Network-Aware Coordination of Virtual Machine Migrations in Enterprise Data Centers and Clouds

Haifeng Chen(1) Hui Kang(2) Guofei Jiang(1) Yueping Zhang(1)

(1) NEC Laboratories America, Inc.
4 Independence Way, Princeton, NJ 08540
{haifeng, gfj, yueping}@nec-labs.com

(2) SUNY Stony Brook University
Stony Brook, NY 11794
hkang@cs.sunysb.edu

***Abstract*—Virtual machine(VM) migration usually requires a considerable amount of system resources such as the network bandwidth. In the case of multiple simultaneous migrations, such resource demands will increase dramatically and are difficult to be satisfied immediately. This paper proposes a scheduling method for multiple VM migrations to guarantee the fast completion of those tasks and hence the reduced impacts on system performance. We discover the best bandwidth sharing policy for each network link, and further propose a bin-packing algorithm to organize bandwidth resources from all the network links. As a result, the migration tasks can fully utilize available resources in the whole network to achieve the fast completion.**

***Keywords*-VM migration; scheduling; simulation; bin packing**

I. Introduction

Multiple VM migrations[1] show up regularly in real system operations. For instance, if some physical machines need to be removed from service for maintenance, all the VMs in those machines have to be migrated to other places. Since applications are nowadays comprised of many VMs distributed across several machines due to load balancing and fault tolerance, the workload surge in an application may require the rearrangement of several VM instances in the system. An even worse situation is that some system faults such as configuration mistakes may trigger a large number of VM migrations. In those cases, it is important to handle concurrent VM migrations in an effective way, so that they can be completed as fast as possible to minimize the total performance degradation time for those VMs.

There are several challenges when multiple VMs request to migrate simultaneously. First, since those migrations may have overlapped links in their migration paths, we need to determine whether to let them share the link by initiating them concurrently, and what is the maximum number of concurrent migrations allowed in that link. The link sharing between multiple migrations can improve the overall utilization of network bandwidth due to the resource multiplexing between migration flows, and thus contribute to the quick completion of migrations. However, the amount of transferred memory pages also increases since each VM is only allocated with a portion of bandwidth in the overlapped links. We need to find a balance in determining the optimal number of concurrent migrations that share the network link.

This paper considers two aspects to address those challenges. First, we analyze the VM migration behavior and build a simulation tool to predict the time of multiple migrations under different links conditions and VM characteristics. By running the simulation tool, we can compare VM migration performance under different conditions, and then generate the optimal sharing policy for each link, i.e., the number of concurrent VM migrations that can achieve the minimal total migration time, based on the link's available bandwidth and VM's memory page dirty rate.

Given the link sharing policy, we further propose a bin-packing algorithm to organize bandwidth resources from all the network links, and allocate them to different migration tasks. The bins in the algorithm represent all the links in the network and the item denotes each migration task. While the capacity of the bin is determined by the available bandwidth in all network links, the size of each item is associated with the bandwidth demand of each migration, which can be estimated from the migration sharing policy in each link along that VM's migration path. Given the bin capacity and item sizes, we use the first-fit decreasing (FFD) heuristic to allocate each migration to a corresponding bin so that the total number of bins to host those migrations is minimized.

We have evaluated our migration scheduling algorithm by simulating different numbers of VM migrations in our test bed systems. Results show that with the help of our migration scheduler, we can achieve the fast completion of multiple VM migrations.

II. Network Link Sharing

Our study focuses on the pre-copy migration technique[1] implemented in common virtualization software such as Xen and VMware. It makes use of an iterative multi-pass algorithm to transfer VM guest memory in successive steps. In each iteration, only the memory that has been dirtied in the interim is sent. When the pre-copy stage is terminated, the final state is sent to the new host and the transfer of control to the new physical machine is completed.

When executing s migrations simultaneously in one network link, the total migration time $T^{(pal)}$ becomes

$$T^{(pal)} = \max\{T_{mig-1},\ T_{mig-2},\ \cdots,\ T_{mig-s}\} \qquad (1)$$

where T_{mig-i} is the duration of the ith migration. Note that the value of T_{mig-i} varies with number of concurrent migrations s due to the differences in allocated bandwidth. In this section, we estimate T_{mig-i} under various setting of s, and identify the best network link sharing strategy, i.e., the s concurrent migrations that lead to the shortest total migration time. We use the normalized value $T^{(pal)}/s$ to compare the overall migration time under different s values.

When $s = 1$, it corresponds to the case when migrations are performed sequentially. With the increase of s value, more migrations are executed in parallel. Running s migrations in parallel can improve the overall utilization of network bandwidth due to the resource multiplexing between migration flows. However, the amount of transferred memory pages also increases since each VM is only allocated with a portion of bandwidth in the overlapped links. In order to find the best s, we need to first predict the VM migration performance under different link sharing strategies. However, due to the complexity of VM migration, its duration depends on several factors such as the available bandwidth in the link and VM memory dirty rates. Table I presents two examples to illustrate this, in which we compare the time of two migrations when they are executed in a sequential order and simultaneously. In the first example, we migrate the VMs with 1GB memory in a link with 1Gbps bandwidth. It shows that when the VMs have 2k memory dirty pages per second, it takes only 20 seconds for parallel migration to complete, whereas sequential migration consumes 22 seconds. However, when the VM memory dirty rate increases to 15k pages per second, parallel migration becomes slower than the sequential one. In the second example, we compare the migration time of two VMs with 1GB memory and 10k memory pages per second dirty rate in different link situations. It shows that while parallel migration is faster when two migrations are executed in a link with 1Gbps available bandwidth, sequential migration becomes much faster when the link bandwidth drops to 300 Mbps.

	VM page dirty rate	link bandwidth	sequential migration	parallel migration
Case 1	2 k/s	1 Gbps	22s	20s
	15 k/s	1 Gbps	27s	38s
Case 2	10 k/s	1 Gbps	23s	21s
	10 k/s	300 Mbps	78s	90s

TABLE I
TOTAL TIME OF VM MIGRATIONS WHEN THEY ARE EXECUTED SEQUENTIALLY AND IN PARALLEL.

Due to nonlinear dependency of migration performance with respect to factors such as the link capacity and the VM memory dirty rate, it is hard to predict the VM migration time by some mathematical formulas. As an alternative, this paper proposes to use software simulation to identify the optimum link sharing policy under different VM and link conditions. Our simulation follows the source code implementation in Xen to predict the VM migration time given the available link bandwidth and VM characteristics. In the case of multiple VM migrations, we incorporate several extra factors in the simulation. For example, some migration overheads, such as the time spent in the initial resource reservation and final VM activation in the target machine, can be saved by the parallelism of multiple migrations. We also model behavior of bandwidth usages when multiple migrations share the network link. By running the simulation tool, we can compare VM migration performance under different conditions, and then generate the optimal sharing policy for each link, i.e., the number of concurrent migrations that can achieve the minimal total time based on the link's available bandwidth and VM's memory page dirty rate.

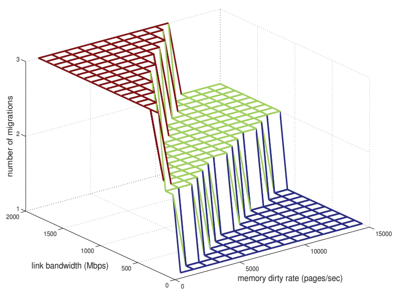

Fig. 1. The optimal link sharing policy for multiple migrations with respect to the link bandwidth and VM memory dirty rates.

By using our simulation tool, we evaluate the total VM migration time under various link sharing strategies, given different VM characteristics and link available bandwidth. From the simulation results, we find that the optimal link sharing mainly depends on two factors: the link available bandwidth, and the memory dirty rates of migrating VMs. This is because that those two metrics determine the size of extra contents, in addition to the original VM memory, that need to be transferred during the migration. After summarizing many simulation scenarios, we obtain the optimal s number of concurrent migrations, given specific link bandwidth and VM memory dirty rates. Some of our simulation results are shown in Figure 1.

III. Global Resource Assignment

In reality, the whole network is comprised of a large number of communication links organized by certain topology design. Meanwhile, each migration usually covers a set of network links in its migration path, i.e., from the source to destination machines. Based on the bandwidth sharing policy in each link, this section attempts to find an optimal assignment of *global* network resources for multiple migrations to achieve the minimal total migration time.

It becomes a combinatorial optimization problem to find the best organization of many migrations with different overlapped links along their migration paths. In this paper, we use a bin-packing algorithm [4] to address that issue. We treat all the links in the network as a bin, and use a multi-dimensional vector $\mathcal{C}$ to represent its capacity. That is, we index each link in the network, and measure the available bandwidth in those links as

$$\mathcal{C} = [c_1, c_2, \cdots, c_r]^\top \tag{2}$$

where r equals to the number of links in the bin. In practice, the value of r depends on the number of physical links in the network as well as the network configurations. For example, when the network is operated in the full-duplex mode, which is typical in network configurations, the value r equals to twice the size of network links due to the differentiation in traffic directions. If the network is configured by equal-cost multipath (ECMP) load sharing [3], we need to combine those multiple links that are used for balancing the load into a logic link, and only include the logical link in the vector (2). We will discuss this more in the section of experiments.

This item in our bin-packing algorithm relates to each migration task. Given the indices of network links, we use a r dimensional binary vector $\mathcal{P}^{(i)} = [1, 0, 0, \cdots, 1]^\top$ to represent the path of migration M_i, in which the value '1' in the ith entry indicates the inclusion of the ith link in the path and '0' vice versa. The end-to-end bandwidth demand for migration M_i is defined as a vector

$$\mathcal{D}^{(i)} = \mathcal{P}^{(i)} \times d^{(i)} = [1, 0, 0, \cdots, 1]^\top \times d^{(i)} \tag{3}$$

$d^{(i)}$ is the expected bandwidth allocated to M_i, which is determined as following.

We denote the links in M_i's path as a set $\{l_1^{(i)}, l_2^{(i)}, \cdots, l_k^{(i)}\}$, each of which has available bandwidth c_j, $j = 1, \cdots, k$. Given the available capacity c_j of link $l_j^{(i)}$ and a migrating VM with memory page dirty rate R_i, we can identify its optimal bandwidth sharing policy $s_j^{(i)}$ in that link from the simulation results in Section II, which represents the optimal number of such VMs that can migrate simultaneously in the link to achieve the minimal total migration time. We then determine the bandwidth demand of M_i in link $l_j^{(i)}$ as

$$d_j^{(i)} = c_j / s_j^{(i)} \tag{4}$$

That is, the local bandwidth demand $d_j^{(i)}$ of M_i is determined to allow $s_j^{(i)}$ such concurrent migrations in the link to achieve the optimal migration time. Once we find the bandwidth demand $d_j^{(i)}$ for all the links $l_j^{(i)}$, $j = 1, \cdots, k$ along M_i's path, the overall bandwidth demand of M_i is determined as the maximum demand among all the local estimations

$$d^{(i)} = \max\left\{d_1^{(i)}, d_2^{(i)}, \cdots, d_k^{(i)}\right\} \tag{5}$$

The intuition here is that the overall demand $d^{(i)}$ should satisfy all the local demands $d_j^{(i)}$ to ensure the quick completion of migrations.

Now given the capacity of the bin and the resource demands of all items, the bin-packing algorithm is to pack those items into the smallest number of bins. By doing so, we can achieve the quickest completion of all migrations, because the number of bins generated by bin-packing represents the total duration of those migrations. In Figure 2, we demonstrate such a process by first assuming that all migrations take the same amount of time T. The x axis in the figure denotes the time, which is divided into b intervals T_1, T_2, $\cdots$,T_b, with equal length T. Since all migrations are assumed to have the

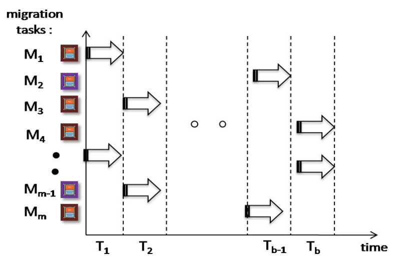

Fig. 2. Migration scheduling is regarded as a bin-packing process.

same duration T, the network has the bandwidth capacity $\mathcal{C}$ at the beginning of each epoch T_i. The y axis in Figure 2 represents the migration tasks. Considering the link sharing policy discussed in Section II, we can only initiate a subset of migrations in each epoch T_i. Our bin-packing solution is to find an optimal assignment of VM migrations into those epochs, so that the total migrations can be completed in the shortest time.

The bin-packing problem is NP-hard, and there have been a number of heuristics [2] to identify its near optimal solution. In this paper we use the first-fit decreasing (FFD) heuristic to schedule the migrations. The FFD method sorts migration tasks in a decreasing order of bandwidth demands, and attempts to place each task in the earliest epoch that can accommodate it. More concretely, the FFD based migration scheduling can be described in the following steps.

1) Transform the resource demand vector $\mathcal{D}^{(i)}$ for each migration M_i, described in equation (3), into a scalar $\eta^{(i)}$, $\mathcal{D}^{(i)} \rightarrow \eta^{(i)}$, where $\eta^{(i)}$ equals to the summation of all the elements in $\mathcal{D}^{(i)}$.
2) Sort the resource demands based on their transformed $\eta^{(i)}$ values ;
3) Scan migration tasks from high to low $\eta^{(i)}$s. For each selected migration task, we try to place it in the earliest time interval T that still has free capacities to host it;
4) Repeat Step 3) until all the migration jobs have been allocated.

In practice, the migration tasks have different time durations due to their variances in VM memory size, memory page dirty rate, and so on. There are no clear boundaries between each time slot, described as vertical dash lines in Figure 2, to synchronize migration tasks. In those general situations, we still use the FFD based heuristics to schedule migrations. However, the start of each migration is triggered by event signals rather than the time. That is, when a migration is completed, it sends a signal to the migration scheduler. Upon receiving that event, the scheduler computes the current available bandwidth in the network links

$$\mathcal{C}^{(new)} = \mathcal{C} - \sum_{\text{active migrations } M_k} D^{(k)} \tag{6}$$

where $\mathcal{C}$ is the original link capacity described in equation (2) and $D^{(k)}$s are the bandwidth demands of ongoing migrations. We regard $\mathcal{C}^{(new)}$ as the current bin size, and scan the ordered migration tasks in the waiting list with an attempt to allocate as many of them as possible to fill the capacity $\mathcal{C}^{(new)}$. The whole process stops when all the VMs have migrated to their target machines.

IV. Experimental Results

We use the data generated from our simulation tool to evaluate the performance of multiple VM migrations in a typical two-tiered data center network as shown in Figure 3. There is an aggregation switch at the top of the network hierarchy, which connects to eight edge switches via 10Gbps network links. Each edge switch in turn connects to sixteen servers via 1Gbps network links. We create background traffic in the network. While the traffic in 1Gbps network links is randomly generated from a predefined range, the traffic in 10Gbps links is the aggregation of incoming traffic from its associated 1Gbps links. There are four virtual machines in each physical server. Each VM has 1GB memory size, and its memory dirty rates is randomly chosen between 4k to 8k memory pages per second. As a result, the system we simulated consists of 64 physical servers and 256 virtual machines in total. Note that since our focus here is the network impact on migration performance, we assume that all the physical machines have enough CPU and memory resources to host VMs.

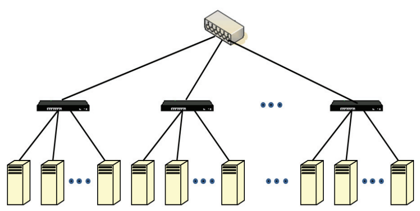

Fig. 3. The system for evaluating the migration performance.

We generate different numbers of VM migrations in the system and simulate the migration time. For each migration, the source VM is randomly selected from machines connecting to the left two edge switches, and its destination is from those machines connecting to the right two edge switches. We use the bin-packing method to schedule those migrations. Here the network links are configured in the full-duplex mode. As a result, the number of links in each bin, i.e., the size of the vector $\mathcal{C}$ in equation (2), is twice the number of physical links in the network. The resource demand of each migration is determined by the VM's memory dirty rate and the available bandwidths of all links along its migration path. Given the bin capacity and resource demand of each migration, we vary the number of VM migrations in the system and simulate the migration time under those different situations. In order to demonstrate the superior performance of our approach, we compare our results with the performance of the fixed k-simultaneous migrations, where maximal k migrations are executed simultaneously in each round and the new round starts only after the previous k migrations are all completed. In the experiments we choose $k = 4$ and 8 for the comparison.

Figure 4 presents the results of VM migrations when they are executed under two different network bandwidth conditions. While in Figure 4(a) we generate around 500 Mbps background network traffic in each second-tier link to create a relatively light-load network, Figure 4(b) increases the background traffic to have a heavily loaded situation.

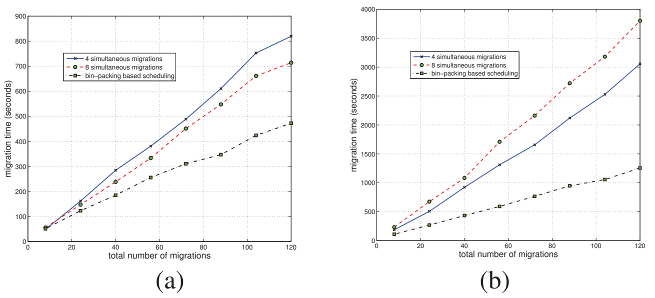

Fig. 4. The duration of different number of VM migrations when they are executed under (a) lightly loaded and (b) heavily loaded network conditions.

The x axis in both figures represent the number migrations that need to execute, and the y axis represents the migration time. The curves are obtained from the average performance of ten repeated simulations, in which the solid and dash lines denote the results of 4-simultaeous migration and 8-simultaeous migration respectively, and the dash dot lines represent the results of our scheduling method. As we can see, when the network is lightly loaded, the 8-simultaeous migration completes faster than the 4-simultaeous migration, because the large network bandwidth allows more simultaneous migrations to share the resource for accelerating the task. On the other hand, when the network available bandwidth is limited, the 4-simultaeous migration performs better than the 8-simultaeous migration, as shown in Figure 4(b). This is due to the large amount of dirty memory pages generated by the 8 simultaneous migration under the bandwidth limited condition. Nevertheless, compared with the strategy of fixed k-simultaneous migrations, our bin-packing based scheduling works much better in both situations. It can automatically adapt to the network conditions and generate the migration schedule with the shortest completion time.

V. Conclusions

This paper has proposed a novel method to coordinate multiple VM migrations in enterprise data centers and clouds. It has considered the migration sharing in each network link, as well as the global network bandwidth assignment for migration tasks. While the network link sharing has been addressed by software simulation, we have proposed a bin-packing algorithm to deal with the global resource assignment. As a result, the total time for those migration tasks can be minimized. The experiments have validated the effectiveness of our approach.

References

[1] C. Clark, K. Fraser, S. Hand, and et. al. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (NSDI '05)*, pages 273–286, Berkeley, CA, 2005.

[2] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1997.

[3] C. Hopps. Analysis of an equal-cost multi-path algorithm, 2000.

[4] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 4th edition, 2007.