2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing

Adaptive Search-based Service Migration with Virtual Moves in Clouds for Mobile Accesses

Yang Wang [♭], Wei Shi [♯], Lingfang Zeng [§]

[♭] *IBM Center for Advanced Studies (CAS Atlantic)*
University of New Brunswick, Fredericton, Canada E3B 5A3
[♯] *Faculty of Business and Information Technology*
University of Ontario Institute of Technology, Ontario, Canada
[♯] *School of Computer, Huazhong University of Science and Technology*
Wuhan National Laboratory for Optoelectronics
E-mail: yangwang@ca.ibm.com, wei.shi@uoit.ca, lfzeng@hust.edu.cn

***Abstract*—In this paper, we study the problem of dynamically migrating a service in Clouds to satisfy a sequence of mobile batch-request demands in a cost effective way. The service may have a single or multiple replicas, each running on a virtual machine. As the origins of the mobile accesses are frequently changed over time, this problem is particularly important to those time-bounded services to achieve enhanced QoS and cost effectiveness. Moving services closer to client locations not only reduces the service access latency but also minimizes the network cost for service providers. However, these benefits do not come without compromise. The migration comes at cost 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 increased monetary costs, we propose a search-based dynamic migration algorithm that can effectively migrate a single or multiple servers to adapt to the changes of access patterns with minimum service costs. The algorithm is characterized by effective uses of historical access information to conduct virtual moves of a set of servers as a whole under a certain condition so as to overcome the limitations of local search in cost reduction. Our simulation results show that the proposed algorithm can effectively achieve the goal by satisfying service request sequences. Moreover, with moderate migration cost, using a single server is more cost-effective to satisfy the requests in Clouds than deploying multiple servers.**

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 to achieve using the traditional technologies due to the intrinsic characteristics of mobile accesses such as the very large scales, time-space variation idiosyncrasies, and high sensitivities to service latency. To cope with these characteristics, it is essential to migrate the service to some vantage locations in the networks that are close to the users in order to minimize the access latency as well as reduce the network cost for service providers. A typical example to illustrate the migration benefits is the multiplayer mobile games, where the game servers may migrate from Asia to Europe and finally to North America depending on the changing locations of dominant access loads at different time frames. Traditionally, achieving such benefits usually employs *process migration* [1], [2] over wide-area network, which is very hard if not impossible. Fortunately, by virtue of the virtualization technologies in clouds, encapsulating the requested service in a virtual machine and migrating it on-demand in the same or across different data centers is becoming a promising way to deploy such services with the afore-mentioned benefits.

Despite the high cost of bulk-data transfer (e.g., server memory image and associated data files) and service disruption, several researchers have demonstrated that it is feasible to migrate virtual machines in a wide-area network [3]–[5]. Furthermore, regarding the service migration, some preliminary results on single server migration have already been achieved with respect to virtual networks [6], [7] and autonomic networks [8], [9]. However, the trade-off between the benefits and the costs (from the monetary cost point of view) of migrating multiple servers as a whole has not been thoroughly studied. In this paper, we investigate this problem and propose a dynamic migration algorithm based on local search techniques to migrate a set of virtual servers. Local search is a meta-heuristic method which is usually used to solve computationally hard optimization problems. In this paper, we exploit these techniques to find efficiently migration targets at runtime as well as use a *virtual move* strategy to adapt to the dynamic changes of mobile access patterns in order to achieve a total service cost reduction.

More specifically, unlike some previous studies [6], [7], [10] on single server migration in distributed approaches, we are particularly interested in an efficient centralized algorithm for migrating a set of κ servers as a whole when κ is upper bounded by $O(\frac{\log n}{\log(1+\Delta_{max})})$, where Δ_{max} is the maximum node degree of a n-node network. This constraint is practical as the number of the deployed service replicas (i.e., virtual servers) is usually not necessary to be proportional to the number of network nodes [11]. With the given central control, we can gain several benefits from the service migration. First, we can reduce the total service cost of the request demands. Second, we can achieve a global load balancing among the servers in an efficient way, which is usually difficult for distributed algorithms. Last, with the foregoing benefits, we can overcome the configuration complexity (i.e., the combination of the server locations) to further ensure that the κ servers will be eventually migrated to a set of suboptimal service nodes and remain there as long as the request pattern is not significantly changed. All these benefits are particularly in favor of a set of cooperative servers to move around in the network to service

CFP13UCC-USB/13 $26.00 © 2013 Crown Copyright
DOI 10.1109/UCC.2013.43

the user requests.

Due to the space limitations, in this paper, we focus squarely on the first aspect of the benefits and show its optimality in a tree-structured network. Other related results can be found in [11]. We validate our findings by conducting extensive simulation studies, in which the numerical results show that the proposed algorithm can well adapt to the changes of mobile access patterns and efficiently satisfy the service requests in a cost effective way.

The remainder of this paper is organized as follows. In Section II, we formulate the service migration problem with the assumptions we make. Section III presents then analyzes our dynamic service migration algorithm given $\kappa \leq O(\frac{\log n}{\log(1+\Delta_{max})})$. We present our empirical studies via intensive simulation in Section IV and then review some related work in Section V. We finally conclude the paper in the last section.

II. Dynamic Service Migration

We consider an arbitrary n-node network $G(V, E)$ as a service infrastructure to provide a mobile service. The service has $\kappa \geq 1$ replicas (also called *servers*), each running on a virtual machine (VM). The set of hosting (physical) machines has a *configuration*, denoted by $\mathcal{H}$, which is the specifications of this set of physical machines that are running the VMs. This set of machines are accessed by a sequence of batch requests $\sigma = \sigma_1\sigma_2...\sigma_m$ issued from a set of external machines (i.e., mobile terminals). The requests arrive in an online fashion and are satisfied by triggering the migration of the servers in $\mathcal{H}$. As a result, a subset of the server locations would be frequently changed over time. We denote the configuration of this subset at time t_i as $\mathcal{H}_i$.

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

Clearly, in order to satisfy σ_i, each request $r \in \sigma_i$ will be eventually routed to certain $h \in \mathcal{H}_i$ via the closest $a_r \in V$. This is typically achieved by the underlying *routing function* determined by ISPs. As a result, the total monetary cost of access (hereafter access cost) of batch request σ_i can be simply calculated as the total connection costs plus transmission costs of all the requests in σ_i along the routing path, which can be expressed as $Cost_{acc}(\mathcal{H}, \sigma_i) = |\sigma_i|\mu + \sum_{r\in\sigma_i} C_{a_r\phi(r)}, \phi(r) \in \mathcal{H}$, where a_r is request r's nearest connect point and $\phi(r)$ is r's service node determined by the routing function $\phi(.)$. Note

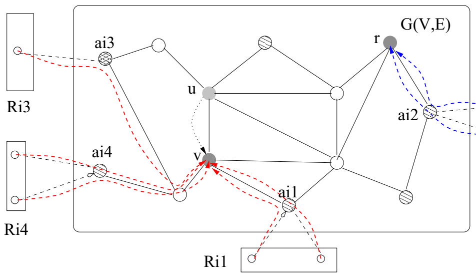


Fig. 1: An example of service accesses and migration model.

that in this equation we implicitly assume that the sizes of requests are so small that the network bandwidth is never the bottleneck for their transmissions, rather, the link latency is the issue. Clearly, this cost is affected by the level of the network latency.

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

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

Therefore, for a sequence of batch requests $\sigma = \sigma_1\sigma_2...\sigma_m$, the goal of the service migration is to determine $\mathcal{H}_1, \mathcal{H}_2, ..., \mathcal{H}_t$ (t is to be determined) to minimize the total service cost defined as $Cost(\mathcal{H}_i) = Cost(\mathcal{H}_{i-1}) + Cost_{acc}(\mathcal{H}_{i-1}, \sigma_i) + Cost_{mig}(\mathcal{H}_{i-1}, \mathcal{H}_i)$, given $Cost(\mathcal{H}_0) = 0$. This recurrence indicates that the total cost to satisfy σ_i is equal to the total cost of satisfying the first $i-1$ requests plus the access cost in configuration $\mathcal{H}_{i-1}$ and then the migration cost from $\mathcal{H}_{i-1}$ to $\mathcal{H}_i$. Clearly, this model is more amenable to the inter-datacenter migration rather than the intra-datacenter migration in reality as the intra-datacenter network latency is low and the bandwidth is usually high. However, it does not mean that the model is not allowed to apply to the intra-datacenter migration. Note that if the κ servers are not allowed to move, this optimization problem is reduced to the classic κ-median problem [12], [13], which is a typical NP-complete problem.

III. Migration Algorithm

In this section, we present our dynamic migration algorithm which services the incoming batch requests in a time sequence

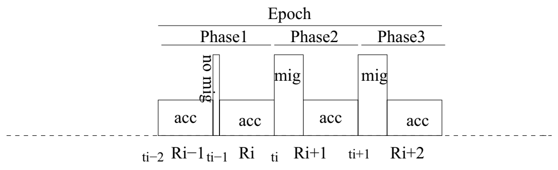


Fig. 2: The relationships between epochs, phases and time stages in our algorithms.

order without needing any a prior knowledge of complete access pattern. We first overview the algorithm and then describe it in depth.

A. Algorithm Overview

1) Definition: To ease the understanding of the algorithm, we first define some concepts and data structures that are used throughout this paper:

Local Space: Given a batch-request σ_t and a configuration $\mathcal{H}_i$ at time t, we first define a neighborhood of $\mathcal{H}_i$ (i.e., *local space*), denoted by $\mathcal{N}(\mathcal{H}_i) = \{H_i^0, H_i^1, ..., H_i^j, \ ...,\}$ Each H_i^j (formally defined in Section III-B) signifies a distinct neighbor's configuration of $\mathcal{H}_i$. The algorithms is to conduct local search in $\mathcal{N}(\mathcal{H}_i)$ to identify certain H_i^j with total cost reduction and migrate all the corresponding elements in $\mathcal{H}_i$ to their targets in H_i^j after serving σ_t by $\mathcal{H}_i$.

Epoch & Phase: The algorithm operates on per-epoch basis along the time-line which is further divided into a sequence of phases. Each phase defines a period of time during which no migration is incurred to satisfy a sub-sequence of requests (i.e., at most κ servers can be migrated between any two adjacent phases), and is hence signified by a server configuration called *pivot configuration* (i.e., $\mathcal{H}_i$ in the following discussion), which is created at the beginning of the phase. An epoch which is composed of one or multiple phases is delimited by some time instance when a certain property is held by the neighborhood of the pivot configuration (discuss later). An example of the relationships between epochs, phases and time stages is shown in Fig. 2 where an epoch consists of three phases, *Phase1* spans across time stages $[t_{i-2}, t_{i-1}]$ and $[t_{i-1}, t_i]$ as there is no migration in $[t_{i-1}, t_i]$ whereas in *Phase2* and *Phase3* only one time stage is covered.

Data Structures: The algorithm maintains two data structures for each configuration in $\mathcal{N}(\mathcal{H}_i)$ on per-epoch basis. One is an *Access Counter* (AC) that is used to monitor and accumulate the access costs in an epoch. The other is an *History Recorder* (HR) used to record the history of the corresponding access requests in the same epoch. During the service, the algorithm progressively accumulates and records for the pivot configuration $\mathcal{H}_i$ the access costs and requests in the pivot counter AC and pivot recorder HR respectively.

2) Basic Idea: The essence of the algorithm is to leverage the rent-or-buy paradigm to determine the service migration and take advantage of a short historical access information to prune the local space for efficient finding the migration target with reduced service cost. The *movement cost* (i.e., the buy cost) is defined as the *maximum* service migration cost from the source pivot configuration $\mathcal{H}_i$ to a target neighbor in $\mathcal{N}(\mathcal{H}_i)$

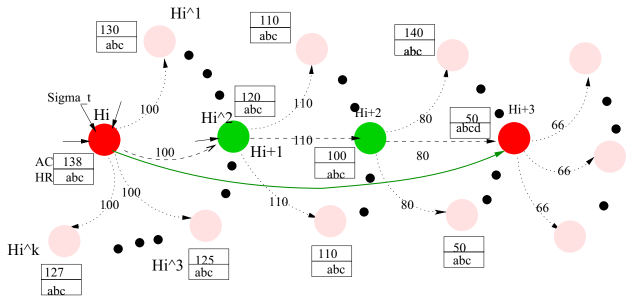


Fig. 3: A working example of the algorithm. For each configuration, its AC and HR are also marked. The algorithm makes three virtual migrations, and the last one is turned into a physical migration because of phase termination.

whereas 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 the goal.

In each epoch the algorithm is composed of multiple iterations, each corresponding a phase. In the beginning of each iteration, the algorithm first leverages the rent-or-buy paradigm to determine the migration by comparing the access cost in the pivot counter and the computed movement cost. If the value of the pivot counter is less than the movement cost, the pivot configuration will be fixed at $\mathcal{H}_i$ to continually services the incoming requests until the value is greater than the movement cost. Otherwise, the pivot algorithm broadcasts its HR (received from the last pivot) to all its neighbor nodes in $\mathcal{N}(\mathcal{H}_i)$. Each neighbor node then overwrites its own HR, and mimics the service to the requests in the HR by accumulating the cost in its AC. After that, each neighbor node sends back its AC to configuration $\mathcal{H}_i$, and the pivot algorithm compares the value of each neighbor's AC with that of the pivot counter. Depending on the comparison outcomes, two cases are distinguished to migrate the servers either virtually or physically,

- *Virtual Moves:* If there is at least one neighbor with the AC value less than that of the pivot AC, the algorithm at pivot randomly selects a migration target from those with the minimum AC as the new pivot configuration for the next phase, and relay the pivot HR to the new phase. We call such migration a *virtual migration* since in this case we can repeat the same algorithm at the new target to pick the next most preferable location, and make the new target to be only a temporary stop (i.e., *virtual configuration*) without requiring the servers to migrate physically.
- *Physical Migration:* Otherwise, if all the neighbors of the virtual configuration have the AC values greater than the virtual counter, the algorithm marks the end of the current epoch. In this case, the servers are directly moved to the nodes in the virtual configuration which is then promoted to a pivot configuration. In this situation, all the data structures will be reset for a new epoch, and the algorithm restarts from scratch as well. On the other hand, when the value of the virtual

Algorithm 1 service migration algorithm

```
 1: procedure MIGRATIONDECISION($\mathcal{H}_i, \sigma_t$)        ▷ Det. mig. at $\mathcal{H}_i$ when accepting $\sigma_t$
 2:     while $i \geq 0$ do
 3:         if $i = 0$ then N                                ▷ Init. new epoch
 4:             for $C \in \mathcal{N}(\mathcal{H}_i)$ do
 5:                 $w_i(C) \leftarrow 0$
 6:                 $d_i(C) \leftarrow \varnothing$
 7:             end for
 8:         end if
 9:         Serve($\sigma_t, \mathcal{H}_i$)                    ▷ serves $\sigma_t$ with $\mathcal{H}_i$ first
10:                                                          ▷ determine migration
11:         $(H, case) \leftarrow$ VirtualMigration($\mathcal{H}_i$)
12:         if $case = PhaseTerm$ then
13:             if $H \neq \mathcal{H}_i$ then
14:                 $H \Leftarrow \mathcal{H}_i$                 ▷ Physical move to H
15:                 $i \leftarrow i + 1$
16:             end if
17:         else
18:             $H \Leftarrow \mathcal{H}_i$               ▷ Epoch ends: Phy. mov. $\mathcal{H}_i$ to H
19:             $i \leftarrow 0$
20:         end if
21:         $\mathcal{H}_i \dashleftarrow H$          ▷ Virtual move, take H as the new $\mathcal{H}_i$
22:     end while
23: end procedure
```

counter is less than the movement cost, it also signifies the completion of a sequence of virtual phases. At this time, the algorithm opens up a new physical phase by informing the original pivot to physically migrate the servers to the new virtual configuration in a direct and minimum-cost way. Note that in this case, the virtual configuration is also promoted to a pivot configuration, but its data structures are not reset in favor of the migrations.

Example: An working example of this algorithm is shown in Fig. 3 where the current phase is marked by the pivot configuration $\mathcal{H}_i$. The algorithm serves σ_t and then makes migration decision by exploiting the historical access information in its pivot recorder based on each of its neighbors. 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 H_i^2 has the minimum service cost (120) to the historical requests (abc) which is less than the pivot counter (138). Then a virtual migration is made from $\mathcal{H}_i$ to H_i^2, a new virtual phase starts marked by $\mathcal{H}_{i+1}$ (i.e., H_i^2). As the virtual pivot recorder of $\mathcal{H}_i$ has been copied to $\mathcal{H}_{i+1}$ (i.e., abc), the same virtual migration process can be repeated at $\mathcal{H}_i$ (as $120 > 110$) to reach the next virtual target $\mathcal{H}_{i+2}$ with the minimum cost (100) from which to the final virtual target $\mathcal{H}_{i+3}$ where this virtual migration process cannot be continued anymore due to the conditions of either the phase termination or the epoch termination (in our case, it's phase termination as $50 < 66$). In this situation, the algorithm turns the virtual phase into an *actual* one by directly migrating servers in $\mathcal{H}_i$ to $\mathcal{H}_{i+1}$, thereby serving the next request d (HR=abcd). From this example, one can see that our algorithm can adapt to the request pattern in a cost-effective way by allowing the servers to be quickly moved to the vantage configuration.

B. *Formal Descriptions*

Our algorithms divide the time-line into epochs, and in each epochs the same procedures are iterated in the different phases. Therefore, we can only describe the algorithms on per epoch basis. We consider the scenario when the local space of the pivot configuration $\mathcal{H}_i$ for phase $\mathcal{P}_i$ can be exhaustively searched when $\kappa \leq O(\frac{\log n}{\log(1+\Delta_{max})})$.

Given a pivot configuration $\mathcal{H}_i$ for phase $\mathcal{P}_i$, the elements in $\mathcal{N}(\mathcal{H}_i)$ are defined based on each $X(l) \subseteq \mathcal{H}_i$, and its one-hop neighbor set $Z(l) = \mathcal{N}_1(X(l))$ for all $0 \leq l \leq \kappa$. For each l, there are $\binom{\kappa}{l}$ $X(l)$s, each corresponding at most Δ_{max}^l $Z(l)$s, and each $Z(l)$ has l nodes, and each node has at most Δ_{max} degrees. Thus when $\kappa \leq O(\frac{\log n}{\log(1+\Delta_{max})})$, the size of $\mathcal{N}(\mathcal{H}_i)$ is upper bounded by $\sum_{l=0}^{\kappa} \binom{\kappa}{l} \Delta_{max}^l = (1+\Delta_{max})^\kappa = O(n)$ (i.e., $|\mathcal{N}(\mathcal{H}_i)| \leq O(n)$).

We use $w_i(H_i^{l_{pq}})$ to represent the access counter of $H_i^{l_{pq}}$ to accumulate the total service cost in the epoch for an incoming request sequence until σ_t, and similarly, use $d_i(H_i^{l_{pq}})$ to denote the corresponding history recorder of $H_i^{l_{pq}}$ to gather the access requests where $l \in [0, ..., \kappa]$, $p \in [1, ..., \binom{\kappa}{l}]$ and $q \in [1, ..., \Delta_{max}^l]$. Let $X - Y = X \setminus Y$ and $X + Y = X \cup Y$, we have $H_i^{l_{pq}} = \mathcal{H}_i - X_p(l) + Z_q(l)$ representing the migration of exactly l servers from $X_p(l) \subseteq \mathcal{H}_i$ to $Z_q(l)$ and reach of $H_i^{l_{pq}}$ while minimizing the overall service cost up to σ_t. Then $\mathcal{N}(\mathcal{H}_i)$ can be written as:

$$\mathcal{N}(\mathcal{H}_i) = \{H_i^{l_{pq}} | H_i^{l_{pq}} = \mathcal{H}_i - X_p(l) + Z_q(l)\} \qquad (1)$$

Note that the subscript p of $X_p(l)$ specifies a particular configuration of the selected migrated servers in $\mathcal{H}_i$ whereas the target locations are specified in the subscript q of $Z_q(l)$. Furthermore, the movement cost for $\mathcal{H}_i$ is defined by:

$$Cost_{mov}(\mathcal{H}_i) = \max_{H \in \mathcal{N}(\mathcal{H}_i)} \{Cost_{mig}(\mathcal{H}_i, H)\} \qquad (2)$$

Suppose that κ servers are initially co-located at $\mathcal{H}_0$, which signifies phase $\mathcal{P}_0$, we have $w_0(H_0^{l_{pq}}) = 0$ and $d_0(H_0^{l_{pq}}) = \varnothing$ for $l \geq 0, p, q \geq 1$. If at time t the algorithm starts the iteration for phase $\mathcal{P}_i$ ($\mathcal{H}_i$), and the incoming batch request is σ_t, we make the migration decision according to the algorithm shown in Algorithm1. 1 and Algorithm2. 2. In the following, we make some remarks on this algorithm and discuss our choices in the design.

1) *Initialization:* We initialize the data structures of each configuration on per-epoch basis in favor of migrations by resetting $w_i(H_i^{l_{pq}}) = 0$ and $d_i(H_i^{l_{pq}}) = \varnothing$ for all elements in $\mathcal{N}(\mathcal{H}_i)$ within $O(n)$ time as the pivot counter would be increased quickly to exceed the movement cost. Of course, another alternative strategy is to reset the algorithm on per-phase basis. However this choice bias towards stationary server configurations.
2) $Serve(\sigma_t, \mathcal{H}_i)$ and $VirtualServe(d_i(C), C)$, $C \in \mathcal{N}(\mathcal{H}_i)$**:** The algorithm increases the pivot counter in $Serve(\sigma_t, \mathcal{H}_i)$ by accumulating the cost for servicing σ_t according to $w_i(\mathcal{H}_i) = w_i(\mathcal{H}_i) + Cost_{acc}(\mathcal{H}_i, \sigma_t, \phi)$. Similarly, by following the same equation, $VirtualServe(d_i(C), C)$ can compute the total (virtual) cost of the request sequence in $d_i(C)$

Algorithm 2 virtual migration algorithm

```
1: procedure VIRTUALMIGRATION($\mathcal{H}_i$)    ▷ recursive func. to find a target config.
2:     if $w_i(H_i) \geq \mathbf{Cost_{mov}}(\mathcal{H}_i)$ then
3:         for $C \in \mathcal{N}(\mathcal{H}_i)$ do    ▷ this loop can run in parallel
4:             $d_i(C) \leftarrow d(\mathcal{H}_i)$    ▷ copy $d_i(H_i)$ to $\mathcal{N}(H_i)$
5:                                    ▷ mimic service reqs in $d_i(\mathcal{H}_i)$
6:             $w_i(C) \leftarrow \mathbf{VirtualServe}(d_i(C), C)$
7:         end for
8:         $H^* \leftarrow \arg\min_{C \in \mathcal{N}(\mathcal{H}_i)} \{w_i(C)\}$
9:             ▷ get the next mig. target, ties are broken at random
10:        if $H^* = H_i$ then
11:            return ($\mathcal{H}_i, EpochTerm$)    ▷ Epoch terminates
12:        else
13:            return VirtualMigration($H^*$)    ▷ Go to next level
14:        end if
15:    else
16:        return ($\mathcal{H}_i, PhaseTerm$)    ▷ Phase terminates at $\mathcal{H}_i$
17:    end if
18: end procedure
```

via mimicking the service provided by C. Since the computation of $VirtualServe(d_i(C), C)$ for each C in $\mathcal{N}(H_i)$ is completely independent, they can be computed in parallel.

3) $Cost_{mov}(\mathcal{H}_i)$**:** Although the semantics of this function is straightforward, the algorithm to realize it is tricky as this function is heavily relied upon $Cost_{mig}(\mathcal{H}_i, H_i^{l_{pq}})$, $H_i^{l_{pq}} \in \mathcal{N}(\mathcal{H}_i)$, which requires us not only to generate *all* the n neighbors in $\mathcal{N}(\mathcal{H}_i)$ but also cope with the situations when two or more servers in $X_p(l)$ share the same neighbor target in $Z_q(l)$. In our design, the access loads are equally shared between the virtual machines if they are co-located in the same host machine.

4) *Find New Phase/Epoch:* We obtain the minimal configuration by local search,

$$H^* = \arg\min_{C \in \mathcal{N}(\mathcal{H}_i)} \{w_i(C)\} \tag{3}$$

The analysis on the existence of such a new pivot configuration that $H^* \neq \mathcal{H}_i$ can be found in [7].

Time Complexity: The complexity of the algorithm comes from several aspects. First, we have the following lemma to bound the complexity of $Cost_{mov}(\mathcal{H}_i)$,

Lemma 3.1: The time complexity of $Cost_{mov}(\mathcal{H}_i)$ is upper-bounded by $O(\kappa n)$.

Proof: Given $l \in [0, ..., \kappa]$, there are at most $\binom{\kappa}{l}\Delta_{max}^l$ neighbors in $\mathcal{N}(\mathcal{H}_i)$. For each such a neighbor, there are at most l migrated servers whose migrating scheme can be computed in $O(l)$. Thus due to $\kappa \leq O(\frac{\log n}{\log(1+\Delta_{max})})$, we have $\sum_{l=0}^{\kappa} l\binom{\kappa}{l}\Delta_{max}^l < O(\kappa n)$. ■

To compute the overall time complexity, we bound the values of the variables in the remaining aspects with the following lemmas,

Lemma 3.2: Given a static network with a connection rate of μ and migration costs β_{uv} to move a server between u and v, we have $|d_i(C)| \leq O(\kappa)$ for $\forall$C $\in \mathcal{N}(\mathcal{H}_i)$ given a pivot $\mathcal{H}_i$.

Proof: Since there are at most κ servers, each with at most cost of β to migrate between two points u and v, the maximum movement cost is thus $\kappa\beta$. On the other hand, given a pivot $\mathcal{H}_i$, for any C $\in \mathcal{N}(\mathcal{H}_i)$, we have $|w_i(C)| \leq \kappa\beta$ because otherwise according to our algorithm, we either leave C for the next smaller neighbor configuration via a virtual migration or stuck at C if there is no such configuration. In either case, $d_i(C)$ is not increased by serving new requests. Therefore, we have $|d_i(C)|\mu \leq w_i(C) \leq \kappa\beta$, then $|d_i(C)| \leq \kappa\frac{\beta}{\mu}$. As both the μ and β are in general determined by ISPs in advance as fixed rates, we have $|d_i(C)| \leq O(\kappa)$ and the conclusion. ■

Consequently, the complexities of $Serve(\mathcal{H}_i)$ and $VirtualServe(d_i(C), C)$ are $O(\kappa|\sigma_t|)$ and $O(\kappa\sum_{j=1}^{|d_i(C)|}|\sigma_j|) \approx O(\kappa^2|\sigma_t|)$ [1] respectively since for each request in σ_t we need to find the nearest server from κ choices.

By following the same arguments in Lemma 3.2, we have a similar result for the length of the virtual migration path:

Lemma 3.3: Given a static network with fixed rate of β_{uv} to migrate a server between u and v, the length of the virtual migration path of any given $\mathcal{H}_i$ is also upper-bounded by $O(\kappa)$.

Proof: Since the access counter is at most $\kappa\beta$, and it is *monotonically* decreased at least $\lambda = \min_{u,v \in V}\{C_{uv}\}$ at each phase along the virtual migration path until to a value of at least $\kappa \cdot \max\{\beta', \mu\}$, we then have $length \leq \kappa(\beta - \max\{\beta', \mu\})/\lambda = O(\kappa)$. ■

Theorem 3.4: Given $\kappa \leq O(\frac{\log n}{\log(1+\Delta_{max})})$ and all pairs of transmission costs in advance, Algorithm 1 triggered by σ_t can make a migration decision within the time complexity of $O(\kappa^2(n + \kappa|\sigma_t|))$.

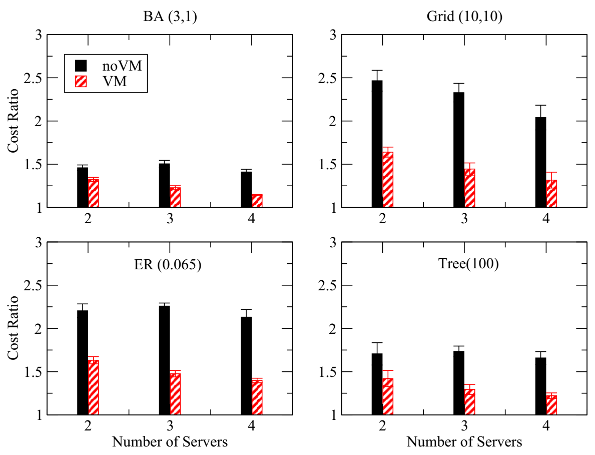
Proof: Since for a given $\mathcal{H}_i$ at time t the complexity of $VirtualMigration(\mathcal{H}_i)$ denoted by $f(i)$ can be recursively written as $f(i) = O(\kappa n + (\kappa+\kappa^2)|\sigma_t|) + f(i-1)$, we then have the overall complexity of the algorithm as $O(\kappa^2(n + \kappa|\sigma_t|))$ according to Lemma 3.3. ■

Theorem 3.5: Given $\kappa \leq O(\frac{\log n}{\log(1+\Delta_{max})})$ under a tree-structured static network, the algorithm will eventually migrate the κ servers to the optimal locations and remain there as long as the request patterns are not changed significantly.

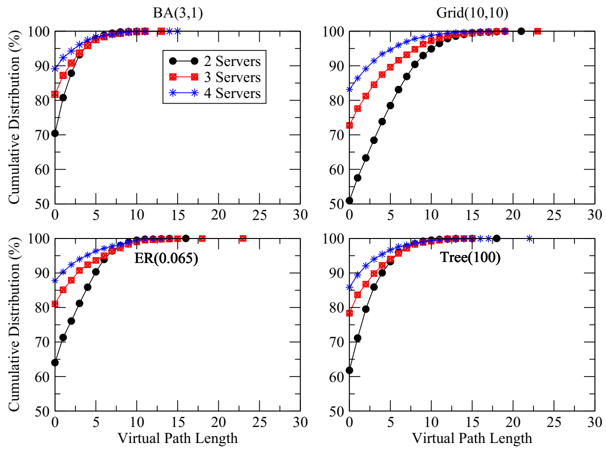
Proof: Given a tree-structured network with $\kappa \leq O(\frac{\log n}{\log(1+\Delta_{max})})$, the algorithm not only ensures that the total service cost is monotonically decreased along the unique shortest path toward the optimal locations but also exhaustively searches all the elements in $\mathcal{N}(\mathcal{H}_i)$, implying that it never gets stuck when creating new epochs unless there is no further minimal configurations. Therefore, the algorithm will migrate the κ servers to the optimal locations and remain there as long as the request patterns are not changed significantly. ■

Remarks: Although the algorithm with the restricted κ will eventually reach this optimality, it does not necessarily means that migration paths of this algorithm is optimal.

[1]It is reasonable to assume that each request in the history recorder has the same order of the size with σ_t.



(a) Cost Ratio Comparisons



(b) Virtual Move Comparisons

Fig. 4: Comparisons between cost ratio and cumulative distribution of virtual path lengths on different networks (100 nodes) with respects to the number of servers, length=0 means the servers are fixed without migrations ($C_{uv} = 2 \cdot sp(u, v), u, v \in V$ and $\mu = 5, \beta_{uv}, v \in \mathcal{N}(u)$ is uniformly distributed in [1,2000]).

IV. Performance Evaluation

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

A. Experimental Setup

We choose networks with four typical topologies: tree, grid, Erodös-Rényi (ER) random graph [14] and Barabási-Albert (BA) graph [15], each of which consists 100 nodes and exhibiting different structural properties to represent a spectrum of communication networks in previous studies [7], [10], [16]–[18].

Both tree and grid network topologies have typical topological characteristics that allow us to observe the behaviours of the algorithms under some extreme yet predictable conditions. In particular, the tree structure, characterized by average node degree, is always the intensively studied structure in different contexts including Clouds [16]–[19]. The grid structure, characterized by its width and height, represents planar networks that have been observed in a surprising number of graphs in the Internet topology zoo [20]. 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.

Access pattern is characterized by a sequence of online batch requests distributed across the network along the time axis. Each batch request is specified by its arrival time instance, batch size as well as distribution of the connect points together with the associated weights. In our experiments, in order to reflect the skewness among nodes, we assume a uniform batch size on $[1, p]$ (i.e., the number of requesting nodes) and a

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

Prof.	BA	Grid	ER	Tree
Dist	2.6	6.67	2.25	6.29
Deg	5.44	3.6	9.74	1.98

Zipf-like distribution, characterized by a parameter $\alpha \geq 1$, to capture the amount of access weight skew of each requesting node.

The major performance metric is defined as the ratio of the total service cost achieved by the proposed algorithm over the one achieved by a sub-optimal off-line algorithm, an effective sampling-based approximation which is on average within 1.06% of the optimal dynamic programming (DP) results for the Zipf access pattern ($\alpha = 1.0, p = 10$) made on the examined networks with a small scale (20 nodes) based on our empirical studies in [11]. We chose this metric as it can not only demonstrate the performance gap in presence and absence of the virtual moves but also measure the relative performance to the potential optimal results.

B. Results

In this section, we show how our algorithm, in various cases, behaves and outperforms the local search algorithm without virtual moves. In the experiments, each data point in the graphs is averaged over five runs by changing the random number seed in the simulator. To measure the distribution of the values for each data point, we also compute the standard deviation of each data point. The average inter-node distances and node degrees of the four above-mentioned of network topologies are listed in Table I.

Compared to the local search algorithm (without virtual moves), Fig. 4 (a) shows how our algorithm with virtual moves reduces the cost ratio with respect to the number of

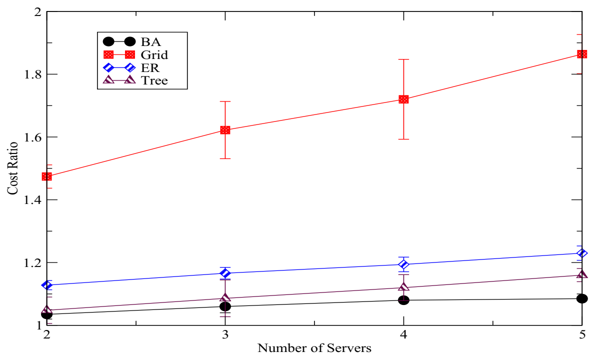


Fig. 5: Cost ratio of our algorithm over the sub-optimal off-line algorithm with respect to different number of servers (Zipf access pattern on 100-node networks).

servers. The algorithm has relatively low performance gains on BA networks because the servers are preferably located at or nearby the hub nodes in such networks to minimize the access cost, and as a result, being highly biased against the migration. This is different from the situation in grid networks where much room is left for migration to adapt to the access patterns since the nodes other than those on the edges are equally important to host the service. On the other hand, the relative large (average) node degree and inter-node distance (Table I) allow the algorithm to have more opportunities for grid networks to benefit from the virtual moves for cost reduction. These conclusions proceed from Fig. 4 (b) where the cumulative distributions of the virtual path lengths are depicted with respect to the number of servers. From the figure, the servers are moved much less in the BA than in the grid.

Furthermore, with the number of servers are inversely proportional to the lengths of virtual moves for all the studied networks. This is clear that when the number of servers increases, the load share on each server is reduced accordingly and as a result, the migration opportunities are also minimized, so is the performance gain of the algorithm.

The performance of the algorithm on ER and tree sits between that of BA and grid, and is consistent with our expectations given the structural feature of the networks. All these results not only prominently demonstrate that the proposed algorithm is cost effective in the service migration but also indirectly evidence that with the virtual moves, the algorithm has a fast convergence speed. A more extensive simulation results can be found in [11] where the performance of the algorithm with respect to different network sizes, access patterns, and other migration parameters is also studied. All results demonstrate the effectiveness of the proposed service migration algorithm in minimizing the total service costs.

We conduct the last set of experiments to examine the algorithm's behavior on a single server migration as this is a frequently studied case in the literature before the researchers explore more general cases. In these experiments, we show that in the context of service migration with moderate costs, a single server can satisfy the sequence of requests in a more cost-effective way than multiple servers. We can conclude this by observing the Zipf pattern results in Fig. 5 where the cost ratio of our algorithm is slowly increased with the number of servers changing from 2 to 5, and the multi-server cases are only slightly better than the single-server case. More specifically, the cost ratios of the single server case < 1.3x those of the multi-server cases, except for the grid network.

We can attribute these results either to some features of the networks that could offset the benefits of deploying multiple servers or to the virtual moves which could minimize the migration cost even if the source and target are far away, equivalently, reducing the multi-server benefits. For example, both BA and ER networks have relatively large average node degrees and short average inter-node distances (see Table I). The algorithm can easily find a server for a request in its 2/3-hop neighborhood even if only a single server is deployed. We can validate this by observing the performance behavior of the algorithm on the grid network. The cost ratio of the algorithm, though higher than those on other networks, is only slowly increased, and exhibits a relatively good result (i.e., < 2) when five servers are available (see Figure 5).

V. Related Work

Service migration in different forms has been studied in a number of related areas, each with similar or different goals [6], [7], [21]–[23]. In the *replica placement problem*, the requests made by users are pre-defined in terms of their frequencies and locations in the network while the locations of the servers with certain constraints to satisfy the sequence of requests with minimum total read and write costs are left to be determined as a goal [17], [21], [24]. The off-line and static nature of this problem, though related, is distinguished itself from our problem.

One typical related problem is the *κ-server problem* that allows κ mobile servers at some nodes of a graph to move on-site to satisfy the request sequence in an online fashion so that the total moving distance of these servers is minimized [22], [25]. Our problem can be viewed as an extension of the κ-server problem in the sense that the mobile servers can not only get accessed remotely but also be moved simultaneously to satisfy a batch request in a cost effective way.

The problem is formulated for the cloud services based on the model introduced in [6] where the service migration is conducted in the context of *virtual networks* (VNet) [26], [27] to minimize the service access latency. To this end, Bienkowski *et al.* [6] presented a randomized online algorithm to migrate a single server in an n-node network. They advocated the competitive analysis on the worst case of the algorithm instead of its general performance, which is our pragmatic concern.

Like in the VNet, service migration is also studied in *autonomic network environments* [8], [9] as a self-managing mechanism to overcome the rapidly growing complexity of the networks. Oikomomou *et al.* [7] proposed a scalable algorithm to service migration in autonomic networks by observing the differential demand traffics on each link between the node hosting the service and its opposite neighbor. Although this algorithm has certain merits in service migrations, it suffers from the slow convergence due to its inefficient one-hop migration. Pantazopoulos *et al.* [10] overcome this downside in their most recent centrality-driven migration algorithm, named cDSMA. However, this algorithm only targets at a single server, and moreover, lacks notion of the migration cost.

Unlike the aforementioned studies which are not directly conducted in the context of clouds, Phan *et al.* [28] proposed a framework, called *Green Monster*, to leverage the dynamic service migrations across the Internet data centers (IDCs) for the energy efficiency issue in cloud computing. The service migration is determined by a developed evolutionary multiobjective optimization algorithm which is able to strike a balance between conflicting optimization objectives. In contrast, our problem has a single objective to minimize the total monetary cost with the service migrations. Although both problems are orthogonal at the first sight, they could be connected inherently as the monetary cost could also be used to model the energy consumption as well. However, this extension is still an open problem to our algorithm.

VI. Conclusions

In this paper, we formulated and studied the service migration problem in cloud platforms for mobile accesses with minimum costs. To this end, we developed an efficient multi-server migration algorithm by leveraging local search techniques. The algorithm is distinct from those existing ones in its effective use of the historical access information to conduct virtual moves of a set of κ servers as a whole when $\kappa \leq O(\frac{\log n}{\log(1+\Delta_{max})})$ in the service migration. We first described the algorithm in depth and then analyzed its time and space complexity as well as its optimality in migrating servers in tree structured networks. Finally, we studied the algorithm via extensive simulations. The results revealed that with virtual moves, the proposed algorithm can effectively adapt to access patterns to reduce the overall service costs. In particular, with moderate migration costs, using a single server is more cost-effective to satisfy the sequence of requests in Clouds than deploying multiple servers. These results are particularly promising to those time-bounded services that are deployed on cloud platforms to achieve enhanced QoS and cost effectiveness.

References

[1] D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process migration," *ACM Comput. Surv.*, vol. 32, no. 3, pp. 241–299, Sep. 2000.

[2] J. M. Smith, "A survey of process migration mechanisms," *SIGOPS Oper. Syst. Rev.*, vol. 22, no. 3, pp. 28–40, Jul. 1988.

[3] T. Hirofuchi, H. Ogawa, H. Nakada, S. Itoh, and S. Sekiguchi, "A live storage migration mechanism over wan for relocatable virtual machine services on clouds," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, 2009, pp. 460–465.

[4] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Proceedings of the 1st International Conference on Cloud Computing*, ser. CloudCom '09, 2009, pp. 254–265.

[5] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe, "Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE '11, 2011, pp. 121–132.

[6] M. Bienkowski, A. Feldmann, D. Jurca, W. Kellerer, G. Schaffrath, S. Schmid, and J. Widmer, "Competitive analysis for service migration in vnets," in *The Second ACM SIGCOMM Workshop on Virtualized Infrastructure System and Architectures (VISA)*, Sept. 2010.

[7] 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.

[8] R. Mortier and E. Kiciman, "Autonomic network management: some pragmatic considerations," in *Proceedings of the 2006 SIGCOMM workshop on Internet network management*, ser. INM '06. New York, NY, USA: ACM, 2006, pp. 89–93.

[9] 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.

[10] 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.

[11] Y. Wang, B. Veeravalli, and C.-K. Tham, "Service migration in the cloud for mobile accesses," University of New Brunswick, Computer Science Technical Series Report, Tech. Rep., 2013.

[12] K. Jain and V. V. Vazirani, "Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation," *Journal of the ACM*, vol. 48, pp. 274–296, March 2001.

[13] M. Charikar and S. Guha, "Improved combinatorial algorithms for the facility location and k-median problems," in *IEEE Conference on Fundations of Computer Sciences*, 1999, pp. 378–388.

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

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

[16] 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, mar 1999, pp. 1282 –1293.

[17] H. Gupta and B. Tang, "Data caching under number constraint," in *the IEEE International Conference on Communications*, 2006.

[18] 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.

[19] K. Oikonomou, I. Stavrakakis, and A. Xydias, "Scalable service migration in general topologies," in *World of Wireless, Mobile and Multimedia Networks, 2008. WoWMoM 2008. 2008 International Symposium on a*, june 2008, pp. 1 –6.

[20] R. Bowden, H. X. Nguyen, N. Falkner, S. Knight, and M. Roughan, "Planarity of data networks," in *Proceedings of the 23rd International Teletraffic Congress*, ser. ITC '11, 2011, pp. 254–261.

[21] A. Benoit, V. Rehn-Sonigo, and Y. Robert, "Replica placement and access policies in tree network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 12, Dec 2008.

[22] M. Chroboak, H. Karloff, T. Payne, and S. Vishwanathan, "New results on server problems," *SIAM Journal on Discrete Mathematics*, vol. 4, pp. 172–181, 1991.

[23] J. Xu, B. Li, and D. L. Lee, "Placement problems for transparent data replication proxy services," *IEEE J.Sel. A. Commun.*, vol. 20, no. 7, pp. 1383–1398, Sep. 2006.

[24] K. Kalpakis, K. Dasgupta, and O. Wolfson, "Steiner-optimal data replication in tree networks with storage costs," in *Proceedings of the International Symposium on Database Engineering & Applications*, 2001, pp. 285–293.

[25] M. Manasse, L. McGeoch, and D. Sleator, "Competitive algorithms for on-line problems," in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, 1988, pp. 322–333.

[26] 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, Mar. 2008.

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

[28] D. H. Phan, J. Suzuki, R. Carroll, S. Balasubramaniam, W. Donnelly, and D. Botvich, "Evolutionary multiobjective optimization for green clouds," in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, ser. GECCO Companion '12, 2012, pp. 19–26.