mobile game. The 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 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 for the service deployment with the aforementioned benefits.

Although the wide-area live VM migration, including server memory image and associated data files, remains expensive to use because of the bandwidth bottleneck, it is still feasible with some advanced technologies to minimize the migration overhead [4]–[8]. For example, R. Bradford *et al.* [4] 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 minimal disruption – 3 seconds in the LAN and 68 seconds over the WAN. This impact was further reduced in the later studies by exploiting different features of the migration [5]–[8]. With these technologies, several preliminary results have demonstrated the benefits of the service migration over virtual networks and autonomic networks [9], [10]. However, the trade-off between the benefits and the costs (from the monetary cost point of view) of 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 the profits of their cloud services.

In this paper, we investigate this problem and propose an autonomous migration algorithm based on local search techniques (**Contribution 1**). Compared with some existing work [9], [10], the proposed algorithm is fully distributed, symmetric, and characterized by the effective use of historical access information to conduct virtual migrations that overcome the limitation of traditional local search in cost reduction (**Contribution 2**). We showed that, via simulation studies, the proposed algorithm exhibits better performance in service

1521-9097/16 $31.00 © 2016 IEEE
DOI 10.1109/ICPADS.2016.101

migration by adapting to the changes of mobile access patterns in a cost-effective way (**Contribution 3**).

The remainder of this paper is organized as follows. We review some related work in Section II, and describe the service migration problem, together with its algorithm, *Dmig*, in Section III. We present the simulation results in Section V, and conclude the paper in Section VI.

II. Related Work

Service migration has been studied in the context of *virtual networks* (VNet) [11], [12] to minimize the service access latency. Bienkowski *et al.* [9] presented a randomized online algorithm to migrate a single server in 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 [13]. Unlike their research, our work concentrates on the performance of distributed approaches in cloud environments.

As with in VNet, the service migration is also studied in *autonomic network environments* [14] as a self-managing mechanism to overcome the rapidly growing complexity of networks. Oikomomou *et al* [10] 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, it suffers from slow convergence due to the inefficient one-hop migration. Pantazopoulos *et al.* [15] overcame this downside in their centrality-driven migration algorithm, named *cDSMA*. However, this algorithm lacks the notion of migration cost in the cloud.

Although our algorithm for single-server migration is still based on the local search techniques, it takes into account both the access and the migration costs while accelerating the migration process via the developed *virtual migrations*. On the other hand, unlike cDSMA, our distributed algorithm, *Dmig*, is fully symmetric with a low message complexity of $\Theta(n)$.

In contrast to the existing studies which are not directly conducted in the cloud, Phan *et al.* [16] propose a framework, called *Green Monster*, to leverage dynamic service migration across the Internet data centers (IDCs) for the energy efficiency. A similar effort is made by Wang *et al.* [17] who presents a decentralized approach to virtual machine migration inside data center for energy saving while maintaining the quality of service. The difference between our results with these two is that we focus on the service migration strategies to minimize the total monetary access cost for CSPs, instead of developing a new migration mechanism.

A more comprehensive study on virtual machine migration in the cloud environments in regards of the benefits, challenges, as well as approaches can be found in a most recent work [18].

TABLE I: Notation frequently used in model and algorithm descriptions

Symbol	Meaning
n	the number of nodes
m	the number of batch requests
C_{uv}	transmission cost between node u and v
C_v	$C_v = \{C_{uv} \mid \forall u \in V \text{ as a connect point}\}$
β_{uv}	migration cost between node u and v
β_v	$\beta_v = \{\beta_{vu} \mid u \in \mathcal{N}(v)\}$
β	$\beta = \max_{u,v \in V}\{\beta_{uv}\}$
β'	$\beta' = \min_{u,v \in V}\{\beta_{uv}\}$
μ	wireless link cost
α	migration parameter ≥ 1
σ	the given 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 by access point a which is recorded in $d(.)$
$\phi(r, \mathcal{L})$	r's service node determined by ϕ given configuration $\mathcal{L}$

III. Service Migration Model

We consider an arbitrary n-node network $G(V, E)$ as a service infrastructure where the provided service is running in a virtual machine (VM) that could be hosted by any compute node. The service is accessed by a sequence of batch requests $\sigma = \sigma_1\sigma_2...\sigma_m$ issued from a set of external machines (i.e., mobile terminals). The requests arrive in an online order and are served in turn by triggering the migration of virtual server. As a result, the location of the server in the network could frequently change over time.

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

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 through 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 the service via the underlying *routing function* determined by ISPs. As a consequence, the total access cost of batch request σ_i can be simply written as

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

where a_r is request r's nearest access point and $\phi(r, \mathcal{L})$ is the service node of request r 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 $\sum_{r\in\sigma_i} C_{a_r\phi(r,\mathcal{L})}$ define the cost for σ_i to access the service 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 a virtual server usually cannot be neglected. Unlike the access cost 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 target $u, u \in \mathcal{N}(v)$ where $\mathcal{N}(v)$ is the neighbor set of v.

Migration Goal: Let $\mathcal{L}_i$ denote the location of the virtual server at time stage i, then for a sequence of batch requests $\sigma = \sigma_1\sigma_2...\sigma_m$, the goal of the service migration is to determine $\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_t$ to minimize the total service cost defined as

$$\begin{aligned} Cost(\mathcal{L}_i) =& Cost(\mathcal{L}_{i-1}) + Cost_{acc}(\mathcal{L}_{i-1}, \sigma_i) \\ &+ Cost_{mig}(\mathcal{L}_{i-1}, \mathcal{L}_i). \end{aligned} \tag{2}$$

where $Cost(\mathcal{L}_0) = 0$.

Note that in this model, we do not consider the workloads of target hosts as well as their runtime costs for the service since these factors, although relevant to the service migration in practice, have no impact on how the model reflects the problem and how the proposed algorithm works.

IV. Autonomous Migration

We now describe our autonomous migration algorithm, called *Dmig*, for a single server migration. The single-server case is common in practice and has been studied in different circumstances [9], [10], [15], [19] as a baseline case. We first overview *Dmig* by defining some concepts and data structures, then introducing the algorithm itself, followed by a running example and some remarks. The frequently used symbols are summarized in Table I.

A. Some Concepts and Definitions

Local Knowledge: For the sake of simple presentation, we assume the following information is available to node v as its local knowledge: 1) *Local Space* $\mathcal{N}(v)$ which is defined as node v's one-hop neighbors; 2) *Access Cost* C_v=$\{C_{uv} | \forall u \in V \text{ as a connect point}\}$; and 3) *Migration Cost* β_v=$\{\beta_{vu} | u \in \mathcal{N}(v)\}$. Both C_v and β_v are pre-defined and provided by ISPs.

Epoch & Phase: The algorithm operates on a per-epoch basis along the time-line, which is 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 node hosting the server, called *pivot node* denoted by *p-node*, which is created at the beginning of the phase. The *epoch* composed of one or multiple phases is 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 epoch, phase and time stage are shown in Fig. 1 where an epoch consists of three phases; *Phase1* spans across time stage $[t_{i-1}, t_i]$ without migration (the initial phase in epoch), whereas in *Phase2* and *Phase3*, a migration is followed by a time stage.

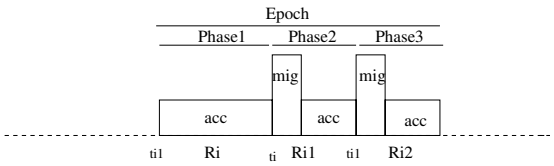


Fig. 1: The relationships between epoch, phase and time stage in our algorithm.

Data Structures: 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 at p-node v progressively accumulates and records the access costs and the request profiles in *AC* (i.e., *p-counter*) and *PR* (i.e., *p-recorder*), respectively.

B. The Algorithm

The essence of the algorithm is to leverage the rent-or-buy paradigm [1] to determine service migration and take advantage of a short historical access information to prune the local space for efficiently finding migration target with a reduced service cost.

Movement Cost: The *movement cost* in our design is defined as the following:

$$Cost_{mov}(v) = \alpha \cdot \max_{u\in\mathcal{N}(v)} \{\beta_{vu}\}. \tag{3}$$

In the design, we select the *degree centrality* of node as migration parameter α to control the movement cost since this degree centrality is often used to measure the quality of nodes that can host the service. 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. As a side effect, this parameter can also improve the stableness of the algorithm, which could prevent the algorithm from making *ping-pang* movements.

How Algorithm Works: In each epoch the algorithm at p-node v (called *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, the

[1]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.

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 recognized to migrate the server, either virtually or physically.

1) *Virtual Migration:* If there are some neighbors with *AC* values less than that of the p-counter, the algorithm at pivot will randomly selects 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 migration *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 actually migrating the server.

2) *Physical Migration:* Otherwise, if all the neighbors of the virtual p-node have *AC* values greater than the virtual p-counter, the algorithm will mark the end of current epoch, and physically move the server to the current virtual node. Then, 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 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.

C. A Working Example of Dmig

A working example of the *Dmig* algorithm is shown in Fig. 2 where the current phase is marked by pivot configuration v_i. The algorithm serves σ_i and then makes a migration decision. In the example, the algorithm finds the value of 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 could 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 [10].

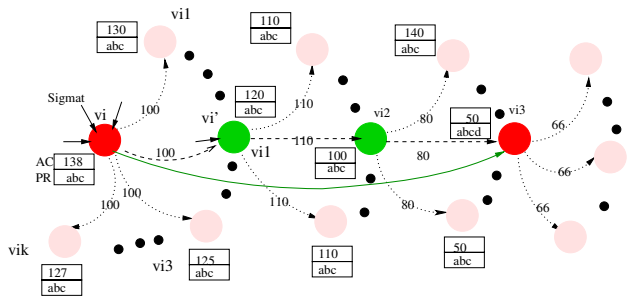


Fig. 2: A working example of *Dmig*. For each node, its *AC* and *PR* are also marked. The algorithm makes three virtual migrations, and the last one is turned into a physical migration because of a phase termination.

D. Remarks on Dmig

The algorithm is fully symmetric to allow the 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 fact that, except for some pathological cases (where $O(n)$ messages are needed for n-node network), *Dmig* does not incur any global operation in migration decisions. Moreover, it leverages a concept of *virtual migration* to overcome the demerits of one-hop local search in the selection of migration targets, and moreover, migrate the virtual server to the optimal service node eventually and remain the server there as long as the access patterns are not significantly changed.

V. 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. The purposes of our evaluation are twofold: 1) to study the algorithm behaviour under various impact factors, and 2) to show the cost effectiveness of the algorithm in service migration compared to some existing algorithms.

A. Experimental Setup

1) Network Topologies: We use the networks with four typical topologies: *Tree(n)*, *Lattice(w,h)*, *Erodös-Rényi random graph* (*ER(n,p)*) [20], and *Barabási-Albert graph* (*BA(n,e)*) [21] to conduct the simulation studies, each network connecting 100 to 1000 nodes and exhibiting different structural properties to represent a spectrum of communication networks [1], [10], [15], [22].

Both *Tree* and *Lattice* have strictly regular structures, allowing us to observe the algorithm behaviours 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 in different sizes.

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

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

2) Access Patterns: An access pattern is characterized by a sequence of online batch requests distributed across the network along the time axis, each being specified by time instance, batch size as well as distribution of access points. Since we are not aware of any open available request trace to migratable service in cloud-scale networks, we carefully crafted synthetic workloads with diverse access patterns for a controlled evaluation of our algorithm.

In our experiments, we refine the workloads with three types of access patterns that are often studied in the related work [10], [15], each with different merits to evaluate the migration algorithms with respect to the performance in handling the impact of network topology, node skewness, and dynamics of mobile accesses.

Uniform(p, q): We generate a batch request by following the uniform distribution for both its size and weight on respective $[1, p]$ and $[1, q]$, to isolate the impact of network topology.

Zipf(p, ν, θ): 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 weighted skews of each requesting node, given the total number of requests ν (spatial skewness).

Zone(p, ν, θ, $\varkappa$): We partition the network graphs into different $\varkappa$ zones to make requests at different time segments in different zones, while in each zone, $Zipf(p, \nu, \theta)$ is followed as shown above. This pattern reflects both the spatial skewness and temporal dynamics of requests.

3) Charging Model: The charging model, defined and provided by the ISPs, is exploited by the ICPs to optimize their service provisioning. In our case, the charging model includes both the access model and the migration model. We assume that each link between a pair of nodes has an equal cost rate, η, then the access cost between 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. The model reflects the QoS requirements that the cost minimization always implies the reduction of request latency.

TABLE IV: Compared algorithms in the experiments.

Alg.	Specification	Remark
Dmig	standard *Dmig* in Section IV-B where virtual migration (VM) is employed	Default
Dmig'	variant of *Dmig* without VM.	Variant
rDmig	recursive *Dmig* with VM.	Variant
Migk	extended algorithm in [9] where $\beta' = \min_{(u,v)\in E}\{\beta_{uv}\}$ is used to control migration	Literature [9]
Migk'	variant of Migk using β instead of β' to control migration where $\beta = \max_{(u,v)\in E}\{\beta_{uv}\}$.	Variant

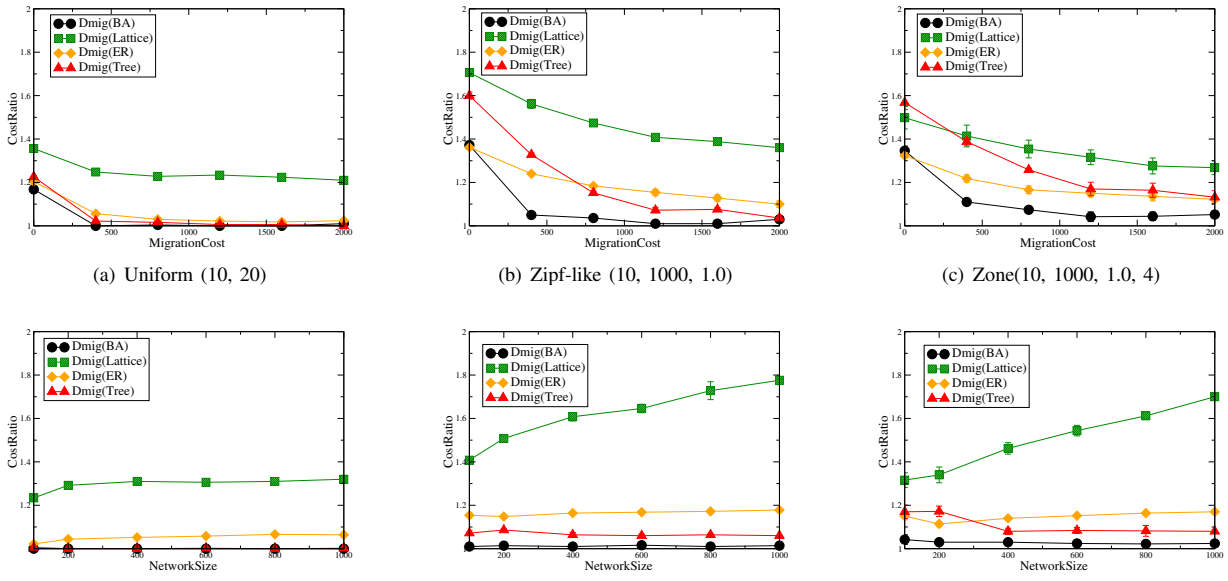
As discussed in the migration model, the migration cost between any pair of neighbor nodes u and v is given in advance by β_{uv}. However, for any pair of non-neighbor nodes u and v, any migration cost is feasible only if it is less than the total sum of the stepwise costs. In our experiments, we assume it is determined by the maximum of migration costs along the path $D(u, v)$, i.e., $\beta_{uv} = \max_{i\in[u,\ldots,v-1]}\{\beta_{i(i+1)}\}$ as the most expensive one is always the pragmatic concern in practice, and also considered in other studies on service migrations [9], [23].

In all experiments, we fix the link cost $\eta = 2$ and $\mu = 5$, respectively, and assume that the migration cost between a pair of neighbor nodes is uniformly distributed when we do not assume any a prior knowledge in this aspect.

4) Reference Algorithms: To fully evaluate the proposed algorithm, we list some existing algorithms for comparison. The first three algorithms in Table IV are compared for service migrations. *Dmig* and its variant *Dmig*' are different, depending on whether or not the virtual migration (VM) is employed. By comparing with these algorithms, we can measure the benefits of the virtual migration. Note that when $\alpha = 0$, *Dmig'(0)* can be simply reduced to an improved *Migration Policy S* in [10], where consecutive movements are performed in terms of one-hop neighbors each time.

rDmig is a recursive version of *Dmig* where if multiple neighbor nodes were eligible for the next moving step, unlike *Dmig*, *rDmig* will *recursively* search for the target node along *all* the possible paths. 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 [9] to handle the heterogeneous migration cost in a single server case, and *Migk'* is a variant of *Migk* to use β instead of β' to control the migration [9] where $\beta = \max_{(u,v)\in E}\{\beta_{uv}\}$ and



(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. 3: Impact of network topology, size and access pattern on the total service costs of *Dmig*

TABLE III: 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

$\beta' = \min_{(u,v)\in E}\{\beta_{uv}\}$.

B. Results

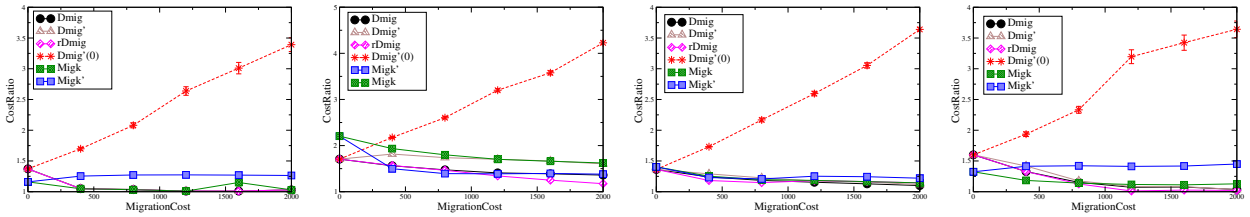
In this section, we show, in various cases, how the proposed algorithm behaves and outperforms the reference algorithms for service migration. The major performance metric is defined as the ratio of the total service cost achieved by the compared algorithms over the cost achieved by the optimal off-line dynamic-programming (DP)-based algorithm. In the experiments, each data point in graphs is averaged over five runs by changing random seed in the simulator, and its standard deviation is also computed.

Network Profiles: The first set of experiments is to study how the network topology and size affect the performance of the *Dmig* algorithm. Fig. 3(a)-(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 varied from 0 to 2000. For all the studied topologies, the algorithm performance 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) in a very limited number of moves. On the other hand, the migration control threshold of node v, (i.e., $deg(v) \cdot \max\{\beta_{vu}\}$) is relatively large, and thus, the server is highly resilient against 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 III. 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 it does not exhibit the power laws for 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, their cost ratios are different (Fig. 3). 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 III), leading to the worst performance. In contrast, for *Tree*, these migration opportunities are reduced because of 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,



(a) *BA(3, 3)* (b) *Lattice(10, 10)* (c) *ER(100, 0.1)* (d) *Tree(100)*

Fig. 4: Performance comparison of *Dmig* with other reference algorithms for Zipf$(10, 1000, 1.0)$ access pattern.

Fig. 3 (d)-(f) show how the algorithm performance changes under different access patterns when the migration cost is fixed as 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 the length of path increases.

Access Patterns: Fig. 3 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. Aside from the lattice networks, we can see from Table III 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 the distribution of request weights, rendering the migration to be beneficial in minimizing the service costs. As the request nodes for each batch are uniformly selected, 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 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 the 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.

Performance Comparison: The performance comparison of *Dmig* with other selected reference algorithms is shown in Fig. 4. 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 [9], [10]. 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 is useful.

From Fig. 4, *Dmig* significantly outperforms *Dmig(*0*)*, an optimized version of the Migration *Policy S* in [10] 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(*0*)*, 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 the cloud environments.

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 the relatively better performance, no one is consistently 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 a better performance.

The benefits of the virtual migration are also shown in Fig. 4 when the migration cost is increased from 0 to 2000. In the figure, *Dmig*, together with its non-virtual migration version and recursive version, (i.e., *Dmig'* and *rDmig*) is evaluated. One can easily observe the performance gaps between whether or not the virtual migration is present for both *Dmig* and *Dmig'*. Depending on the network topology and migration cost, such a gap can be as large as up to 28.44% for *Lattice* (*Dmig* is overlapped with Migk' in Fig. 4(b)) and 10.20%

for *Tree* (Fig. 4(d)), respectively, illustrating the values of the virtual migration.

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 Fig. 4. 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 not worthwhile to deploy in practice.

VI. Conclusions

In this paper, we formulated and studied the service migration problem in the cloud platforms so that the mobile accesses could be adaptively serviced with minimum costs. To this end, we developed an efficient autonomous 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 overcome the limitation of local search. Our extensive simulation results showed that compared with several existing algorithms, *Dmig* can significantly reduce the overall service costs for all the studied access patterns made on some representative networks in the cloud.

Acknowledgment

This work was supported in part by the China National Basic Research Program (973 Program, No. 2015CB352400), NSFC under grant No. 61672513 and No. 61572377, Science and Technology Planning Project of Guangdong Province (2015B010129011, 2016A030313183).

References

[1] 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.

[2] Z. Xu and L. Bhuyan, "Qos-aware object replica placement in cdns," in *Global Telecommunications Conference, GLOBECOM '05. IEEE*, vol. 2, Dec. 2005, pp. 861–866.

[3] A. Benoit, V. Rehn-Sonigo, and Y. Robert, "Replica placement and access policies in tree networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 12, pp. 1614–1627, Dec. 2008.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, 2008.

[12] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, Apr. 2010.

[13] 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.

[14] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Trans. Auton. Adapt. Syst.*, vol. 1, no. 2, pp. 223–259, Dec. 2006.

[15] 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.

[16] 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.

[17] X. Wang, X. Liu, L. Fan, and X. Jia, "A distributed virtual machine migration approach of data centers for cloud computing," *Mathematical Problems in Engineering*, vol. 2013, 2013.

[18] R. Boutaba, Q. Zhang, and M. F. Zhani, "Virtual machine migration in cloud computing environments: Benefits, challenges, and approaches," in *Communication Infrastructures for Cloud Computing*, H. T. Mouftah and B. Kantarci, Eds. Hershey: IGI Global, Nov. 2014, pp. 383–408.

[19] 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*, 2011, pp. 2–2.

[20] P. Erdös and A. Rényi, "On random graphs I." *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.

[21] A. L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[22] 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.

[23] 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.