7-14-2010

Live Migration Of Parallel Applications

Raul Fabian Romero
Purdue University, rromero@purdue.edu

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Fabian Romero

Entitled Live Migration of Parallel Applications

For the degree of Master of Science

Is approved by the final examining committee:

Thomas J. Hacker
Chair

John A. Springer

Eric T. Matson

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Thomas J. Hacker

Approved by: Gary R. Bertoline 07/09/2010
Head of the Graduate Program Date

Graduate School Form 20
(Revised 1/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:
Live Migration of Parallel Applications

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Teaching, Research, and Outreach Policy on Research Misconduct (VIII.3.1)*, October 1, 2008.*

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Fabian Romero
Printed Name and Signature of Candidate

07/13/2010
Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/viii_3_1.html

LIVE MIGRATION OF PARALLEL APPLICATIONS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Fabian Romero

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

August 2010

Purdue University

West Lafayette, Indiana

To my wife, parents, and sisters for their encouragement, motivation, and support that greatly inspired me to achieve my academic goals.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Romero, Raul F. M.S., Purdue University, August, 2010. Live Migration of Parallel Applications. Major Professor: Thomas J. Hacker.

It has been observed on engineering and scientific data centers that the absence of a clear separation between software and hardware can severely affect parallel applications. Applications that run across several nodes tend to be greatly affected because a single computational failure present in one of the nodes often leads the entire application to produce incorrect results or to even die. This low observed reliability requires a combination of a proactive and reactive solution in order to preserve the state of parallel jobs running on degraded nodes; therefore it is possible to avoid runtime errors in parallel applications.

This thesis addressed the critical problem of low reliability in parallel jobs by implementing a fault tolerance approach based on OpenVZ virtualization. By using virtual machines on which parallel applications were running, this study showed that it was feasible to make parallel jobs independent of any particular hardware/software implementation; therefore when a degraded node is detected, the virtual machine(s) running on this degraded node(s) may be migrated with its parallel jobs to a healthier node. This study examined the correctness and performance of implementing live migration on hosts loaded with parallel jobs, and determined that it is possible to efficiently save the state of parallel applications after live migration of virtual machines to a more reliable node.

CHAPTER 1. INTRODUCTION

For many years computational systems were unable to meet many of the computational power requirements for scientific and engineering applications, resulting in tremendous delays in obtaining the expected results of important calculations. High performance computing systems came to the rescue providing enormous amounts of power; but a disadvantage was as the number of computational components increased, the mean time to failure decreased, resulting in a poor reliability (Hacker, Romero, Carothers, 2009). As parallel applications run across several computational nodes, the potential for a failure in these distributed programs is even more likely. This reality became the inspiration of this study by motivating an approach based on virtual live migration to move parallel processes from degraded to healthier hosts.

1.1. Background

After joining together many computing nodes to serve as a single large computing system, high performance computers were able to satisfy many of the scientific and engineering requirements, providing the means to obtain faster and precise solutions to the complex calculations submitted by scientists and engineers. Nonetheless, this large collection of computers, usually spread around several locations, tend to experience a very high rate of component failure, which hardly impacts the calculations of large parallel applications.

For this thesis, the use of an operating system-level virtualization environment *OpenVZ* was investigated to perform live migration of containers or virtual machines (VMs), on which multi-processor parallel applications were running. By using this virtual technology (OpenVZ) this thesis focused on investigating a way to help parallel jobs succeed when a hardware failure is detected on the system, based on the following list of objectives:

1. Validate the correctness implications of the live migration of parallel jobs.
2. Measure the performance of live migration.
3. Verify the feasibility of implementing live migration on MPI based systems.
4. Test the speed-viability relationship after implementing multiple live migrations of parallel applications.
5. Measure the efficiency of implementing multiple live migrations of virtual machines to keep applications running on the most reliable nodes.

By the time this study was written, many reactive solutions such as checkpoint/restart were present in the market, offering expensive high availability implementations to recover systems in the event of a component failure. Nonetheless, HPC systems may fail several times per hour, making reactive solutions somehow ineffective to satisfy the computational needs on time to save the state of critical applications; due to the high rate of failure occurrences.

For this reason, Hacker et al's (2009) prediction model was complementary to this study, because their study provided a solution to identify nodes under risk of failure. Therefore, by detecting a degraded node on time and by taking actions to save the state of its computations, it was possible to find a way to improve the chances of success for critical scientific jobs in order to satisfy the expectations of the scientific community.

This thesis investigated and demonstrated how virtualization technology can greatly improve the chances of success of scientific and engineering applications by using migration of virtual machines without turning down the system. To do this, the study used a *Linux* based virtualization technology started by *SWsoft* (the company that owns the commercial virtualization software Virtuozzo) *called OpenVZ*. It is a light and flexible paravirtualization software that the study demonstrated to work well with parallel applications. One of the benefits of using this operating system (OS) based virtualization technology was that it did not require a dedicated allocation of memory (RAM) and there was only one kernel installed in the physical machine, avoiding unnecessary layers. When there are fewer layers for the data to go through, this means that it is processed through fewer cycles, avoiding unnecessary steps and improving processing speed (Fischer & Mitasch, 2006).

The basic architecture of OpenVZ is shown in figure 1.1, consisting of only two added OpenVZ layers when compared with traditional no-virtual systems. As shown in this figure, the containers (VMs) were created on top of an OpenVZ template and can be spread among many hosts.

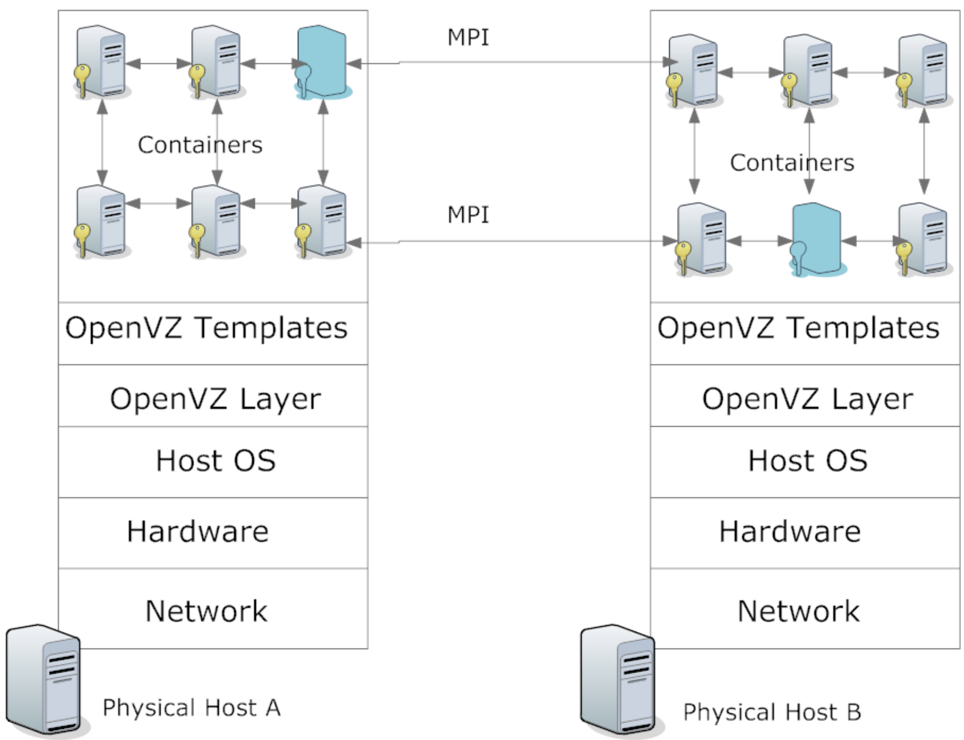

Figure 1.1 OpenVZ-Based Experimental Architecture.

To install and run this proactive virtualization approach the project used four servers: Two *Dell 1950* with 64-bit 8-Core Intel Xeon processors, and two HP Proliant DL-165C with 8-core Quad-Core AMD Opteron. The network fabrics used were based on a *10-Force* network gigabit switch, and high speed, low latency *Myricom* 10Gb/s switch. The virtualization software *OpenVZ* was installed over a *CentOS 4.5* OS, and 30GB-based virtual machines (VM) were then created on top of the virtualization kernel. Over this structure the VMs were configured to work together, exchanging parallel messages with MPI, as showed above in figure 1.1. Later, there were created two experimental scenarios, one based on High Performance Linpack (HPL), which was compiled and set up on one system to simulate a parallel work while running the migration tests, and OMEN, a parallel application software that was compiled and implemented on another system to assess and compare the live-migration behavior of different parallel environments.

1.2. Significance

This study found that the stability of scientific and engineering applications was highly dependent on the performance of large scale computing systems, therefore it is important to avoid any hardware related failure that might affect the applications. Unfortunately, as the number of components increases in this large scale computing system, the mean time to failure decreases due to the increment of components susceptible to fail (Hacker et. al. 2009).

The approach presented in this thesis addressed this reliability problem and explored the use of virtual live migration to move applications from a degraded node to a healthier one. By doing so it was possible to avoid incorrect termination of applications and kept them running independently from the hardware platform. As mentioned in the previous section, this study served as a complementary work to Hacker's study, on which a prediction model was explored in order to identify computational nodes under risk of failure. This prediction algorithm worked based on three computational statuses: *up*, *down*, and *degraded* where *up* meant that the host was alive and working normally, *down* was used for a dead or out of service host, and finally *degraded* status was used to refer to a still operational host that had been identified under risk of failure. The *degraded* state was of most interest for the purposes of this investigation due to the work presented based on the migration of VMs from degraded nodes, that is, this thesis explored a combination of a reactive with a proactive solution to save the state of parallel applications running on degraded computational node(s).

As an overview to the problem and proposed solution presented in this section, following are the most important factors regarding importance of counting with a better reliability for parallel applications:

- Critical large scale parallel applications should not rely on the health of a single hardware platform.
- Operating systems and its applications should have the ability to transfer to reliable nodes when a failure is predicted.
- Concurrent migration of several VMs can help preserve the state of the entire parallel application by running parallel jobs on the most reliable hosts only.

1.3. Scope

For this study the author based his assumptions on primary and secondary literature sources and the thesis followed a deductive scientific method on which hypothesis and theory were tested according to the principles of the quantitative research methodology.

This study intended to decrease the failure rate observed in MPI-based parallel applications, which are affected by their tight dependency on hardware platforms. The alternative provided to separate the enormous dependency of applications from hardware consisted upon exploring the use of virtual live migration to move parallel jobs out of affected hardware.

For the purposes of efficiently live migrating VMs running parallel jobs, this study investigated the operating system-level virtualization environment OpenVZ. The study focused on the correctness and performance of live migrating single and multiple VMs that were running parallel applications in multiprocessor systems and were communicating via MPI over TCP.

1.4. Personal Statement of Research Interest

The author planned to accomplish this thesis project in the field of High Performance Computing (HPC) for two purposes: first to put into practice his recently acquired technical and organizational skills, and second, to assemble an interesting solution for the scientific community that minimizes the poor reliability observed on HPC systems. He chose a Master thesis project as a means of challenging his project management skills while implementing an interesting and well organized project that involved the use of state of the art technologies.

1.5. Research Question

Is it possible to schedule parallel applications to run on the most reliable large-scale supercomputing system and reduce the rate of unsuccessful parallel jobs? By implementing a VM based live migration approach to move applications out from suspected failing nodes; would it be possible to overcome the low reliability observed in large spread parallel applications?

It is important to keep parallel applications up and running even if their current computational processors have been predicted to fail. The impact of computational hardware failures on parallel applications is tremendous, because usually in a system composed of 100 hosts, a single node failure can lead to the unsuccessful termination of a whole parallel application even if it had been running for months.

Consequently, while parallel applications evidence a very low mean time to failure, non-parallel applications that work on a single computational host have a larger mean time to failure. In order to understand this high impact and high probability of failure in parallel application, assume that there is a data center with 20 nodes and 19 of them are predicted to fail (as an example for incorrect environmental conditions), in this instance, there would be at most 1/20 probability of successfully completing the application. Of course, for this successful case to be possible, the application would have to run only in the working or *up* system. On the other hand, if we ran a parallel application on the same 20 hosts-based data center and only a single host was predicted to fail, the entire parallel application would at risk for incorrect termination if the jobs in the failing host cannot be moved. As demonstrated, the chances of the failure parallel application are high, and this is not good for the scientific community that deserves strong reliable systems.

From a hardware point of view, when the reliability of the computational hardware is assessed, it has been observed on supercomputer systems that there is a significant cost behind large amounts of processing power. The probability of experiencing a component or node failure on a HPC cannot be compared with the probability of a standalone system failure. If a personal computer on average is affected once every three years, an HPC system with thousands of nodes might fail several times per day (Liang, Zhang, Xiong, & Sahoo, 2007). According to this comparison, applications running on large computing systems have a high probability of failure, and therefore it is important to discover a solution that helps minimize the impact of unexpected component failure.

1.6. Assumptions

The assumptions inherent to the study are:

1. Based on reported UNIX computing logs, many component failures can be predicted before any catastrophic event takes place.
2. This study assumed that a prediction algorithm has been implemented in a large scale supercomputing system in order to identify functioning nodes that were at risk for failure.
3. Because a node was predicted to fail, there were enough computer resources available to fulfill the parallel application requirements on the target host.

4. Because this investigation was interested in assessing the efficiency of live migration during the execution time of a parallel application, the study took into consideration the total runtime of the parallel application plus the total time to pursue the live migrations.
5. The two parallel applications used for this study (High Performance Linpack and OMEN) effectively manipulated MPI messages and worked flawlessly over an OS-based virtualized environment.

1.7. Limitations

The limitations for the study of reducing the fault rate experienced by parallel applications included:

1. Accuracy of HPL and OMEN to execute parallel calculations.
2. Performance of migrating operating system-level virtualization environments based on OpenVZ.
3. Accuracy of UNIX based time managers to calculate the total time involved in migrating VMs and completing a parallel application.

1.8. Delimitations

The delimitations for the study of reducing the fault rate experienced by parallel applications included:

1. Computing resources available in the High Performance Computing laboratory of the Computer and Information Technology department of Purdue University, during the period of Fall 2009 to Spring 2010.
2. Four computational servers that were configured to work with the operating system-level virtualization environment OpenVZ.

3. Examination of only two different types of parallel applications: HPL (Dongarra, Bunch & Stewart, 2009) and OMEN (Klimeck & Luisier, 2010).
4. The creation of two similar virtualization environments to assess the correctness, performance, and reliability implications of the proposed approach with HPL and OMEN.
5. Configuration of two network paths for the virtual environments: a 1Gb/s Ethernet network and a Myricom 10Gb/s SR fiber network.

1.9. Definitions of Key Terms

Checkpoint – This is a position in the log that indicates a point at which all filesystem structures are stable and consistent. After all modified information, including the index block, data blocks and so on is written to the log, the system writes a checkpoint region to a fixed block on disk. This information is useful at start up time and particularly after a system failure. (Preston, 1999).

Correctness – This term is used in computer science with respect to an algorithm to say that the algorithm behavior and output is free of errors so it is correct with respect to a specification.

Cyberinfrastructure - This is a term originated by the National Science Foundation (NSF) to describe the information technology resources used by researchers, clinicians, engineers and artists to create new knowledge. (Solomonides, 2008).

Grid Computing – This new technology emerged in the late 90's, underpins distributed problem-solving solution. Research sharing, coordinating problem solving and dynamic multi-institution are basic characteristics of grid computing. (Jin, Pan, Xiao & Sun, 2004).

Parallel Computing - A form of computation in which many computations are carried out simultaneously (Almasi & Gottlieb, 1989).

Paravirtualization – This is a type of virtualization in which the underlying operating system is modified to provide virtualization capabilities.

Virtualization: Refers to one piece of hardware running multiple kernels on top of a lower layer of software that manages their access to the hardware. Each kernel, called a guest, acts as if it has the whole processor to itself. (Adelstein & Lubanovic, 2007).

1.10. Summary

Chapter 1 is an overview of the fundamentals concepts of this study. It provides a general explanation of the problem as well as the scope, assumptions, limitations, and delimitations of the expected solution. The next chapter provides an overview of related work, and uncovers certain aspects of the literature that contributed to the creation of this research project.

CHAPTER 2. REVIEW OF RELEVANT LITERATURE

2.1. Approach to This Review

The following literature review is based on primary and secondary literature sources that contributed to the construction of this document and served as a point of reference to find contributing or contradictory ideas.

2.2. Related Work

Previous studies in this field noted the importance of improving reliability for High Performance Computing (HPC) systems. For instance, Liang et al. (2006) performed a similar study about failure prediction on an IBM BlueGene/L supercomputer system. They provided three failure prediction models that operated based on distinguishable classifications of the hardware component, which raised a flag after detecting a failure. It is certainly an impressive approach that draws attention to the efforts made on regulating the low reliability of a large computing system (*IMB BlueGene* supercomputer). Liang et at.(2005) authored another motivating paper on the same topic that discussed the failure behavior observed in a large computing system. They provided several models and methodologies that certainly helped to distil the most relevant error messages over a large collection of events. These works contributed to this research by providing different methods to identify or predict computational hosts in degraded state, which is essential before implementing the proposed migration approach.

Other research analyzed the benefits of implementing checkpoint and restore, intended to save the state of the machine for prevention or recuperation purposes. The paper developed by Hacker et al. (2007) had a structure based on mathematical models, where there was a useful investigation about checkpoint/restore and queue structures functionalities. This study served as a point of departure for this thesis by means of the reactive strategies that were useful for supercomputer systems.

Virtual machines have the potential to increase in popularity because of its uneven number of advantages and disadvantages. As proof of this potential, Fischer and Mitasch (2006) proposed two OpenVZ-based virtual environments that consisted on a virtual/virtual scenario executed over multiple physical machines, and a physical/physical with fail/switchover of virtual machines. Even though their experiment was focused on High Availability (HA), the proposed approach served as the starting point of the expected utilization of VMs to improve the low reliability of HPC systems. By implementing the virtual/virtual scenario shown in figure 2.1, they sought to eliminate the existence of a single processing node by allowing another host to hold another virtual machine (Fischer & Mitasch, 2006).

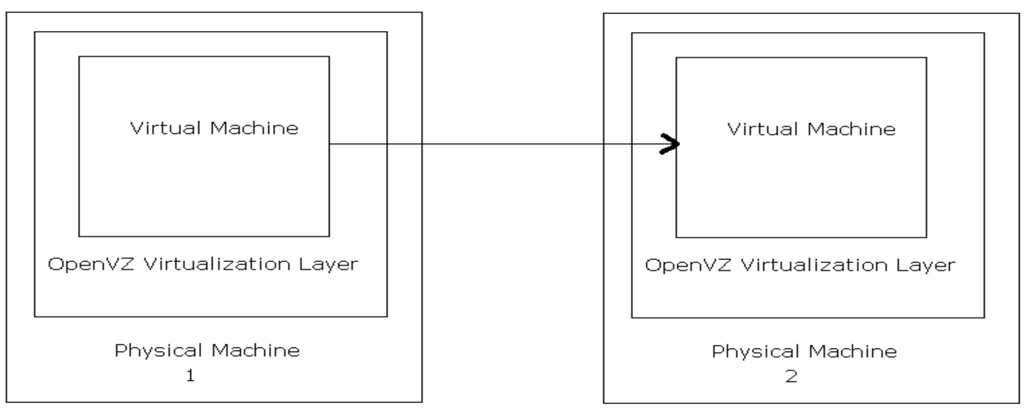

Figure 2.1 Virtual/Virtual scenario (Fischer & Mitasch, 2006)

The other scenario can also be highlighted for its contributions to this study. This scenario was based on a physical/physical with fail/switchover configuration as shown in figure 2.2. The experiment this time consisted in the migration of all the VMs contained in a single physical machine to a different physical server. This is especially important to this investigation because the authors of the scenario also contemplated the possibility of a catastrophic failure affecting a whole computing system. Fischer and company claimed that if a failure can be pro-actively predicted, several virtual machines running in a single physical machine can also be migrated to a healthier machine. Their work differed from this study in that they did not contemplate exclusively live migrations of VMs, nor did they assess the impact of virtualization on parallel applications.

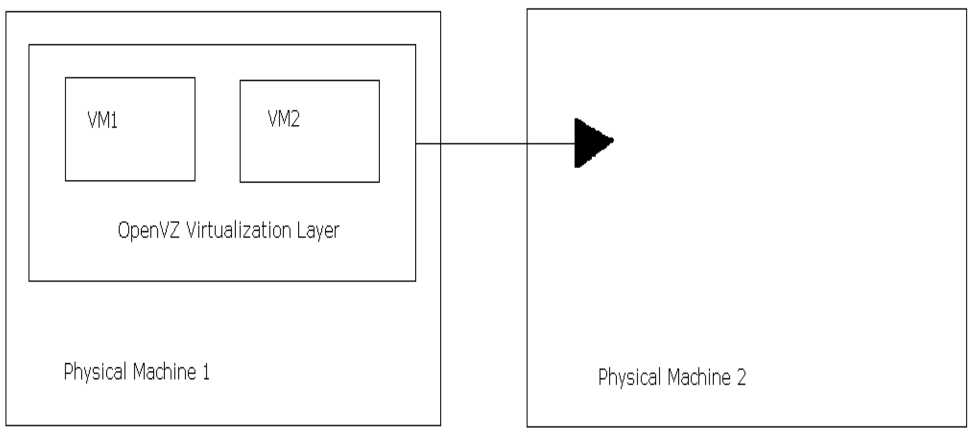

Figure 2.2 Physical/Physical with Fail/Switchover (Fischer & Mitasch, 2006).

Recall that the main objective of this work is to assess a live migration approach to improve the low reliability observed in parallel systems. Another study ran similar tests on live migration of virtual machines, Clark et al. (2005) focused on cluster environments and observed the importance of separating hardware from software. This enabled administrators to remove a physical machine from service (including applications running) by transferring its load to another physical host. Clark et al. (2005) claimed that their approach left the original machine free and ready for maintenance purposes. Their study differed from this thesis in that it highlighted the availability of using Xen VMM virtualization software to significantly improve manageability instead of OpenVZ. They accomplished a minimum service downtime of 60ms by carrying out live migration over *Quake 3* on a commodity cluster; and 210ms over the SPECweb benchmark. Their calculations were made with two physical machines connected through a high performance communication switch.

The approach of Clark et al. (2005) also differed from this study in that it did not assess the performance of different network fabrics nor did it use different interactive applications (other than OMEN and HPL) to test the performance of live migration. The study also concluded that the performance made live migration a practical tool “even for servers running interactive loads” (Clark et al., 2005). Finally, there was a difference in that a paravirtualized tool (Xen) does not allow free efficient resource usage and density. Xen required fixed memory and disk definitions (Xen has hard, fixed caps), while *OpenVZ* had burstable memory usage, which made it possible to subscribe more users to a server running on top of OpenVZ.

2.3. Summary

Chapter 2 is a collection of several related literature sources that influenced the flow of this research. Some of the sources served as a notable point of reference for comparative purposes and/or better comprehension of the problem. The next chapter provides the fundamental framework and methodologies used to implement the proposed virtual live migration approach and list the most important details about the data collection process.

CHAPTER 3. FRAMEWORK AND METHODOLOGY

3.1. Theoretical Framework

This section provides details about the research background and elements used to build the experimental environments of this project, such as methodology, data, variables, and population among others.

3.1.1. Approach to Research and Methodology

This study followed a quantitative research approach by systematically investigating the properties and phenomena of live migration of virtual machines on which multi-processor parallel applications were running. A correlational quantitative research was conducted to determine if a relationship existed between the quantifiable variables that influenced the performance, correctness, and reliability of live migration and to what degree.

The methodology employed to run the measurements was based on two varying parallel scenarios that exchanged MPI messages. As show in figure 3.1, the first scenario was based on High Performance Linpack (HPL) and the second (not shown in this figure) was based on OMEN parallel application. For both of these parallel scenarios the same tests were performed in order to have comparable results. The tests executed under each of these parallel scenarios were separated into two groups or virtual networked environments, configured to communicate over one of two network paths: a Gigabit Ethernet or a 10 Gb/sec network fabric.

The idea of conducting these experiments over two different networks was to assess and compare the effects of network bandwidth and latency during the execution of parallel applications and the live migration of parallel jobs.

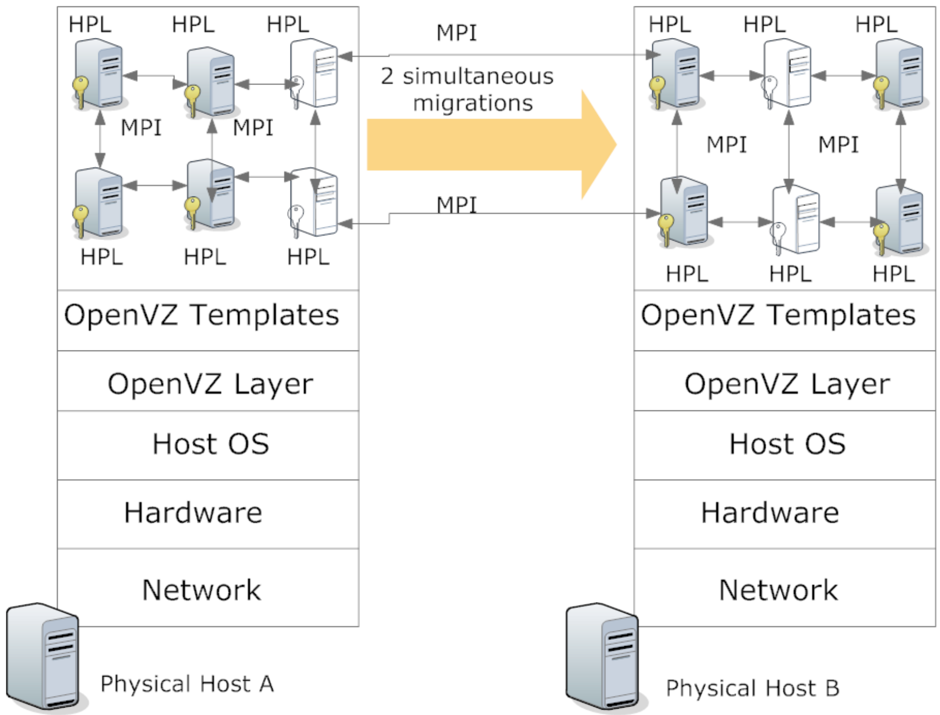

Figure 3.1 MPI-Based Connection and Transmission of Two VMS

The series of experiments conducted to understand the effects of live migration on parallel applications and the ability to complete without errors, consisted in varying the following four major variables:

- Type of parallel application or benchmark (HPL or OMEN).
- Number of VMs allocated to the MPI parallel application or benchmark, ranging from 2 to 11 virtual containers.
- Network fabric. A series of experiments were conducted over the 1Gb/sec, and then over the 10Gb/sec networks.
- Number of simultaneous live migrations performed during the runtime of the parallel application. The migration options were 1,2,4, or 8 simultaneous live migrations, and for each, a total of 8 migration cycles were invoked to be migrated, independently from the number of simultaneous migrations started.

To validate the correct completion at the parallel applications, the study observed and assessed the final results of the computations upon termination. The results were compared in experimental trials with and without live migrations running. After the calculations were completed in both experimental scenarios, the observed results were always the same, therefore it was possible to determine that every job with OMEN and HPL were successfully completed with the same results.

3.1.2. Hypothesis

Because the high rate of failure occurrences on large scale supercomputing systems affects the behavior of applications (Hacker, 2007) and the traditional reactive approach might work together with a proactive approach to overcome this problem, the null and alternative hypothesis that this study wanted to address were the following:

H_o: Multiple live migration of VMs on which parallel applications are running does NOT reduce the fault rate experienced by parallel applications.

Ha: Multiple live migration of VMs on which parallel applications are running reduces the fault rate experienced by parallel applications.

If the hypothesis H_o is rejected, eventually the effect of large scale failure occurrences on parallel applications can be reduced by implementing virtual live migrations. The results will be examined in chapter 4 using values obtained from different networks and from experiments based on HPL and OMEN. Eventually, hypothesis H_o will be rejected based on the results, which demonstrated that it is possible to reduce the fault rate of parallel applications.

3.2. Method

The following topics provide details about the information sources and the manipulation of data.

3.2.1. Population

The population consists of a collection of 34 *OpenVZ*-based virtual containers loaded with *CentOS 5* and running a parallel application. These virtual machines (VMs) were distributed among four servers and communicated using *MPI* over *TCP*.

3.2.2. Sample

The sample of interest corresponds to two sets of 17 virtual machines that were specifically created for each of the two virtual environments. Each virtual environment is based on a different parallel application, and was configured to communicate over both Gigabit Ethernet network, and 10 Gb/s network.

3.2.3. Data Collection

Based on the script included on appendix C, all the total live migration times (seconds) were collected and added to a spreadsheet which simultaneously included and excluded, parallel computations running over different scenarios. Depending upon the total number of virtual machines (processors) sequentially allocated to calculate the benchmark (*HPL* or *OMEN*), the total execution time of completing 8 migration cycles during the runtime of the parallel application was recorded.

3.2.4. Data Instruments

The first data instruments were a series of BASH shell scripts that were written to automatically start the timer, launch the migrations, run the parallel calculations, and allocate VMs to the appropriate MPI based communication environment.

Microsoft Excel (2007) and the R statistical software (Dalgaard, 2008), were also used to plot and create bar graphs to reflect the correlation and differences observed from the collected data.

3.2.5. Data Analysis

The correlation between the number of simultaneous migrations and the total time to migrate while executing the parallel calculations is illustrated by measuring the direction and strength of the linear relationships among the series of experiments. To better understand the behavior and correlation of the collected data, bar chart diagrams were used along the phases of this investigation which effectively plotted the total execution time of the live migrated parallel application. Additionally, bar charts were used to illustrate the standard deviation and identify outliers in the dataset.

3.3. Timeline and Dates

Below is a timeline framework that describes work distributed among three academic semesters commencing Summer 2009:

- Five months to configure the virtualization environments, investigate related work and outline the first three chapters of the thesis.
- Five months to run the experiments and collect results after repeating each test a minimum of three times.

- Three months to classify, identify patterns in data, execute statistical analysis, and write respective technical and analytical chapters of the thesis.
- Three months to get final conclusions, and present/defend the thesis.

3.4. Variables

The independent variable that was manipulated along this study was the “number of virtual machines”, which were sequentially allocated to each experimental trial (each virtual machine was configured to use only one processor). This independent variable consisted of a range of 2 to 11 VMs that were invoked for each series of live migrations. Additionally, the dependent variable that was influenced by the number of virtual machines invoked in each test was the “parallel application execution time” which consisted of a measurement in seconds of the total time for each experimental trial to perform 8 live migration cycles during the runtime of the parallel benchmark.

CHAPTER 4. DATA ANALYSIS

This chapter presents the findings for different metrics used to evaluate the correctness and performance of live migrating parallel applications. It further presents the summary of the differences observed among the virtual environments.

4.1. Correctness

The parallel programs used in this study, HPL and OMEN, must be able to successfully complete their operations after some of the parallel jobs have been live migrated to a different node. Correctness was measured here, using the output of each of the parallel programs that reported the ending status of its expected arithmetic or physical calculations.

The HPL script *xhpl* worked based on a configuration file called HPL.dat, which contained the configuration parameters of the HPL arithmetic calculations. Among this parameters, it was possible to specify an output file to automatically generate final status reports after concluding the linear calculations. Therefore and as shown in table 4.1, it was possible to verify that no errors were reported during the arithmetic calculations for all experiments conducted with or without running virtual migrations.

Table 4.1

HPL finished 48 tests

48	Tests completed and passed residual checks
0	Tests completed and fail residual checks
0	Tests skipped because of illegal input values

The results observed in table 4.1 are a good example of what was observed during the execution of HPL based experiments. In this example 48 tests of linear arithmetic operations were conducted and once finished, HPL reported all tests were completed successfully. No tests were skipped because of illegal input values, as showed in the HPL output example.

For the experiments conducted with OMEN, this study followed a different approach than the one previously described for HPL. Since OMEN did not automatically provide termination status as HPL did, in order to validate the correct execution of OMEN this study observed the number of successfully computational phases completed, which were reported in the OMEN output files. This number was then compared from the OMEN experiments were no migrations were executed and compared to the number reported when live migration was included in the OMEN experiments. All cases yielded the same number of successfully computational phases completed.

It is important to note that for each set of experiments (OMEN or HPL based) the state of the virtual containers was checked after live migration, and the author of this thesis concluded that each virtual machine successfully recovered from migration. This included verifying the memory state, network interfaces, operating system, file systems, and running applications. All of these resources worked properly in the destination node after conducting the live migrations.

4.2. Performance

This study focused on the performance of applying live migrations of virtual machines in containers running MPI-based parallel programs in order to measure the effectiveness of the live migration approach. This study interpreted performance as the successful termination of parallel calculations by a migrated computational host compared with the time and resources used. Performance was measured based on the number of processors involved in the parallel calculations and the total time taken to complete the parallel program (runtime) while live migrations were executed.

4.2.1. Number of Processors

This metric consisted of exploring the significance of impact by utilizing fewer or more virtual machines in the parallel calculations. For all experiments only one processor per VM was configured, therefore it was simpler to measure the statistical difference using the different numbers of VMs while simultaneously migrating some of them. By using this configuration this study sought to discover the performance impact of migrating a total of eight VMs during the parallel benchmark running time. The purpose of maintaining this constant number of eight migrations was to have the same point of reference for all experimental trials. Further in this study it was easier to compare the results obtained from all the tests that involved different number of VMs for parallel calculations, while migrating the same number of eight VMs during the total execution of the parallel benchmark.

4.2.2. Time to complete benchmark with migrations

This analysis explored the possibility of a statistically significantly difference in total time spent to execute the parallel tasks with and without migrations. This thesis implemented the same metrics under HPL and OMEN to measure the time to execute parallel jobs.

4.2.2.1. <u>HPL based</u>

Output was collected from each of the 240 HPL executions after running multiple live migrations with 1, 2, 4, and 8 simultaneous migrations; over the 1Gb/s and 10Gb/s networks. All the HPL computations were successfully completed as each *xhpl* application output confirmed. This means that HPL can tolerate OpenVZ based live migration regardless of the number of simultaneous migrations. The dataset corresponding to HPL-based figures can be found in appendix A.

Following the experimental methodology described in a previous section, HPL benchmark performance with and without simultaneous live migrations was assessed. The results of assessing the time to execute HPL with only one VM migrated at a time over eight migration cycles are shown in figure 4.1. In this figure, the first two of each set of five bars represent the total HPL execution time over the 10Gb/s and 1Gb/s networks respectively. For each category, a total of eight live migration cycles was conducted sequentially (one by one) during the total benchmark execution time. Each bar shows the average of three experimental trials, with error bars on top of each, representing the resulting standard deviation.

There was selected a number of three experimental repetitions due to the long time it took to complete each trial of the parallel application. The *HPL* runtime (without including live migration nor manipulation tasks) lasted about 15 minutes, and it was necessary to run it 30 times per group of experiments (1VM, 2VM, 4VM, and 8VM migrations) resulting in 120 *HPL* executions only over the 1Gb/s network, so a total of 240 times including the experiments conducted over both of the network fabrics. The same number of experiments were performed for the *OMEN* based experiments, with the difference that *OMEN*-based tests lasted about 30 minutes each one. Therefore, to have an estimate of the total time to run the experiments for this research in an ideal scenario when neither errors, migrations nor manipulation time were included, the entire phase of experiments lasted about 180 computational hours.

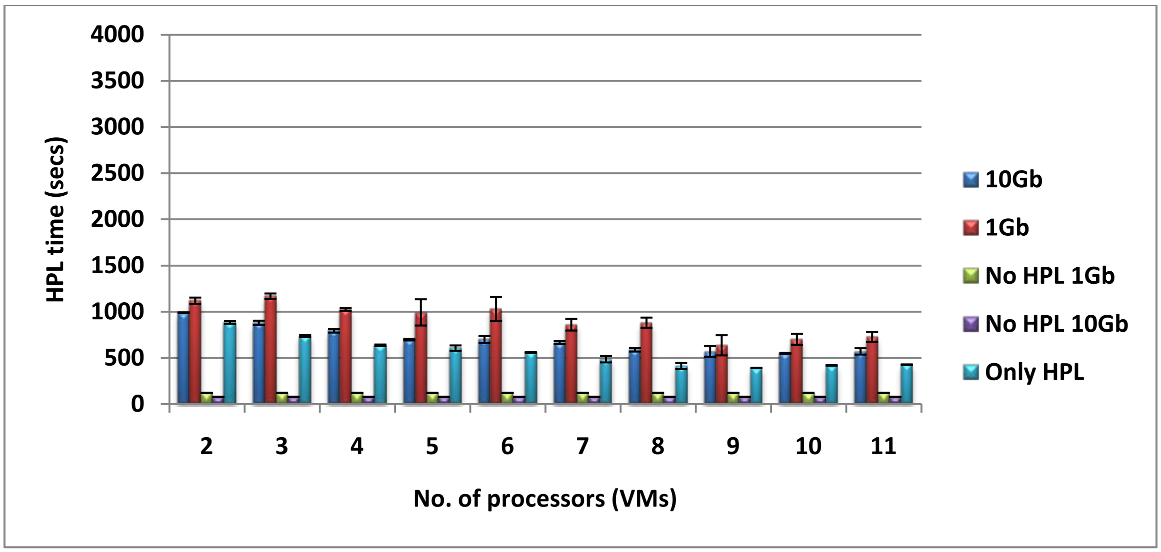

Figure 4.1 HPL -One VM Migrated over Eight Migration Cycles

As shown in figure 4.1, the third and fourth bars of each set correspond to *No HPL 1Gb/s* and *No HPL 10Gb/s* represent only the total time of eight sequential migration cycles. These bars, however, do not include the time to execute the parallel benchmark. The final category in each set of five bars is *Only HPL*, which represents the total time to execute HPL without running any live migration. The only variable progressively modified is the number of VMs that were used for the parallel computations ranking from 2 to 11 VMs. From this figure it is simpler to identify the total time required to move the VMs, which was significantly less than the total time to complete the parallel calculations.

Figure 4.2 shows the results of running HPL with two simultaneous live migrations over eight migration cycles. Note that in this figure the bars were not as linear as they were in the one migration based trial because, the HPL does not scale well with two simultaneous migrations. In general, the total execution performance of HPL over both of the networks was very poor.

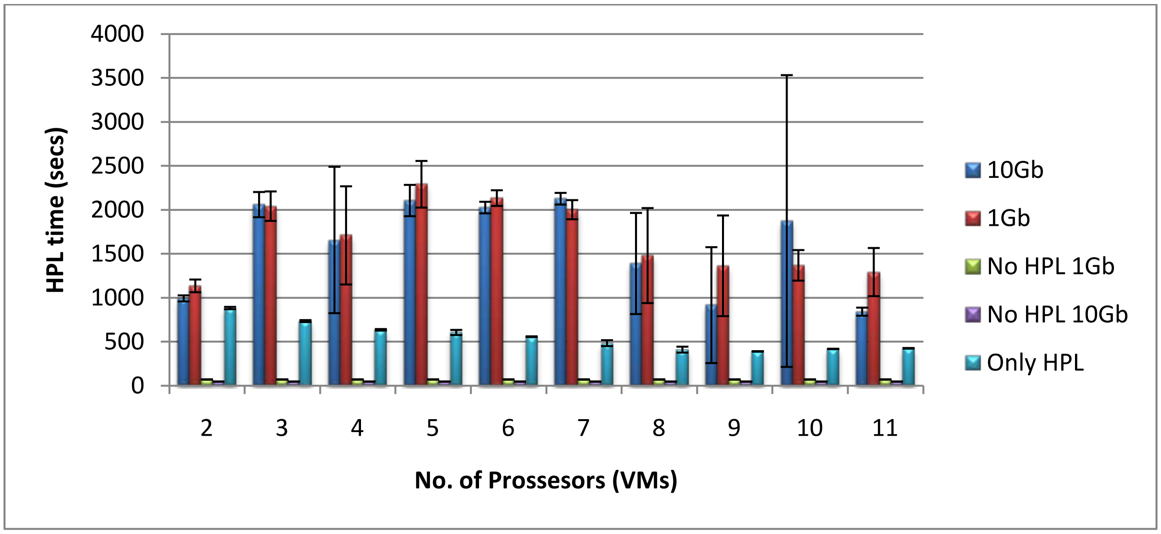

Figure 4.2 HPL -Two Simultaneous VMs Migrated over Eight Migration Cycles

Also note that in figure 4.2, it took 201.6% more time to complete the HPL calculations over the 10Gb/s network when using five processors than it was in the one migration experiments, which of course is not good to experience a prolonged running time. Even though the performance for the two migration based experiments decreased, HPL was able to complete successfully as confirmed by the output showed after completing the linear calculations. It is important to emphasize here that the number of live migration cycles remained the same, which was eight for all experiments in this study; however the only factor that indeed varied was the amount of simultaneously triggered migrations (out of eight).

Figure 4.3 shows that HPL execution time was very inconsistent with four simultaneous migrations, as was also the case for the two migration experiments showed in figure 4.2. At best, HPL running time decreased 32.6% for the four processors-based experiment compared with the single migration. At worst, HPL running time increased 247.8% for the five processors. Experiments were also conducted with eight simultaneous migrations that are not shown here but the HPL execution time was also unstable, showing again that the HPL performance was negatively affected by simultaneous VM live migrations.

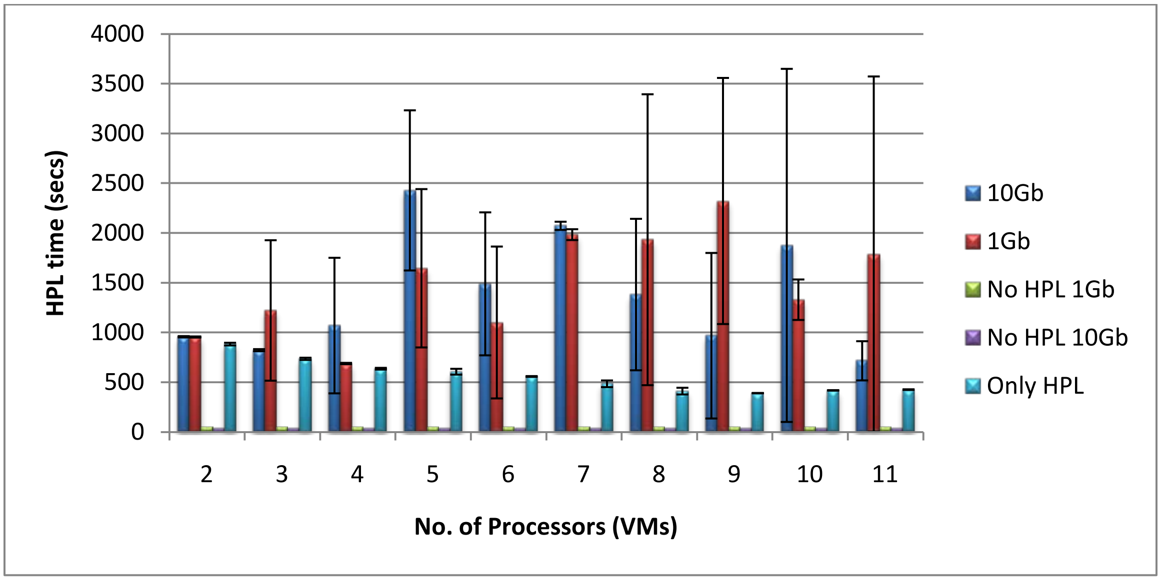

Figure 4.3 HPL -Four Simultaneous VMs Migrated over Eight Migration Cycles

To summarize the results, HPL worked well with single live migrations but did not scale well. The results observed from the two or more simultaneous live migrations, showed that the performance of HPL was significantly degraded.

4.2.2.2. <u>OMEN based</u>

Similar to the experiments conducted with HPL, this thesis tested the performance of the OMEN parallel application using MPICH2. Following the same methodology used for HPL, a Bash shell script program was developed in order to manipulate the live migrations. A copy of one of the versions of this script can be found in appendix C.

Figure 4.4 shows the results of the first execution of OMEN experiments, when migrating only one VM at a time over a cycle of eight live migrations. The categories were the same as those used for the HPL experiments with 10 (2 to 11) sets of five bars, starting with the total execution time when migrations were included over the 10Gb/s and 1Gb/s networks. The third and four bars represented the total time to live migrate one VM at a time out of eight migration cycles. Finally, the fifth bar of each set showed the total time to complete OMEN without executing VM live migration. The dataset corresponding to OMEN-based figures can be found in appendix B.

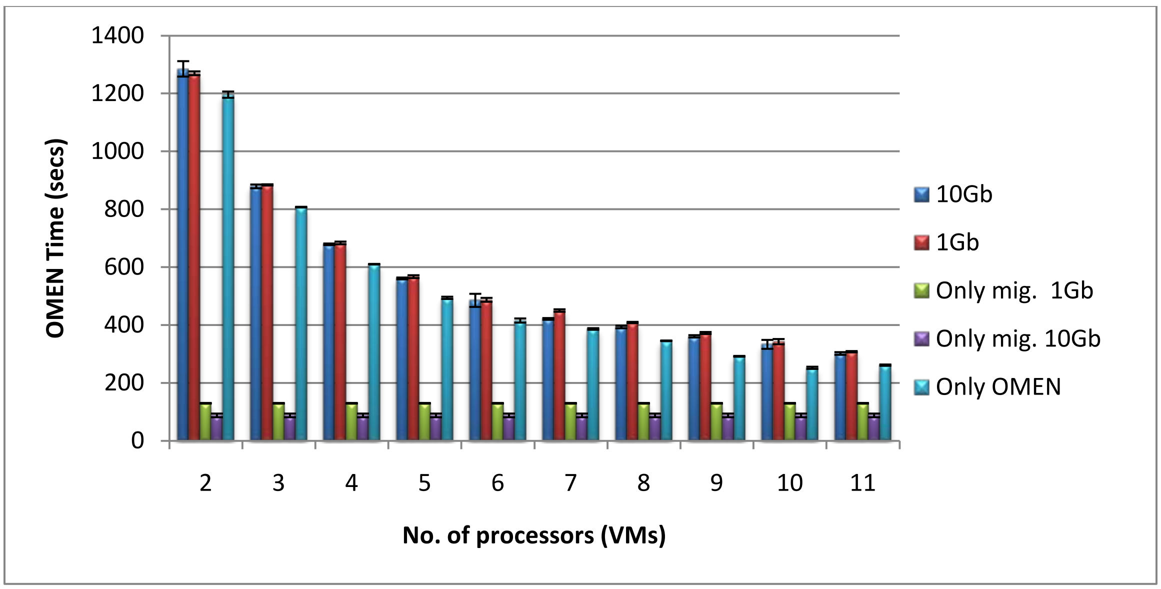

Figure 4.4 OMEN -One VM Migrated over Eight Migration Cycles

Similar to the results of HPL showed in figure 4.1, OMEN tolerated a single live migration, and was able to scale well when live migrations were included during the runtime of the application. From this figure a slight difference in performance between the networks is observed, showing only a small advantage in favor of the 10Gb/s (3-11). On average there was only a 1.3% execution time advantage to 10Gb/s over the 1Gb/s network.

Figure 4.5 shows the performance of OMEN when migrating two VMs simultaneously. This graph may be compared with Figure 4.2, where the same categories were used to assess the performance of running HPL with eight cycles of two simultaneous migrations, but this time the experiment was conducted with OMEN instead of HPL.

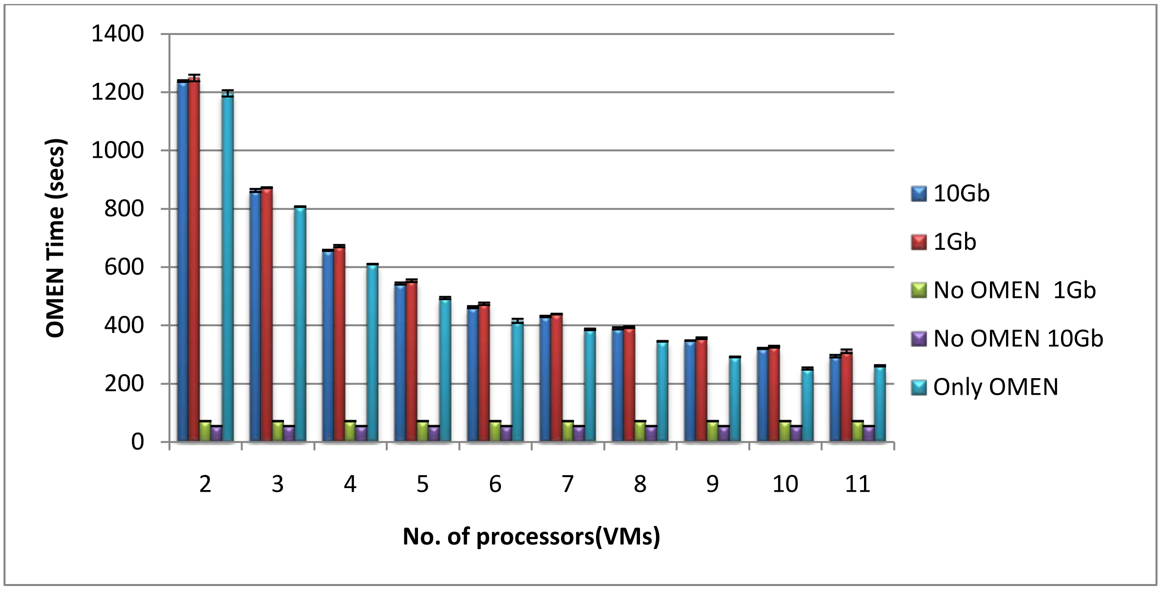

Figure 4.5 OMEN -Two VMs Simultaneously Migrated over Eight Migrations Cycles

OMEN tolerated two simultaneous VM migrations, scaled well and finished successfully with no errors reported during the experiments. Additionally, Figure 4.5 shows that over the 10Gb/s network the performance was just 2.6% more efficient than it was over the one VM migration based. This two VM based trial also provided a 2.18% performance gain over the 1Gb/s network.

In contrast to the behavior of HPL after executing the same sets of experiments with four and eight simultaneous migrations as demonstrated in Figures 10 and 11, this study found that OMEN tolerated multiple simultaneous migrations very well when performing experiments over the 10Gb/s network. The performance gain of four and eight migrations was 3.65% and 3.79% (respectively) better than the performance provided in the single migration experiments.

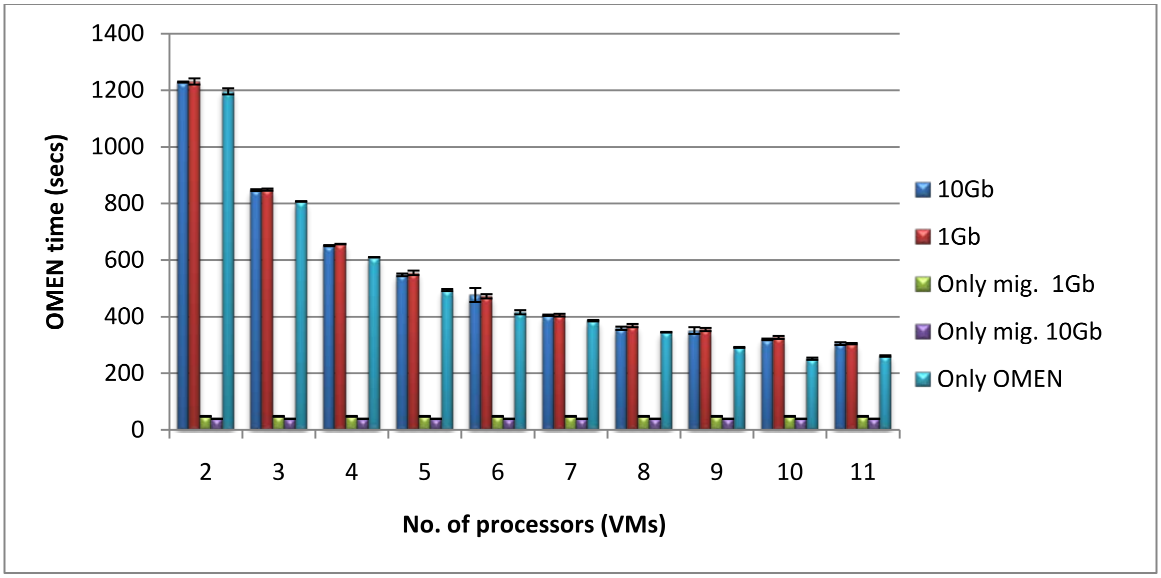

Figure 4.6 OMEN -Four VMs Simultaneously Migrated over Eight Migrations Cycles

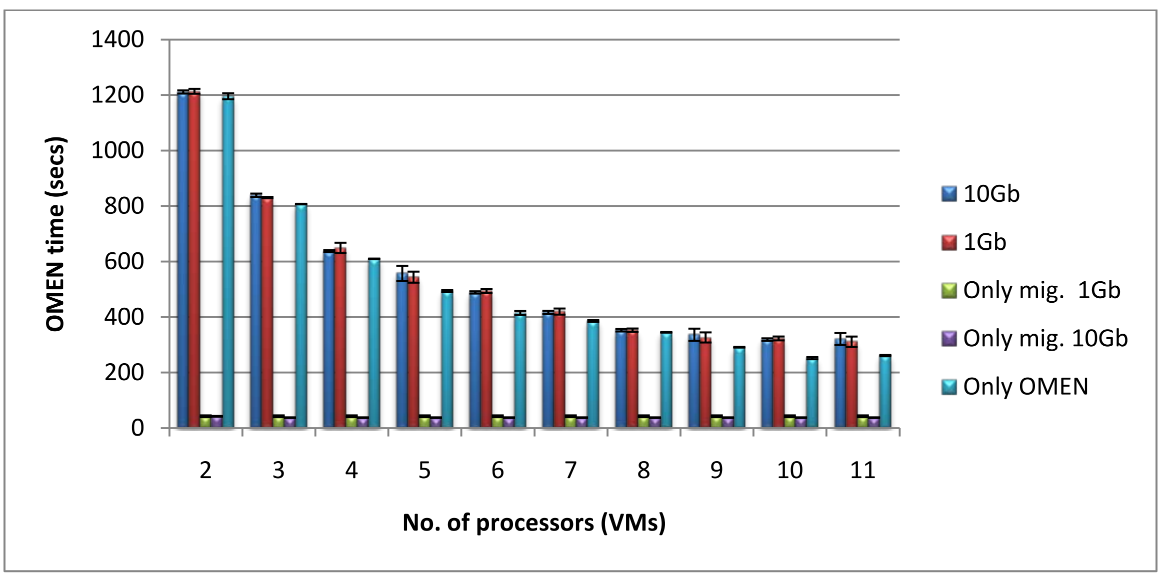

Figure 4.7 OMEN -Eight VMs Simultaneously Migrated over Eight Migrations Cycles

Likewise, over the 1Gb/s network a 4.31% and 5.28% better performance was noted. In order to complete OMEN calculations with multiple live migrations, not only does it terminate successfully but it also required less time to complete than the experiments with a single migration. Therefore, this thesis concluded that OMEN tolerated multiple live migrations much better than HPL.

4.2.2.3. Runtime without a parallel benchmark

Figure 4.8 depicts a closer view of the results after running eight migration cycles in series of one, two, four, and eight simultaneous live migrations over each of the networks fabrics (1Gb/s Ethernet and 10Gb/s Myricom). For these experiments no parallel benchmark was running in order to have a point of reference regarding the time lapsed to complete live migrations.

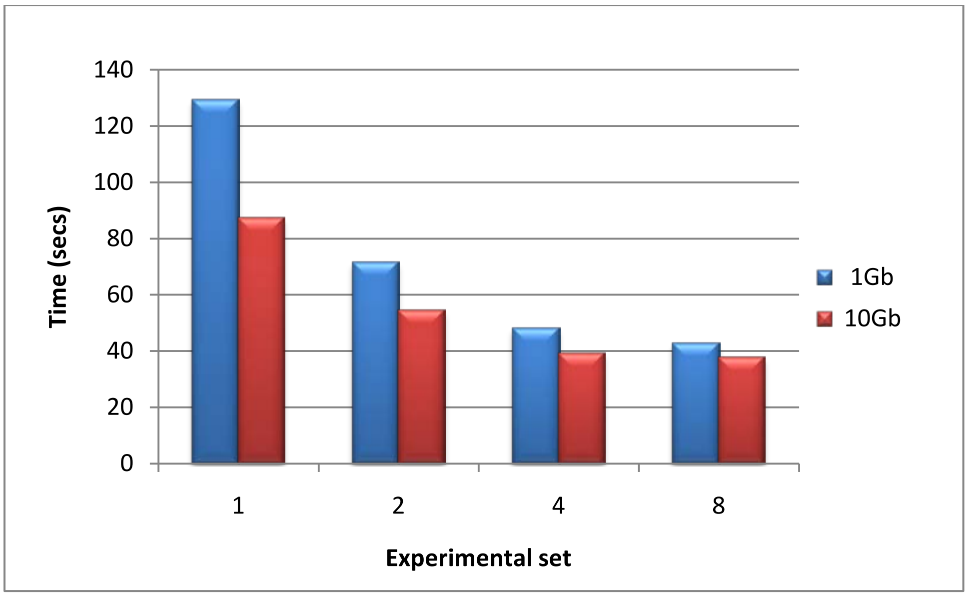

Figure 4.8 Experimental Trials of Multiple Simultaneous Migrations out of Eight Migration Cycles -No Parallel Jobs Were Running

The performance of the 10Gb/s network was better than the 1Gb/s network across all experiments. As shown in Figure 4.4, the 10Gb/s network was 32.3% more efficient than the 1Gb/s network when live migrating only one VM at a time out of eight migration cycles. This difference was narrower as the number of simultaneous migrations increased, 24.4%, 18.2%, and 11.6% time was gained when migrating two, four, and eight VMs respectively, over the Myricom 10Gb/s network.

4.3. Summary

Chapter four presented the quantitative analysis and outcome obtained after executing a series of experiments and comparing the results with the major patterns found. It provided an analysis of the correctness and performance of live migration with and without parallel jobs. The next chapter presents a summary of the most important issues found in this study and concludes with recommendations for future continuation of this research.

CHAPTER 5. CONCLUSIONS, DISCUSSIONS, AND FUTURE RECOMMENDATIONS

This chapter summarizes the major findings observed in this work. A discussion section and recommendations for future continuation of this research is offered.

5.1. Conclusions

The author of this thesis evaluated the viability of using live migration of virtual machines as an alternative to keeping parallel applications healthy by preventing them from failing. This study was specifically focused on evaluating the efficiency and correctness of a proactive approach in correlation to traditional reactive methodologies.

As long as failure is predicted on time or for maintenance purposes, It is possible to successfully migrate parallel jobs from one or multiple degraded hosts to one or multiple healthy ones. Therefore, virtualization proved to be a good alternative to save the state of the parallel calculations by allowing them to complete on a different host. For this reason this study rejects the hypothesis H_o of chapter 3, because multiple live migration of VMs on which parallel applications were running were able to reduce the fault rate experienced by parallel applications.

The performance of simultaneous live migrating multiple VMs is more efficient than the sequentially migration of a single VM. After conducting several series of experiments based on sequential migration of only one VM at a time and multiple simultaneous migrations, the results accounted for the gain in performance of multiple simultaneous live migrations. This means that if a degraded computer machine holds 10 VMs, all of them could be effectively live migrated at once, instead of migrating them one by one, which improves the total execution time of the parallel application.

OMEN demonstrated better tolerance than HPL to the live migration of parallel jobs. As the number of processors or VMs involved in the parallel computation improved, the total time to complete the benchmark decreased. Overall, OMEN scaled well with different numbers of simultaneous migrations without affecting the final results of the computations.

LAM/MPI and MPICH-2 worked very well with virtual machines. The connections, synchronization, transfer, and manipulation of data was fluently achieved among the virtual containers. Overall the results of the experiments indicated that it was possible to successfully live migrate a parallel application using these powerful platforms of high performance computing.

The total execution time of parallel calculations with VMs migrated during the runtime of the application was impacted by the network fabric. The 10Gb/s based experiments were constantly more efficient than the experiment conducted over the traditional 1Gb/s network, though the difference was always small. In general, the time to live migrate VMs that run parallel jobs is dependent upon the bandwidth and latency of the network.

5.2. Discussion

This thesis tested the power of *OpenVZ* virtualization and found that the live migration process worked flawlessly with parallel jobs independent of the number of simultaneous migrations and the successful completion of parallel applications. This study also determined that there was a positive effect on the rate to calculate the parallel tasks depending upon the network fabric used during the live migration process. Even though virtualization was a good alternative to increase the mean time to failure of parallel applications, this is still a new technology that requires further research to improve its scope and enhance its scalability.

There was some uncertainty about the behavior of *MPI* processes during the live migration. It was not clear if there were packages lost when the receiver process was unresponsive while it was frozen within a migrating container. After investigating this issue, the conclusion was that there was an advantage to using *MPI* over *TCP* because part of the *TCP* protocol behavior is to automatically attempt to retransmit lost packets when the receiver was unresponsive.

Virtual live migration can provide many benefits, however, not without a cost. The percentage overhead ranges from 11.35% for two processor experiments to 65.21% for the eight processor experiments. Other fault tolerant approaches, such as checkpoint and restart are compelled to frequently checkpointing the entire application, which usually takes an unreasonable amount of time and slows the system down with a massive load. In contrast, live migration of virtual machines only transfers precise nodes used by a parallel application, which requires only a fraction of the bandwidth consumed by the traditional checkpoint approach. The advantage is that aside from a small increase in total execution time, this process can be done while the system operates and will not affect the normal behavior of the applications. For these reasons, the extra cost of implementing live migration is insignificant when compared with the inherent benefits.

5.3. Faults Experienced

There were observed two major types of faults experienced during experiments: time considerations for the parallel applications and resource allocation for the VMs.

First, for the initial experiments the total execution time of the parallel applications was not enough to allow for the execution of eight migrations cycles when more than seven processors (VMs) were involved in the parallel calculations. As mentioned in the results section, HPL and OMEN always terminated successfully when live migrations were conducted; nonetheless these correct results were not reflecting the execution of the same number of eight migrations for all of the experiments, which causes inconsistencies in the final results. To solve this issue, the total execution time of the parallel applications was increased enough to allow for eight live migrations even when eleven processors (VMs) were involved in the calculations. This total time varied from 895.4 to 424.9 seconds (2 to 11 VMs) for *HPL*, and from 1205.7 to 259.1 seconds (2 to 11VMs) for the *OMEN* experiments.

Second, during the initial experiments with *OMEN* there was not enough memory allocated for the VMs. For the first experiments with *HPL*, 10GB of memory allocated per VM was fine to compile *HPL*, complete the parallel calculations, and migrate each VM. Unfortunately this was not the case for *OMEN* because it could not even be compiled over 10GB-based VMs. Therefore, to solve this lack of memory issue, it was necessary to increase the amount of memory of each VM to 30GB and repeat all of the *HPL*-based tests to avoid inconsistencies with *OMEN*-based experiments.

5.4. Future Recommendations

The virtualization application used in this study was only the UNIX operating system-based *OpenVZ*. The study can be further improved by testing the live migration behavior with several virtualization platforms and over different operating system instances. Because the platform requirements to run parallel applications greatly vary, using several experimental platforms could allow testing the speed and parallel job migration behavior over many different technologies.

Another interesting experiment would be to implement an entire reliability system for large computing systems. By establishing a prediction model that accounts for the detection of degraded hosts, with a virtualization environment that automatically launches VM live migration to keep the parallel jobs working until completion. This experiment would have the potential to join together a traditional reactive approach with the proactive solution presented in this study.

5.5. Summary

This final chapter included the major finding in this research. It also presented a discussion section followed by recommendations for further improvements on the presented research.

LIST OF REFERENCES

LIST OF REFERENCES

Adelstein, T., Lubanovic, B. (2007). Linux System Administration. *Solve real-life linux problems quickly*. O'Reilly Media.

Almasi, S., Gottlieb, A., (1989). High Parallel Computing. *IBM systems Journal*. Benjamin-Cummings. 29, 1. Redwood, CA.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., & Warfield, A. (2005). Live migration of virtual machines. *Proceedings of the 2005 Conference on Symposium on Networked Systems Design & Implementation*. *2*, 273-286.

Dalgaard, P. (2008). Introductory Statistics with R. *Statistics and Computing. 2nd edition, Springer. 16, 364.*

Dongarra, J. C., Bunch, J., &Stewart, P. (2009). High Performance Linpack. online article. HTTP://WWW.NETLIB.ORG/LINPACK.

Figueiredo, R. J., Dinda, P. A., & Fortes, J. A. (2003). A Case for Grid Computing on Virtual Machines. *Proceddings of the 23rd International Conference on Distributed Computing Systems*.

Fischer, W., & Mitasch, C. (2006). High availability clustering of virtual machines-posibilities and pitfalls. 12th *Linuxtag*, Wiesbaden,1-14.

Gray, J., & Shenoy, P. (2000). Rules of Thumb in Data Engineering. *Advanced technology division, Microsoft Corporation*. 16th Internet conference on Data Engineering, 3-12.

Hacker, T. J., & Meglick, Z. (2007). Using Queue Structures to Improve Job Reliability. ACM, *Proceedings of the 16th International Symposium on High Performance Computing*, 43-54.

Hacker, T. J., Romero, F., & Carothers, C. D. (2009). An analysis of clustered failures on large scale supercomputing systems. *Journal of Parallel and Distributed Computing*. 1-14.

Jin, H., Pan, Y., Xiao, N., & Sun, J. (2004). Grid and Cooperative Computing. *Proceedings of the Third International Conference of Lecture Notes in Computer Science*.

Kangarlou, A., Xu, D., Ruth, P., & Eugster, P. (2009). Taking Snapshots of Virtual Networked Environments with Minimal Downtime. *Proceddings of the 3rd International Workshop on Virtualization Technology in Distributed Computing*. ACM. Reno, Nevada.

Kleiman, S.R. (1986). Vnodes: An Architecture for Multiple File System Types in Sun UNIX. *Computer Science Division Sun Microsystems*, 1-5.

Klimeck, G., Luisier, M. (2010). Atomistic Modeling of Realistically Extended Semiconductor Devices with NEMO/OMEN. *IEEE Computing in Science and Engineering (CISE),* 12, 28-35.

Liang, Y., Zhang, Y., Jette, M., Sivasubramaniam, A., & Sahoo, R. (2006). Blue Gene/L failure analysis and prediction models. *Dependable Systems and Networks International Conference*, 425-435.

Liang, Y., Zhang, Y., Sivasubramaniam, A., Sahoo, R.K., Moreira, J., & Grupta, M. (2005). Filtering failure logs for a Blue Gene/L prototype. *Dependable Systems and Networks. Preceedings International Conference*, 476-485.

Liang, Y., Zhang, Y., Xiong, H, & Sahoo, R. (2007). An Adaptive Semantic Filter for Blue Gene/L Analysis. *Parallel and Distributed Processing Symposium*. IPDPS 2007. IEEE International, 1-8.

Mirkin, A., Kuznetsov, A., Kolysh, K. (2008). Containers Checkpointing and Live Migration. *Proceddings of Ottawa Linux Symposium*. 1(2), 85-90.

Nagarajan, B., Frank, M., Christian, E., & Stephen, S. (2007). Proactive Fault Tolerance for HPC with Xen Virtualization. *Proceddings of the 21st Annual Conference on Supercomputing*.

Patterson, D. A., Gibson, G., & Kartz, R.H. (1988). A Case for Redundant Arrays of Inexpensive Disks (RAID). *ACM SIGMOD Record*, 17 (3), 109-116.

Petitet, A., Whaley, R., Dongarra, J., & Cleary, A. (2009). HPL-A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. Online article. HTTP://WWW.NETLIB.ORG/BENCHMARK/HPL.

Preston, C. (1999). Unix Backup & Recovery. Protecting your Filesystem, Database, and Operating System Data. O'Reilly Media, 242.

Singh, R., Graham, P. (2008). Performance Driven Partial Checkpoint/Migrate for LAM-MPI. *22nd International Symposium of High Performance Computing Systems and Applications*. IEEE. (pp.110-116). Winnipeg, MB, Canada.

Solomonides, T. (2008). Studies on Health Technology and Informatics. *Proceedings of Healthgrid* 2008. 138.

Vaughan, N. S., New Approach to Virtualization is a Lightweight. *Computer Conference*, 39(11) 12-14.

Zhao, Y., Raicu. I., & Foster, I. (2008). Scientific Workflow Systems for 21st Century, New Bottle or New Wine?. *IEEE Congress on Services*, 1, 467– 471.

APPENDICES

Appendix A

HPL Dataset

Running time of HPL with no live migration during the run time:
12ps= 6000 5000 6000 5000 6000 5000 6000 5000 6000 5000 6000 5000

VMs	No Migration HPL	No Migration HPL	No Migration HPL	Average Only HPL - No Migration	Standard Deviation
2	888.801	895.495	869.5825	884.6261667	13.4512579
3	743.293	742.974	723.5885	736.6185	11.28543819
4	641.948	642.699	627.352	637.333	8.651951861
5	623.814	624.101	573.7115	607.2088333	29.00989655
6	556.611	557.263	563.571	559.1483333	3.843990288
7	466.513	466.163	524.4775	485.7178333	33.56731215
8	392.938	392.809	451.0415	412.2628333	33.5833724
9	390.725	390.872	393.8645	391.8205	1.771681193
10	420.402	421.379	417.175	419.652	2.200063408
11	425.673	424.964	430.825	427.154	3.198882774
			Average:	556.1542	

HPL 1 Migration
Migrations over the 10 GB/s network:

VMs	No Migration	HPL 10Gb 1Mig	HPL 10Gb 1Mig	HPL 10Gb 1Mig	Average 10Gb	Standard Deviation
2	869.5825	985.707	990.086	993.586	989.793	3.947663486
3	723.5885	875.198	860.013	906.198	880.4696667	23.53946491
4	627.352	774.029	810.43	795.799	793.4193333	18.31680404
5	573.7115	707.856	696.006	691.285	698.3823333	8.537255433
6	563.571	656.409	720.264	723.068	699.9136667	37.70222302
7	524.4775	678.744	673.37	647.491	666.535	16.71003953
8	451.0415	580.102	575.504	609.923	588.5096667	18.68645323
9	393.8645	547.975	636.483	528.837	571.0983333	57.42762173
10	417.175	555.047	549.059	543.573	549.2263333	5.738829962
11	430.825	604.91	572.311	536.533	571.2513333	34.20081435
				Total avg:	700.8598667	

Migrations over the 1 GB/s network:

VMs	HPL 1Gb 1Mig	HPL 1Gb 1Mig	HPL 1Gb 1Mig	Average 1Gb	Standard Deviation
2	1158.993	1101.362	1100.892	1120.41567	33.409777
3	1198.965	1138.657	1166.602	1168.07467	30.180959
4	1030.426	1036.162	1006.1	1024.22933	15.960262
5	1094.78	831.261	1053.905	993.315333	141.82347
6	881.602	1129.609	1080.93	1030.71367	131.40825
7	833.053	933.46	815.653	860.722	63.590899
8	879.329	827.942	938.561	881.944	55.355844
9	658.571	519.106	733.809	637.162	108.94083
10	660.037	771.629	674.55	702.072	60.673627
11	665.663	759.718	756.295	727.225333	53.342009
			Total avg:	914.5874	

Only migrations over the 1 GB/s network:

Trial	Only Migration / No HPL 1Mig 1Gb	Only Migration - No HPL 1Gb	STDEV
1	120.95	121.5756	0.46039907
2	121.933	121.5756	0.46039907
3	122.086	121.5756	0.46039907
4	121.315	121.5756	0.46039907
5	121.594	121.5756	0.46039907

Only migrations over the 10 GB/s network:

Trial	Only Migration / No HPL 1Mig 10Gb	Only Migration - No HPL 10Gb	STDEV
1	77.697	78.8242	2.317325
2	77.659	78.8242	2.317325
3	77.961	78.8242	2.317325
4	82.964	78.8242	2.317325
5	77.84	78.8242	2.317325

HPL 2 Migrations

Migrations over the 10 GB/s network:

VMs	No Migration	HPL 10Gb 2Mig	HPL 10Gb 2Mig	HPL 10Gb 2Mig	Average 10Gb	Standard Deviation
2	869.5825	1032.948	984.135	966.547	994.5433333	34.40237277
3	723.5885	2052.009	1921.752	2208.017	2060.592667	143.3254062
4	627.352	730.713	2339.139	1906.095	1658.649	832.2744088
5	573.7115	1919.022	2273.222	2128.555	2106.933	178.0871785
6	563.571	2099.297	1969.729	2012.553	2027.193	66.01298248
7	524.4775	2117.096	2198.703	2067.27	2127.689667	66.35380699
8	451.0415	1726.265	727.418	1719.841	1391.174667	574.8391091
9	393.8645	1678.11	566.699	509.192	918.0003333	658.9019599
10	417.175	3720.423	1392.381	509.669	1874.157667	1658.709402
11	430.825	842.734	798.747	890.268	843.9163333	45.77195423
				Total avg:	1600.284967	

Migrations over the 1 GB/s network:

VMs	HPL 1Gb 2Mig	HPL 1Gb 2Mig	HPL 1Gb 2Mig	Average 1Gb	Standard Deviation
2	1055.113	1193.305	1161.571	1136.663	72.38483
3	2145.32	2133.871	1849.646	2042.94567	167.50027
4	1069.749	2087.071	1975.596	1710.80533	557.96199
5	2352.447	2521.684	2001.878	2292.003	265.122
6	2121.624	2052.887	2229.658	2134.723	89.110518
7	2057.786	1877.439	2073.561	2002.92867	108.96309
8	1872.194	1706.541	864.901	1481.212	540.1293
9	742.428	1868.171	1484.121	1364.90667	572.26163
10	1515.983	1417.059	1177.923	1370.32167	173.80859
11	1140.127	1131.308	1609.787	1293.74067	273.73967
			Total avg:	1683.02497	

Only migrations over the 1 GB/s network:

Trial	JM 2Mig 1Gb	Only Migration 1Gb	STDEV
1	72.254	72.7206	1.26950868
2	74.438	72.7206	1.26950868
3	72.589	72.7206	1.26950868
4	71.013	72.7206	1.26950868
5	73.309	72.7206	1.26950868

Only migrations over the 10 GB/s network:

Trial	JM 2Mig 10Gb	Only Migration 10Gb	STDEV
1	49.102	49.3738	0.91501
2	50.294	49.3738	0.91501
3	49.252	49.3738	0.91501
4	50.177	49.3738	0.91501
5	48.044	49.3738	0.91501

HPL 4 Migrations

Migrations over the 10 GB/s network:

VMs	No Migration	HPL 10Gb 4Mig	HPL 10Gb 4Mig	HPL 10Gb 4Mig	Average	Standard Deviation
2	869.5825	953.006	956.358	965.836	958.4	6.654287941
3	723.5885	827.81	811.693	831.379	823.6273333	10.48835899
4	627.352	677.587	1857.378	677.614	1070.859667	681.1448573
5	573.7115	2031.165	1900.849	3355.042	2429.018667	804.602371
6	563.571	660.65	1917.482	1891.75	1489.960667	718.3193373
7	524.4775	2090.633	2102.622	2026.028	2073.094333	41.1990756
8	451.0415	1884.152	506.695	1756.139	1382.328667	761.0174575
9	393.8645	501.111	1929.078	476.334	968.841	831.6819086
10	417.175	3879.858	505.106	1244.099	1876.354333	1773.992072
11	430.825	787.316	868.424	494.211	716.6503333	196.8605336
					1378.9135	

Migrations over the 1 GB/s network:

VMs	HPL 1Gb 4Mig	HPL 1Gb 4Mig	HPL 1Gb 4Mig	Average	Standard Deviation
2	950.693	955.552	961.631	955.958667	5.4803279
3	806.933	823.824	2037.449	1222.73533	705.61328
4	685.487	685.585	697.159	689.410333	6.7107211
5	2219.384	738.274	1981.996	1646.55133	795.4961
6	656.089	666.461	1982.779	1101.77633	762.98831
7	1923.229	2002.605	2027.665	1984.49967	54.521297
8	1856.984	511.282	3431.656	1933.30733	1461.6823
9	3750.203	1627.217	1590.269	2322.563	1236.5105
10	1118.68	1348.458	1524.314	1330.484	203.41346
11	747.886	769.763	3849.828	1789.159	1784.6252
				1497.6445	

Only migrations over the 1 GB/s network:

JM 4Mig 1Gb	Only Migration 1Gb	STDEV
48.051	48.3916	0.60096031
48.008	48.3916	0.60096031
49.374	48.3916	0.60096031
48.565	48.3916	0.60096031
47.96	48.3916	0.60096031

Only migrations over the 10 GB/s network:

JM 4Mig 10Gb	Only Migration 10Gb	STDEV
37.808	38.3646	1.218796
40.284	38.3646	1.218796
38.509	38.3646	1.218796
38.24	38.3646	1.218796
36.982	38.3646	1.218796

Appendix B

OMEN Dataset

Running time of OMEN with no live migration during the run time:

VMs	No Migration OMEN	No Migration OMEN	No Migration OMEN	Average Only OMEN - No Migration	Standard Dev.
2	1197.449	1184.191	1205.759	1195.7997	10.878184
3	807.335	808.262	806.193	807.26333	1.0363601
4	610.02	610.996	609.185	610.067	0.9064144
5	493.444	497.845	490.004	493.76433	3.9303028
6	410.443	423.719	412.022	415.39467	7.2521862
7	384.856	389.441	384.259	386.18533	2.8352471
8	346.415	345.46	344.186	345.35367	1.118298
9	293.22	290.561	291.451	291.744	1.3534981
10	250.337	256.372	249.319	252.00933	3.8123125
11	260.909	263.723	259.166	261.266	2.2993801
			Average:	505.88473	

OMEN 1 Migration
Migration over 10Gb/s network:

VMs	No Migration	OMEN 10Gb 1Mig	OMEN 10Gb 1Mig	OMEN 10Gb 1Mig	Average 10Gb	Standard Deviation
2	1195.7997	1315.522	1272.361	1266.742	1284.875	26.6893658
3	807.26333	886.192	873.561	876.911	878.888	6.543465213
4	610.067	680.065	675.563	680.737	678.78833	2.813356951
5	493.76433	563.682	561.434	557.396	560.83733	3.185193453
6	415.39467	511.154	473.391	470.632	485.059	22.64099797
7	386.18533	422.915	422.378	417.409	420.90067	3.03576915
8	345.35367	397.098	388.385	393.549	393.01067	4.381374708
9	291.744	365.045	360.376	357.161	360.86067	3.964283079
10	252.00933	321.555	350.561	326.792	332.96933	15.45822093
11	261.266	306.478	297.189	300.999	301.55533	4.669422912
				Total avg:	569.77443	

Migrations over the 1 GB/s network:

VMs	OMEN 1Gb 1Mig	OMEN 1Gb 1Mig	OMEN 1Gb 1Mig	Average 1Gb	Standard Deviation
2	1276.98	1264.125	1267.961	1269.689	6.599346963
3	886.362	883.215	882.493	884.0233	2.057265742
4	678.136	686.587	685.771	683.498	4.661517671
5	567.012	571.911	561.963	566.962	4.974188477
6	480.807	485.757	494.212	486.9253	6.778440701
7	450.741	453.788	445.263	449.9307	4.319882676
8	407.761	406.953	411.066	408.5933	2.179168725
9	369.84	376.406	370.623	372.2897	3.586282523
10	332.262	345.533	349.535	342.4433	9.041496687
11	306.323	306.462	310.238	307.6743	2.22128799
			Total avg:	577.2029	

Only migrations over the 1 GB/s network:

Trial	Only Mig. / No OMEN 1Gb	Av Only Mig - No OMEN 1Gb	STDEV
1	128.992	129.4092	1.135838325
2	130.657	129.4092	1.135838325
3	127.884	129.4092	1.135838325
4	130.413	129.4092	1.135838325
5	129.1	129.4092	1.135838325

Only migrations over the 10 GB/s network:

Trial	Only Mig. / No OMEN 1Mig 10Gb	Av Only Mig. - No OMEN 10Gb	STDEV
1	98.763	87.5708	6.420597
2	86.208	87.5708	6.420597
3	83.778	87.5708	6.420597
4	86.16	87.5708	6.420597
5	82.945	87.5708	6.420597

OMEN 2 Migration
Migration over 10Gb/s network:

VMs	No Migration	OMEN 10Gb 2Mig	OMEN 10Gb 2Mig	OMEN 10Gb 2Mig	Average 10Gb	Standard Deviation
2	1195.7997	1240.584	1234.866	1238.883	1238.111	2.936131639
3	807.26333	867.433	856.945	863.838	862.73867	5.329721975
4	610.067	655.1024	658.284	658.862	657.41613	2.024485726
5	493.76433	546.735	539.38	544.237	543.45067	3.740019563
6	415.39467	459.155	466.206	462.168	462.50967	3.537895184
7	386.18533	427.713	432.07	431.806	430.52967	2.442873786
8	345.35367	392.597	391.189	385.626	389.804	3.686106211
9	291.744	347.014	348.073	348.593	347.89333	0.804686481
10	252.00933	318.754	322.514	321.648	320.972	1.969043423
11	261.266	290.63	293.22	299.151	294.33367	4.368300623
				Total avg:	554.77588	

Migrations over the 1 GB/s network:

VMs	OMEN 1Gb 2Mig	OMEN 1Gb 2Mig	OMEN 1Gb 2Mig	Average 1Gb	Standard Deviation
2	1238.521	1260.819	1246.106	1248.482	11.3372939
3	870.944	873.953	872.191	872.3627	1.511827481
4	669.253	669.947	676.402	671.8673	3.942437106
5	555.075	547.688	556.465	553.076	4.717619421
6	471.96	471.02	478.906	473.962	4.307348604
7	439.989	438.555	437.41	438.6513	1.292195935
8	395.666	395.897	390.421	393.9947	3.097040577
9	359.205	353.452	354.952	355.8697	2.984264789
10	325.034	325.057	330.554	326.8817	3.18035475
11	306.528	307.962	318.33	310.94	6.439966149
			Total avg:	564.6087	

Only migrations over the 1 GB/s network:

Trial	Only Mig. / No OMEN 1Gb	Av. Only Mig - No OMEN 1Gb	STDEV
1	70.468	71.49	0.745166089
2	71.952	71.49	0.745166089
3	71.206	71.49	0.745166089
4	72.425	71.49	0.745166089
5	71.399	71.49	0.745166089

Only migrations over the 10 GB/s network:

Trial	Only Mig. / No OMEN 1Mig 10Gb	Av. Only Mig. - No OMEN 10Gb	STDEV
1	54.606	54.5354	0.509359
2	53.872	54.5354	0.509359
3	55.296	54.5354	0.509359
4	54.459	54.5354	0.509359
5	54.444	54.5354	0.509359

OMEN 4 Migration
Migration over 10Gb/s network:

VMs	No Migration	OMEN 10Gb 4Mig	OMEN 10Gb 4Mig	OMEN 10Gb 4Mig	Average 10Gb	Standard Deviation
2	1195.7997	1230.649	1226.999	1228.282	1228.6433	1.851633423
3	807.26333	846.591	843.924	849.927	846.814	3.007706601
4	610.067	648.235	651.465	652.258	650.65267	2.130973095
5	493.76433	545.789	543.665	553.207	547.55367	5.009788153
6	415.39467	503.515	456.421	469.216	476.384	24.35151693
7	386.18533	407.772	402.866	405.093	405.24367	2.456467857
8	345.35367	352.434	359.792	364.824	359.01667	6.231282479
9	291.744	364.009	341.967	347.124	351.03333	11.5292934
10	252.00933	319.67	316.866	323.081	319.87233	3.112436398
11	261.266	306.834	308.032	298.798	304.55467	5.021275668
				Total avg:	548.97683	

Migrations over the 1 GB/s network:

VMs	OMEN 1Gb 4Mig	OMEN 1Gb 4Mig	OMEN 1Gb 4Mig	Average 1Gb	Standard Deviation
2	1218.112	1237.043	1236.819	1230.658	10.86573196
3	846.746	853.167	846.633	848.8487	3.74021314
4	657.828	656.586	655.376	656.5967	1.226034801
5	552.434	563.88	547.543	554.619	8.384811328
6	463.374	475.562	476.506	471.814	7.324478411
7	403.786	402.054	411.5	405.78	5.028792698
8	366.156	364.076	375.747	368.6597	6.225296807
9	358.493	347.544	358.2	354.7457	6.238546652
10	332.168	327.519	321.149	326.9453	5.53185415
11	306.927	302.859	303.637	304.4743	2.159398373
			Total avg:	552.3141	

Only migrations over the 1 GB/s network:

Trial	Only Mig. / No OMEN 1Gb	Av. Only Mig - No OMEN 1Gb	STDEV
1	47.815	48.0642	1.236962691
2	46.139	48.0642	1.236962691
3	48.155	48.0642	1.236962691
4	48.812	48.0642	1.236962691
5	49.4	48.0642	1.236962691

Only migrations over the 10 GB/s network

Trial	Only Mig. / No OMEN 1Mig 10Gb	Av. Only Mig. - No OMEN 10Gb	STDEV
1	40.348	39.2704	1.029436
2	40.313	39.2704	1.029436
3	38.667	39.2704	1.029436
4	38.024	39.2704	1.029436
5	39	39.2704	1.029436

OMEN 8 Migration
Migration over 10Gb/s network:

VMs	No Migration	OMEN 10Gb 8Mig	OMEN 10Gb 8Mig	OMEN 10Gb 8Mig	Average 10Gb	Standard Deviation
2	1195.7997	1214.091	1204.523	1214.776	1211.13	5.732071441
3	807.26333	844.53	832.183	839.914	838.87567	6.238646034
4	610.067	633.99	640.419	639.337	637.91533	3.442217648
5	493.76433	563.801	527.675	581.607	557.69433	27.47969595
6	415.39467	486.602	486.683	493.585	488.95667	4.008458848
7	386.18533	415.335	423.913	412.853	417.367	5.803246333
8	345.35367	356.332	354.664	347.658	352.88467	4.602617661
9	291.744	345.542	311.81	352.817	336.723	21.87977566
10	252.00933	314.36	322.322	321.125	319.269	4.293240385
11	261.266	336.701	296.118	329.989	320.936	21.75344982
				Total Avg:	548.17517	

Migrations over the 1 GB/s network:

VMs	OMEN 1Gb 8Mig	OMEN 1Gb 8Mig	OMEN 1Gb 8Mig	Average 1Gb	Standard Deviation
2	1212.777	1205.132	1222.792	1213.567	8.856465153
3	829.201	829.551	833.651	830.801	2.474368606
4	670.845	641.132	636.217	649.398	18.73551875

5	530.698	534.61	566.963	544.090333	19.90465012
6	492.356	489.016	502.19	494.520667	6.848569583
7	419.279	431.412	409.537	420.076	10.95925695
8	349.119	359.872	350.765	353.252	5.791858855
9	315.079	317.531	347.729	326.779667	18.18403149
10	317.45	321.011	331.045	323.168667	7.049656044
11	303.413	297.456	332.723	311.197333	18.87822042
			Total Avg:	546.685067	

Only migrations over the 1 GB/s network:

Trial	Only Mig. / No OMEN 1Gb	Av. Only Mig - No OMEN 1Gb	STDEV
1	44.964	42.7602	3.366060784
2	43.559	42.7602	3.366060784
3	45.548	42.7602	3.366060784
4	42.63	42.7602	3.366060784
5	37.1	42.7602	3.366060784

Only migrations over the 10 GB/s network

Trial	Only Mig. / No OMEN 1Mig 10Gb	Av. Only Mig. - No OMEN 10Gb	STDEV
1	37.983	37.7784	1.225916
2	36.774	37.7784	1.225916
3	37.991	37.7784	1.225916
4	36.532	37.7784	1.225916
5	39.612	37.7784	1.225916

Appendix C

Bash Shell Script

```
#!/bin/bash
# This Bash shell script executes live migration of OpenVZ VMs,
# and controls the execution of OMEN and/or HPL parallel benchmarks
# over two different network fabrics - 1Gb/s and/or 10Gb/s
# By Fabian Romero
# Fall 2009

#Debugging Function:
debug ()
{
    if [[ "$DEBUG" == "true" ]]; then
      if [[ "$1" == "on" ]]; then
          set -x
      else
          set +x
      fi
    fi
}



##############  Migration Steps  ################

myAA ()
{
        FPROB="MyriAA"
        echo
        TM1=$(date +%F | sed 's/-//g')
        TM2=$(date +%T | sed 's/://g')
      echo
      echo "Starting migration cycle 1..."
        ssh root@${HOST[7]} "vzmigrate -r no --online -v ${HOST[8]}
$VM1 >> /tmp/$FPROB-$TM1-$TM2.txt" &
      pid=$!
      ssh root@${HOST[7]} "vzmigrate -r no --online -v ${HOST[8]} $VM2
>> /tmp/$FPROB-$TM1-$TM2.txt" &
      ppdd=$!
#     ssh root@${HOST[7]} "vzmigrate -r no --online -v ${HOST[8]} $VM3
>> /tmp/$FPROB-$TM1-$TM2.txt" &
#     ssh root@${HOST[7]} "vzmigrate -r no --online -v ${HOST[8]} $VM4
>> /tmp/$FPROB-$TM1-$TM2.txt" &
      #wait
      wait $pid
      wait $ppdd
        echo "cycle 1 completed"

}

myBA ()
{
        FPROB="MyriBA"
        echo
```

```
        TM1=$(date +%F | sed 's/-//g')
        TM2=$(date +%T | sed 's/://g')
      echo
      echo "Starting migration cycle 2..."
        ssh root@${HOST[8]} "vzmigrate -r no --online -v ${HOST[7]}
$VM1 >> /tmp/$FPROB-$TM1-$TM2.tx " &
      pid=$!
      ssh root@${HOST[8]} "vzmigrate -r no --online -v ${HOST[7]} $VM2
>> /tmp/$FPROB-$TM1-$TM2.tx " &
      ppdd=$!
#     ssh root@${HOST[8]} "vzmigrate -r no --online -v ${HOST[7]} $VM3
>> /tmp/$FPROB-$TM1-$TM2.txt" &
#     ssh root@${HOST[8]} "vzmigrate -r no --online -v ${HOST[7]} $VM4
>> /tmp/$FPROB-$TM1-$TM2.txt" &
      #wait
      #pid=$!
      wait $pid
      wait $ppdd
        echo "cycle 2 completed"

}

myCopyAA ()
{
        FPROB="MyriCAA"
        echo
        TM1=$(date +%F | sed 's/-//g')
        TM2=$(date +%T | sed 's/://g')
      echo
      echo "Starting migration cycle 3..."
        ssh root@${HOST[7]} "vzmigrate -r no --online -v ${HOST[8]}
$VM1 >> /tmp/$FPROB-$TM1-$TM2.txt" &
      pid=$!
      ssh root@${HOST[7]} "vzmigrate -r no --online -v ${HOST[8]} $VM2
>> /tmp/$FPROB-$TM1-$TM2.txt" &
      ppdd=$!
#     ssh root@${HOST[7]} "vzmigrate -r no --online -v ${HOST[8]} $VM3
>> /tmp/$FPROB-$TM1-$TM2.txt" &
#     ssh root@${HOST[7]} "vzmigrate -r no --online -v ${HOST[8]} $VM4
>> /tmp/$FPROB-$TM1-$TM2.txt" &
      #wait
        #pid=$!
      wait $pid
      wait $ppdd
        echo "cicle 3 completed"
}

myCopyBA ()
{
        FPROB="MyriCBA"
        echo
        TM1=$(date +%F | sed 's/-//g')
        TM2=$(date +%T | sed 's/://g')
      echo
      echo "Starting migration cicle 4..."
        ssh root@${HOST[8]} "vzmigrate -r no --online -v ${HOST[7]}
$VM1 >> /tmp/$FPROB-$TM1-$TM2.tx " &
```

```
      pid=$!
      ssh root@${HOST[8]} "vzmigrate -r no --online -v ${HOST[7]} $VM2
>> /tmp/$FPROB-$TM1-$TM2.tx " &
      ppdd=$!
#     ssh root@${HOST[8]} "vzmigrate -r no --online -v ${HOST[7]} $VM3
>> /tmp/$FPROB-$TM1-$TM2.txt" &
#     ssh root@${HOST[8]} "vzmigrate -r no --online -v ${HOST[7]} $VM4
>> /tmp/$FPROB-$TM1-$TM2.txt" &
      #wait
        #pid=$!
      wait $pid
      wait $ppdd
        echo "last cycle 4 completed, waiting for OMEN to finish"
}


########## LAM/MPI HPL or OMEN execution ###########
mpi2 ()
{
   echo "Modifying HPL.dat..."
   ssh bob@${VMHOST[1]} sed -i -e '3s/[a-z]*[0-9]*/myri2/' $HPL
   ssh bob@${VMHOST[1]} sed -i -e '12s/[0-9]*/2/' $HPL
   echo
}

mpi3 ()
{
   echo "Modifying HPL.dat..."
   ssh bob@${VMHOST[1]} sed -i -e '3s/[a-z]*[0-9]*/myri3/' $HPL
   ssh bob@${VMHOST[1]} sed -i -e '12s/[0-9]*/3/' $HPL
   echo
}

mpi4 ()
{
   echo "Modifying HPL.dat..."
   ssh bob@${VMHOST[1]} sed -i -e '3s/[a-z]*[0-9]*/myri4/' $HPL
   ssh bob@${VMHOST[1]} sed -i -e '12s/[0-9]*/4/' $HPL
   echo
}

mpi5 ()
{
   echo "Modifying HPL.dat..."
   ssh bob@${VMHOST[1]} sed -i -e '3s/[a-z]*[0-9]*/myri5/' $HPL
   ssh bob@${VMHOST[1]} sed -i -e '12s/[0-9]*/5/' $HPL
   echo
}

mpi6 ()
{
   echo "Modifying HPL.dat..."
   ssh bob@${VMHOST[1]} sed -i -e '3s/[a-z]*[0-9]*/myri6/' $HPL
   ssh bob@${VMHOST[1]} sed -i -e '12s/[0-9]*/6/' $HPL
   echo
}
```

```
mpi7 ()
{
   echo "Modifying HPL.dat..."
   ssh bob@${VMHOST[1]} sed -i -e '3s/[a-z]*[0-9]*/myri7/' $HPL
   ssh bob@${VMHOST[1]} sed -i -e '12s/[0-9]*/7/' $HPL
   echo
}

mpi8 ()
{
   echo "Modifying HPL.dat..."
   ssh bob@${VMHOST[1]} sed -i -e '3s/[a-z]*[0-9]*/myri8/' $HPL
   ssh bob@${VMHOST[1]} sed -i -e '12s/[0-9]*/8/' $HPL
   echo
}

mpi9 ()
{
   echo "Modifying HPL.dat..."
   ssh bob@${VMHOST[1]} sed -i -e '3s/[a-z]*[0-9]*/myri9/' $HPL
   ssh bob@${VMHOST[1]} sed -i -e '12s/[0-9]*/9/' $HPL
   echo
}

mpi10 ()
{
   echo "Modifying HPL.dat..."
   ssh bob@${VMHOST[1]} sed -i -e '3s/[a-z]*[0-9]*/myri10/' $HPL
   ssh bob@${VMHOST[1]} sed -i -e '12s/[0-9]*/10/' $HPL
   echo
}

mpi11 ()
{
   echo "Modifying HPL.dat..."
   ssh bob@${VMHOST[1]} sed -i -e '3s/[a-z]*[0-9]*/myri11/' $HPL
   ssh bob@${VMHOST[1]} sed -i -e '12s/[0-9]*/11/' $HPL
   echo
}

############  Migration  ##########################
migration ()
{
    echo
    echo "****  Myranet  *******"
                myAA
                #myAB
                myBA
                #myBB
                myCopyAA
                #myAB
                myCopyBA
                #myBB
}

testo ()
{
```

```
    echo
    echo "TEST  Executing migration migration migration"
    echo "Executing migration migration migration"
    echo "Executing migration migration migration"
    echo "Executing migration migration migration"
    echo "Executing migration migration migration"
}

###################### TODO #####################
todo ()
{
    for((J=3; J<=3; J++))
      do
      for((I=9; I<=11; I++))
      do
      TM1=$(date +%F | sed 's/-//g')
      TM2=$(date +%T | sed 's/://g')
      echo
      echo
      echo "--------- STARTING NEW PROCCESS I= $I ----------"
      echo
      echo "Wait: adjusting mpdboot to n = $I ..."
        ssh bob@${VMHOST[1]} "mpdboot -n $I -f mpd$I.hosts" &
      wait
      echo "Running mpirun..."
      INICIO=$(date +%s.%N)
        ssh bob@${VMHOST[1]} "mpirun -np $I ./OMEN_steele-pgi64-mpich2
transmission.cmd >> $PA1/OMGigDos$J-$I-$TM1-$TM2" &
      migration
      wait
      FIN=$(date +%s.%N)
        DIFFE=$(echo "$FIN - $INICIO" | bc)
        echo "difference took $DIFFE secs" >> $LPATH/OMEN-GigDos$J-$I-
$TM1-$TM2

      done
    done
}


#####################################################
############ This is the MAIN script ################
echo
debug on
HPL="/home/bob/HPL.dat"
PA1="/home/bob"
LPATH="/root/ovz-mig"
echo
PWDIR="/root/ovz-mig"
echo "Output files will be in: ${VMHOST[1]} - $PA1"
TM=20 #This is the delay time in seconds
FPROB="GigaA"

######## 1Gb-Giga hosts ##########
HOST[7]=128.210.135.164
NAME[7]="openvz164"
HOST[8]=128.210.135.165
```

```
NAME[8]="openvz165"
VMHOST[1]=128.210.135.206
VM1=730
VM2=731
#################################

######  10Gb-myri hosts ##########
#HOST[7]=192.168.0.101
#NAME[7]="Myr-openvz164"
#HOST[8]=192.168.0.105
#NAME[8]="Myr-openvz165"
#VMHOST[1]=128.210.135.206
#VM1=730
#VM2=731
#################################
todo
echo "Output files are in: $LPATH"

debug off
# Script completed
```