On Service Migration in the Cloud to Facilitate Mobile Accesses

Yang Wang [♭] and Wei Shi [♯]

[♭]*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
E-mail: yangwang@ca.ibm.com, wei.shi@uoit.ca

***Abstract*—Using service migration in Clouds to satisfy a sequence of mobile batch-request demands is a popular solution to enhanced QoS and cost effectiveness. As the origins of the mobile accesses are frequently changed over time, 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.**

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. More specifically, 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.

I. Introduction

A new generation of Cloud enabled mobile services are in general hard to achieve using the traditional technologies due to the intrinsic characteristics of the mobile accesses such as the very large scales, time-space variation idiosyncrasies, and high sensitivities to service latency. Some of these characteristics include:

- The rapid changes of service requests: the mobile service requests are in general highly time and location varying. They are continuously changing with respect to the time and location of the mobile users;
- The diversity of service access patterns: the Internet is made up *small world* in the sense that the very different users can be gathered in small communities sharing common interest (e.g., approximately the same set of data items), and thus the access patterns are different from world to world [1];
- The availability of unreliable network resources: the unreliability is caused by multiple factors, including dynamism that introduces unpredictable and changing behaviors, failures that have an increasing probability of occurrences as system/application scales increase, and instability of network states that is intrinsic to large mobile environments.

Fig. 1: Mobile game: an example of service migration

Clearly, providing network services without considering these factors may significantly increase the access delays and much worse impose a large amount of access traffics on the network which might course a service disruption.

To cope with these characteristics, it is essential to migrate the service to some vantage locations in the network that are close to the users for minimizing the access latency and in the meantime reducing the network cost for service providers. A typical example to illustrate the migration benefits is the multiplayer mobile games where the game server 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* [2], [3] 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 aforementioned benefits. Several researchers have demonstrated that it is feasible to migrate virtual machines in a wide-area network [4]–[6] despite the high cost of bulk-data transfer (e.g., server memory image and associated data files) and service disruption. Furthermore, regarding the service migration, some preliminary results on single server migration have already been achieved with respect to virtual networks [7], [8] and autonomic networks [9], [10].

978-1-4799-0898-1/13/$31.00 ©2013 IEEE

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 propose a dynamic migration algorithm based on local search techniques for a set of virtual servers. Local search is a meta-heuristic method which is usually used to solve computationally hard optimization problems. However, in this paper we exploit these techniques to efficiently find migration targets at runtime and in the meanwhile, develop a *virtual move* strategy to overcome their limitations to adapt to the dynamic changes of mobile access patterns for total service cost reduction. Unlike some previous studies [7], [8], [11] 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 $O(\frac{\log n}{\log(1+\Delta_{max})})$, here 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 [12]. With the given central control, we get three benefits from such service migration:

1) reduce the total service cost of the request demands;
2) achieve a global load balancing among the servers in an efficient way, which is usually difficult for distributed algorithms;
3) 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 remained 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 the user requests.

Due to the space limitations, in this paper, we only 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 [12].

II. Dynamic Service Migration Model

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 consist of a *configuration*, denoted by $\mathcal{H}$, which 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 fashion and are satisfied by triggering the migration of the servers in $\mathcal{H}$. As a result, the subset of the server locations would be frequently changed over time. We denote the subset $\mathcal{H}$ 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

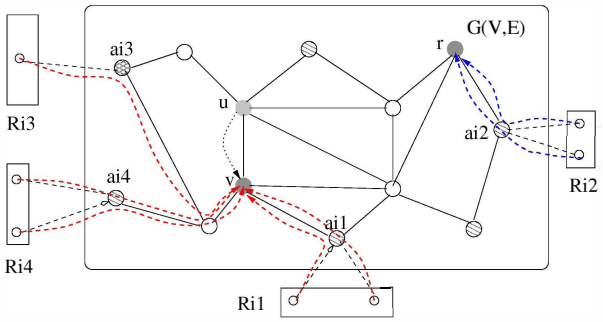

Fig. 2: An example of service accesses and migration model.

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. 2 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 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 [13], [14], which is a typical NP-complete problem.

III. Algorithm Description

In this section, we present our dynamic migration algorithm which services the incoming batch requests in a time sequence order without needing any a prior knowledge of complete access pattern. We first overview the algorithm and then describe it in depth.

A. Result Outlines

According to our model, it is possible to access the server remotely without needing for the server to move to the request points (represented by the corresponding access points). Therefore, the basic idea of the algorithms can be built upon this fact by selecting the *optimal* locations in the network with respects to the current batch request and then migrate servers to these locations for the subsequent requests. Our algorithms are based on the *rent-or-buy* paradigm in the *ski rental problem* to determine the migration and leverage the *local search* technique with historical access information to select the target configurations.

Due to the complexity of this problem, we first concern ourselves with the restricted situation when κ is upper bounded by $O(\frac{\log n}{\log(1+\Delta_{max})})$. is the maximum node degree of network G. Under this constraint the defined local space can be *exhaustively* searched by our algorithm in a simple yet efficient way and a migration scheme triggered by σ_t can be served within $O(\frac{n\log^3 n}{\log^3(1+\Delta_{max})}|\sigma_t|)$ time complexity. Furthermore the algorithm ensures that the virtual servers are eventually migrated to the optimal service nodes and remain there as long as the request sequence is not significantly changed. Then, we remove this constraint and extend the result to a more general case (i.e., $\kappa \leq O(n)$). We show that in this situation with the configuration complexity, the local space can only be *partially* searched according to some heuristics and a migration scheme can be determined within $O(\kappa^4\Delta_{max}(|\sigma_t| + \Delta^2_{max}))$ without the guarantee of the eventual migration to the optimal service locations.

B. Definition

To ease the understanding of the algorithm, we first define some used concepts and data structures as follows:

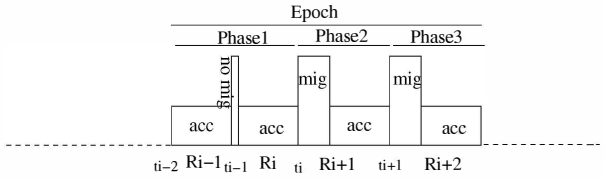

Fig. 3: The relationships between epochs, phases and time stages in our algorithms.

Local Space: Given a batch-request σ_t and server 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 signifies a distinct neighbor configuration of $\mathcal{H}_i$, and 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 $\mathcal{H}_i$. The algorithm has the knowledge of the network topology. As such given $\kappa \leq O(\frac{\log n}{\log(1+\Delta_{max})})$, $\mathcal{N}(\mathcal{H}_i)$ can be computable in $O(\kappa n)$ time.

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. 3 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.

C. 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.

D. Basic Ideas

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 $\mathcal{N}(\mathcal{H}_i)$ 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,

1) 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.

2) 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 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.

IV. Time Complexity Analysis

The complexity of the algorithm comes from several aspects. The first one is $Cost_{mov}(\mathcal{H}_i)$. The second one is the recursive $VirtualMigration()$, which is dependent of the *depth* of the recursion, and the final one is the $VirtualServe()$ being invoked inside $VirtualMigration()$, which is determined by the product of $|\mathcal{N}(\mathcal{H}_i)|$ and $|d_i(\mathcal{H}_i)|$. First, we have the following lemma to bound the complexity of $Cost_{mov}(\mathcal{H}_i)$,

Lemma 4.1: The time complexity of $Cost_{mov}(\mathcal{H}_i)$ is upper-bounded by $O(\kappa n)$.

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

Lemma 4.2: 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 $\forall \mathtt{C} \in \mathcal{N}(\mathcal{H}_i)$ given a pivot $\mathcal{H}_i$.

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 4.2, we have a similar result for the length of the virtual migration path:

Lemma 4.3: 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 $O(\kappa)$.

Theorem 4.4: Given $\kappa \leq O(\frac{\log n}{\log(1+\Delta_{max})})$ and all pairs of transmission costs in advance, our migration algorithm triggered by σ_t can make a migration decision within the time complexity of $O(\kappa^2(n + \kappa|\sigma_t|))$.

Theorem 4.5: 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.

V. Conclusions

In this paper, 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 conclude that with moderate migration costs, using a single server is more cost-effective to satisfy the sequence of requests in Clouds than deploying multiple servers. Our results are particularly promising to those time-bounded services that are deployed on cloud platforms to achieve enhanced QoS and cost effectiveness.

References

[1] K. Kleinberg, "The small-world phenomenon: an algorithm perspective," in *Proceedings of the 32nd annual ACM symposium on Theory of computing*, 2000.

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

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

[4] 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.

[5] 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.

[6] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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