2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops

SmartShuffle: Managing Online Virtual Machine Shuffle in Virtualized Data Centers

Peng Zhang, Hongbo Wang, Junbo li, Jiankang Dong, Yangyang Li, Shiduan Cheng
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications, Beijing 100876, China
Email: {zhp, hbwang, lijunbo, dongjk, yyli, chsd}@bupt.edu.cn

***Abstract*—Virtual machine (VM) live migration provides spatial flexibility by rearranging VM placement (i.e., VM shuffle) in several scenarios, including server consolidation, power consumption saving, fault tolerance, QoS management and network congestion resolving. However, VM live migration would consume scarce bandwidth and even cause network congestion. Since the bandwidth used by VM migration is usually the same as the services running in the VM, migration traffic would dominate network path and affect other application traffic as the traffic of a VM migration is usually several GBs. It gets worse in VM shuffle where plenty of VMs are needed to be moved. In this paper, we explore the opportunity to manage online VM shuffle and minimize the impact to data center networks. An efficient online VM shuffle scheduling method named *SmartShuffle* is presented. SmartShuffle tries to minimize the VM shuffle duration by coordinating VM migration in a proper scheduling order. VMs benefiting others maximally are migrated preferentially. We employ the simulated annealing algorithm to search for a solution for SmartShuffle. Our evaluation shows that SmartShuffle decreases the shuffle duration dramatically.**

Keywords- cloud computing; virtualized datacenter; virtual machine live migration; virtual machine shuffle; scheduling

I. INTRODUCTION

Virtualization is a technique to run several virtual machines (VMs) simultaneously on one physical server. It can offer many benefits, including application isolation, resource sharing, fault tolerance, portability and cost efficiency [1]. With its potential to reduce capital expenses and energy costs, Virtualization has become an indispensable practice in the design and operation of modern data centers. Both cloud providers and enterprises are looking to gain economical revenues from underutilized IT resources.

Another advantage of virtualization is VM live migration which allows VMs to be moved transparently from one physical server to another, while the VMs are continuously running. It provides a new spatial flexibility by rearranging VM placement (i.e., VM shuffle) on the fly in several scenarios, including power consumption saving, server consolidation, fault tolerance, QoS management and network congestion resolving [2]-[9].

Although VM live migration is widely used, it does not come along without any negative impact, but consumes scarce bandwidth and even causes network congestion. Since the bandwidth used by VM migration is the same as the services running in the VM, migration traffic would dominate network path and affect other application traffic as the traffic of a VM migration is usually several GBs. It gets worse in VM shuffle where plenty of VMs are needed to be moved. And Results in [9] suggest that 5-10% of VMs in the data center are needed to be moved to resolve network congestion every round. Therefore, managing VM shuffle is an important issue. However, few works has focused on it.

In this paper, we explore the opportunity to manage online VM shuffle and try to minimize its impact to data center networks. We choose to minimize the VM shuffle duration time by coordinating a proper scheduling sequence for VM migration. The duration is the total elapsed time to move all VMs to their target hosts. Shorter VM shuffle duration means smaller impact to data center network because of less occupation of scarce bandwidth.

We formulate VM shuffle scheduling issue as an optimization problem, which is shown to be a variation of the NP-hard Travelling Salesman Problem (TSP). We therefore design *SmartShuffle*, an efficient online VM shuffle scheduling method. We employ the simulated annealing algorithm to search for a solution for SmartShuffle.

The basic idea of SmartShuffle is the VMs benefiting others maximally should be migrated preferentially. Because the VM shuffle duration is not only related to VMs to be moved, but also the VM scheduling order. Different with single VM, VMs in the same shuffle probably interact with each other. Once a VM is moved, the traffic will be moved from the source physical server to the target physical server. The traffic of physical links would be changed. Then the left VMs migration may be benefited. Then the VM shuffle duration time will be reduced.

The remainder of this paper is organized as follows. In Section II we introduce related works. The SmartShuffle design is presented in Section III, including the motivation, the problem formulation and the SmartShuffle method. We evaluate our design using a discrete event simulator in Section IV. Finally, Section V concludes the paper.

II. RELATED WORK

A. VM Live Migration

Live migration allows VMs to be moved transparently from one physical server to another, while the VMs are not stopped. Virtualization use *pre-copy* [10] to enable VM live migration. Pre-copy consists of the following two phases:

(1) Pre-copy phase.

At this stage, VM memory is iteratively copied from the source to the target server while the VM continues to run. It starts with transferring all active memory. Then pages dirtied

978-0-7695-5023-7/13 $25.00 © 2013 IEEE
DOI 10.1109/ICDCSW.2013.51

have to be re-sent in an additional round to ensure memory consistency.

There are three pre-copy stop conditions and when any one of them is met the pre-copy phase is stop: a) the num number of pre-copy cycles exceeds the pre-defined threshold ($n > n_{th}$). b) The total amount of memory that has already been transmitted exceeds a pre-defined threshold ($v > v_{th}$). c) The number of pages dirtied in the previous round falls below a pre-defined threshold ($p < p_{th}$).

(2)Stop-and-copy phase. At this stage, the hypervisor suspends the VM to stop page dirtying and copies the remaining dirty pages as well as the state of the CPU registers to the destination server. After the migration process is completed, the hypervisor on the target server resumes the VM.

B. VM Placement & Online VM Shuffle

Recently, several studies leverage VM placement to optimize power consumption saving, server consolidation, fault tolerance and QoS management [2]-[5]. However, all of these works do not take into account the network resource. In [6], Meng *et al*. presents an approach of manipulating VM placement to localize large chunks of traffic and thus reduce network load. In [7], Shrivastava *et al*. propose an efficient mechanism for balancing load of physical machines while minimize the network traffic inside data centers. In [8], Jiang *et al*. combine VM placement and routing and get the benefits of the joint design. However, all the above work only rearranges VM placement but not consider the cost of multiple VM migration which cannot be ignored.

The most related work to ours is VirtualKnotter [9]. VirtualKnotter focuses on controllable migration traffic while minimizing continuous congestion by enabling online VM placement at the granularity of tens minutes. However, VirtualKnotter just limit the number of moved VMs but do not consider the process of VM shuffle. While VM shuffle is what this paper focuses on and we try to find out the best VM scheduling order to minimize the impact of VM shuffle to data center network.

III. SMARTSHUFFLE DESIGN

In this section, we present *SmartShuffle*, a heuristic method to the online VM shuffle scheduling problem which tries to shorten VM movement duration by calculating a proper VM sequence. We first introduce the idea behind SmartShuffle then formulate the VM shuffle scheduling problem using optimization language and analyze its complexity. Finally, the simulated annealing algorithm is employed to search for a solution.

A. Motivation

VMs may compete for link bandwidth when they are migrated, so it is better to schedule VMs sequentially instead of migrating them simultaneously. VMs in the same shuffle probably interact with each other. Once a VM is moved, the traffic will be moved from the source physical server to the target physical server. The traffic of physical links would be changed. Then the left VMs migration may be affected.

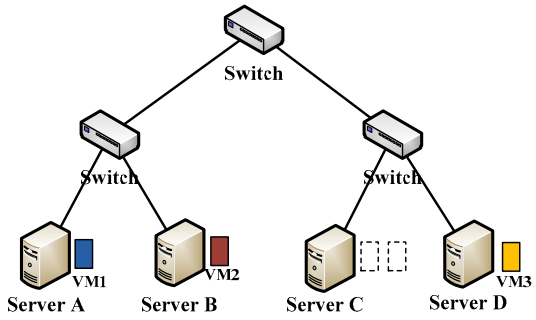

Figure 1. A simple scenario of SmartShuffle (both VM1 and VM2 will be moved to Server C)

The basic idea of SmartShuffle is VMs benefiting others more should be migrated preferentially. Then the latter VMs will get more bandwidth for migration. So the VM shuffle duration will be reduced. It should be noted that the benefit may be negative. But the idea of SmartShuffle holds true since the bigger "negative benefit" (i.e., negative gain) brings less impact on others.

Before introducing SmartShuffle method, we present a simple example to point the effectiveness of SmartShuffle, as shown in Fig.1. There are two VMs needed to be moved. VM1 and VM2 are located on server A and server B respectively and both of them will be migrated to server C. There is application traffic among VMs. VM1 sends data at 200Mbps to VM3 and VM2 also sends data at 400Mbps to VM3. The capacity of all the links is 1 Gbps. It is easily seen that the links among switches is the bottleneck for VM migration and the maximum available bandwidth for migration is 400Mbps.

We assume that both VMs are identical. The memory size and dirty page rate are 2GB and 50Mbps respectively. And all migration lasts five pre-copy cycles. If both VMs are moved simultaneously, the migration rate of two VMs is 200Mbps under the assumption that the available bandwidth is divided fairly. So the migration time of two VMs is 106.6s and the shuffle duration is 106.6s. If VM1 is scheduled firstly, the migration time of VM1 is 45.7s. After VM1 migrating, the available bandwidth of bottleneck link is increased to 600Mbps. So the migration time of VM2 is 29.1s. The sum is 74.8s and is reduced by 29.8% to simultaneously moving. In the same way, if VM2 is moved first, the migration time of VM2 is also 45.7s. After VM2 migrating, the available bandwidth of bottleneck link is increased to 800Mbps. So the migration time of VM2 is 21.3s. The sum is 67s and is reduced by 37.6% to simultaneously moving and reduced by 10.4% to VM1 first. As we can see from this example, the bandwidth gain of VM2 is bigger than VM1, so VM2 migration benefits the other VM more and should be scheduled preferentially.

B. Problem Formulation

In this subsection, we formulate the VM shuffle scheduling problem using optimization language and analyze its complexity.

TABLE I. KEY NOTATION AND ITS MEANING

Symbol	Description
X'	Current VM placement. A binary value $X_{i,s}$ indicates whether VM i is located on server s.
X	Target VM placement.
V	The VM set will be migrated from X' to X.
G	Network topology
C_l	The bandwidth capacity of link l
P	Network routing. A binary-value function $P_{s,d}(l)$, meaning whether the traffic path from server s to d traverses through link l.
M	VM traffic matrix. $M_{i,j}$ denotes the traffic volume from VM i to VM j.
B_i	Migration rate of V_i
D_i	Memory size of V_i
R_i	Dirty page rate of V_i
n	The number of pre-copy cycles in a VM migration
n_{th}	The first pre-copy cycle threshold of pre-copy cycles
v_{th}	The second pre-copy cycle threshold of the total amount transmitted data
p_{th}	The third pre-copy cycle threshold of pages dirtied in the previous round
t_i	The migration time of V_i
Q	The scheduling VM sequence in VM shuffle
T	The VM shuffle duration time

Let the current VM placement in the data center be represented by X' and the optimized VM placement by X. Let V be the set of VMs to be migrated in the shuffle.

We define $X_{i,s}$ as

$$X_{i,s} = \begin{cases} 1, & \text{if } V_i \text{ is placed on server } s \\ 0, & \text{otherwise} \end{cases}.$$

$X_{i,s}$ is a binary value indicating whether VM i is located on server s.

Then we can get V and the migration sources and the targets from X' and X.That is,

$$V = \{V_i \mid i \in \{n \mid X'_{n,s} - X_{n,s} = 1\}\}.$$

Hence, the following constraints must be satisfied:

$$\sum_s X_{i,s} = 1, \forall i, V_i \in V. \quad (1)$$

This equation dictates that each VM must be placed on at most one physical server. The same is hold for $X'_{i,s}$.

Next, we assume data center is connected with a hierarchical structure G, such as multi-root tree, which is mostly used currently. We further assume a deterministic routing P is applied in data center network. We denote the network routing by a binary-value function $P_{s,d}(l)$, meaning whether the traffic path from server s to d traverses through link l. And l is a physical link between two switches.

We let the traffic matrix among VMs represented by M. $M_{i,j}$ denotes the traffic volume from VM i to VM j. we assume the $M_{i,j}$ for a certain period of time is stable. The bandwidth capacity of link l is C_l. When V_i is migrated from server s to server d with a rate B_i, the following constraints must be satisfied:

$$P_{s,d}(l)\sum_{i,j} X_{i,s}(M_{i,j} + B_i)X_{j,d} \le C_l. \quad (2)$$

Furthermore, we analyze the migration time of a single VM. Our analysis is according to [11]. But our pre-copy stop condition is based on XEN [14] and analysis of [11] is based on VMware vMotion [15]. The memory size of V_i is denoted by D_i and the page dirty rate is by R_i. Assume that pre-copy technique is used in VM live migration, so there are n pre-copy cycles and one final stop-copy cycle. In the first cycle, the migration traffic equals to the entire memory size D_i and takes time D_i/B_i. During that time, D_iR_i/B_i amount of memory becomes dirty, and hence, the second cycle results in $D_i(R_i/B_i)^2$ amount of traffic. So the k-th cycle will result in $D_i(R_i/B_i)^k$ for $k \ge 1$.

The total traffic generated by migration VM V_i, including n pre-copy and one stop-and copy phases, is

$$\sum_{k=0}^{n+1} D_i(R_i/B_i)^k = D_i\frac{1-(R_i/B_i)^{n+1}}{1-R_i/B_i}.$$

So the migration time t_i of V_i is,

$$t_i = D_i\frac{1-(R_i/B_i)^{n+1}}{B_i - R_i}. \quad (3)$$

Next we will analyze the actual pre-copy cycles n. Because of the pre-copy stopping conditions, the actual pre-copy cycles may smaller than the pre-defined threshold n_{th}. From the first pre-copy stopping condition, we have

$$n = n_{th}.$$

From the second pre-copy stopping condition, we have

$$D_i\frac{1-(R_i/B_i)^{n+1}}{1-R_i/B_i} > v_{th},$$

Where v_{th} is the pre-defined threshold of the total amount transmitted data, then

$$n > \log_{R_i/B_i} 1 - \frac{v_{th}}{D_i}(1-\frac{R_i}{B_i}),$$

so $$n = \left\lceil \log_{R_i/B_i} 1 - \frac{v_{th}}{D_i}(1-\frac{R_i}{B_i}) \right\rceil.$$

From the third pre-copy stopping condition, we have

$$D_i(\frac{R_i}{B_i})^n < p_{th},$$

where p_{th} is the pre-defined threshold of pages dirtied in the previous round, then

$$n > \log_{R_i/B_i}\frac{p_{th}}{D_i},$$

so $$n = \left\lceil \log_{R_i/B_i}\frac{p_{th}}{D_i} \right\rceil.$$

Thus, from the above three conditions, the number of pre-copy n is

$$n = \min\left(n_{th}, \left\lceil \log_{R_i/B_i} 1 - \frac{v_{th}}{D_i}(1-\frac{R_i}{B_i}) \right\rceil, \left\lceil \log_{R_i/B_i}\frac{p_{th}}{D_i} \right\rceil\right). \quad (4)$$

Eq.(3) and Eq.(4) dictates that t_i is closely related to available bandwidth B_i. But B_i is variable because once any VM is moved, the available bandwidth of whole data center networks will be updated.

Finally, VMs are moved according to the scheduling sequence and ones in front of sequence are migrated preferentially. It does not mean VMs are moved one by one, but are moved in parallel as long as there is enough available bandwidth. VM scheduled preferentially uses as much as available bandwidth for migration. If there is available bandwidth left, the next VM in the queue is moved then available bandwidth is updated. When a VM migration completes, the VMs being moved should be updated first in order to get more bandwidth for migration. Then VMs in the sequence is scheduled if there is available bandwidth left.

We let T be VM shuffle duration time, i.e., the time of the last VM completes migration. T is a function of Q so we let $T(Q)$ be the function. Now the optimization framework is defined as:

$$minimize\ T(Q). \tag{5}$$

This equation tries to minimize the VM shuffle duration time by scheduling VMs migration in the proper sequence Q.

Complexity analysis: The above optimization problem can be seen as a job scheduling problem. The data center network is the "machine" and each VM migration is a "job". It is equivalent to the Travelling Salesman Problem (TSP) which is a known NP-hard problem in combinatorial optimization studied in operations research and theoretical computer science [13].

Different with classic TSP problem where the distance between cities is known, the "distance" of VM shuffle optimization problem is dynamic because those VMs in the same shuffle interact with each other. Once a VM is moved, the traffic will be moved from the source physical server to the target physical server. The traffic of physical links would be changed. Then the left VMs migration may be affected. So this problem is more complicated than classic NP-hard TSP problem. Therefore, we resort to an intuitive heuristic approach which is described in next section.

C. SmartShuffle Method

In this subsection, we present *SmartShuffle*, a heuristic method to the online VM shuffle scheduling problem which tries to shorten VM movement duration by calculating a proper VM sequence.

We have shown that the problem is inherently NP-hard and no efficient exact solution can scale to the size of a typical VM shuffle. Therefore, we resort to an intuitive heuristic approach. We employ the simulated annealing algorithm to search for a solution on SmartShuffle. The simulated annealing algorithm is known efficient in searching in an immense solution space.

The detail procedure of simulated annealing for SmartShuffle is shown in Algorithm 1. We first get the VM set V needed to be moved. Then a random combination of V is generated as the initial solution. The VM shuffle duration time T is the "energy" and the function *shuffle_time*() calculates and returns the VM shuffle duration time of a given VM sequence. Specifically, VMs are moved according to the scheduling sequence and ones in front of the sequence are migrated preferentially. If there is available bandwidth left, the next VM in the queue is moved and available bandwidth is updated. When a VM migration completes, the VMs being moved should be updated first in order to get more bandwidth for migration. Finally, the time of the last VM completes its migration is returned.

Algorithm 1: Simulated Annealing for SmartShuffle

Require:
X', X, G, P, M, D, R
Algorithm:
$V \leftarrow \{V_i \mid i \in \{n \mid X'_{n,s} - X_{n,s} = 1\}\}$
$Q_{init} \leftarrow$ a random combination of V
$Q_{cur}, Q_{best} \leftarrow Q_{init}$
$T_{cur}, T_{best} \leftarrow shuffle\_time(Q_{init})$
$K \leftarrow K_{max}$
while $K > 0$ **do**
  $Q_{new} \leftarrow neighbor(Q_{cur})$
  $T_{new} \leftarrow shuffle\_time(Q_{new})$;
  if $calc\_pro(T_{cur}, T_{new}, K) > rand()$ **then**
    $Q_{cur} \leftarrow Q_{new}$
    $T_{cur} \leftarrow T_{new}$
  end if
  if $T_{new} < T_{best}$ **then**
    $Q_{best} \leftarrow Q_{new}$
    $T_{best} \leftarrow T_{new}$
  end if
  $K \leftarrow K - 1$
end while
return Q_{best}, T_{best}

The function *neighbor*() swaps a random VM pair in a given VM sequence. In each iteration, a neighboring state is generated. Then moving to a neighboring state with a certain acceptance probability which is got from the function *calc_pro*(), which is defined as

$$calc\_pro(T_{cur}, T_{new}, K) = \begin{cases} 1, & \text{if } T_{new} < T_{cur} \\ e^{(T_{cur} - T_{new})/K}, & \text{otherwise} \end{cases} \tag{5}$$

The function *calc_proc*() computes the acceptance probability based on the shuffle duration time of current and neighboring VM sequence as well as current temperature K. This function lets the probability of accepting a move to worse sequence decreases as the temperature is decreased in each iteration.

IV. EVALUATION

In this section, we evaluate the performance of SmartShuffle by conducting simulation experiments. In order to simulate a VM migration accurately, we write a discrete event simulator using C++. The whole migration process can be simulated as events, including migration start, the n-th pre-copy cycle, stop-and-copy and migration completion.

Moreover, the dynamical change of available bandwidth caused by VM migration start and completion is also simulated as events.

A. Simulation Setup

The physical topology is a three-level tree with an oversubscription ratio of 4:1. Edge to aggregation is 2:1 and aggregation layer to core is 2:1. From top to down, a core switch connects four aggregation switches with a 4Gbps link respectively. Then an aggregation switch connects four edge switches with a 2Gbps link respectively. An edge switch connects 4 physical servers with a 1Gbps link respectively. Every physical server hosts 4 VMs. So there are 64 physical servers and 256 VMs.

It is reasonable to take multi-root tree as single-root one. This is because that to take advantage of the path diversity in multi-rooted trees, data centers spread outgoing traffic to or from any host as evenly as possible among all the core switches. Moreover, protocols like Multipath TCP [12] offer the ability to use all possible paths in a single data transfer.

Current VM placement X' and target VM placement X are generated randomly. The number of VMs to be moved is set to 10, 20, 30, 40, 50 and 60 respectively. VM memory size is uniformly distributed on the interval from 2 to 8 GB. The dirty page rate is related to the application offered by the VM and is assumed to uniformly distributed on the interval from 0 to 100 Mbps. Pre-copy parameters are set according to the default values of XEN [14]. Pre-copy cycle threshold is set to 30. The transmitted data threshold is set to thrice memory size. The minimum dirty pages threshold is set to 200KB.

B. Evaluation Results

We evaluate the performance of SmartShuffle under two different traffic patterns: 1) *global traffic pattern*, in which each VM communicates with every other with the same probability; 2) *partitioned traffic patter*, in which VMs form isolated partitions, and only VMs within the same partition communicate with each other with the same probability. The communication rate between VMs is randomly generated following an exponential distribution. This is reasonable since the number of mice flows are much more than elephant flows in a typical data center.

We conduct simulation experiments to compare results of *SmartShuffle* to *Random*. In Random, VMs are scheduled to migrate based on a randomly generated order. The simulated annealing for SmartShuffle is iterated about 100000 cycles. We also perform Random 10000 times and evaluate the mean result.

1) Global Traffic Pattern

In this traffic pattern, every VM sends to other 0 to 4 VMs which are chosen randomly from all the VMs except itself. The rate follows an exponential distribution with a mean of 30 Mbps.

Fig.2 shows the shuffle duration time results for the simulation experiments. As shown in the figure, the SmartShuffle outperforms Random significantly on the shuffle duration. The shuffle duration of SmartShuffle on average is 38.3% smaller than Random, as shown in Fig.3.

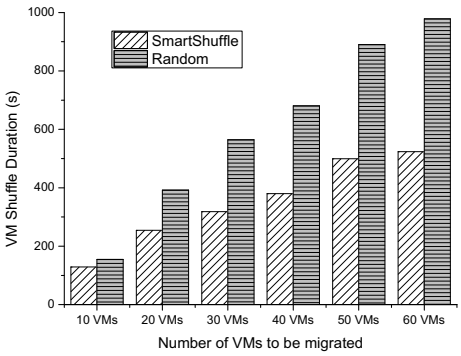

Figure 2. The evaluation results on shuffle duration

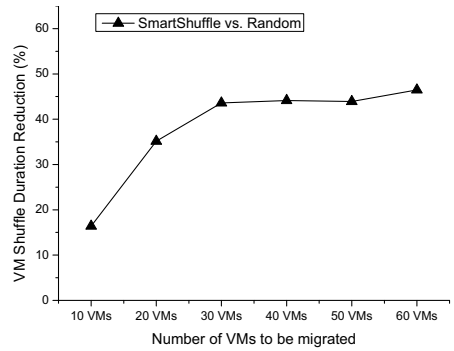

Figure 3. The normalized reduction of VM shuffle duration

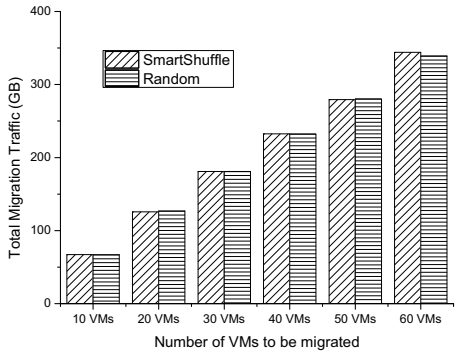

Figure 4. The total migration traffic in the VM shuffle

Additionally, the performance of SmartShuffle gets better when the number of VM to be moved is bigger.

Fig.4 depicts the total migration traffic in the shuffle. As shown in the figure, the total migration of both SmartShuffle and Random are almost identical. In other word, SmartShuffle gets better improvement while cause very little extra migration traffic.

2) Partitioned Traffic Pattern

In this traffic pattern, VMs are partitioned into 8 groups. Every VM also sends to other 0 to 4 VMs. But only VMs within the same partition communicate with each other and the target VMs are chosen randomly from the same isolated partition. The rate also follows an exponential distribution with a mean of 30 Mbps.

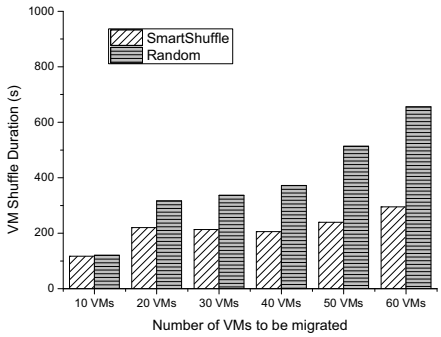

Figure 5. The evaluation results on shuffle duration

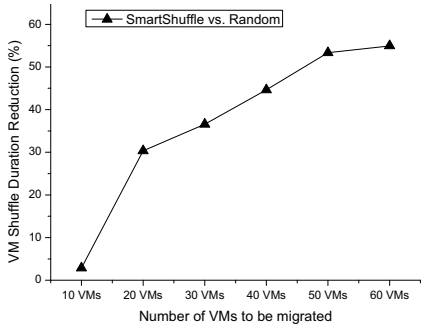

Figure 6. The normalized reduction of VM shuffle duration

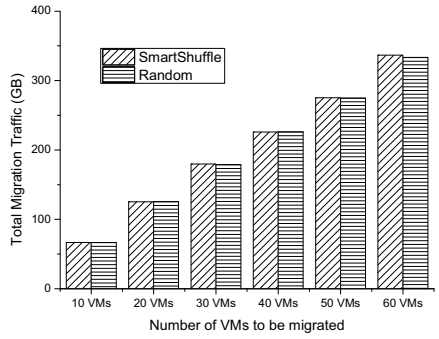

Figure 7. The total migration traffic in the VM shuffle

The simulation results in partitioned traffic pattern are similar to global traffic pattern. Fig.5 shows the shuffle duration time results for the simulation experiments. As shown in the figure, the SmartShuffle outperforms Random significantly on the shuffle duration. The shuffle duration of SmartShuffle on average is 37.1% smaller than Random, as shown in Fig.6. As the number of VMs increases in the shuffle, the benefit of SmartShuffle increases, even up to 55.0% when 60 VMs are to be moved.

Fig.7 depicts the total migration traffic in the shuffle. As shown in the figure, the total migration of both SmartShuffle and Random are almost identical. That is, SmartShuffle gets better improvement while cause very little extra migration traffic.

V. CONCLUSION

To explore the opportunity to minimize the impact of online VM shuffle to data centers, this paper designs SmartShuffle, an efficient online VM shuffle scheduling method. SmartShuffle tries to minimize the VM shuffle duration by coordinating the proper sequence of VM migration. VMs benefiting others maximally are migrated preferentially. We employ the simulated annealing algorithm to search for a solution to SmartShuffle. Our evaluation shows that SmartShuffle decreases the shuffle duration dramatically in both global traffic pattern and partitioned traffic pattern.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (No. 61002011); the 863 Program of China (No.s 2013AA013303, 2011AA01A102); the Open Fund of the State Key Laboratory of Software Development Environment(No. SKLSDE-2009KF-2-08), Beijing University of Aeronautics and Astronautics; the 973 Program of China (No. 2009CB320505).

REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, and R. Neugebauer, I. Pratt and A. Warfield. “Xen and the art of virtualization,” in Proceeding of SOSP, 2003.

[2] R. Nathuji and K. Schwan, “Virtualpower: coordinated power management in virtualized enterprise systems,” in Proceeding of SOSP, 2007.

[3] N. Bobroff, A. Kochut, and K. Beaty, “Dynamic Placement of Virtual Machines for Managing SLA Violations,” in Integrated Network Management 2007.

[4] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, “Proactive fault tolerance for hpc with xen virtualization,” in Proceeding of ICS, 2007.

[5] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, “Enacloud: An energy-saving application live placement approach for cloud computing environments,” in Proceeding of CLOUD, 2009.

[6] X. Meng, V. Pappas, and L. Zhang, “Improving the scalability of data center networks with traffic-aware virtual machine placement,” In Proceeding of INFOCOM, 2010.

[7] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, “Application-aware virtual machine migration in data centers,” In Proceeding of INFOCOM Mini, 2011.

[8] J.W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, “Joint VM Placement and Routing for Data Center Traffic Engineering,” In Proceeding of INFOCOM, 2012.

[9] X. Wen, K. Chen, Y. Chen, Y. Liu, Y. Xia, and C. Hu, “VirtualKnotter: Online Virtual Machine Shuffling for Congestion Resolving in Virtualized Datacenter,” In Proceeding of ICDCS, 2012.

[10] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, “Live migration of virtual machines,” In Proceeding of NSDI, 2005.

[11] V. Mann, A. Gupta, P. Dutta, and A. Vishnoi, “Remedy: Network-Aware Steady State VM Management for Data Centers,” In Proceeding of IFIP Networking, 2012.

[12] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, and M. Handley, “Data center networking with multipath TCP,” In Proceeding of HotNets, 2010.

[13] M. L. Pinedo, “Scheduling: Theory, Algorithms, and Systems,” Fourth Edition. Springer, 2012.

[14] XEN. http://www.xen.org.

[15] VMWare. http://www.vmware.com.