Distributed Shared Memory based Live VM Migration

Tariq Daradkeh
Department of Electrical and Computer Engineering
Concordia University
Montreal, Canada
t_darad@encs.concorda.ca

Anjali Agarwal
Department of Electrical and Computer Engineering
Concordia University
Montreal, Canada
aagarwal@encs.concorda.ca

***Abstract*—Live virtual machine migration is an essential tool for dynamic resource management in current data centers. Many techniques have been developed to achieve this goal with minimum service interruption. In this paper, we propose a pre-copy live VM migration using Distributed Shared Memory (DSM) computing model. The setup is built using two identical computation nodes to construct the environment services architecture namely the virtualization infrastructure, the shared storage server, and the DSM and High Performance Computing (HPC) cluster. The custom DSM framework is based on a low latency memory update Grappa. HPC cluster with OPENMPI and MPI libraries support parallelization and auto-parallelization work load by using CPUs computation nodes. The DSM allows the cluster CPUs to access the same memory space pages resulting in a lower memory data updates based on locality attributes updates, which reduces the amount of data transferred through the network. This model achieves a good enhancement of the live VM migration metrics. Downtime is reduced by 50% in the idle workload of Windows VM and 66.6% in case of Ubuntu Linux idle workload. In general, this model not only reduces the downtime and the total amount of data sent, but also does not degrade other metrics like the total migration time and the application performance.**

***Keywords*-Virtual Machine; Total Migration Time; Down Time; Distributed Shared Memory; Physical Machine; HPC;**

I. Introduction and Related Work

Live Virtual Machine (VM) migration process is a major service provided by modern cloud service providers. It can be defined as transferring the Virtual Machine (VM) state while it continues to run and serve clients from one physical machine to another physical machine without disrupting the clients accessing that VM.

The VM state is dynamically changed during the live migration process. As a result of serving live clients, these changes affect the memory state, the virtual VM CPU (vCPU) registers and state, and network state. The way to transfer these three work spaces safely while continuing to run the VM is to maintain sending the changes in a coherent way until a stop condition occurs.

A lot of work has been done since 2005 when Clark [1] proposed the pre-copy method of live migration, which is based on transferring the memory state iteratively. After that, many optimization methods [2] were proposed to enhance the way how memory image and the CPU state are moved.

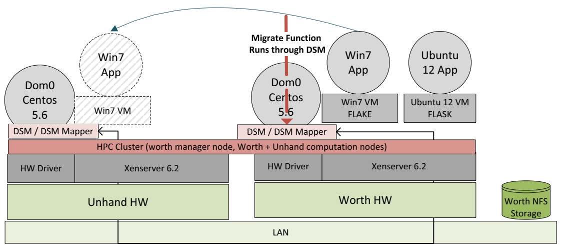


Figure 1. System Design Model Overview

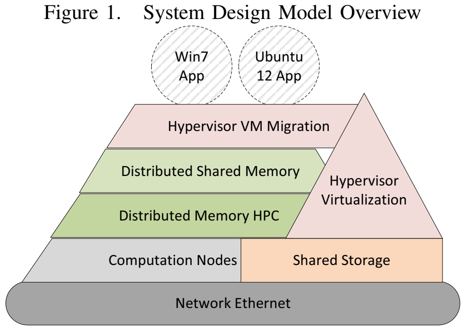


Figure 2. System Logical Modules

Another approach which is based on CPU logging and replay [3] requires synchronization and not much work has been done using this approach. All these approaches consider the following four performance metrics:

1) Total Migration Time: Time since starting the migration process until finished. Objective is to reduce the total migration time.

2) Down Time: Time when the vCPU execution is suspended in the source Physical Machine (PM) until vCPU is resumed in the destination PM. Objective is to reduce the down time.

3) Data Transferred Size: Size of data moved through the network during the total live migration time. Objective is to reduce the data transferred size.

4) VMs Application Performance: The migrated VM application response. Objective is to keep the application performance.

Following sections discuss the proposed model and how it works.

2159-6190/16 $31.00 © 2016 IEEE
DOI 10.1109/CLOUD.2016.114

Table I
SETUP HARDWARE SPECIFICATION

Dell Precision T1700 Specifications	
CPU	Intel(R) Xeon(R) CPU E3-1271 v3 @ 3.60GHz (4 Cores)
Memory (RAM)	32G Byte Kingston 1600 MHz (0.6 ns)
Storage (Hard Disk)	1T Byte ATA Disk
Network	Intel Ethernet Connection I217-LM 10/100/1000Mbps
OS	Citrix Xenserver (Linux Centos 5.6 Custom)
Xen Kernel	2.6.32.43-0.4.1.xs1.8.0.835.170778xen

Table II
WORKLOAD BENCHMARK

Workload	Benchmark
Idle OS	Run guest OS with idle state (for both OS types)
CPU intensive task	For Linux Compiling XEN source code For Windows Installing Cygwin
Memory Intensive task	Playing video (for both OS types)
Network Intensive Task	Web server (for both OS types)

II. SYSTEM DESIGN MODULES

In this work we deployed the pre-copy migration of Citrix Xen hypervisor using DSM running on top of High Performance Computing (HPC). The DSM used is modified to fit our requirements. This system architecture runs four services (NFS, hypervisor XenMotion, HPC and the DSM), which work in a cooperative way to handle the live VM migration of XenServer hypervisor in an optimized way, the enhancement of the XenServer motion using the NFS shared storage, and the DSM HPC cluster to speed up the XenServer VM motion. The building block architecture of the proposed live VM migration is composed of three services layers to facilitate the migration through running the hypervisor migration process as a job in the DSM HPC cluster computation. Figure 1 shows the conceptual architecture of the module's block components and communication flows.

A. System Setup Physical Components

In this work, two identical Dell workstations with high speed processor 4 Cores of Intel Xeon 3.6 GHz speed, connected by Linksys Ethernet switch with port speed 100Mbps are deployed. Table I shows summary of hardware specifications.

B. Logical Overview

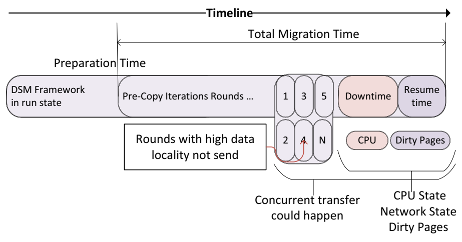
Figure 2 depicts the logical modules as layers architecture to build the VM migration. The first part is the shared storage NFS Protocol, which is a transparent protocol that allows the shared storage server update to be synchronized with all virtual members. Second part is the Virtualization Infrastructure using Citrix XenServer version 6.2 hypervisor, which is used to create the virtual machines, and managed by Citrix Xen-Center management console, which is a software for managing VMs and virtual machines templates. The shared storage and virtualization modules provide basic setup for normal live migration.

Third part is the HPC Cluster Distributed Memory with Message Passing. In general HPC clustering the distributed memory concept is mandatory to support parallel programming, which requires the use of explicit message passing (MP), to allow processors to communicate. The standard communication between processors is Message Passing Interface (MPI) that is supported by all high performance computing vendors. The role of HPC cluster is to provide parallelization and auto-parallelization services to the codes segments, based on the code attributes using OPENMPI library.

Fourth part is the Distributed Shared Memory framework. The DSM model works with the HPC cluster to provide the shared memory accessibility for all cluster nodes and processors. DSM changes the inter-processor communication based on shared memory communication paradigm, which allows all processors in the HPC cluster to access the same memory space. In such a model the process migration only requires moving the process state from CPU scheduling ready queue on one computation node processor to the ready queue on the other node processor, since process control block PCB, code and stack are all in the same memory address space, and shared virtual memory is a single address space shared by number of processors. Any processor can access any memory location in the shared address space directly. The role of DSM module is to provide memory state update in a consistent and coherent way, where the pre-copy method starts by moving the VM memory pages iteratively. The DSM helps moving the memory pages by avoiding sending the dirty pages.

III. LIVE VM MIGRATION USING DSM

The source machine (Worth) runs two VMs (Flake and Flask); the guest OS (Win7) of Flake will be migrated. The process will start by triggering the migration function of the XenServer XAPI, which then starts the migration process to do the pre-copy live VM migration. DSM will provide a shared memory access to all cluster computation nodes that enable direct access to all memory pages for both source and destination virtualization servers. The live VM migration process is localized by the source physical machine, the OPENMPI scan the serialized code to find any parallel segment to start the speed up. The DSM is used as a global memory space between the HPC cluster computation nodes that can help the pulling process migration of the VM. Meanwhile the source physical machine continues to push the virtual CPU state of the migrated VM and the Network state as a priority task, then the remaining dirty pages is moved as a lower priority task. The source and destination virtualization servers will be loaded with the migration task to start and terminate the live VM migration based on the destination VM memory image consistency threshold,



Timeline
Total Migration Time
Preparation Time
DSM Framework in run state
Pre-Copy Iterations Rounds ...
1
3
5
Downtime
Resume time
2
4
N
CPU
Dirty Pages
Rounds with high data locality not send
Concurrent transfer could happen
CPU State
Network State
Dirty Pages

Figure 3. DSM Pre-copy Time Progress

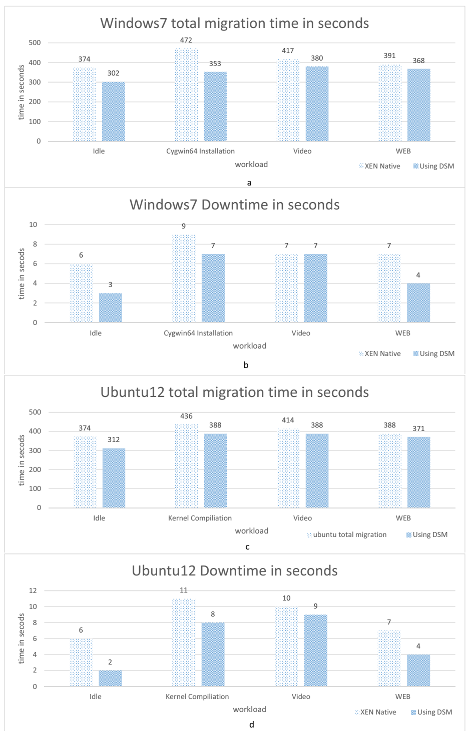


Figure 4. Total Migration Time and Downtime

which is bounded by three parameters: maximum number of iterations, less memory pages changes (less dirty pages generation), and consistent memory image at destination.

The DSM enhances the pre-copy Live VM migration process by customizing some attributes of the Grappa [4] DSM. The first modification is to update the message size. DSM migration process works on moving memory pages with an average size of 4KB instead of 32 bytes as used in Grappa. The update message size is made same as memory pages size. This will reduce the number of update messages. Second modification is related to the memory pages selection based on locality management. The data accessed with low locality is usually modified on its home processor rather than sending a copy of the code image to the destination processor, where as the data accessed with high locality is send to the requested processor. In our approach the best is not to move the high data locality memory pages, but its more efficient to first send the low locality memory pages with lower memory dirtying pages rate to the destination processor. This will make the update frequency lower. The remaining higher memory dirty pages are later transferred during the suspend and resume state. The automatic parallelization provides parallel task of sending and receiving VM memory pages. Figure 3 describes the pre-copy with the customized DSM approach as time-space diagram. This way our approach reduces the number of dirty pages send from home processor to the destination processor.

IV. Performance Measurement

The live VM migration is done for both guest OSs, the Windows7 VM Flake and the Ubuntu12 Linux Flask, between the two host servers Unhand and Worth. The experiment is run for each different case for six times to make sure the measured values do not contain any special errors, and then an average is calculated.

A. VM Workload Benchmarks

The guest VM OS is loaded with four different workloads to test the live VM migration under different case scenarios, which load migrated VM by applications that have variant execution behaviors as benchmark, as given in Table II.

1) OS Idle Workload: In idle OS the memory dirty pages generated have lowest time of generation during the VM migration without running any heavy workload. The OS load generation of memory changes in idle case is used to evaluate the DSM live VM migration model.

2) CPU Intensive Workload: Compiling source code (Linux VM) and Cygwin installation (Windows VM) are a heavy CPU load and in these cases of workload the CPU context of compiling time is higher than other CPUs jobs.

3) Memory Intensive Workload: The memory intensive task workload is achieved by running video during the live migration for both types of OSs, Windows and Linux VM, in addition to the disk intensive read load for the video from the hard disk.

4) Network Intensive Workload: For network intensive task network state of each client connection to the web server must be maintained until the client terminates the connection. The data transferred in web workload mainly has a big network state to save the web server clients connection which increases memory foot print with higher locality attribute.

As shown in Figure 4a) the total migration time is reduced in all workloads but the best is with Cygwin installation and idle workloads, which is about 25% and 20% of enhancement respectively, whereas with web and video work load the enhancement is 6% and 8% respectively. Figure 4c) depicts the Linux VM total migration time. With DSM it is less with all workloads and the best ratio is for idle VM workload with 16.6% of enhancement reducing the total

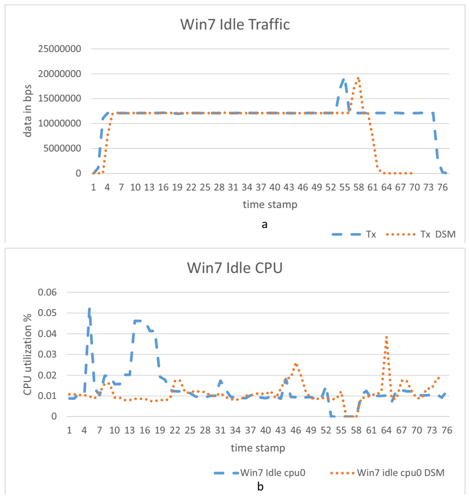


Figure 5. Windows Idle Workload

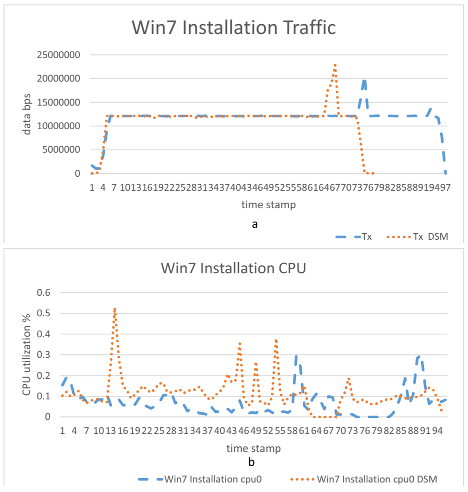


Figure 7. Windows CPU Intensive Workload

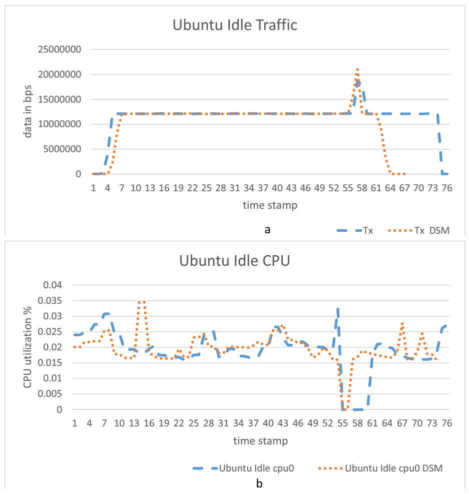


Figure 6. Linux Ubuntu Idle Workload

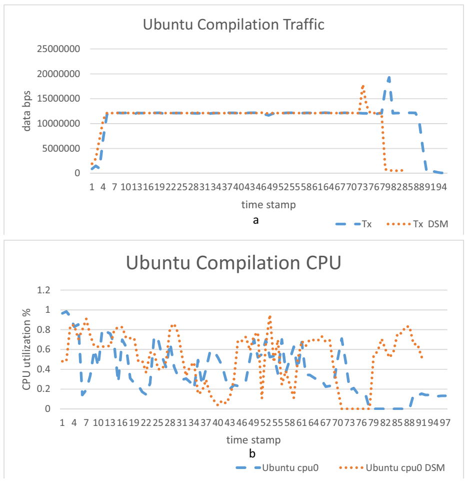


Figure 8. Linux Ubuntu CPU Intensive Workload

migration time. Since Linux memory footprint is less than Windows, the update of high locality dirty pages is lower in Linux than in Windows. This makes DSM work better with Windows VM.

In Figures 4b) and 4d), the downtime measured for Windows and Linux with DSM has a good reduction in all VM workload cases. It achieves about 42.8% of downtime reduction in the Web work load for both windows VM and Linux VM. In windows video playing there is no enhancement because GPU is working with the memory load and video images marshalling. But with Linux it achieves 10% of enhancement knowing that with Linux the video does not stream in a smooth way, because of the basic graphic driver work behavior. DSM reduces the down time by reducing the number of faulty pages after the suspend and resume state. The target processors can see all the memory pages directly.

Figures 5 to 12 show the network and VM CPU performance during the live VM migration. The time slot is normalized as fixed time slot to show the period of the starting time of live VM migration, downtime and total migration time. We can see clearly the relation between the VM CPUs performance and traffic activity. In all cases the network traffic is bounded to a fixed bandwidth of 12Mbps to protect the network, and during downtime period the traffic rate is increased to reduce the downtime.

V. Conclusion

A novel method is proposed to run the live VM migration in the cloud data centers using a distributed shared memory high performance computing cluster. In this way, memory of each computation nodes is shared and accessible to all nodes CPUs with high abstraction for nodes local memory.

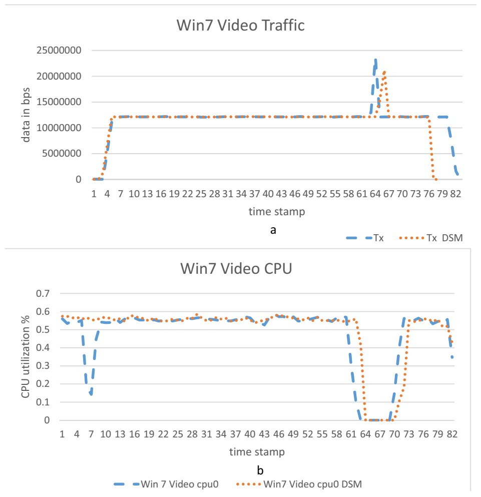


Figure 9. Windows Memory Intensive Workload

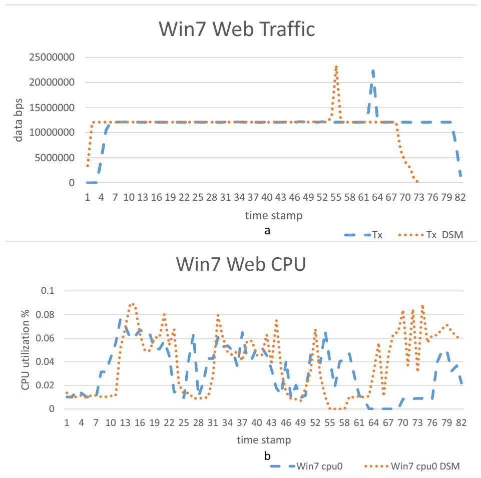


Figure 11. Windows Network Intensive Workload

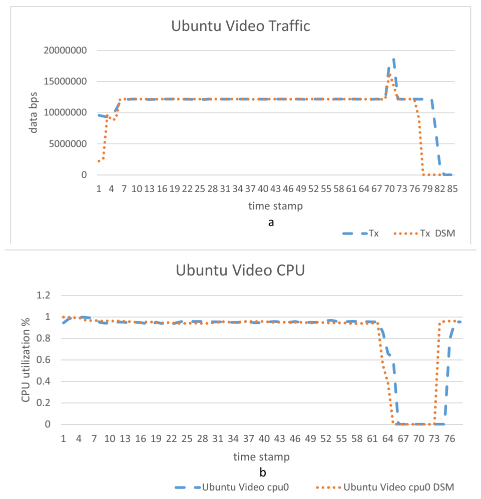


Figure 10. Linux Ubuntu Memory Intensive Workload

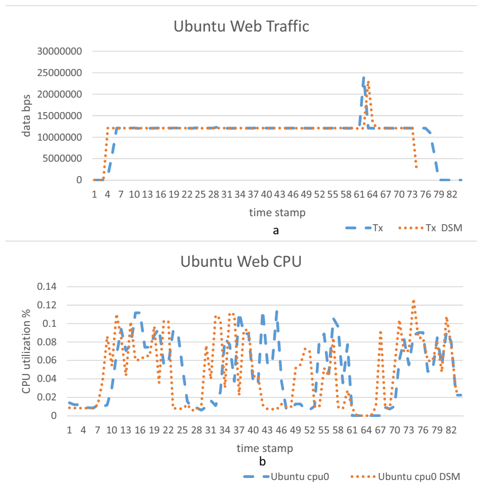


Figure 12. Linux Ubuntu Network Intensive Workload

In the DSM model the same pre-copy method is used, so the iteration phase provides the same memory stable state with minimum transferring of dirty pages, because of the proposed DSM locality attributes, which does not send any highly changed memory page. Furthermore, the speed up is achieved by using automatic parallelization of applicable and memory accessibility mapping between source machine and destination machine in sending memory pages or CPU and network state. The proposed model is built and integrated with virtualization architecture using the share storage. Our future work is to integrate the DSM HPC cluster with the VM migration as one unit.

References

[1] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," *Symposium on Networked Systems Design*, pp. 273–286, 2005.

[2] J. Hai, D. Li, W. Song, S. Xuanhua, and P. Xiaodong, "Live virtual machine migration with adaptive, memory compression," *Conference on Cluster Computing and Workshops*, pp. 1–10, 2009.

[3] X. Min, M. Vyacheslav, S. Jeffrey, V. Ganesh, W. Boris, and V. Inc, "Retrace: Collecting execution trace with virtual machine deterministic replay," *Workshop on Modeling, Benchmarking and Simulation*, 2007.

[4] J. Nelson, B. Holt, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin, "Latency-tolerant software distributed shared memory," *USENIX Annual Technical Conference*, pp. 291–305, 2015.