Network Aware Load-Balancing via Parallel VM Migration for Data Centers

Kun-Ting Chen[2], Chien Chen[12], Po-Hsiang Wang[2]
[1]Information Technology Service Center, [2]Department of Computer Science
National Chiao Tung University, Hsin-Chu, Taiwan
quentin2007.cs96g@g2.nctu.edu.tw, chienchen@cs.nctu.edu.tw, btc889874.cs98g@g2.nctu.edu.tw

***Abstract*—It becomes a challenge to design an efficient load balancing method via live virtual machine (VM) migration without degrading application performance. Two major performance impacts on hosted applications that run on a VM are the system load balancing degree and the total time till a balanced state is reached. Existing load balancing methods usually ignore the VM migration time overhead. In contrast to sequential migration-based load balancing, this paper proposes using a network-topology aware parallel migration to speed up the load balancing process in a data center. We transform the VM migration-based multi-resource load-balancing problem into a minimum weighted matching problem over a weighted bipartite graph. By obtaining the minimum weighted matching pairs through the Hungarian method, we parallel migrate multiple VMs from overloaded hosts to underutilized hosts to reduce the time it takes to reach a load balanced state. The experimental results show that our algorithm not only obtains a compatible multi-resource load balancing performance but also improves the balanced time which results in at most a 10% throughput gain by assuming a large batch application running on all VMs.**

I. INTRODUCTION

Cloud data centers employ virtualization-based technology to consolidate hardware resource usage to provide application hosting for multiple service providers. In a cloud data center, thousands of commodity computers work in parallel with host virtual machines (VMs) that support different applications. The CPU speed, memory size, and network bandwidth of different commodity computers are widely heterogeneous. Besides, a physical host may run multiple services with different types of resource demands. Without proper allocation, the loads of different resources may become unbalanced among different physical hosts. Even with careful resource provisioning at the beginning, due to dynamic arriving and leaving of running workload, the cloud system could still become a load unbalanced state later. Thus, a cloud system may have quite a few overloaded hosts while lots of underutilized hosts are still available.

Load-balancing mechanisms improve system performance by reducing resource contention (e.g., CPU, network bandwidth, and memory) on overloaded hosts. More importantly, it can prevent a higher hardware failure rate in overloaded hosts.

Load-balancing mechanisms are effectively studied in many research areas, including distributed systems, web servers, and cloud systems, to maximize resource utilization and minimize the variance of the load of multiple servers [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]. While most recent research focuses on the VM migration as a mean to achieve load balancing in cloud systems [4] [5] [6] [7] [8] [9] [10] [11] [20][21], a few of them focus on the impact of the total time required to reach system balance. Normally, the existing load balancing algorithms search for a VM to instantiate a VM migration from overloaded hosts to under-loaded hosts. They usually select and start the next VM migration when the previous migration completely terminates. These schemes aim to achieve an egalitarian state without considering the waiting time for other hosts to offload their workload. This paper calls such schemes "sequential migration-based load balancing methods." These methods could lead to all overloaded hosts taking a long time to offload their workload. Consequently, the applications running on those overloaded hosts will contend for resources for a longer time. Therefore, users could experience application performance degradation.

In this work, we study the multi-resource load balancing problem for a cloud data center with heterogeneous hardware capacity. Compared with sequential migration-based load balancing methods, this paper proposes a parallel VM migration approach that focuses on minimizing the joint multi-resource imbalance and the time it takes to reach a balanced state. In addition, the migration delay based on the network topology of a data center is considered, especially if the migration pairs are between machines with a large hop distance. In order to improve the application performances, this paper minimizes the time till a balanced state is reached by migrating VMs parallel between independent pairs of hosts while considering the underlying network topology. To be specific, this paper models the load balancing problem with a weighted bipartite graph (*WBG*). The over- and underutilized hosts in a data center are selected as two disjoint sets of vertices: Trigger (*TR*) node and Non-Trigger (*NTR*) node sets. A physical host is said to be a trigger node if its resource utilization exceeds a system threshold on any dimensions of resources; otherwise, it is a non-trigger node. Each edge between the two sets is designated with a weight according to the multi-resource requirements of VMs, the various resource capacities of the hosts, and the network cost between two hosts. By solving the minimum weighted matching problem using the Hungarian method [12], this paper could migrate a set of VMs between over- and underutilized hosts in parallel.

The experimental results show that our algorithm achieves a compatible multi-resource load balancing while improving the balancing time, which results in at most a 10% running application's throughput gain compared to conventional algorithms which perform load balancing in a sequential manner.

The rest of this paper is organized as follows. Section II describes related works. Section III elaborates the motivation

978-1-4799-3572-7/14/$31.00 ©2014 IEEE

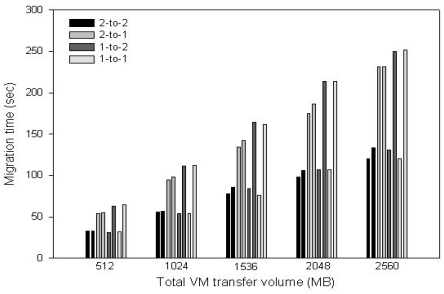

Fig. 1. Migration time vs. VM memory size

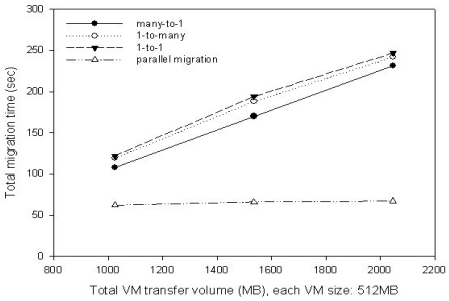

Fig. 2. Total elapsed migration time vs. number of VMs

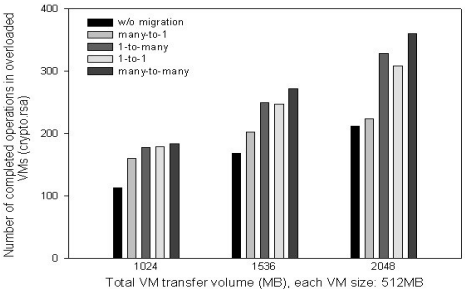

Fig. 3. Number of completed operations vs. number of VMs

behind this work. Section IV describes the network-aware bipartite matching load balancing algorithm in detail. Experimental results of the simulation will be presented in section VI. Finally this paper concludes with some remarks in section VII.

II. Related Works

A number of researches have proposed load balancing methods [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] for cloud data center. Most of these works focus on balancing the resource requirement of VMs on physical machines. Based on their algorithms, they migrate either single or multiple VMs in each round of the load balancing procedure. These load-balancing schemes for cloud computing measure the load on physical hosts in different ways. However, none of their efforts address the multi-resource load balancing issue [1] [2] [3]. Zhao and Huang [2] use the number of VMs of a host as their load measurement. Recent study [3] mimics the animal behavior of honeybees foraging and scouting to harvest foods for load-balancing. Though the scheme works for a dynamic workload, they assume each host is homogeneous and deal with single resources only. And yet [4] considers the multi-resource demands of VMs and heterogeneous capacity in each host. They propose a load balancing method called *VectorDot* (*VD*). Their goal is to bring overloaded hosts below their threshold by migrating one or more VMs sequentially. They extend the Toyoda heuristic for a single knapsack problem to solve a multidimensional knapsack problem. *VD* treats a VM in an overloaded host as an item to be placed into an appropriate node. Then, this VM is migrated to the selected destination node. This procedure is repeated until there are no more overloaded hosts. From a practical perspective VM migration takes a while to finish. Thus migrating VMs in a one-by-one manner could take a long time until the system is balanced, and thereby exacerbate the running applications that are hosted on overloaded machines.

Recent works that consider VM migration overhead for load balancing can be found in [7] [8] [9] [10] [11]. Since a cloud system involves cloud operators and network operators with different goals in mind, [7] studies the problem of making both objects meet. On one hand, the cloud operators wish to assign a number of VMs to some selected hosts for the purpose of meeting the VM scheduling deadline. Thus, the cloud operators wish to pair each VM with a destination host that minimizes the migration time. On the other hand, the network operators wish to minimize the network bandwidth. To make both ends meet, [7] models a balanced stable matching problem for both requirements of cloud operators and network operators. VM migration overhead is derived from the network topology and the transfer memory volume. However, their matching does not consider the bandwidth of NIC sharing at the destination host. Thus they could migrate several VMs from multiple distinct hosts to an identical host which could degrade into sequential migration. In contrast, we consider finding independent pairs between the end hosts for load balancing via parallel migration.

Recent works [8] [9] conduct a number of experiments on the cloud testbed to study the factors that influence VM migration time and impact of live migration on application performances. In [8], they show that VM migration could contend for resources with the running workload on the source and destination host, especially as the number of concurrently migrating VMs between one source and one destination host increases. Hence they study the problem of selecting a number of VMs to migrate that minimize the resource contention overhead. [8] models a VM migration as processes that run at source and destination hosts. They show that by carefully assigning VMs they could reduce the VM migration overhead. Though, it is difficult to formulate the impact of migration time given different combination of jobs and VM migrations running on machines with various resource configurations. As the cloud system gets larger, the resource contention model would become more difficult to build. [9] shows that the destination host CPU reserved for the migration process has little to do with migration time. The source host CPU impacts the migration time when the host is overloaded, especially when its memory dirty rate is high. However, they only suggest a direction to balance multi-resource load without relating it to the performance overhead incurred by concurrent VM migrations. In contrast to [8] [9], this paper performs an experimental study of live migrations and application throughputs with multiple, concurrent VM migrations (in next section). These experiment results provide a motivation to a new parallel migration-based load balancing algorithm.

Recent studies [10] [11] formulate the load balancing problem with migration overhead being the distance measured by the number of hops on the Internet. Nevertheless these schemes often do not consider the heterogeneous capacity of multi-resource in each host. In addition, the concurrent migration approach they adopt still has a chance to migrate multiple VMs to an identical destination host or from a single source to multiple distinct destinations.

III. Motivation

Even through the modern cloud administrators migrate multiple VMs concurrently, if the hosts of migration pairs do not choose carefully, concurrent migration still has a chance to degrade into sequential migration. The worst case circumstance

in load balancing is to migrate many VMs from different hosts to one under-utilized host (many-to-1) concurrently. In this section, we measure the real application performance and compare parallel migration with a range of many-to-1 and 1-to-many concurrent migration. The experiment is conducted on a real testbed based on Xen [13]. In this system, we reserve the network bandwidth as 100 Mbps at each host NIC for VM migration. Each VM runs the crypto.rsa application in SPECjvm [14]. VMs serving the same application type are allocated with identical resources. During live migrations, applications running on VMs persist in execution with a small interruption time compared to non-live migration [13]. We first measure the time taken for each individual VM during concurrent migration by observing different combinations of VM pairs. Then we observe the application performance for an increasing number of concurrently migrating VMs. A conclusion that leads to the work in this paper follows thereafter. Each evaluation is an averaged result for 10 times.

Fig. 1 shows the individual VM migration time for migrating two VMs with increasing total migrating memory volume. Consistently with [8], as each VM memory size increases from 256 to 1280 MB (where the total migrated size increases from 512MB to 2048MB, respectively,) VM migration time increases linearly proportional to the volume of transferred memory. As two distinct hosts migrate to a single host (i.e., 2-to-1), it suffers from a linearly increasing time for both VM migrations since the network capacity at the destination host's NIC is shared by the two migrations. When migrating from a single host to two distinct hosts (i.e., 1-to-2), one VM does not start migrating until another finishes migrating. This is the mechanism implemented by the Xen platform[1] which would start VM migration after the previous one finishes, that behaves the same as sequential migration in the case of migrating two VMs from one host to the other host. As opposed to the above cases, the parallel VM migration scheme (i.e., 2-to-2) achieves the smallest migration time without suffering from additional network delays.

Given that migrating two VMs could degrade into sequential migration in the 2-to-1 and 1-to-2 cases, we would like to know the performance impact of migrating multiple VMs concurrently in a larger scale, especially in an overloaded system environment. We assume that an overloaded environment consists of overloaded hosts and idle hosts. In the following experiments we assume that all the VMs run on those overloaded hosts. In such an overloaded environment, the workload on one VM contends resources with all other VMs resident on the same host. Thus the time until a VM is migrated to an idle host could affect the application performance running on that VM.

[1]Xen provides migrating modules for synchronous and asynchronous modes. The only difference is whether the migrating thread immediately returns to user thread or returns until migration completes. Both implementations migrate VMs in sequential manner even they are issued concurrently.

We study the impact of total migration time of different number of migrating VMs on the performance of workload on the migrating VMs, where the total migration time denotes the longest completion time for migrating a number of VMs. In the many-to-1 case, we concurrently migrate 2, 3, and 4 VMs each from 2, 3, and 4 overloaded hosts to one idle host. On the contrary, the 1-to-many case exhibits that 2, 3, and 4 VMs in one overloaded host are migrated to 2, 3, and 4 idle hosts respectively. The 1-to-1 case let 2, 3, and 4 VMs migrate from an overloaded host to an idle host. Parallel migration independently migrates 2, 3, and 4 VMs each from 2, 3, and 4 overloaded hosts to 2, 3, and 4 idle hosts. Each VM migration is instantiated by our controller 1 minute after the workload starts running. During VM migration, we measure the application performance obtained on migrating VMs. In order to measure the impact on the running application, the iteration duration parameter of the crypto.rsa application is set to last after all VMs finish migrating. This parameter ensures that the VM continues to demand resources during VM migration. The baseline performance is approximately 108 operations under baseline single core CPU at 2.67GHz.

Fig. 2 shows the total migration time under 2, 3, and 4 VM migrations. As the number of VM migrations increases, the completion time of migration in the many-to-1, 1-to-many, and 1-to-1 cases increases. However, the total migration time of parallel migration remains almost the same. It's the smallest among the other cases. Consistently with [8], as the total transfer memory size of concurrent VM migrations increases, the total migration time also increases. Given that the total transfer volume of memory between two hosts is fixed, they observe that the total migration time increases as the number of concurrent VM migrations increases. This is due to the resource contention between those migration processes and running workload on the source and destination host. In order to improve application performance, they study the VM assignment problem by selecting different pairs of VM migrations to lower the impact of resource contention.

Fig. 3 shows the total number of completed operations of crypto.rsa observed on migrating VMs. The case of a single host to multiple distinct hosts (1-to-many) completes 4%, 9%, and 10% fewer operations than parallel migration, as the number of concurrently migrating VMs increase from 2, to 3, to 4. Although the many-to-1 case experiences a smaller total migration time than that of the 1-to-many as shown in Fig. 2, the number of completed operations of many-to-1 decreases more severely than that of 1-to-many with 15%, 34%, and 61% fewer operations done than parallel migration. Further, the many-to-1 case is also close to the worst case without migration (i.e. no load balancing). The 1-to-many case completes more operations than the case of many-to-1 where it suffers from a longer migration delay for each individual VM as shown in the 2-to-1 case in Fig. 1. Among them, parallel migration shows the highest workload completion rate. These observations suggest that maneuvering parallel VM migrations is a feasible way to improve application performances.

IV. Network-aware Bipartite Matching Load-Balancing Algorithm

Load balancing for multi-resource requirements while considering heterogeneous resource capacity has been a challenging problem, especially in a large scale cloud hosting environment. In this paper, we study the load balancing problem for a cloud data center. Especially, we try to reduce the time taken for a cloud data center to reach its load balance state. Since a live migration of VM takes some time to accomplish, sequentially migrating VMs from trigger nodes to non-trigger nodes may take a long time to offload workload for overload hosts. Further, the migration delay between two hosts varies

according to the underlying network architecture and the transfer volume of VM memory. Thus, we propose a *Network-Aware Bipartite matching* (*NABM*) load balancing algorithm. This paper first transforms the load-balancing problem into a minimum weighted matching problem. According to the minimum weighted matching obtained from the Hungarian method, this paper migrates VMs from overloaded hosts to underutilized hosts in parallel. Furthermore, this paper adds the hop distance between source and destination hosts to reflect the impact of the network to the VM migration time. By reducing the VM migration time, it can shorten the time for a cloud system to reach its load balanced state.

A. Preliminary

The notations for use in *NABN* will be defined as follows. Let $H=\{h_1,h_2,\ldots,h_m\}$ and $V=\{vm_1,vm_2,\ldots,vm_v\}$ denote a set of m hosts and v VMs in a cloud system, respectively, with an existing allocation $A:V\rightarrow H$ where $A(vm_i)$ denotes the host where vm_i resides. The n different dimensions of resources consumed by a VM vm_β is represented by $S_\beta=\{s_\beta^i \mid i\in\{1,2,\ldots,n\}\}$. The input to our load balancing algorithm includes a capacity value, a current utilization value and a suggested threshold fraction between 0 and 1. Let $C_a=\{c_a^i \mid i\in\{1,2,...,n\}\}$ and $U_a=\{u_a^i \mid u_a^i\in[0,1], i\in\{1,2,\ldots,n\}\}$ denote the vectors of capacity and resource utilization along n dimensions of resources for a Host h_a, respectively. $T=\{t_i \mid t_i\in[0,1], i\in\{1,2,...,n\}\}$ denotes the vector of the system threshold along each dimension of resources. Our load balancer will maintain the usage of the resources below those thresholds. Moreover, the available capacity for a host h_a in any dimension of resources i can be represented as in (1).

$$F_a=\{f_a^i \mid f_a^i=(1-u_a^i)\cdot c_a^i, i\in\{1,2,\ldots,n\}\}. \quad (1)$$

B. Bipartite Matching Load-Balancing

Similar to [4] , *NABM* collects multi-dimensional resource utilization information of hosts and resource requirements of VMs as input. In the real testbed deployed with Xen, we send XML-RPC requests to collect load information from any host in the same server farm through a control domain. Xen implements multiple control domains in charge of sending and receiving end hosts' control messages. To construct a weighted bipartite graph (*WBG*), *NABM* leverages three decision policies: participation, candidate selection, and location, as defined in [15]. The participation policy decides which hosts are involved in the load balancing process. In this paper, we specify two node sets for over- and under-loaded hosts. They can be defined as a triggered set: $TR=\{h_a \mid h_a\in H, \exists i(u_a^i>t_i), i\in\{1,2,\ldots,n\}\}$ and a non-triggered set: $NTR=\{h_a \mid h_a\in H, h_a\notin TR\}$, respectively. The triggered set is a set of hosts for which resource utilization exceeds the system threshold in any dimension of resources. The hosts that are not marked as the triggered nodes belong to the non-triggered set. These two sets together form two vertex sets for the weighted bipartite graph used in *NABM*.

Both the candidate selection policy and the location policy play the intermediate roles of generating edges and edges' weight for the *WBG*. The candidate selection policy chooses the VMs to transfer to alleviate an overloaded hot spot. A VM is a candidate if the removal of this VM turns a triggered host into a non-triggered host. However, if the removal of any VM in a trigger node couldn't turn the trigger host into a non-trigger host, then all tenant VMs in this triggered host are chosen for the candidate set. For a triggered host h_a and any dimension of resource i, we identify the candidate set as (2).

$$CE_a=\begin{cases}\left\{\begin{array}{l}vm_\beta \mid A(vm_\beta)=h_a, h_a\in TR\\ \forall i(\max\{u_a^i-s_\beta^i,0\}\le t_i), i\in\{1,2,\ldots,n\}.\end{array}\right\}\\ if\ \exists\beta, \forall i(\max\{u_a^i-s_\beta^i,0\}\le t_i), i\in\{1,2,\ldots,n\},\\ \left\{\begin{array}{l}vm_\beta \mid A(vm_\beta)=h_a\end{array}\right\}, h_a\in TR\\ if\ \forall\beta, \exists i(\max\{u_a^i-s_\beta^i,0\}>t_i), i\in\{1,2,\ldots,n\}\end{cases} \quad (2)$$

The location policy selects the destination host to which the candidate VM previously selected is migrated. A Host h_a in *NTR* is said to be feasible if after migration its capacity constraint C_a is not violated. For any $vm_\beta\in CE_a$ the location policy determines the set of potential destination hosts from the feasible hosts in *NTR* set. Since we attempt not to increase the number of triggered nodes after the migration of VMs, a feasible node can be a potential destination node only if the VM migration wouldn't turn it into a triggered node. The set of potential destination hosts for any candidate VM $vm_\beta\in CE_a$ is shown in (3).

$$EP_\beta=\left\{\begin{array}{l}(vm_\beta,h_a) \mid \forall i(\max\{u_a^i+{s_\beta^i}/{c_a^i},0\})\le t_i,\\ i\in\{1,2,\ldots,n\}, A(vm_\beta)\ne h_a,\\ h_a\in NTR.\end{array}\right\}. \quad (3)$$

This policy then associates a cost with each (vm_β, h_a). The cost is denoted as the sum of the resource requirements of vm_β divided by the residual capacity on n dimensions of the potential destination host h_a. It indicates that a host with scarce remaining resources exhibits a higher cost. The formal load balancing cost function for a VM migration for a pair of hosts (h_{src}, h_{dst}, vm_β) can be defined in equation (4).

$$Cost_{LB}(h_{src},h_{dst},vm_\beta)=\sum_{i=1}^{n}\frac{s_\beta^i}{f_{dst}^i} \quad (4)$$

where vm_β resides on h_{src} which belongs to *TR*, h_{dst} belongs to *NTR*, $s^i{}_\beta$ denotes resource demands of vm_β over each resource dimension i, $f^i{}_{dst}$ denotes the available resource of h_{src}, and n denotes the total number of machine resources. For the candidate VM $vm_\beta\in CE_a$, the location policy selects a destination host out of the potential destination host set according to cost. We can apply one of the strategies in the

best-fit, first-fit, worst-fit, and relaxed-best-fit [4] to determine the destination host. With the best-fit, we select a destination host with the smallest cost. Thus, the potential destination host with higher cost is less likely to be selected as a destination for the candidate VM. However, it requires a linear search from a set of potential destination hosts. Thus, the relaxed-best-fit exhibits a better search time by investigating the smallest cost from a much smaller set of the potential destination hosts which are randomly chosen from the original potential destination hosts. With the relaxed-best-fit, it also reduces the chance of picking the same best potential hosts among different VMs and thereby increases the number of valid edges in a weighted bipartite graph. *NABM* applies only relaxed-best-fit (*RBF*) as in [4].

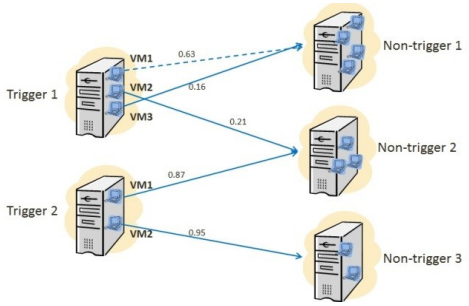

Fig. 4. An illustration of our WBG

For a candidate VM in CE_a, relaxed-best-fit selects a destination host to form an edge in EP_a. Since multiple candidate VMs residing at the same host could select the same destination host, it causes multiple edges between triggered and non-triggered nodes. We discard all such edges except for the edge with the smallest weight. Fig. 4 shows an illustration of *WBG* with two triggered nodes and three non-triggered nodes. Since candidate VM 1 and VM 2 in the Trigger node 1 select the Non-Trigger node 1 as their potential destination host, two potential edges associated with the weights are formed between the Trigger node 1 and Non-Trigger node 1. However, since the solid edge has a smaller weight than the dotted edge, the dotted edge is discarded from our final weighted bipartite graph. For any node pair, the edge set is computed as (5).

$$E = \left\{ \begin{array}{l} (h_{src}, h_{dst}) \mid\ h_{src} \in TR, h_{dst} \in NTR, \\ \exists vm_\beta (\forall vm_j (\beta \neq j, \\ (Cost_{LB}(h_{src}, h_{dst}, vm_j) \geq Cost_{LB}(h_{src}, h_{dst}, vm_\beta))), \\ vm_\beta, vm_j \in CE_{src} \end{array} \right\}. \quad (5)$$

Finally, we construct a weighted bipartite graph: $G=(V,E,W)$ where $V = TR \cup NTR$ and E is the edge set obtained from (5), W is the cost function defined in (4).

After constructing the *WBG, NABM* applies the Hungarian algorithm [12] to solve the minimum weighted matching problem. Since the Hungarian algorithm solves the instance of *WBG* with perfect matching, *NABM* must ensure that the *WBG* created above has an equal number of nodes for *TR* and *NTR*. Thus *NABM* algorithm adds pseudo nodes to either *TR* or *NTR*, whichever one has the smaller number of nodes, until they have the same number of nodes. Pseudo edges associated with a significantly large value also are added from a pseudo node to all nodes in the other set. Based on the outcome of each match from the Hungarian algorithm except for the pairs connected with pseudo edges, *NABM* migrate VMs from triggered nodes to their corresponding non-triggered nodes in parallel for balancing load. Compared with the sequential migrations described earlier, *NABM* not only migrates VMs concurrently but also prevents from exhibiting the migration overhead due to contention at the end hosts' NIC.

NABM iterates the steps of constructing a weighted bipartite graph, finding its minimum weighted bipartite match, and parallel migrating the corresponding VMs until either there's no triggered node left or the match consists of nothing but the pseudo edges. We omit the pseudo-code of *NABM* due to space limit.

C. Network-aware Extension

In addition to the weighted bipartite match that increases the total number of VM migration pairs in each round, the length of the duration between each parallel migration round also plays an important role in reducing total time till balance load. The time between each round depends on the longest migration time among all migrations. Since the network hop distance between two hosts will affect the migration time [7], we further consider network hop distance in the location policy. Like [6], we assume that the traffic patterns rarely change in the production data center. Even though the traffic distribution could be highly uneven, a balanced traffic distribution could still be obtained in the data center network. For example, the cloud operators could reassign traffic flows among communicating VMs via equal-cost-multiple-path (ECMP) [16] or a centralized network controller such as NOX [17].

Under even traffic distribution assumption, VM migration between end hosts with a long network path exhibits a higher probability of long network delay. According to (6), it indicates that the migration time is proportional to the size of VM transfer memory and communication distance while being disproportional to the bottleneck of end hosts' NIC bandwidth. *NABM* extends the previous cost function in (4) to account for migration cost which is related to the network hop distance and the size of VM transfer memory as defined in (6).

$$Cost_{mig}(h_{src}, h_{dst}, vm_\beta) = \frac{s^{mem}_\beta}{\min(f^{bw}_{src}, f^{bw}_{dst})} \cdot D(h_{src}, h_{dst}). \quad (6)$$

where vm_β resides on h_{src} which belongs to *TR*, h_{dst} belongs to *NTR*, $s^{mem}{}_\beta$ is memory volume of vm_β, $f^{bw}{}_{src}$ and $f^{bw}{}_{dst}$ denote the available bandwidth for VM migration in source and destination NIC, respectively, and *D(i, j)* is network hop distance between Host *i* and *j*. An example of *D(i, j)* is to

account for the hop count between end hosts in some data center network architectures such as fat-tree [18] and VL2 [16]. This paper implements *fat-tree* and *VL2* as the underlying network architectures. The extension of *NABM* takes a normalized cost of (4) and (6) as in (7).

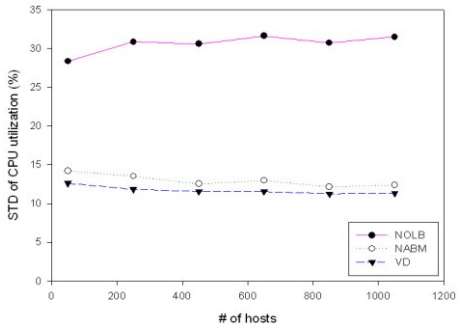

Fig. 5(a). STD of CPU utilization

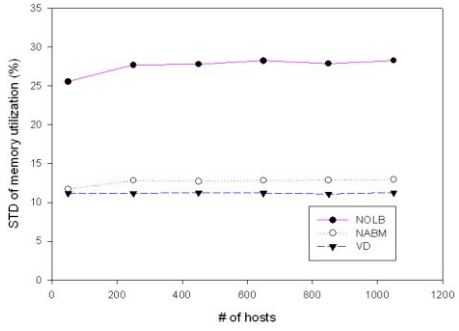

Fig. 5(b). STD of memory utilization

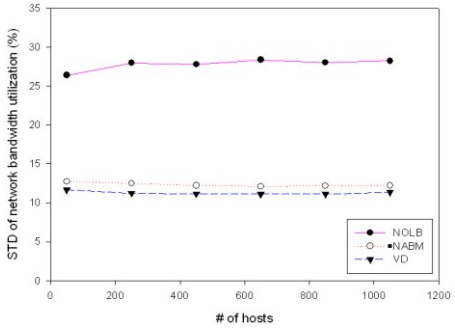

Fig. 5(c). STD of network utilization

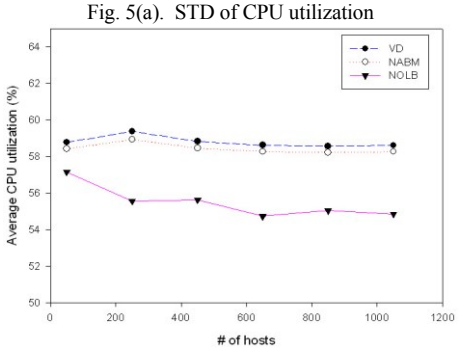

Fig. 6(a). Average CPU utilization

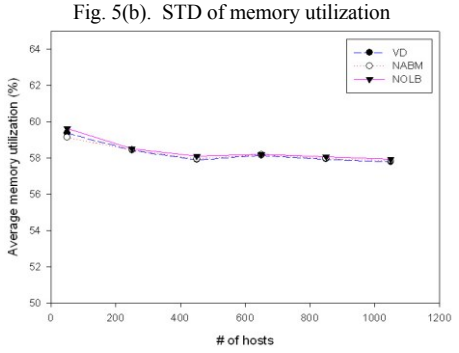

Fig. 6(b) Average memory utilization

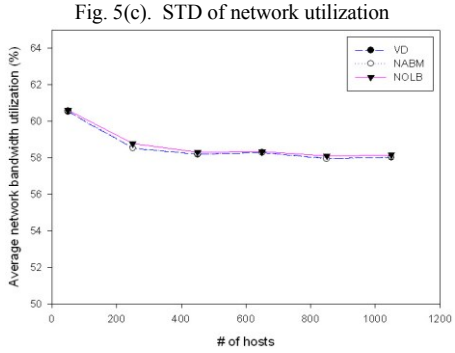

Fig. 6(c). Average network utilization

$$Cost\left(h_{src}, h_{dst}, vm_{\beta}\right) = (1-\alpha)\cdot Cost_{LB}(h_{src}, h_{dst}, vm_{\beta}) + \alpha\cdot(\rho\cdot Cost_{mig}(h_{src}, h_{dst}, vm_{\beta})) \quad (7)$$

where ρ is an adjustable parameter for normalizing the migration cost with load balancing cost; andα is the parameter for adjusting the relative importance between the two costs. For example, it considers there to be no network cost if α is zero. During the construction of the *WBG*, the edge weight is substituted with $Cost(h_{src}, h_{dst}, vm_{\beta})$ in (7) in the location policy. This paper conducts a number of event-driven simulation analyses to study how parameter α affects the time till system is balanced, the mean VM migration time between each round, and the application performances in section VI.

V. Experimental Evaluation

This section studies the effectiveness and scalability of the *NABM* algorithms by comparing them with *VectorDot* (*VD*) [4] using the *NetworkCloudSim* [19] simulation platform.

A. Experimental Settings

To vary the system scale we simulate with 50~1050 hosts and 157~3244 VMs. The baseline host is equipped with a 2.8GHz (approximately 12,000 MIPS) quad-core CPU, 4 GB memory, and 1024 Mbps NIC. The multi-resource and heterogeneous capacity of each host is generated as baseline capacity*(1±heterogeneous degree.) The heterogeneous degree is 0.2. Each VM consumes 12%~25% multi-resource demands of the baseline host capacity. Each VM runs a large-scale Bag-of-Task (*BoT*) application which is independent without the need to communicate with other VMs. The workload consists of 287,712,000 million instructions which take approximately 9 hours to finish for a VM with 8880 MIPS CPU. Typical examples of such workload are biological computation, data mining, and scientific engineering applications. The 37% of hosts are over provisioned, which are considered to be overloaded hosts. The average system utilization along each resource dimension is approximately 57%. In order to accommodate the variance of multi-resource load balancing, the triggered node threshold is set to 75% along each dimension of resources.

For the network parameters, we assume that the workload in VM consumes on average 57% of host NIC bandwidth. This bandwidth is reserved and not used by VM migrations. The network topologies fat-tree and VL2 used in our simulation are built according to [6]. The network parameter α is set from 0 to 1 with 0.2 increment. While $\alpha = 0$, *NABM* would not consider any impact of the network topology. We take a modest value of network weight α equal to 0.6 for the simulation results in Fig. 5, 6, and 7. The available bandwidth of core and aggregate switch reserved for VM migration is set to 1184 Mbps in the following experiments. The parameter ρ is determined as follows. We observe that load balancing cost $Cost_{LB}$ in general is in the range (0, 1.8]. In order to combine the costs of load balancing and migration, we normalize the migration cost to be in the range (0, 1.8]. Thus we take ρ as 1.8 divided by the maximum value of migration cost which is in the range [1000, 2800] in our experiments. For the rest of simulation, we fix ρ = 1.8/1500.

All the performance results are an average of 10 repeated runs and obtained at the moment of system convergence. We use the following metrics to evaluate *NABM* and *VD*: the system utilization and standard deviation of multi-resources of the hosts, the time it takes for the load balancing process to reach a balanced state, the number of VM migrations taken during the load balancing process, and the total application completion time.

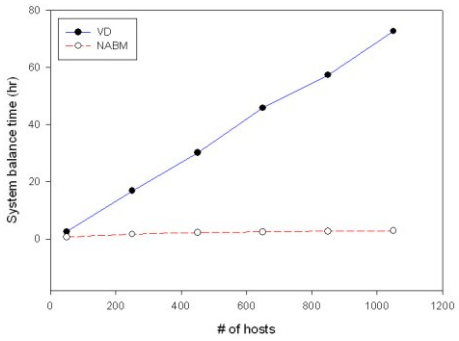

Fig. 7(a) Total time taken to reach a balanced state

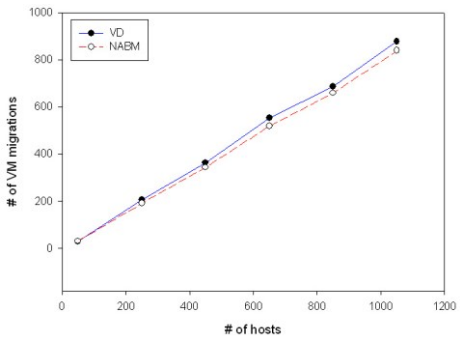

Fig. 7(b) Number of VM migrations to reach a balanced state

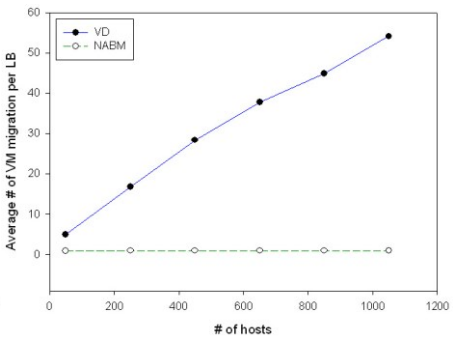

Fig. 7(c) Number of average VM migrations per round

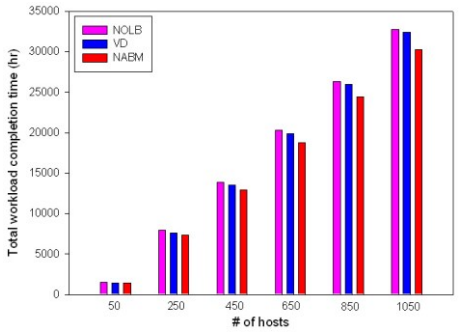

Fig. 7(d) Total batch completion time in different number of hosts

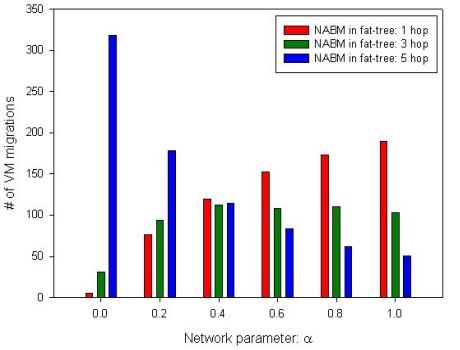

Fig. 8(a) Fat-tree: the number of VM migrations with different hop count

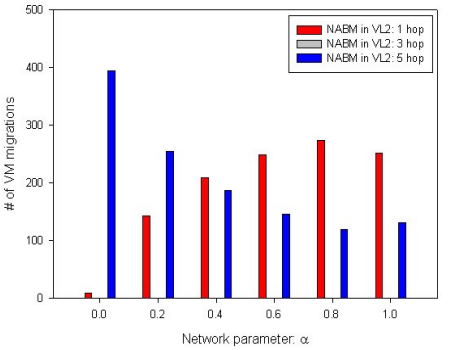

Fig. 8(b) VL2: the number of VM migrations with different hop count

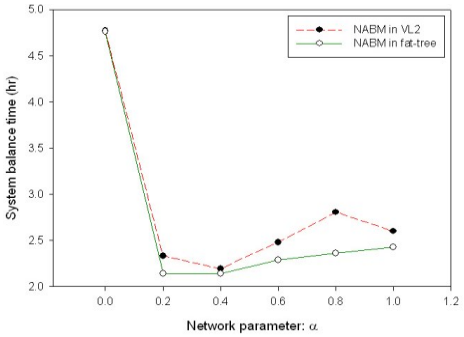

Fig. 8(c) Total time taken to reach a balanced state in fat-tree and VL2

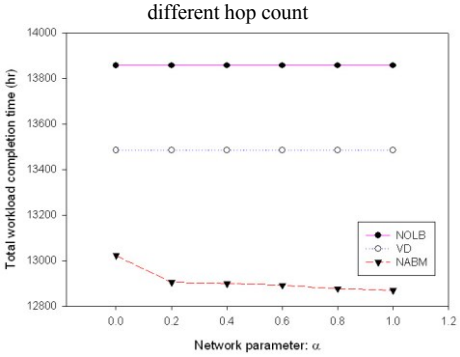

Fig. 8(d) Total batch completion time in fat-tree topology

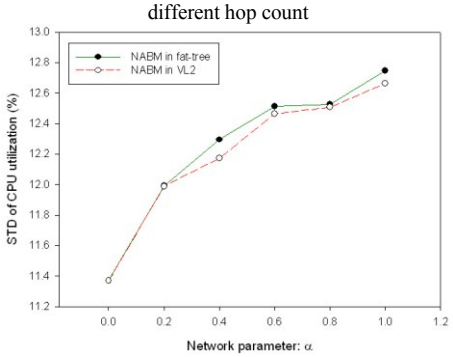

Fig. 8(e). STD of CPU utilization in different network parameters

B. Experimental Results

Fig. 5(a), (b), and (c) and Fig. 6(a), (b), and (c) show the average standard deviation of resource usage of CPU, memory, network bandwidth, and their average resource utilization, respectively, for different numbers of hosts. Both *NABM* and *VD* exhibit a lower resource standard deviation than *NOLB*, which does not do any load balancing. The differences of average standard deviation between *NABM* and *VD* are less than 5% which means that *NABM* still maintains a good degree of imbalance at balanced states compared with *VD*.

Fig. 7 (a) shows that the time for *NABM* to get the system to a balanced state is much less than the time for *VD* especially in a data center with a large number of hosts. For *VD*, the balance time increases along with increases of the number of VM migrations. This is because *VD* sequentially migrates one VM at a time. In contrast, *NABM* concurrently migrates VMs according to the number of independent matching pairs. While the average number of VM migrations per round for *VD* is constant (equal to 1), the average number of VM migrations per round for *NABM* increases as the number of hosts increases as shown in Fig. 7(c). This is because as the number of hosts increases, the matching pairs in the minimum weighted matching also increases. Fig. 7(d) shows that the total application completion time of the *NABM* algorithm is better than VD by 1%, 3%, 5%,7%, 9%, and 10%, respectively, under different numbers of hosts. This is because *NABM* handles multiple overloaded hosts at a time, decreasing the load on those hosts and thus decreasing the degree of resource contention amongst the applications running on the VMs. In contrast, *VD* handles only one trigger node at a time and the algorithm needs to spend a longer times to decrease the amount of trigger nodes. When the number of hosts is increased, there

are more VMs in the hotspot competing for the resources. For *VD*, it will take more rounds (i.e. time) to alleviate the hot spot.

Fig. 8 shows the VM migration time and the time it takes to reach a load balancing state under different weights of network parameter α. As the weight of network parameter α increases, *NABM* can select more hosts with shorter paths to migrate VMs for both fat-tree and VL2 as shown in Fig. 8(a) and 8(b). Fig. 8(c) shows that the time till the system reaches the balanced state is

reduced as the network parameter α increases. It's because migrating VMs along a shorter path could improve the mean migration time. Especially, compared with no network cost or very small network cost cases, considering the network hop distance ($\alpha > 0.2$) saves us more than half of the time it takes to reach a balancing state. With the decline of migration time, network-aware further improves application performances as shown in Fig. 8(d). However, a large network parameter value could still hurt the time till the system reaches balance and application performance. This is why we pick network weight α equal to 0.6 in our previous experiment setting. Fig. 8(e) shows the standard deviation of CPU utilization verse network weight while *NABM* reaches a balance state. When network weight α increases, the standard deviation of CPU utilization also increases. It indicates the tradeoff between VM migration time and load balancing degree.

VI. Conclusion & Future Works

We propose a network-aware multi-resource load-balancing scheme using a parallel VM migration. We transform the parallel VM migration to a minimum weighted matching problem of a weighted bipartite graph in a cloud system. Our algorithm migrates VMs parallel with each other to minimize the time to get the system to a balanced state and thus increases the throughput of overloaded hosts. Since the network hop distance between two hosts will affect the migration time, we further consider network hop distance in our cost function. Simulation results show that our *NABM* algorithm improves the throughput on overloaded machines up to 10% compared with VD. In the future, we will take network bandwidth into consideration. In order to avoid multiple migrations sharing a same network link, a greedy algorithm will be applied to select one VM migration pair at a time until no more migration pairs can be selected. However, the VM migrations still can be performed in parallel.

ACKNOLEGEMENT

This work has been supported in part by a garnt from D-Link Systems, Inc.

References

[1] T. Chieu, A. Mohindra, A. Karve and A. Segal, "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment," in *ICEBE*, 2009.

[2] Y. Zhao and W. Huang, "Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud," in Fifth International Joint Conference on INC, IMS and IDC, 2009, pp. 170-175.

[3] M. Randles, et al., "A comparative study into distributed load balancing algorithms for cloud computing," in IEEE 24th International Conference on Advanced Information Networking and Applications Workshops, 2010, pp. 551-556.

[4] A. Singh, et al., "Server-storage virtualization: integration and load balancing in data centers," in Proceedings of the 2008 ACM/IEEE conference on Supercomputing, 2008, p. 53.

[5] V. Shrivastava, et al., "Application-aware virtual machine migration in data centers," in Proceedings IEEE INFOCOM, 2011, pp. 66-70.

[6] X. Meng, et al., "Improving the scalability of data center networks with traffic-aware virtual machine placement," in Proceedings IEEE INFOCOM, 2010, pp. 1-9.

[7] H. Xu and B. Li, "Egalitarian stable matching for VM migration in cloud computing," in IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2011, pp. 631-636.

[8] S.-H. Lim, et al., "Migration, assignment, and scheduling of jobs in virtualized environment," *ACM USENIX workshop HotCloud*, 2011, vol. 40, p. 45, 2011.

[9] K. Ye, et al., "Live migration of multiple virtual machines with resource reservation in cloud computing environments," in IEEE International Conference on Cloud Computing, 2011, pp. 267-274.

[10] D. Arora, et al., "On the benefit of virtualization: Strategies for flexible server allocation," in Proc. USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), 2011.

[11] M. Bienkowski, et al., "Competitive analysis for service migration in vnets," in Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures, 2010, pp. 17-24.

[12] H. W. Kuhn, "The Hungarian method for the assignment problem," Naval research logistics quarterly, vol. 2, pp. 83-97, 2006.

[13] Xen. Available: http://www.cl.cam.ac.uk/research/srg/netos/xen/

[14] SPECjvm. Available: http://www.spec.org/

[15] K. P. Bubendorfer and J. H. Hine, "A compositional classification for load-balancing algorithms," technical report, No.CS-TR-99-9, 1998.

[16] A. Greenberg, et al., "VL2: a scalable and flexible data center network," in ACM SIGCOMM Computer Communication Review, 2009, pp. 51-62.

[17] NOX. Available: http://www.noxrepo.org/

[18] M. Al-Fares, et al., "A scalable, commodity data center network architecture," in ACM SIGCOMM Computer Communication Review, 2008, pp. 63-74.

[19] S. K. Garg and R. Buyya, "NetworkCloudSim: Modelling Parallel Applications in Cloud Simulations," in Fourth IEEE International Conference on Utility and Cloud Computing, 2011, pp. 105-113.

[20] D. Kliazovich, et al., "DENS: data center energy-efficient network-aware scheduling," in Cluster Computing, Volume 16, Issue 1, March 2013, pp 65-75.

[21] D. Kliazovich, et al., "Accounting for load variation in energy-efficient data centers, " in IEEE International Conference on Communication (ICC), 2013, pp. 1154-1159