Global Journal of Computer Science and Technology
Volume 11 Issue 13 Version 1.0 August 2011
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

The Evaluation of Network Performance and CPU Utilization during Transfer between Virtual Machines

By Igli Tafa, Elinda Kajo, Elma Zanaj, Aleksandër Xhuvani
Polytechnic University of Tirana

Abstract - The aim of this paper is to test the performance of network communication between virtual machines during the live migration while a controlled failure occurs. Also we want to test the CPU utilization between the virtual machines in different hosts and to make a comparison with the live migration in different physical nodes. We want to introduce a script in C++ which will improve the performance of migration during the failure controller phase. Another script is built to test the CPU utilization on host computers when memory utilization is increased or decreased. We have used Httperf benchmark to test this script. Requests are sent from client machine to server machine with IPs of C class located in different hosts. These requests are some files in C which execute a multiplication of 2 square matrices with range 5, at the moment they arrive in the destination node. We have used Para-virtualization approach (Xen-Hypervisor), because it gives more flexibility and tolerance to researchers.

Keywords : *Live migration, Network Communication, CPU Utilization, Controller Phase, Hypervisor.*

GJCST Classification : *D.4.8, C.4, D.2.8, I.6*

Strictly as per the compliance and regulations of:

The Evaluation of Network Performance and CPU Utilization during Transfer between Virtual Machines

Igli Tafa[α], Elinda Kajo[Ω], Elma Zanaj[β], Aleksandër Xhuvani[ψ]

Abstract : The aim of this paper is to test the performance of network communication between virtual machines during the live migration while a controlled failure occurs. Also we want to test the CPU utilization between the virtual machines in different hosts and to make a comparison with the live migration in different physical nodes. We want to introduce a script in C++ which will improve the performance of migration during the failure controller phase. Another script is built to test the CPU utilization on host computers when memory utilization is increased or decreased. We have used Httperf benchmark to test this script. Requests are sent from client machine to server machine with IPs of C class located in different hosts. These requests are some files in C which execute a multiplication of 2 square matrices with range 5, at the moment they arrive in the destination node. We have used Para-virtualization approach (Xen-Hypervisor), because it gives more flexibility and tolerance to researchers.

Keywords : Live migration, Network Communication, CPU Utilization, Controller Phase, Hypervisor.

I. Introduction

Live Migration is one of the most important tools in Virtual Technology. It means that data can be transferred from one virtual machine to others, while there are working. Virtual Machine is above the Hypervisor. In our paper we have used Xen-Hypervisor, which operates above the bare hardware. Host Operating System on the bare hardware is called Dom0 and it is a privileged zone. Above Xen can be located the Guest Operating System, it is called DomU, and is built above the Hypervisor.

Xen can support a lot of virtual machines and every virtual machine has it`s own dedicated virtual memory. Xen offers para-virtualization approach, as ESX_Server does, localized between Full Virtualization and OS virtualization and offers more flexibility and tolerance for researchers. Virtualization offers some advantages such as: Flexibility, High Scalability, good utilization of the Hardware, Safety etc, but on the other hand it has the disadvantage of delays. In this paper we have studied live migration between virtual machines in different hosts. We have tested the performance of the system by using a C script which multiplies two square matrices. At first all the calculations are performed between 2 virtual machines in different hosts and then between 2 physical machines. Of course we should take in consideration the interface of network communication. To monitor live migration process we have used heartbeat tool. We can improve the performance of live migration by modifying a script in Heartbeat tool. By using this tool we can verify and precaution the failures. In this way if a virtual machine fails, then the transfer of the virtual machine's data into another virtual machine can be safe. Heartbeat tool can help us to monitor the CPU and Memory Utilization in host computers. Httperf tool generates messages (in our case the message is the script in C) with different size (by calling a script which dynamically changes the size of the memory used) and rate from one machine to another by offering different values of CPU and Memory Utilization.

We propose and implement a script in heart-beat tool which offers a better performance than Xen in the live migration between virtual machines in different hosts. Then we evaluate the performance between nodes. All these nodes create a cluster. This paper is organized as follows. In the second section is introduced the Related works. The third section is the Pre-Experimental Phase. In the forth section is implemented a Script, in the fifth is presented the Experimental Phase and in the sixth section are given the Conclusions and Future Works. At the end are listed the References.

II. Related Works

In [1] are shown three types of virtualization and components of infrastructure virtualization. In [2] is presented the design and evaluation of a set of primitives implemented in Xen. *XenMon tool* accurately measures per-VM resource consumption, including the work done on behalf of a particular VM in Xen's driver domains and *SEDF-DC* scheduler accounts for aggregate VM resource consumption in allocating CPU [2]. So the *ShareGuard tools* limits the total amount of resources consumed in privileged and driver domains based on administrator specified limits.The performance evaluation indicates that tool effectively enforce performance isolation for a variety of workloads and configurations.

In [3] is described the construction of a general and transparent high availability service that allows

Author α Ω β ψ : Polytechnic University of Tirana ,Information Technology Faculty ,Computer Engineering Department.
E-mails : itafaj@gmail.com , e_kajo@yahoo.com , hakikpaci@gmail.com, axhuvani@yahoo.com

existing, *unmodified* software to be protected from the failure of the physical machine on which it runs. *Remus* provides an extremely high degree of fault tolerance, to the point that a running system can transparently continue execution on an alternate physical host in the face of failure with only seconds of downtime, while completely preserving host state such as active network connections. They created an approach which encapsulates protected software in a virtual machine, asynchronously propagates changed state to a backup host at frequencies as high as forty times a second, and uses speculative execution to concurrently run the active VM slightly ahead of the replicated system state. In [4] is described a lightweight software mechanism for migrating virtual machines with direct hardware access. They based their solution on *shadow drivers*, an agent in the guest OS kernel that efficiently captures and restores the state of a device driver. On the source machine, the shadow driver monitors the state of the driver and device. After migration, the shadow driver uses this information to configure a driver for the corresponding device on the destination machine. Shadow driver migration requires a migration downtime similar to the driver initialization time, short enough to avoid disrupting active TCP connections. The performance overhead, compared to direct hardware access, is negligible and is much lower than using a virtual NIC. In [5] is designed and implemented a continual migration strategy for virtual machines to achieve automatic failure recovery. By continually and transparently propagating virtual machine's state to a backup host via live migration techniques, trivial applications encapsulated in the virtual machine can be recovered from hardware failures with minimal downtime while no modifications are required.They show that virtual machine in a continual migration system can be recovered in less than one second after a failure is detected, while performance impact to the protected virtual machine can be reduced to 30%. In [6] is evaluated the performance of several virtual machine technologies in the context of HPC. A fundamental requirement of current high performance workloads is that both CPU and I/O must be highly efficient for tasks Such as MPI jobs. This asks two virtual machine monitors, OpenVZ and KVM, specifically focusing on I/O throughput. Reference [7] presents the reduction of the virtualization overhead and achieves the co-existence of performance and manageability through VM technologies. They are focused on I/O virtualization, designing an experimental VM-based computing framework, and addressing performance issues at levels Of the system software stack. They propose high performance VM migration with Remote Direct Memory Access (RDMA), which drastically reduces the VM management cost. Based on these references we want to test the CPU performance between virtual machines in different nodes, the network performance while a controller failure occurs and the impact of this failure During the file transferation.

III. Software Installed for the Experimental Phase

The system that we want to test is not complex and expensive, thus it is not a persistent system, nevertheless this is not a problem for our tests.

We are using 4 PCs with x86 architecture, each of them has 2 GB RAM and 2 intefaces Gigabit. Three computers are the nodes of cluster and the forth is used as a shared storage. We have used a computer as a shared storage because we couldn`t find a real one. This is associated with 2 problems, the first one is the decrease of the performance compared to SAN (Storage Area Network) and the second is that we have no backup because our shared storage can`t offer RAID technology over data since we are using a single disk. Nevertheless, security and backup are not our aims. All the nodes can communicate together by a Gigabit Switch which offers a good performance. To create cluster nodes can exchange UDP packets between them. This is a condition of the Heartbeat software. Thus we create a second LAN which is dedicated only for the communication between nodes.

At first we have configured ***common disk***. In this common disk we can storage the images of different Virtual Machines in order to access it from all nodes. We have used the forth computer for this purpose and have configured iSCSI protocol. It is known as a target computer and the third remaining computer are known as initiators. Previously we have configured the target computer and then the nodes. We have used CentOS 5.6 which includes iSCSI protocol. While configuring iSCSI protocol we have assigned the path of the fourth computer's hard disk as a shared storage. We have assigned this computer as a server by using the command ***yast2 iscsi-server***. Than we have tested the configuration by using the command **cat/proc/net/iet/volume**. It must give as a reply the volume name of the hard disk. Afterward we have configured the other three nodes. In all the nodes we have installed CentOS 5.6. All the nodes are configured as client computers by using the command ***yast2 isci-client***. We can choose setup option service at the moment of computer's boot. Each computer has got it`s own class C IP. ***Heartbeat*** should be installed and configured in each of computer except the fourth one by using the command ***yast2 heartbeat*** . All the nodes can communicate with each other by dedicated lines using UDP protocol. The last configuration step is the copy configuration of Heartbeat file to all nodes by **editing /etc/hosts, /usr/lib/Heartbeat/ha_propogate and /etc/init.d/heartbeat start**.

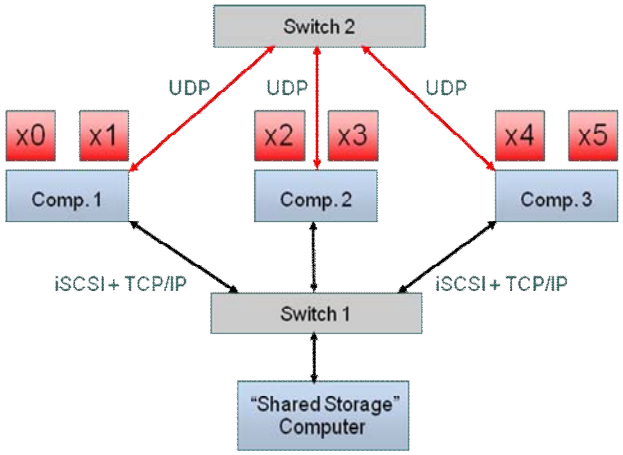

Fig.1 : The graphic of network

To configure and install Xen we have used **yast2 xen** command and after that we have setup 2 virtual machines above each host. The first virtual machine is a WindowsXp and the second one is Ubuntu 10.10 Desktop. Each virtual machine is configured by it`s own parameters.

IV. The Implementation of a Script

We have evaluated only the tests according to the controlled failure. We have evaluated: the CPU performance in Virtual Machines, the effect of a controlled failure in the network performance. Below is given an algorithm on how Heartbeat works:

1. Shutdown node 0, this is a controlled failure.
2. At this moment heartbeat notices all the nodes around for this situation and presents the virtual machine that has failed.
3. In node 0 heartbeat executes/etc/init.d/x0 stop and/etc/ init.d/x1 stop, so the virtual machines x0 and x1 will stop.
4. Heartbeat chooses the host nodes where the virtual machine should be migrated.
5. Let`s say node 1 is setup to support x0 virtual machine and node 2 to support x1 virtual machine.
6. Heartbeat executes this script in node 1: /etc/init.d/x0 start and this in node 2: /etc/init.d/x1 start.

Live migration should take in consideration two factors:

a. Virtual Machines cannot stop, because if they stop their services become unavailable. They should live migrate from one node to another.

b. Heartbeat in all the virtual machines of the failed nodes executes a script ***init.d stop*** and waits to stop these machines.

We want to modify the init.d script to give the possibility of live migration between virtual machines. This script will return the stop execution code to the heartbeat while the virtual machines continue the execution. So heartbeat will suppose that the virtual machines are stopped, but in reality they work. It will execute ***init.d start*** script and assign the destination nodes. At this moment starts the live migration of virtual machines between nodes. At the start moment no virtual machine is executed so we should take in consideration the setup of virtual machines. We implement in C++ a start script by using the follow steps with virtual machine x0 as an example.

- X0 is executed on node 1.
- Send a broadcast messages and migrate x0 to node2.
- Start the migration phase.
- Locate xo to node 2.

This process supports just a command "MIGRATE VIRTUAL MACHINE". If we want to use this test widely it should be improved in reliability and time execution.

V. Experimental Phase

- CPU performance during the live migration.
- The evaluation of network performance during a physical node failure.

a) CPU performance by increasing and decreasing Memory Utilization

First we will evaluate the CPU performance. Thus we should test the CPU activities by increasing and decreasing the utilization of the memory in virtual machines. We are using two nodes. At the first one is located x0 vm which is the client computer with Ubuntu 10.10 desktop installed on it and in host 2 is located the x2 server computer with WinXP. In this computer we have installed XAMP 1.7 version. In apache 2.2.16 into the Web-server machine we have included memory_balloon.c module. This module will serve as a tool to increase and decrease the memory utilization of the virtual machines. Previously we are located at **/etc/ apache2/apache.conf** , then we have to install a tool: **tool-sin apxs2** and compile it by command **apxs2-c-l-a memory_balloon.c**. We can configure test file in **/etc/ apache2/httpd.conf** as multithreading process.

For each call of the increase-handler function, the memory of Server increases with 10 MB. The IP address of x0 is 192.168.1.1 and of x2 is 192.168.1.2. In x2 is installed XAMP. From x0 we call 5 times the x2 virtual machine by using the command http://192. 168. 1.2/increase_mem of benchmark Httperf so the memory used by x2 will increase up to 50 MB. As a sample we are using a script in C which multiplies 2 square matrices with dimensions 5. So this benchmark should manage the CPU consumption in the Physical host during the dynamic utilization of memory. In the same way if we call http://192.168.1.2/decreas_mem , the virtual memory in x2 would decrease with 10 MB per time. We evaluate Response time by MemAccess Benchmark. We have presented all the results in table 1:

Table 1: Communication between 2 virtual machines in Different hosts

Memory Utilization in Appache DomU2	Response time	CPU Consuming
10 MB	0,046 ms	44 %
20 MB	0,050 ms	44 %
30 MB	0,067 ms	44 %
40 MB	0,099 ms	44 %
50 MB	0,141 ms	44 %

From table 1 we take Response time and CPU consuming when memory in apache web server increases from 10 MB to 50 MB. It seems that CPU consuming is fixed to 44 % and response time grows slightly in a linear way. If we repeat again the experiments with physical nodes in network we will get other results, presented in table 2.

Table 2 : Communication between 2 physical hosts

Memory Utilization in Appache DomU2	Response time	CPU Consuming
10 MB	0,022 ms	41 %
20 MB	0,036 ms	41 %
30 MB	0,047 ms	41 %
40 MB	0,081 ms	41 %
50 MB	0,110 ms	41 %

If we compare table 1 and table 2 we can observe that the response time in the second case is slightly smaller because the hypervisor introduces an additive time. This additive time corresponds to the CPU consuming. In table 2 CPU consuming is just 41% (41%<44 %). because of the absence of hypervisor.

b) Evaluation of the network performance during a failure

The main aim of Xen during live migration of virtual machine from one node to another one is to minimize the interrupted services. Referred to the above algorithm, initially the virtual machine is available in the source node, but it can decrease the network performance because it utilizes some bandwidth. Then, the virtual machine crosses over the network, so it is not available. Finally the virtual machine is executed in destination node. To test the network performance we are referred to figure 1. Initially we will test the response time before, during and after the migration from node 1 to node 2. So from the shared storage computer we will send the ICMP packets towards virtual machines by using the command: **ping –c 3000 –I 0.01 x0**.

In 30 seconds we will transmit 3000 ICMP packets. After this command, we will evaluate the round trip time from the virtual machine to the shared storage machine and vice versa. During 30 seconds of experiment we want to simulate the controlled failure of x0 virtual machine. This failure will give the possibility of migration from node1 to node2. The failure will be accomplished after the execution of a script in **/etc/init.d/heartbeat stop**. So we will evaluate the network response time during the migration as shown in table 3. All the results are evaluated by some network tools but most significantly by Net-Flow and PacketTrap tool.

Table 3 : The affect of a controlled failure in the round trip time, when we have modified the script.

Time duration	Round Trip Time for packet stream
0-18 sec: Time before x0 virtual machine failure	0,25 ms
18-21 sec: During this time has happened the failure	1 ms
21-30 sec: The failure h as finished	0,5 ms down to 0,25 ms

Table 4 : The affect of a controlled failure in the round trip time, before modifying the script.

Time duration	Round Trip Time for packet stream
0-18 sec: Time before x0 virtual machine failure	0,25 ms
18-21 sec: During this time has happened the failure	1,3 ms
21-30 sec: The failure h as finished	0,6 ms down to 0,25 ms

In the above tables we have presented the effect of live migration into the round trip time of ICMP packet. In the table 3 the migration process between sec 18 and 21 has affected in the increase of the response time to 1 ms. It has happened because of the controlled failure. The situation is localized in 0,5 ms after the sec 21. For inertia reasons it takes few time to stabilize from 0,5ms to 0,25 ms. Moreover if we analyze tables 4, the round trip time increases to 1,3 ms. The reason is that the modified script offers a better performance.

VI. Conclusion and Future Work

The performance of live migration between different physical hosts can be improved significantly if nodes don't stop their services. It can be accomplished by including a script on the heartbeat tool. The time duration of live migration from one physical host to another one is very important and it should be as small as possible. If we refer to table 3 and table 4 we conclude that the inclusion of our script in heartbeat tool decreases the round trip time when a controlled failure occurs from 1,3 ms to 1 ms.

The response time presented in table 2 is slightly smaller than that presented in table 1. As we

mentioned above the reason is that the inclusion of the hypervisor introduces an additive time. This additive time corresponds to the CPU consuming. In table 2 CPU consuming is just 41% (41% < 44 %) because of the absence of hypervisor that means our script has improved the CPU utilization. In the future we want to test the performance of CPU utilization, Memory Utilization with Pre-copy and Post- Copy [8] iteration in the WAN.

References References Referencias

1. www.cc.iitd.ernet.in/misc/cloud/hypervisor_performance.pdf
2. "Diwaker Gupta1, Ludmila Cherkasova2, Rob Gardner2, and Amin Vahdat University of California, an Diego, CA 92122, USA Hewlett-Packard Laboratories" Enforcing Performance Isolation across Virtual Machines in Xen.
3. "Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield_ Department of Computer Science The University of British Columbia", Remus: High Availability via Asynchronous Virtual Machine Replication.
4. "Asim Kadav" Live Migration of Direct-Access Devices.
5. "Wenchao Cui, Dianfu Ma, Tianyu Wo, Qin Li School of Computer Science and Engineering Beihang University Beijing 100191, China" Enhancing Reliability for Virtual Machines via Continual Migration.
6. "Nathan Regola" Center for Research Computing 111 ITC Bldg, Univ. of Notre Dame Notre Dame, IN 46556 Jean-Christophe Ducom Center for Research Computing 124 ITC Bldg, Univ. of Notre Dame Notre Dame, IN 46556.
7. "Wei Huang, M.Sc (Tech)" High Performance Network I/O in virtual machines over modern interconnects "Qiang Li1,2, Qinfen Hao1, Limin Xiao1, Zhoujun Li11 School of Computer Science, Beihang University, Beijing, 1000832 School of Mathematics and Computer Science, Hunan Normal University, Changsha, 410081" VM-based Architecture for Network Monitoring and Analysis.
8. Michael R. Hines, Umesh Deshpande, Kartik Gopalan, "Post-Copy Live Migration of Virtual Machines"

This page is intentionally left blank