Analyzing Virtual Machine Live Migration in Application Data Context

Mutiullah Shaikh
Faculty of Electrical, Electronic and Computer Engineering,
Mehran University of Engineering and Technology,
Jamshoro, Pakistan

Muhammad Ali Memon
School of Information Technology,
Shaheed Benazir Bhutto University,
Shaheed Benazirabad, Pakistan

Asadullah Shaikh
College of Computer Science and Information Systems,
Najran University,
Najran, Saudi Arabia

Farah Deeba
Faculty of Engineering Science and Technology,
Hamdarad Institute of Engineering and Technology,
Karachi, Pakistan

***Abstract*—Virtualization plays a very vital role in the big cloud federation. Live and Real-time virtual machine migration is always a challenging task in virtualized environment, different approaches, techniques and models have already been presented and implemented by many re- searchers. The aim of this work is to investigate various parameters of Real-time and live data migration of virtual machines in stateful and data context at the application level. The migration of one virtual machine to another requires some time depending on the network bandwidth, guest availability, hardware limitation overcomes, resource allocation, server reallocation, hypervisor compatibility and many more. To enhance and ensure the performance and optimization of the time this work presents the some analysis in the form of different time stacks in multiple piece of data stored in the virtual machines. To optimize the migration time virtual machine checkpoints are used in order to achieve the better results by using the xen hypervisor memory technique which dynamically allows the migration of the configured memory while the allocated memory could be discarded for a while. By this the bad memory remains un-migrated only the good memory consisting the used data would be migrated by means of Real-time.**

Keywords—component; Cloud Computing; Virtualization; Virtual Machine Monitor VMM; Xen; VMResume; Xen Save and Restore; DC Data Centers Copy on Write CoW

I. INTRODUCTION

As the demand of cloud computing is increasing, storage and communication resources within data centers (DC) are developing new ways for the distributed resources of computing and sharing infrastructure by using virtualization. Virtualization actually was deployed for the cost saving. But very soon organizations realized that it is also effective in terms of speed, flexibility and robustness. In general," virtualization" refers to the process of turning a hardware-based entity into a software-embedded component and this is encapsulated in an entity called Virtual Machine (VM). By using Virtual Machines technique the resources are utilized in much more effective manner [2]. Virtualization has attracted considerable interest in recent years, particularly from the data centers and cluster computing communities. Since clusters are costly to own, therefore transferring and sharing access to a single general cluster is an optimal solution when demands vary time by time [3]. In other words, sharing of access or clusters is known as migration of virtual machines, means moving a VM from one source host to another sink host. If one VM has lot of load to carry, it can move and share some of load to another VM for better performance and results. Migration is also useful in maintenance of VMs. Additionally, if one VM fails, then through live migration the VM host failure recovery could be achieved. Live migration makes these invisible and seamless to users and end users [4]. Hence, this research typically focuses on the problems and different approaches to analyze the performance of the parameters for the Real-time on live data. Additionally, virtual machine migration between the single/multiple virtual machines on the basis of data availability, state maintenance in terms of time using multiple scenarios of data context in virtualized environment.

Virtual machine live migration in the cloud federation virtualized environment is always a much spirited task. Live migration of Virtual Machines plays vital role by providing virtual machine robustness. The main objective behind this research is to investigate and analyze different parameters achieved after implementation of live virtual machine migration. Specifically, this work aims to conduct the analysis for the optimization of migration time and live migration down time in the multiple scenarios such as:

- Time required for the Data Migration
- Time required for the State Maintenance
- Time Required for the Network Migration

In order to achieve the optimization and results in terms of time required by migration this research aims to perform the Migration between the virtual machines with the different amount of data and memory.

To implement the Virtual Machine Migration in the Cloud environment ini- tially tools required are:

– Ubuntu Cloud server 14.04 LTS TrustY

– Virtual Machine Monitor/Hypervisor Xen

Xen is the enhanced and updated type1 hypervisor which helps in creation of guest VMs and provide full access to the created guest VMs. It has the capability to save a Virtual Machine in a running state. After taking the snapshot of the saved Virtual Machine xen migrates that Virtual Machine to the another Virtual Machine in Real-time. This technology of xen called the xen Virtual Machine Xen save and resume.

II. Related Work

This section aims to describe some related work about the virtualization in the big cloud federation as well as the virtual machine live migration in the virtualized cloud environment and cluster based system. The idea behind this chapter is to present the earlier work for the sharing of resources between the virtual machines in terms of real-time and some discussion on enhancement in live migration. For the shared resources and to avoid the congestion of huge work load from a virtual machine various models and techniques have been specified by numerous researchers to study the workload and to present the overhead solutions.

A. Background & overview: live migration

As far as migration is concerned, it also took place in terms of offline migration, typically non-real-time. Web suspends/resume highlights on saving/resumption the current computing position and state on unspecified hardware [5]. Sapuntzakis et. al. attends to end user mobility and system management by encapsulating the computing atmosphere into capsules that could be shared between distinctive guests [6]. Schmidt et. al. by deploy capsules, in the form of interrelated processes with their network addresses i.e., IP and the entire network states, as the shape of migration decades [7]. In the same way, Zap uses process groups (pods) as well as their state at the kernel level in the shape of migration decades [8]. In all these proposed preceding, the running execution suspended for a while and the processes which is in use in the form of applications within the VM remained un-progressed.

To get the always availability of the data and other computing resources, presently many techniques of live migration exits in the virtualization-based environment [9, 10]. From all two of the illustrations are live migration in Xen [11] i.e., xen motion and migration of VMwares i.e., Vmotion [12], which migrates in the same manner as pre-copy strategy. For the duration of migration, pages of physical memory transfers from the one primary sink host to the another new host (backup), while the state of the VM is running on the sink host (pri-mary).Memory pages which are replicated during the migration of VMs should ensure their consistency and integrity. After that iterative procedure of VM shar- ing phase, stop-and-copy phase will be initiated and executed for a while and that caused VM suspended, the remaining pages of the configured memory are transferred , due to this the machine monitor (Hypervisor) of the destination (backup) VM generate a signal for the resumption of the executed VM. Though, the pre-copy takes minimal downtime of VM being migrated in comparison of the others [13].

In addition with pre-copy procedure according to Kemari, there are some previous related techniques which proposed the better solution for optimization during the migration [14–20]. Such as post-copy migration technique has pointed out the cons of pre-copy migration [21, 22]. Some experimental results shows that the downtime taken by post-copy migration for a VM being migrated is less than the time spend by the pre-copy migration [21]. However, pre-copy implementation supports the (PV) para-virtualized users as the catching memory method is involved which accesses and manages a memory based pseudo-paging system within a guest. Since, as the upgraded/patched version of OS needed by the post- copy procedure so due to this it could not be commonly used as the pre-copy. Hines et. al. introduces the design by combining the post-copy and pre-copy mutually [21]. By this combination an adaptive pre-paging method proposed by them, which maintains the access patterns of the user applications.

B. Improved live vm migration

Now apart from that Remus is of the official Xen warehouse [23]. It gains the huge and always availability by keeping an copy of the updated VM along with its running state on the secondary host computer i.e., (backup), which is alert and activated whenever the primary host get failed and its state being destroyed. LLM initially updates the memory after that copied dirty data and then uses the pages of the memory excluding copying [24]. Although, the tracking of the bad data is not so much efficient, the onwards goal is to further enhance and save the memory and processing time as well as power by analyzing the performance in the different decades of data context within the VMs would be migrated from sink to backup.

Lu et al. also achieved the always availability by using three state memory synchronization [28], like in systems Remus: bad memory tracing, active VM backup along with tentative transfer of the state. This method describes main idea of the proposed work in the paper and tells us about the actual pros of the migration, however, it implies with the associated memory migration overhead. For instance, in the shared environment the swapping of workload and its estimation, the memory overhead is more than the 50%. Since the main and configured memory is the essential resource, the ratio in high percentage overhead is a trouble. To overcome the memory issues in the systems which are Xen-based, several ways are available to inflexibility memory redundancy in guest VMs, like as patching and sharing of memory pages. Some previous efforts have shown the potential memory sharing in virtualized systems. Some changes in working sets were inspected and their results demonstrated that changes were essential for the host to host VMs migration [11, 26]. For the guest virtual machine with 512MB allocated

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors*).

memory, roughly changed low load with 20MB, roughly changed medium load with 80MB, roughly changed high load with 200MB. Therefore, the workloads normally take places between these boundaries. The previous evaluation also makes known the amount in memory changes with different workload running in the VMs (within minutes) [11, 26]. Within two minutes none of the VM make changes in the memory more than 4MB. The Content-Based Page Sharing (CBPS) technique also revealed the memory sharing potentially [21]. CBPS typically based on the technique known as Compare-by-hash introduced in [23, 24]. As claimed, the CBPS was capable to recognize form all pages 42.9% as much as sharable, and reclaimed from the all pages 32.9% doing real-world workload in the ten instances of Windows NT. Nine guests VMs were capable to illustrate the sharable pages as much as 29.2% and reclaimed when decreased from nine to five guest VMs as 18.7%, and the resultant numbers were 10.0% and 7.2%, correspondingly.

For Sharing the memory pages in the efficient manner, nowadays, the technique known as CoW (Copy-on-Write) was broadly adopted in Xen Hypervisor [27]. Unlike the OS that uses CoW method for the sharing of memory pages in a conventional way, in virtualized environment, pages are shared between the multiple guest VMs. as an alternative of using CoW to migrate the memory pages between the VMs, here we use the same idea but in the more efficient manner like by sharing the pages between smaller blocks. The Difference Engine illustrated the potential saving in memory obtainable from the leveraging a mixture of page patching, sharing and in-core level memory compression [13]. It also reveals the vast potential of exploiting memory redundancy in guest VMs. On the other hand, Difference Engine also faces difficulty problems when using the patching technique because some additional modification required by Xen.

III. CONTRIBUTIONS

In this section of paper we aim to describe the work done during the conduct of this research. Specifically, few proceeding overviewed and detailed described in this section in order to achieve our aims and objectives. To overcome the issues faces during the Live migration we discussed an adaptive way of Live migration to improve the Load balancing, optimize the Downtime, VM disk/data migration. Furthermore, this section also defined resumption of a VM into another VM in the Real-time. The key idea behind this research work i.e., Live migration in data context is discussed in this section as well.

A. Virtual machine resumption

While during the resumption of saved VM using a stored checkpoint file from slow-access storage, the saved states in a checkpoint file should be retrieved. Those saved states are virtual/shared CPU states, the states for emulated de- vices as well as contents of memory disk of VM. Usually, most of the data saved in the checkpoint file retrieves from contents of VM memory. Therefore, a straightforward method for the resumption of VM is to restore first all the saved memory data from the saved check-point file of VM, and then retrieve the rest of data which includes device and CPU data. The VM cannot start without the required device and CPU states, it cannot be initialized until all data have been retrieved from memory along with its all previously stored memory pages have been set up. Presently, Xen hypervisor uses this method for VM resumption from a check-point file.

we can summarize that the same issue of the check-pointing VM mechanism also takes place in VM resumption: as amount of contents of VM memory dominate the stored and saved data in check-point file, when assigned memory to VM increases, the time consumed on restoring its saved data rapidly develop into bottleneck. As illustrated in figure 1, with the increase of size of VM memory, the time took by command xm restore would also increase linearly. For small amount of memory (i.e. with a size of VM memory 128MB), but while retrieving in gigabytes from a saved check-point file (i.e. 10s in the 1GB), there is a significant increase in time to resumption.

However, in the first solution, which typically restores data from memory before the device and CPU data, which results in-effective. Also, it is very difficult to consider data before the CPU and device while restoring the data memory in reverse order.

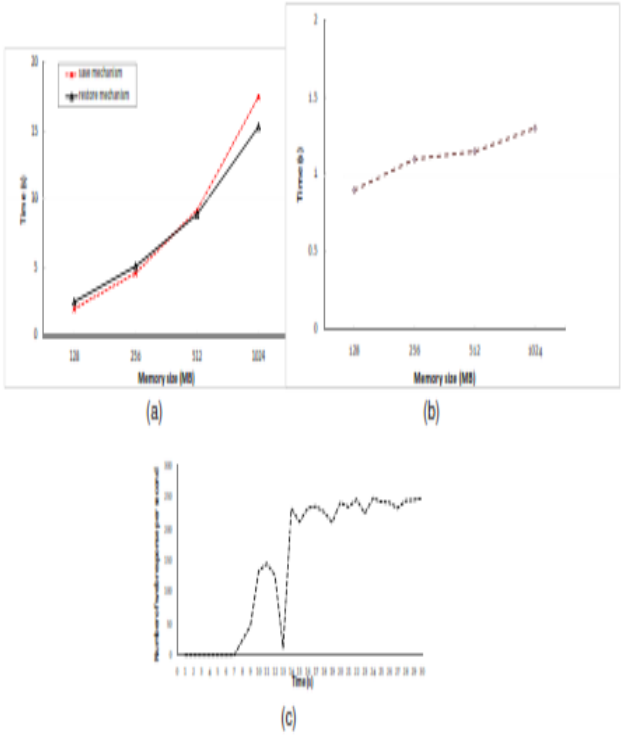

Fig. 1. Comparison of mechanisms for VM resumption

That means, letting the VM to boot initially to load required necessary devices and CPU states, after this loading, now restore the data of memory pre saved in check-pointed file after VM starts. In a certain case, when the VM requires to access a page from memory which has not still been loaded, then this corresponding

data is retrievable from on-disk check-point file and get sets up the pages. This solution provide the benefit as it starts of VM very quickly, and always keeps it in running state while restoring the data memory. Moreover, since in this manner, the VM only require the restoration of small and less amount of necessary device and CPU states to start VM, its performance would not be affected by the size of VM memory.

However, in contrast with the first solution, the second one has some demerits. In the first solution, after the startup of VM (although it takes 10s or even few minutes), the works well and as better as before checkpointing. In comparison, with the second solution, the VM appears and supposed to be in running state after restoring the necessary device and CPU states. However, when it wants to accesses a page from memory which has not still be stored, a page fault immediately occurs. Then the in process execution must be paused by hypervisor, then the checkpoint file restores the memory pages, and then it will be resumed. Since at the beginning significant number of page faults occur due memory data is not restored by VM at first, which degrades somehow VM performance. Our performed experiments shows, for a VM consist of 1GB RAM, the VM runs so much slow to be useful in duration of first 10 seconds. Almost all of this time in seconds was consumed on restoring the necessary required memory data.

To pick the dual benefits of both solutions for VM resumption and to over- come their limitations, a mechanism for dual purpose is hybrid resumption called as VMresume.

Our aim is to start a VM in the running state as soon as possible, but to avoid degradation in performance caused by page faults when VM starts. Our basic aim and purpose is to examine the memory pages which have high possibility to access at the beginning period after the VM startup, restore those pages form check-point file, and then boot the VM by loading the necessary required device and CPU states. By reason of preloading all likely-to-be-accessed pages of memory, we ensure the after the startup of VM, there could not be as much as page fault found in the second solution. Also, we can ensure the earlier startup of VM compared with first solution, this is due to reason we do not preload all the data from memory which is saved in the checkpoint file before the restoring the device and CPU states. This hybrid mechanism for resumption provides some benefits in data context by ensuring the high availability of memory pages during the resumption in the running state.

So now, for the likely-to-be-accessed pages from memory, how it can be determined? Their determination can be achieved by principle of temporal locality, pages recently updated are likely to get updated in near future. Therefore, we can trust on the facts on the recent activities of memory access to forecast the upcomin activities for memory access. For suppose we get an upgraded checkpoint and now we wish to VM resume from the latest checkpoint. According to the principle temporal locality, pages which would have highest possibility to be access during the initial period, so those memory pages get accessed during the latest checkpoint interval. Thus, while receiving the checkpoint file from a checkpointing interval, we maintain a record of the memory pages recently likely to be accessed during that interval, and use that maintained record to forecast such memory pages which are likely to accessed after the VM resumption. This needs a predictive mechanism for checkpointing.

B. Virtual machine disk migration

In comparison with live migration in LANS (Local Area Networks), migration for VM possess additional challenges in WANS (Wide Area Networks). While migrating a VM within a LAN network, the storage in disk for both source and destination/target VMs are often shared by network-attached storage (NAS) or SAN (Storage area Network) media. Therefore, in LAN-based migration most part of the data that requires to be migrated is derived from run-time state of memory of VM. However, while migrating a VM within a WAN network, besides the state of memory, entire disk data, along with file system and I/O devices state, should also be migrated, this is due to they are not shared between both source and destination/target VMs. The disk data, particularly for I/O likewise applications, is commonly very large (e.g., in order of 100s of GBs). Therefore, LAN-based VM migration approaches that can only migrate the data from memory (usually in order of GBs) may not be go well with when applied to WAN-based VM migration.

A straightforward method to migrate a VM at the data context level is to suspend the VM on source host machine, then transfer the stored data in local disk memory (in the form of a self-contained image file) on the network, them towards the destination/target host machine, after then reload memory and file system to resume the VM [6]. Although, these stop-and-resume faces long downtime. In order to decrease the larger amount of data within a disk that is going to be transmitted, many optimization techniques have also been earlier introduced-i.e. data compression while migration and content-based comparison between disk data and memory [25, 26]. However, sometimes these optimization techniques introduces either computational or memory overheads. Therefore, it really needs to develop new scenarios for migration of VMs with their potentially larger file systems with acceptable overhead and minimal downtime.

In order to achieve quick live migration within WANs, the disk data with larger amount that is going to be transferred over the WAN should be reduced. Traditional migration techniques in LAN-based systems include in previous work uses checkpointing/resumption approaches discussed earlier to migrate data of memory [23, 24]. Moreover, to migrate the shared disk they use some incremental checkpointing mechanism to decrease the updated memory data that has to be transferred during each migration stage. These incremental checkpointing are often used for the virtual disk migration to gain the low downtime, but the problem occurs when larger

data in disk and memory combined can still prevents in unacceptable total migration time.

C. Live migration in data context

In order to get effective live migration as well as high availability of computational resources along with rapid resource allocation VM resumption is the most important feature in virtualization. Also, it is a straightforward approach for the maintenance of data consistency in data sharing context and application context from source machine to the destination/target machine. However, by implementing the predictive checkpointing technique and hybrid solution for VM resumption i.e. VMresume, we get the data in run-time with application context without any unacceptable migration downtime. This is our main goal behind this work.

Since in contrast with previous work and citations, we aim to present a new technique based on incremental checkpointing mechanism to share the data pages during the resumption i.e. a predictive checkpointing for resumption of VM mechanism. In this mechanism, we assume VMresume, when the system initialized, first of all the complete image of data in VM memory as well as emulated devices/CPU states saved by the VMresume to on-disk file, which becomes the VM initial checkpoint. After that, it checkpoints the VM at constant and fixed frequency. Then all off the memory pages are set to be as read-only at the start of the checkpointing interval (i.e. which typically shows the time between the previous and leads to next checkpoint). Thus, if there is found any write mode in memory pages, it triggers an alert, that alert is coded for page fault. By leveraging shadow-paging feature in Xen, VMresume captures whether a page settle as read-only and tracks either it a dirty or otherwise. Whenever write mode found in a read-only page, alert triggers a page fault and it would be reported to VMM, then that page is set up as writable. Thereafter, VMresume adds the address of the triggered faulted page in the list of changed pages and discards the write mode protection from the page for the application proceeding in write. The list of changed pages which are modified during that particular interval updated, at the end of that interval. VMresume copies to the checkpoint state of all changed pages, and resets again all pages as read-only. This provides high availability of data resumption during the migration.

By using this incremental checkpointing approach, it helps to find the entire write mode accessed pages in memory during the latest checkpoint interval. These all write accessed pages are probably to get accessed after VM resumption. However, write accessed pages are often a small section of pages that are likely to get accessed after resumption of the VM. Besides this, there are some more pages in memory which are purely read accessed during the same checkpoint interval. The pages which are read accessed are not recorded by in our used checkpoint method, but they should also get preloaded while resuming the VM to decrease the potential paging faults on those pages.

Live migration in data context perspective from our work proposes a quick VM resumption by taking the snapshot/VM image. Here in our performed experiments as shown in f, A and B, we take 3 host VMs in the active state and enough configured. From the data context perspective, we assigned VMs with memory of 1GB, 2GB and 3 GB respectively. The assigned memory is al- located memory which is a complete memory of that particular VM. But, from the allocated memory some portion of memory either has data pages or otherwise. Portion of memory that has some data pages is configured memory. We proposed VMresume with the mechanism of allocated and configured memory. As discussed above about the read-accessed and write-accessed memory pages. The entire allocated memory is read-accessed while the configured memory is write accessed. While performing the Live migration from the source machine to the target/destination machine some our work focus on transferring the con- figured memory pages which has the actual data that has the high possibility to be accessed rather than the allocated memory. This solution overcomes and overheads unacceptable downtime during the migration, data duplication and trigger page faults.

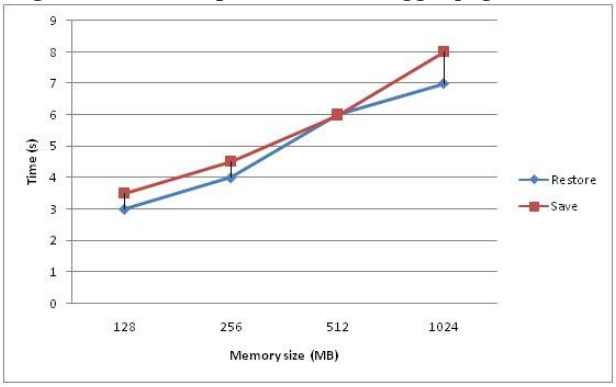

Fig. 2. Comparison of VM resumption from configured memory

Therefore, figure 2 illustrates the migration downtime needs to transfer the configured memory. The configured memory is the main focus behind our pro- posed work, which includes the facts and figures that does not involve any bad memory. All of the memory is configured means the data is also configured data (useful for the user). As shown in the figure 2, the comparison of configured data migration is meanwhile seems to be not too high, the both lines are coinciding with each other, this could happened due to the memory migration based on the configuration memory which saves the computational and migration downtime in the meanwhile which can also overcome the load balancing of VMs and also helps in performance degradation.

Additionally figure 3 illustrates comparison of VMs with allocated memory rather than the configured. The allocated memory consist all the memory storage which is assigned to a particular VM, and data within the allocated memory consist all over the memory that can be either user needed or otherwise. As shown in figure

the migration downtime consumption is quite high due to big storage and raw data and the lines are too far to each other, so in the resultant it presents performance degradation due to the high storage and raw allocated migration.

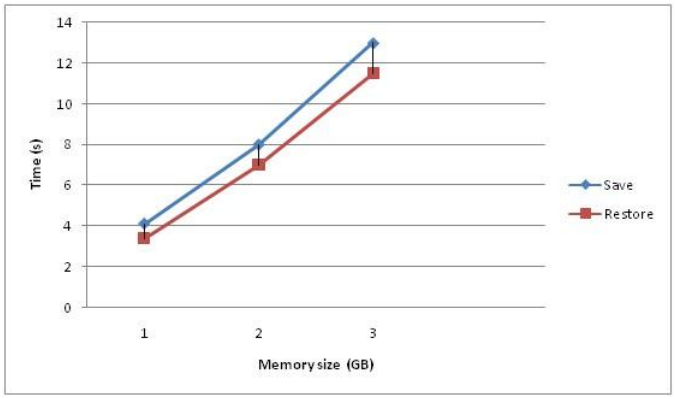

Fig. 3. Comparison of VM resumption from allocated memory

Moreover in the summary of our performed experiments aims the migration of configured memory (i.e. memory that has the actual data and write-accessed pages) rather than the allocated memory (i.e. memory that has whole VM data read-accessed and write-accessed pages), while migration of entire al-located memory results a high downtime etc. Therefore, to perform an efficient migration we propose VMresume mechanism with migration of configured memory technique. Therefore, By implementing this, at the end of checkpoint interval, for the data pages (i.e. write-accessed) in the interval, VMresume saves in checkpoint file and observes their R/W bit. For those memory pages whose A bit set to 1, then VMresume determines whether they are read-accessed or write-accessed. If pages are write-accessed, then they are already saved in check-point file and going to be migrated. If they are read-accessed, VMresume keeps copy of those pages read-accessed pages for future prediction purpose whenever resuming VM from corresponding checkpoint file. It is completely unnecessary to save the read-accessed memory pages contents because they are not updated and modified during migration checkpoint interval.

While resuming the VM, VMresume initially reloads all of the write-accessed pages (i.e. they are newly saved in checkpoint file), also other likely-to-accessed by tracking the record of all read-accessed pages. Then the VM will be resumed and started with required CPU/devices states along with the data resumption which is typically configured and likely-to-be-accessed with any unacceptable downtime and delay.

IV. Experiments & Observations

In this section, the performance from the proposed work and techniques (i.e. Xen Hypervisor) and estimated results are presented. We measured some overhead and migration downtime by using under FGBI, and analyzed and compared the achieved results with that under Remus and LLM.

A. Experimental setup

We have designed an experimental setup which includes two hosts. One host is primary or master and second one is used as backup. The two hosts are Intel core2 Duo processor 2.6 GHz and 4 GB RAM. The two hosts are connected through a 2 Mbps network connection. The network connection is used for migration of the Primary host to the second host.

TABLE I. Specifications of Guest VMs

Parameter / VM	Guest vm1	Guest vm2	Guest vm3
ID:	2	3	4
Name:	Testvm1	Testvm2	Testvm3
Hypervisor:	Xen	Xen	Xen
OS Type:	Hvm (Ubuntu)	Hvm (Ubuntu)	Hvm (Ubuntu)
State:	Running	Running	Running
CPU:	1	1	1
CPU Time:	11.2 s	12.5 s	14 s
Virtual Memory (RAM):	524288 KiB	524288 KiB	524288 KiB
Allocated Memory (ROM):	1048576 KiB	2097152 KiB	3145728 KiB
Disk Space:	1 GB	2 GB	3 GB
UUID:	192.168.0.60	192.168.0.30	192.168.0.20

B. Experimental results

Here VMs are migrated in two situations. The first one when there is no work load on VMs and second one when there is workload of different applications on VMs.

VM Migration with no Load Live Migration is the practice of transfer of the Virtual Machines active memory state and accurate execution state over a high- speed network, which permits the VM to shift from running on the source host to destination host. Live migration of a VM has some time parameters named as Real, User and Sys. These parameters have different values, depending on the virtual RAM and hard disk of the VM.

TABLE II. Specifications of Guest VMs

	Real	User	Sys
Testvm1	10.378s	2.068s	5.252s
Testvm2	19.789s	3.015s	9.651s
Testvm3	28.651s	3.859s	14.798s

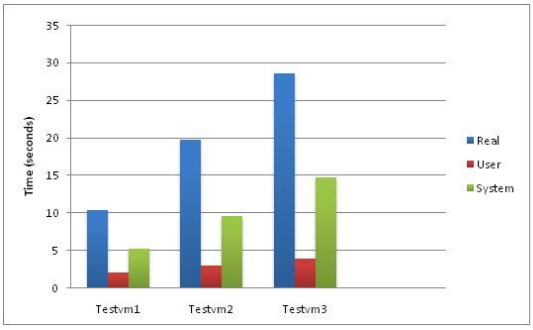

Fig. 4. Migration Time taken by XenServer in our system

Real: It is the time taken by host VM for live migration of guest VM. User: It describes the time taken by host VM for state migration of guest VM. Sys: It expresses the time which is taken by host for memory migration of guest VM.

The table 2 specifies the three migration time parameters of the entire guest VMs. The parameters specify that how much time it takes to migrate the entire VM from source to destination.

By plotting the above parameters data in pictorial form it looks like as follows:

From figure 4, one can estimate the total migration time, state migration time and memory migration time taken by the VMs in our system. The three VMs named as Testvm1, Testvm2 and Testvm3. All the VMs are of different sizes consecutively of 1GB, 2GB and 3GB. Greater the virtual hard disk is longer time is taken by Xen Hypervisor to migrate, as shown in fig 4. Here in the result, the three parameters we achieved are the Real (time taken by Data migration), user (time taken by the Network migration), system (time taken by the State migration). All the parameters and their data in accordance with time are shown meanwhile in figure 4.

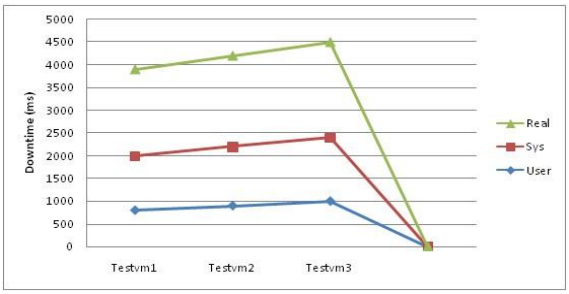

Fig. 5. Downtime of VMs with different memory size m

While the VM is migrated from source host to destination host, the both hosts will be down. In other words, the hosts will not be able to communicate or run any application. This downtime also depends on the RAM and hard disk of virtual machine (VM). The graph below (Fig 5) shows the down time of migration time parameters of the three VMs. The Testvm1 is smaller is size as compared to Testvm2 and Testvm3, therefore it takes lesser downtime of the parameters (Real, Sys and User).

Live migration consists of three parameters: migration of entire VM, state migration and memory migration. The memory migration also depends on the hard disk and RAM of VM. As mentioned earlier and our main focus is to overall migration the figure 5 states that how much time VMs take to transfer the memory state. Moreover, it is illustrated from figure 5 the individual parameters are plotted of each of all three VMs. Hence the size of each VM is different the time taken by each parameter for each depends on the size and processing speed in order to achieve the better mean time while performing the migration.

Hence the primary motivation behind this proposed work is to perform live migration the data context level. So figure 6 shows the memory and data migration of all of three VMs. Therefore, here two parameters Real and sys are plotted which typically shows the data migration and memory migration. In order to mi- grate the data between the VMs the hypervisor will check either the memory is configured or otherwise then the migration will be performed accordingly. If the hypervisor found the memory which configured and then perform migration so we can assume this strategy ensures the best results and performance.

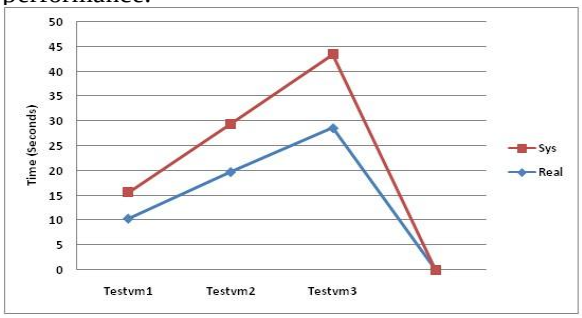

Fig. 6. Memory Migration of Xen Hypervisor

V. Conclusions and Future Work

This work presents some set of methods and approaches for the analysis off VM live migration in data sharing perspective between the VMs in run-time. Further- more live migration in data sharing perspective without any unacceptable halt, delay and performance degradation. This happened possible with VM resumption techniques i.e. save and restore technique and VMresume hybrid solution. The design, implementation and analysis of our proposed techniques ensure the high availability with unacceptable downtime and performance degradation. This happened possible with the help of Xen hypervisors save and restore commands and VMresume mechanism. In this research we analyzed and aim to propose that, from our performed experiments and evaluations it can be analyzed that we can achieve minimal downtime with unacceptable performance degradation and high availability of data and resources by migrating the configured memory of VM (i.e. likely to be accessed memory pages or real/occupied data) rather than the allocated memory (i.e. entire assigned data may be blank space) while performing the live migration

REFERENCES

[1] What is cloud computing?? Luit Infotech

[2] Diego Perez-Botero, A Brief Tutorial on Live Virtual Machine Migration From a Security Perspective, Princeton University, Princeton, NJ, USA

[3] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen Eric July, Christian Limpach, Ian Pratt, Andrew Warfield, Live Migration of Virtual Ma- chines, University of Cambridge Computer Laboratory 15 JJ Thomson Avenue, Cambridge, UK, Department of Computer Science University of Copenhagen, Denmark

[4] NSRC, Virtual Machine Migration

[5] Mahadev Satcyanarayanan, B. Gilbert, M. Toups, N. Tolia, D.R. OHallaron, Ajay Surie,A. Wolbach, J. Harkes, A. Perrig, D.J. Farber, M.A. Kozuch, C.J. Helfrich, P. Nath, and H.A. Lagar-Cavilla. Pervasive personal computing in an internet suspend/resume system. Internet Computing, IEEE, 11(2):1625, 2007.

[6] Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S.Lam, and Mendel Rosenblum. Optimizing the migration of virtual computers. SIGOPS Oper. Syst.Rev. 36(SI):377390, December 2002.

[7] Brian Keith Schmidt. Supporting ubiquitous computing with stateless consoles and computation caches. PhD thesis, Stanford, CA, USA, 2000. AAI9995216.

[8] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh. The design and im- plementation of Zap: a system for migrating computing environments. SIGOPS Oper. Syst. Rev, 36:361376, December 2002.

[9] Wei Huang, Qi Gao, Jiuxing Liu, and Dhabaleswar K. Panda. High perfor- mance virtual machine migration with RDMA over modern interconnects. In CLUSTER 07:Proceedings of the 2007 IEEE International Conference on Clus- ter Computing, pages 1120, Washington, DC, USA, 2007. IEEE Computer So- ciety,

[10] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu. Live migration of virtual machine based on full system trace and replay. In Proceedings of the 18th ACM international symposium on High performance distributed computing, HPDC 09, pages 101110, New York, NY, USA, 2009. ACM.

[11] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation - Volume 2, NSDI05, pages 273286, Berke- ley, CA, USA, 2005. USENIX Association.

[12] Michael Nelson, Beng Hong Lim, and Greg Hutchins. Fast transparent migration for virtual machines. In ATEC 05: Proceedings of the annual conference on USENIX Annual Technical Conference, pages 2525, Berkeley, CA, USA, 2005. USENIX Association.

[13] Diwaker Gupta, Sangmin Lee, Michael Vrable, Stefan Savage, Alex C. Snoeren, George Varghese, Geoffrey M. Voelker, and Amin Vahdat. Difference engine: harnessing memory redundancy in virtual machines. Commun. ACM, 53:8593, October 2010.

[14] Renato J. Figueiredo, Peter A. Dinda, and Jose A. B. Fortes. A case for grid computing on virtual machines. In Proceedings of the 23rd International Con- ference on Distributed Computing Systems, ICDCS 03, pages 550,Washington, DC, USA, 2003. IEEE Computer Society.

[15] John R. Lange and Peter A. Dinda. Transparent network services via a vir- tual traffic layer for virtual machines. In Proceedings of the 16th international symposium on High performance distributed computing, HPDC 07, pages 2332, New York, NY, USA, 2007. ACM.

[16] A. Feldmann R. Bradford, E. Kotsovinos and H. Schioeberg. Live wide-area mi- gration of virtual machines including local persistent state. In VEE07: Proceed- ings of the third International Conference on Virtual Execution Environments, pages 169116, San Diego,CA, USA, 2007. ACM Press.

[17] Yoshiaki Tamura, Koji Sato, Seiji Kihara, and Satoshii Moriai. Kemari: Vir- tual machine synchronization for fault tolerance using DomT (technicalre- port).http:/wiki.xen.org/xenwiki/Open Topics For Discussion?action=Attach File&do=get&target=Kemari 08.pdf, 2008.

[18] Franco Travostino, Paul Daspit, Leon Gommans, Chetan Jog, Cees de Laat, Joe Mambretti, Inder Monga, Bas van Oudenaarde, Satish Raghunath, and Phil Yonghui Wang. Seamless live migration of virtual machines over the man/wan. Future Gener. Comput. Syst.22 (8):901907, October 2006.

[19] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. cost of virtual machine live migration in clouds: A performance evaluation. In Proceedings of the 1st International Conference on Cloud Computing, Cloud- Com 09, pages 254265, Berlin, Heidelberg, 2009. Springer-Verlag.

[20] Ming Zhao and Renato J. Figueiredo. Experimental study of virtual machine migration in support of reservation of cluster resources. In VTDC 07:Proceedings of the 2nd international workshop on Virtualization technology in distributed computing, pages 5:15:8, New York, NY, USA, 2007. ACM.

[21] Michael R. Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In Proceed- ings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, VEE 09, pages 5160, New York, NY, USA, 2009. ACM.

[22] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. Re- active consolidation of virtual machines enabled by postcopy live migration. In Proceedings of the 5th international workshop on Virtualization technologies in distributed computing, VTDC 11, pages 1118, New York, NY, USA, 2011. ACM.

[23] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchin- son, and Andrew Warfield. Remus: high availability via asynchronous virtual machine replication. In Proceedings of the 5th USENIX Symposium on Net- worked Systems Design and Implementation, NSDI08, pages 161174, Berkeley, CA, USA, 2008. USENIX Association.

[24] Bo Jiang, Binoy Ravindran, and Changsoo Kim. Lightweight live migration for high availability cluster service. In Proceedings of the 12th international conference on Stabilization,safety, and security of distributed systems, SSS10, pages 420434, Berlin, Heidelberg,

[25] Hai Jin, Li Deng, Song Wu, Xuanhua Shi, and Xiaodong Pan. Live virtual machine migration with adaptive, memory compression. In Cluster Computing and Workshops, 2009. CLUSTER 09. IEEE International Conference on, pages 1-10,31 2009 sept 4, 2009

[26] Pierre Riteau, Christine Morin, and Thierry Priol. Shrinker: Improving live mi- gration of virtual clusters over wans with distributed data deduplication and content-based addressing. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, Euro-Par 2011 Parallel Processing - 17th International Confer- ence, Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011, volume

[27] Yifeng Sun, Yingwei Luo, Xiaolin Wang, Zhenlin Wang, Binbin Zhang, aogang Chen, and Xiaoming Li. Fast live cloning of virtual machine based ceedings of the 2009 11th IEEE International Conference on High Performance Computing and Communications, pages 392399, Washington, DC, USA, 2009. IEEE Computer Society.

[28] Maohua Lu and Tzi cker Chiueh. Fast memory state synchronization for virtualization-based fault tolerance. In Dependable Systems Networks, 2009.DSN 09. IEEE/IFIP International Conference on, pages 534543, 2009