International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 8 | ISSN : 2456-3307
DOI : https://doi.org/10.32628/CSEIT18387

ABDM : Agent Based Live Migration of Virtual Machines in Cloud Computing For Multimedia Data

K. Syed Ibrahim*[1], Dr. A. R. Mohamed Shanavas[2]
*[1]Research Scholar, Baharathidasan University, Tiruchirappalli, India
[2]Associate Professor, Jamal Mohamed College, Tiruchirappalli, India

ABSTRACT

Migration time is one of the metric to measure the performance of the algorithm for live migration. In this paper we have introduced a new parameter for live migration of virtual machines (VM) called the 'Exit Time' which is defined as the time to eject the state of one or more VMs from the source node. Exit Time defines how rapidly the VM can be taken out from the source node and its resources are freed for reallocating other tasks. We present an Agent Based Live Migration which disconnects the source node from the destination node during migration to reduce the exit time if the destination is slow. The source distributes the memory of VMs to multiple intermediate nodes organized by a middleware. Simultaneously, the destination collects and merges the VMs' memory from the intermediate nodes. Thus exit from the source node is no longer resisted by the receiving speed of the destination. We support simultaneous live exit of multiple VMs and our ABDM implementation in the CloudSim platform reduces the exit time by a considerable amount against the traditional pre-copy and post-copy migration at the same time keeping the total migration time when the destination node is sluggish than the source.

Keywords : Exit Time, Live Migration, Virtual Machine, QoS

I. INTRODUCTION

Virtual Machines (VMs) live migration [3], [4], [5], [6] is used in datacenters for providing best Quality of Service (QoS) with respect to system maintenance, consolidation, load balancing and power savings. The performance of live migration is generally measured using the conventional parameters like downtime, network traffic, total migration time overhead, and performance degradation of applications.

In this paper, we have used a new parameter called 'exit time', which can be defined as the time taken to absolutely exit the state of one or more VMs being migrated from the source node. In most of the situations rapid removal of VMs from the source node is very important in order to save the resources in the host like energy, memory and processing cycles. We have used an agent based divide and merge VM live migration technique for quickly ejecting the entire VMs from the source node and merging at the destination node. For instances, cloud administrators may wish to save energy by switch off additional resources in a server [7], [8], [9], [10], [11], [12], quickly remove hotspots by scaling down the physical resources for performance assurance [13], quickly remove the lower priority VMs to accommodate other higher priority ones, perform emergency maintenance [14] or handle imminent failures.

A. Defining Live Migration

Live migration denotes the process of transferring a running virtual machine or application from one

physical machine to another physical machine without disconnecting the client or application. Memory, storage, and network connectivity of the virtual machine are transferred from the original computer called as the source node to another computer called as destination node.

There are two major types of Live Migration namely pre-copy migration and post-copy migration.

In pre-copy memory migration, the migration manager typically copies all the memory pages from source to destination while the VM is still running on the source. If some memory pages change or making fault during this process, they will be re-copied until the rate of re-copied pages is greater than the page dirtying rate. [1].

Post-copy VM migration is originated by suspending the VM at the source node. With the VM suspended, a least subset of the execution state of the VM with CPU state, registers and memory state is transferred to the destination node. The VM is then restarted at the destination. Simultaneously, the source dynamically pushes the remaining memory pages of the VM to the destination. This process is called as pre-paging. At the destination, if the VM tries to access a page that yet to be received, it generates a page-fault. These faults, called as network faults, are caught at the destination and redirected to the source. On seeing this request the source node resends the faulted page. Performances of applications running inside the VM are degraded by too many network faults. Hence pre-paging can dynamically adapt the page transmission order to network faults by actively pushing pages in the vicinity of the last fault [2].

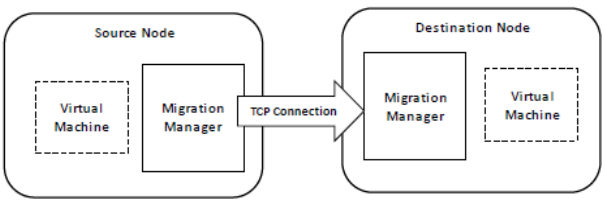

Figure 1. Conventional VM migration

In the conventional live VM migration methods [3], [4], [5], the exit time is equal to the total time for migration, which can be defined as the period from the time when the source node starts transferring the VM to the time point when VM restarts the execution after the destination node completely received the entire state of the VM. The migration time will be very slow if the source node straightly transfers the VM's state to the destination host over a TCP network. Therefore VM transmission will be as slow as the slower node and the source is connected to the destination for the full duration of the VM migration.

There are many reasons for a destination node to receive a VM slower than a source can transmit it:

i. The resource in the destination node may be busy with other jobs which might not be predicted by the VM migration algorithm.
ii. The network may be slow due to congestion.
iii. The destination node may be a consolidation server that is simultaneously receiving VMs from several sources, which cause the speed reduction for individual VMs.
iv. Finally, importance may be given to transmit the idle or less important VMs from the source to vacate resources for more significant VMs.

If the source and destination nodes are connected for a long time for migration to be completed, it will create unnecessary problems. For example, system optimization will not be effectively taking place, if it relies on the transmission time. Moreover, long duration migration affects the performance of the other VMs that exist in the node.

In this paper we propose a novel idea which makes the VM migration very fast if the destination host is slower than the source node. This process contains

three steps. As the first step the source and the destination nodes are disconnected. In the second step an agent distributes the state of the VM from the source node to the intermediate nodes as quick as possible with the maximum speed of the source node. Finally in the last step, the agent collects the VMs from the intermediate nodes, merges and sends to the destination when the local resources become available.

In our proposed technology we present an Agent Based Divide and Merge VM live migration which decreases the ejection time by migrating VMs using the intermediate nodes. A middleware component which offers a brokerage service matches the demand and availability of the memory devices for VM live migration. The intermediate nodes are used as temporary buffer for the migrated VMs from the source. Once the source node started the migration, it sends a request to the agent for memory devices. The agent divides the VM to multiple nodes including the destination node. The intermediate nodes are the other nodes in the cloud like network caches or some storage devices. The memory and bandwidth required for the VMs transmission is acquired by accumulating all the buffers in the intermediate nodes. Simultaneously the agent collects and merges the memory from the intermediate nodes and sends back to the destination node. Hence by temporally disconnecting the source and destination the source can eject the VMs at its maximum speed even though the destination is not as fast as the source. Our main contributions are given below.

We have used agent based memory virtualization for live migration in this paper. A Migration Agent Middleware (MAM) layer collects the information about the free memory and CPU resources among all the intermediate nodes and keeps an index of the available resources. In datacenters, more amounts of free memory and CPU resources are available since all the datacenters have surplus infrastructures [15] and those machines are used as intermediate nodes by the MAM.

II. RELATED WORKS

In this section we discuss some of the existing research in minimizing the time for live migration. In [5] and [6] the authors proposed the post-copy migration which decreases the total migration time and network overhead in contrast with the pre-copy migration given in [3] and [4]. In [17], [21] and [22] the authors presented methods to optimize the live migration of multiple VMs using various techniques like memory compression, memory deduplication, and maintaining the order of VMs migration order. A number of optimizations for storage and memory are integrated in XvMotion [16]. Optimizations such as ballooning [5] dropping the guest cache [19] deduplication [20] and compression [39] minimize time form migration and the network traffic and migration time. Post-copy VMs migration is presented in SnowFlock [23] which implements cloning of VMs in multiple hosts to execute High Performance Computing (HPC). The partial migration technique is proposed by Jettision [7] in which only the working set of an idle VM is migrated to save the energy by combining idle VMs from number of PCs at a central server. In [18] a traffic-sensitive migration is proposed which monitors the traffic at the nodes to choose any one from pre-copy or post-copy of VMs. The researches have demonstrated In the post-copy [5], [6], migration method, the VM is first adjourned in the source node and the state of the CPU is transferred to the destination where the VMs are received ad re-assembled immediately.

III. Proposed ABDM Architecture

In the conventional live VM migration shown in figure 1, the source node directly sends the VM's

state to the destination node. The source and destination nodes have Migration Managers for each VM being migrated.

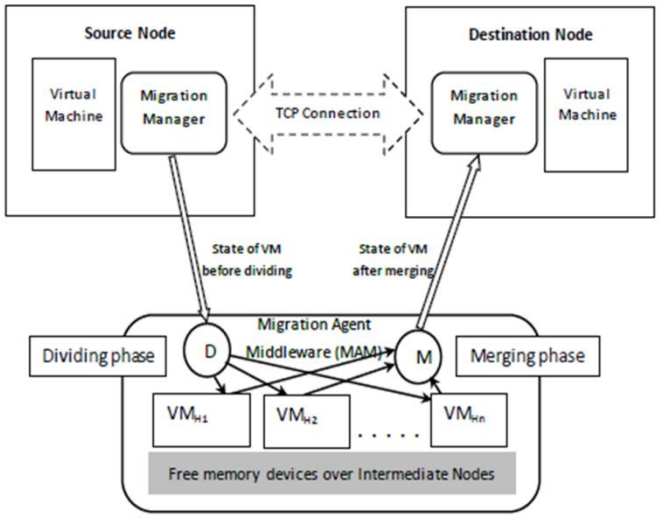

Figure 2. Architecture of ABDM VM live migration

A TCP connection between the Migration Managers transmits the data about VM's memory and state of the CPU and control information such as synchronization, handshakes etc. This connection is put off only after the destination receives the entire VM. The ABDM approach uses a Migration Agent Middleware (MAM). The architecture of ABDM approach is shown in figure 2. When the source node wants to migrate a VM, it sends a migration request to the MAM. The MAM receives the request and allocates the memory and CPU resources as per the requirement. Then the source node sends the state of the VM to MAM. The divider 'D' receives the entire VM and distributes it among the allocated resources. At the same time the merger 'M' collects the parts of the VM from the intermediate resources and merges them and sends to the destination node. The agent MAM is acting as a broker as well as speed matcher between the source and the destination. The functioning of ABDM is given below:

i. A customer node connected with the cloud requested a multimedia streaming data.
ii. The source node which is already part of the cloud sends the requested data to the customer node.
iii. The customer started moving so that the source node needs to migrate the VM contents to other node to which the customer node connects while moving.
iv. The source node request the agent MAM for VM migration since the network speed between the source node and the destination nodes is not matching.
v. The divider in the agent allocates the memory and CPU resources and divides the VMs based on the availability of the resources.
vi. The merger in the agent merges the parts of the VM and sends back to the destination with the speed that matches with the destination.

3.1 Migration Agent Middleware (MAM)

Migration Agent Middleware (MAM) is an intermediate layer, through which the source node transfers the VM's memory to the destination. The MAM layer simplifies by modularizing the architecture. The MAM layer collects the available free memory of all intermediate nodes and keeps an index. The index contains the address of each device with the amount available memory. When the migration manager at the source node sends a request to the MAM for VM migration the MAM allocates the required amount of memory by dividing the VM into pieces. Then the source node writes the VM's memory to the allocated devices. The migration manager at the destination concurrently reads the VM's memory from the devices. No physical memory is reserved in advance at the intermediate nodes; instead, the MAM layer at the source uses the memory availability information at the intermediate nodes to dynamically decide where to distribute the memory pages.

3.2. Dividing Phase

The objective of the dividing phase is to quickly eject the VM's memory and execution state from the source node. Initially a TCP connection is established between the Migration Managers at the source and the destination nodes. Next, the migration manager sends a migration request to MAM to start the migration process. The divider 'D' is responsible for dividing the VM's memory and distributes them among the intermediate devices. The CPU state of the VM is conveyed to the destination node in which the VM is going to resume immediately as in the post-copy migration. Since the VM's memory still exist in the source node, the VM in the destination node generates page-faults its memory. The destination node's migration manager transmits all page-fault requests as a control message during the dividing phase to the source node's migration manager over the TCP network, which then redirects the same to the MAM. The MAM layer responds for this control message and resends the error page to the destination. This process is as similar as how the demand-paging is handled in the conventional post-copy migration. Each page written to the MAM layer is sent to one of the intermediate nodes based on its offset in the VM's memory. The fault-tolerance of migration is improved by replicating each page to multiple intermediate nodes. For each page sent to the MAM, the divider 'D' at the source node directly sends the corresponding control information to the destination's Migration Manager over the TCP network. The control message includes the address of each page that was divided and its status such as optimization by compression or deduplication, applied to the page. This message is used later by the Merger 'M' at the destination node to collect the VM's pages from the MAM layer. Once the entire VM has been ejected, the resources used at the source node are freed reused for other VMs.

3.3 Merging Phase

The merging phase recovers the VM's memory pages from the intermediate node. This phase runs simultaneously with the divider phase at the source. The destination node starts executing the VM immediately as it receives the VM's execution state from the source node. The merger phase contains two modules: (a) Active collection or pre-paging the VM's pages from the intermediaries and (b) Recollecting the faulted pate or demand-paging from the source node.

In pre-paging, the merger 'M' at the destination node's migration manager collects the VM's state from the intermediate nodes merge them as the full VM and sends it to the migration manager of the destination. Moreover, the migration manager hears the control TCP connection on which the 'D' at the source node sends information about the divided pages. The destination's migration manager uses the control message received from the source to copy the received pages from the MAM into the VM's memory.

The demand-paging module is working as follows. The merger phase is responsible for recollecting the VM's page merging and giving as full VM to the destination. This has to be done quickly to minimize the migration time and to maintain QoS. Hence, if the VM faults on any page during this merging phase, the Migration Manager at the destination node directly sends a request to the source node for the faulted pages. These requests are called as demand-paging which will be again handled by the MAM layer at 'D' which resends the error page over the TCP connection to the source. In order to reduce the latency in handling page faults, higher priority is given by the source node's migration manager for the faulted pages.

II. IMPLEMENTATION

We have implemented ABDM using CloudSim virtualization platform. We describe the implementation details of the MAM and the Migration Managers in the following sections.

4.1 Migration Agent Middleware (MAM) Layer

We have created the MAM as a distributed peer-to-peer communicating system among computing nodes in an Ethernet network which shares the free memory in the devices. For every VM being migrated, the MAM client module at the source and destination nodes shows a dummy device to the migration managers. The dummy device is a logical representation of the aggregated memory; no physical memory is reserved in advance at the intermediate nodes. The MAM servers periodically broadcast resource announces their identity and memory availability to MAM clients. The source MAM client uses this information to divide and sent the VM pages. The dummy device simply allows the migration managers to transfer pages without knowing the identity of each intermediate device and the location of each page.

Every MAM server acts as an index server as well as a content server. The index server keeps the record of the location of a page whereas the content server saves the content of the page. The contents and their location is maintained by the index server by mapping of page offsets to content hash values for each memory page within its offset range. This mapping is used to locate the content server for a given page offset.

4.2. Migration Manager

The migration manager is created one for each VM that interfaces between the VM and the hypervisor for processing the VM migration. The migration managers both at the source node and the destination node open the dummy device sent by the MAM layer to proceed with the divide and merge phases. We adapt a general post-copy implementation from the Yabusame project [4] to implement the migration managers. In the migration process, the migration manager at the source node uses a TCP connection with the destination to transfer the control information about each page, which includes the physical address of the page in the VM's memory and its offset address in the MAM layer.

III. Results and Performance Evaluation

In this section, we evaluate the performance of ABDM migration by comparing with the standard pre-copy and post-copy migration. We evaluate the exit times when migrating single and multiple VMs, degrading the performance on both migrating and co-located VMs and the impact of using multiple intermediates. For all the experiments, each data-point shows an average performance over six iterations. We run all the experiments using dual quad core servers with 2.3GHz CPUs, 8GB DRAM, and 1Gbps Ethernet cards. Ubuntu 14.04.2 is used for running all VMs with Linux kernel 3.2, have 2 virtual CPUs (vCPUs) and for both the hard disk and network adapter we have used Virtio - An I/O virtualization framework for Linux.

Figure 3 shows that the total migration time of ABDM is only slightly higher than pre-copy and post-copy (by up to 10%). This overhead is reasonable because of two reasons. First, the VM pages are transmitted in two hops to the destination, but in pre-copy and post-copy they are transmitted in single hop. Secondly, our implementation of the ABDM presently provides around 750 Mbps to 800Mbps throughput in a 1Gbps Ethernet when the intermediate nodes concurrently handle sending and receiving whereas direct TCP connection between

source node and the destination node can achieve up to 900Mbps throughput. Figure 4 shows the ejection times and the total migration times for the migration of a 5GB busy VM.

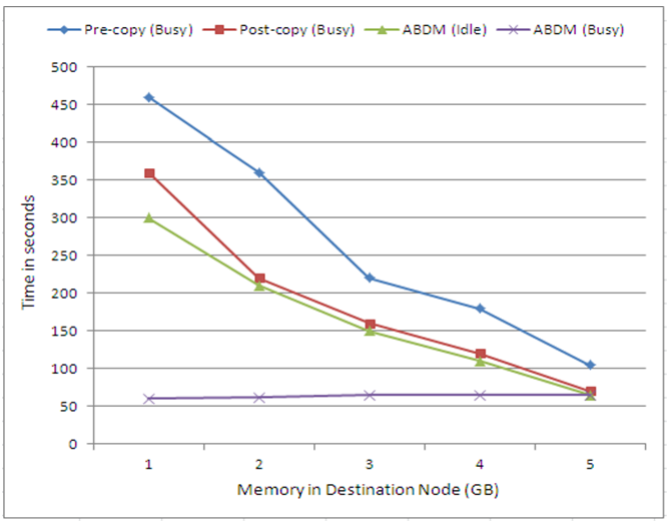

Figure 3. Ejection time and Total Migration Time (TMT) for migrating 5GB **idle** VM to a busy destination

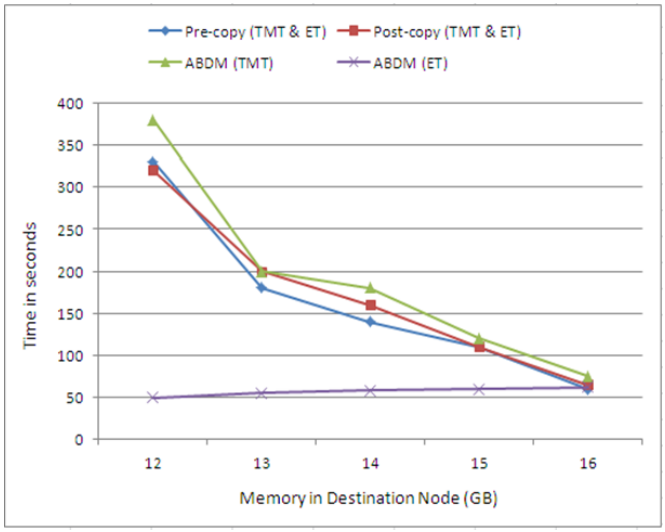

Figure 4. Ejection time and Total Migration Time (TMT) for migrating 5GB **busy** VM to a busy destination

Figure 5 shows the ejection time for the migration of multiple VMs. For migrating up to 3 VMs, the destination node does not feel any memory burden, therefore all techniques, excluding pre-copy migration of a busy VM are good. For the migrating 4 and 5 VMs, ABDM delivers a lower ejection time than pre-copy and post-copy because the link between the source and MAM layer is free and non-congested even though the destination is under memory burden.

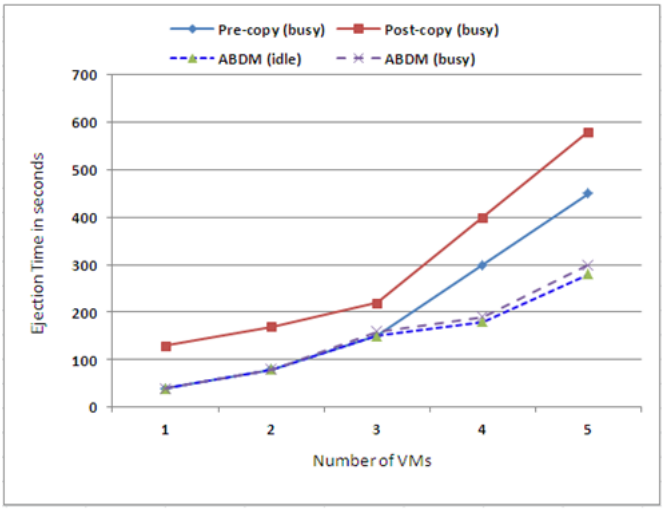

Figure 5. Ejection Time for increasing number of VMs

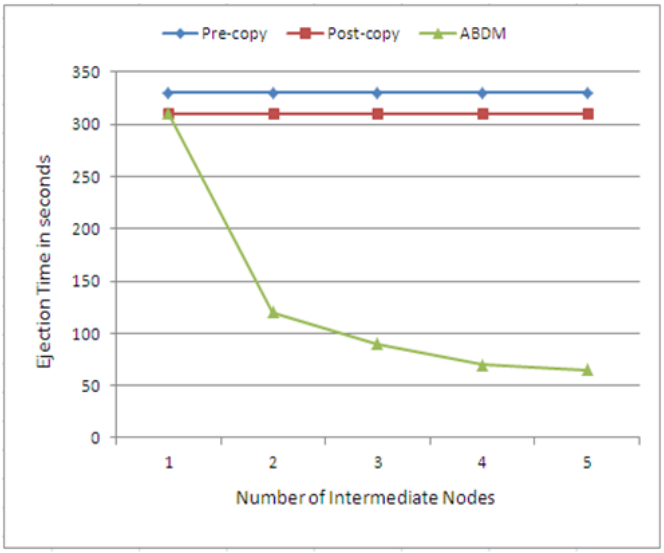

Figure 6. Reduction in ejection time when using multiple intermediates

Figure 6 illustrates that the migration of one idle 5GB VM to a memory reserved destination while increasing the intermediate noes. The destination has 12GB of memory and hosts two 5GB VMs. During migration, the destination swaps out pages to accommodate the incoming VM.

IV. CONCLUSION

Maintaining QoS is an important agenda in cloud computing platform. There are several parameters we use to measure the performance of the cloud platform. It is still more important when accessing the multimedia content using mobile devices. Virtual Machine migration is a key technique for maintaining QoS in mobile cloud environment. Traditionally the performance of live migration of virtual machines is measured using several parameters. In this paper we have used the exit time as one of the metric to measure the migration of VM from the source node. We have used an agent based migration technique to smoothly distribute and merge the states of the VMs and the effectiveness of our approach is tested by conducting experiments using CloudSim simulator and the simulation results illustrates that the migration time is reduced considerably in the rate of 4 while maintaining the quality of the content of the media stream.

V. REFERENCES

[1] Moghaddam, F.F. and Cheriet, M., 2010, April. Decreasing live virtual machine migration down-time using a memory page selection based on memory change PDF. In *Networking, Sensing and Control (ICNSC), 2010 International Conference on* (pp. 355-359). IEEE.

[2] Hines, M.R., Deshpande, U. and Gopalan, K., 2009. Post-copy live migration of virtual machines. *ACM SIGOPS operating systems review*, *43*(3), pp.14-26.

[3] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in Network System Design and Implementation, 2005.

[4] M. Nelson, B. H. Lim, and G. Hutchins, "Fast Transparent Migration for Virtual Machines," in USENIX Annual Technical Conference, 2005.

[5] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy Live Migration of Virtual Machines," SIGOPS Operating System Review, vol. 43, no. 3, pp. 14–26, 2009.

[6] T. Hirofuchi and I. Yamahata, "Yabusame: Postcopy Live Migration for Qemu/KVM," in KVM Forum, 2011.

[7] N. Bila, E. J. Wright, E. D. Lara, K. Joshi, H. A. Lagar-Cavilla, E. Park, A. Goel, M. Hiltunen, and M. Satyanarayanan, Energy

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

[9] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems," in Middleware, 2008.

[10] T. Das, P. Padala, V. Padmanabhan, R. Ramjee, and K. G. Shin, "LiteGreen: Saving Energy in Networked Desktops Using Virtualization," in USENIX Annual Technical Conference, 2010.

[11] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu, "Delivering Energy Proportionality with Non Energy- Proportional Systems - Optimizing the Ensemble." HotPower, 2008.

[12] A. Jaikar, D. Huang, G.-R. Kim, and S.-Y. Noh, "Power efficient virtual machine migration in a scientific federated cloud," Cluster Computing, vol. 18, no. 2, pp. 609–618, 2015.

[13] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi, "Reactive Consolidation of Virtual Machines Enabled by Postcopy Live Migration," in Virtualization Technologies in Distributed Computing, June 2011.

[14] S. Setty and G. Tarasuk-Levin, "vMotion in VMware vSphere 5.0: Architecture, Performance and Best practices", in VMworld 2011, Las Vegas, Nevada, USA, 2011, p. 24

[15] J. Hwang, A. Uppal, T. Wood, and H. H. Huang, "Mortar: Filling the Gaps in Data Center Memory," in Proc. of Virtual Execution Environments (VEE), 2014.

[16] A. J. Mashtizadeh, M. Cai, G. Tarasuk-Levin, R. Koller, T. Garfinkel, and S. Setty, "XvMotion: Unified Virtual Machine Migration over Long Distance," in Proceedings of the USENIX Annual Technical Conference, 2014.

[17] U. Deshpande, B. Schlinker, E. Adler, and K. Gopalan, "Gang Migration of Virtual Machines using Cluster-wide Deduplication," in International Symposium on Cluster, Cloud and Grid Computing, May 2013.

[18] U. Deshpande and K. Keahey, "Traffic-Sensitive Live Migration of Virtual Machines," in International Symposium on Cluster Computing and the Grid Environments, 2015.

[19] C. Jo, E. Gustafsson, J. Son, and B. Egger, "Efficient Live Migration of Virtual Machines Using Shared Storage," in Virtual Execution Environments, March 2013.

[20] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live Virtual Machine Migration with Adaptive Memory Compression," in Cluster Computing and Workshops, August 2009.

[21] R. K. Hui Lu, Cong Xu and D. Xu, "vHaul: Towards Optimal Scheduling of Live Multi-VM Migration for Multi-tier Applications," in 8th IEEE International Conference on Cloud Computing (Cloud 2015), New York, NY, June 2015.

[22] H. Liu and B. He, "VMbuddies: Coordinating Live Migration of Multi-Tier Applications in Cloud Environments," Parallel and Distributed Systems, IEEE Transactions on, vol. 26, no. 4, pp. 1192– 1205, 2015.

[23] H. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan, "SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing," in EuroSys, 2009.