UNIVERSITY OF OSLO
Department of Informatics

Management of high availability services using virtualization

Master thesis

Espen Braastad
Oslo University College

May 22, 2006

Abstract

This thesis examines the use of virtualization in management of high availability services using open source tools. The services are hosted in virtual machines, which can be seamlessly migrated between the physical nodes in the cluster automatically by high availability software. Currently there are no complete open source solutions that provide migration of virtual machines as a method for repair.

The work is based on the high availability software Heartbeat. In this work, an add-on to Heartbeat is developed, allowing Heartbeat to be able to seamlessly migrate the virtual machines between the physical nodes, when shut down gracefully. This add-on is tested in a proof of concept cluster, where Heartbeat runs Xen virtual machines with high availability. The impact of migration has been measured for both TCP and UDP services, both numerically and heuristically. The outages caused by graceful failures (e.g. rebooting) are measured to be around 1/4 seconds. Practical tests are also performed. The impression is that the outages are not noticed by the users of latency critical services as game servers or streaming audio servers.

Acknowledgements

First and foremost, I would like to thank my supervisor Assoc. Professor Hårek Haugerud for his support throughout this thesis. His feedback has been of great help and is highly appreciated. Many thanks to Per Buer, the manager of the system administration group at Linpro, for spending his valuable time helping me with technical problems, providing equipment and for doing the basic configuration of the cluster. Thank you, Kyrre Begnum for countless valuable discussions, helpful tips, enthusiasm, and also for being a one of a kind motivator for me and the rest of the class. Professor Mark Burgess for believing in me and insisting on high standards, making me aim higher and demand more from myself every time.

Thanks to Gard Undheim, Sven Ulland and Jon-Henrik Bjørnstad for fruitful collaboration and inspiration in these two years. Maurice Wörnhard for beneficial discussions on methodologies and measurements, Ilir Bytyci for great companionship in weekly workouts throughout the period of this work, giving my mind much needed breaks from virtual machines and high availability, and the rest of the teachers and students here at the Network and System administration MSc course for making the time here at Oslo University College truly fabulous.

I would also like to thank my friends Morten Haraldsen, Erik Fosser, Oddmar Lid and my family for the constant support in this time and for showing interest in my work and activities at all times. Last but not least, I owe my dear girlfriend and friend, Renate Larsen, my expression of gratitude for bearing out with me even though way too much of my time in this period has been dedicated to this thesis. I love you very much!

Oslo University College, May 2006

Espen Braastad

Contents

Preface

My interest in virtual machines increased last year during a lab project in the *Network Infrastructure and Security Lab* course at OUC. UML virtual machines were used to create large networks to get hands-on experience with the configuration engine *cfengine*. Our network consisted of 5 servers and 50 clients that ran virtually on one single physical computer in the lab. The virtual machines were easily built using the tool *MLN*.

During the project I became aware of the endless possibilities given by virtualization. When Linpro, through Kyrre Begnum, provided me with this master project about virtualization in high availability clusters, I gratefully accepted.

The target audience is the reader who has some interest in service quality and brief knowledge about computer network concepts. However, an effort is made to make the terminology used in this report understandable to as many readers as possible.

Chapter 1

Introduction

High availability computing and *business critical services* are two terms that traditionally involve significant cost and effort. This is mainly due to costly hardware redundancy and complex installations requiring constant endeavor from the system administrators.

A typical scenario is that each high availability service is hosted by two x86 servers, where one of these is inactive and serves as a backup server in case of a failure on the active one. Due to business critical data hosted on these high availability servers, separation is often required. This means that each pair of servers can only host one service, or a group of services that share the same business critical data (see figure 1.1). The consequence is that the hardware resources are being utilized inefficiently. The average server utilization today is at 25% according to Bittman and Scott in Gartner[1] [1].

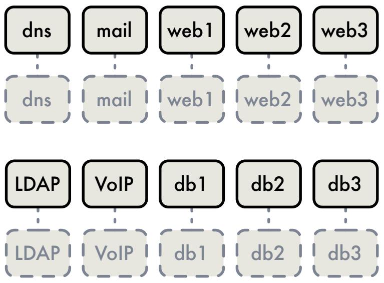


Figure 1.1: The traditional way of hosting business critical high available services. Recognized by separation of services on different physical computers and redundant set of physical computers for each service. 10 services require 20 computers.

The traditional way of hosting high available and business critical services are haunted by several disadvantages:

- As figure 1.1 shows, many computers are needed. The rule of thumb is

[1]Information Technology research and advisory company

twice the amount of computers as services. This is expensive in terms of the high demand for power usage, real estate, air cooling systems, etc.

- Each pair of computers has their own specialized configuration optimized for the specific service. This implies significant effort when changing or upgrading computers to increase capacity or to repair hardware failures.
- If the demand for capacity suddenly increases on one of the services, the only way to solve it is to upgrade the hardware running that service. This is not flexible or scalable.
- The administration effort to reconfiguration and maintenance is high.
- The hardware resources available are inefficiently utilized.

The above mentioned disadvantages are present at Linpro, a company that is hosting business critical services for their customers. The system administration group at Linpro is considering alternative topologies to avoid the disadvantages that come along with the traditional setup. A cluster manager for virtual machines that could have solved several of the disadvantages has previously been discussed [2], but no one has actually implemented it using open source tools at the time of writing.

1.1 High availability services using virtualization

Virtualization is a technique for running more than one operating system instance on a single computer simultaneously, making one computer appear as two, three or several more computers.

The two main advantages of virtualization that are interesting in this respect are:

- that it is possible to run several virtual machines on one physical computer [3].
- that virtual machines can be migrated to other physical computers seamlessly using a method called *live migration* [2].

This thesis aims to utilize these advantages by implementing a high availability cluster based on virtualization using *Heartbeat* for high availability and *Xen* for virtualization. Migration of complete virtual machines is relatively simple compared to the alternatives because the virtual machine and user applications are treated as a whole with "no loose ends". Connection tables, kernel states and in-memory data are migrated together with the virtual machine automatically. The opposite of virtual machine migration is *process migration*, where single processes are migrated. This may seem simpler, but is in reality far more complicated [4].

Xen [2, 3, 5, 6] is an open source x86 virtual machine monitor that is used to create high performance virtual machines with resource isolation and performance guarantees. This means that the virtual machines running on the same hardware are strictly separated, and that all of them are guaranteed to get a fair share of the hardware resources even though one of them is trying to acquire all of the resources.

Heartbeat [7–9] is an open source software package used to create high availability clusters. It runs as a daemon on all of the nodes and uses broadcast messaging for intercommunication in between the nodes. It is given control of a set of resources (the virtual machines), and makes the physical nodes collaborate on keeping them highly available.

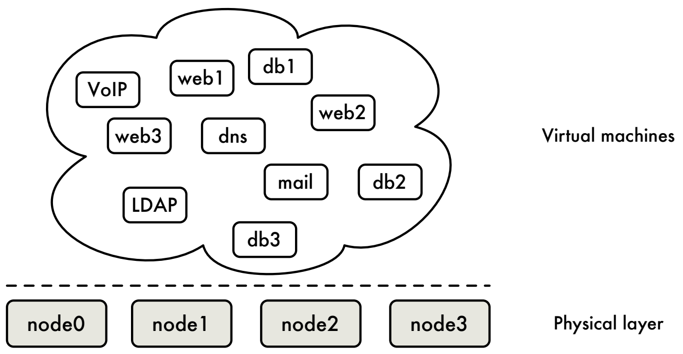


Figure 1.2: A fully functional high availability cluster setup where *node0*, *node1*, *node2* and *node3* are physical computers collaborating on running the virtual machines that hosts all of the services.

The high availability services are hosted on the virtual machines, which run on the physical computers. The heartbeat daemons monitor the health of the virtual machines, and repair unavailability automatically if a failure[2] occurs. Since *live migration* is one of the methods used for repair, availability will be maintained even though certain physical nodes are shut down due to maintenance or other reasons. The idealized model of the topology is illustrated in figure 1.2 by virtual machines floating between the physical nodes utilizing the available hardware resources without constraint to any particular of the physical machines.

Other advantages with this approach besides high availability are that:

- fewer physical computers are required.

[2]Failure is defined as scenarios that are causing outages, i.e. shutdown, reboot, hardware failures or power failures. The term "failure" as used in this thesis is further explained in section 3.6.

- high flexibility is achieved due to separation of applications and hardware. The applications can be moved between hardware without reconfiguration.
- failures are quickly repaired.
- hardware can be added, removed or modified without affecting the users. The physical computers are all configured equally, making it easy to setup new computers.
- the hardware is utilized more efficiently.
- peak demands can be coped with automatically using load balancing software that can re-allocate hardware resources within seconds.
- less effort is required from the system administrators due to automatic repair, no physical nodes with specialized installations, fewer physical computers, fewer instances of each machine (Two equal physical nodes in the traditional setup translates into one virtual machine in this setup).

Some service level agreements (SLA)[3] require dedicated servers for critical data for maximized integrity and security. Using virtualization it is possible maintain integrity and security even if several virtual machines are hosted on the same physical computer. This is due to strict separation between the virtual machines running on the same hardware.

What the proposed topology can achieve is explained using the following simple example.

Given a scenario where a company hosts critical services for 40 customers. Due to business critical data and services, each of them require a dedicated server for themselves plus redundancy. This adds up to a total of 80 physical computers, putting a heavy load on the system administrators, power usage, heat generation and space requirement (real estate).

Using our proposed setup, these business critical services and data can be hosted on virtual machines in a high available cluster where the nodes cooperate in hosting the virtual machines. The result is fewer physical computers, less maintenance cost, increased scalability, flexibility and utilization.

[3]An SLA is a contract between the provider and the user of a service about how the service should be provided.

Since the separation between the virtual machines is so powerful, it is possible to host for example 4 customers on one single physical server, giving them one virtual machine each. In our example with 40 customers, this approach lowers the amount of physical computers to 10 plus redundancy.

Since the physical hosts are equally configured, it may be enough with one extra physical computer as backup - making the total number of physical computers 11.

1.2 Objectives

Three objectives for the thesis are formalized and will be used throughout this report:

1. Review previous work on high available clustering using virtualization and open source tools.
2. Implement an experimental high availability cluster using Heartbeat and Xen.
3. Do an assessment with a scientific approach to validate the setup and use statistical methods to analyze the results if objective 2 is completed successfully.

Important: Heartbeat does not support controlling virtual machines as of this writing, and the major part of the work will be to develop and implement a tool that makes Heartbeat support Xen virtual machines - including the functionality of *live migration*. This is an innovative approach to high availability clustering using virtualization and open source tools.

1.3 Thesis outline

The background and previous research on the topic is located in chapter 2. Chapter 3 explains the methodology used, including the software and hardware, topology design and validation methodology. Chapter 4 is an analysis of the problem and explanation of how it is solved. Chapter 5 elaborates on the measurements and results used to validate the topology, to make it scientifically approved for production environments. The discussion in chapter 6 summarizes the findings, experiences and form a basic set of guidelines for bringing the topology closer to a production stable state. The conclusions are located in chapter 7.

Convention:	**Is to be interpreted as:**
Physical node	A physical computer in a cluster
Host operating system	The operating system of the physical computer
Guest operating system	The operating system running on a VM
HA	High Availability
VM	Virtual Machine

Table 1.1: Conventions used in this document

The conventions used in this paper are listed in table 1.1.

Chapter 2

Background material and previous work

In this chapter we will look at previous work, and introduce the reader to the history behind some of the tools available for creating and managing high availability clusters and the virtualization technology available today. Towards the end, a summary of previous research is provided to see what is missing for being able to configure a high availability clusters using virtualization and open source.

2.1 High availability

High availability computing is, as the term implies, a highly available computer system. How available the system should be depends on what the system should provide. For ISPs, high availability means 24/7 availability, while for other businesses it may stand for availability between for example 8am and 8pm each day. Depending on the service, an outage of 1 second might be insignificant or disastrous. Therefore, the degree of availability should match the purpose and suit the business needs of the company. The service level is the degree of service that shall be provided by the system according to the service level agreement. A service level agreement (SLA) is a formal document of the promise made by the service provider to the customer about service provision policy [10, 11]. It is important that planned and unplanned outages do not exceed this service level degree [12].

Services that are considered as *business critical* are often categorized as high availability services. Computers, software and networks are unreliable, making it difficult to achieve 100% availability. However, systems running business critical services should be planned and designed from the bottom with the goal of achieving the lowest possible amount of planned and unplanned downtime [12]. One example is redundant power supplies on the computers connected different power sources. If the electricity on one power source is lost, the servers will be unaffected and still available. In case of hard-drive failure, a backup disc is connected in mirror RAID[1] that will do the job and leave the users unaffected from the failure. The buzzword is redundancy everywhere. The availability of a system is given by the elements that the system *depends* on. These dependencies have to be analyzed in terms of backup and redundancy possibilities [13]. A critical dependency that acts as a single point of failure should not occur in a high availability system [12]. By making the dependencies redundant, we end up with one single physical computer that is completely redundant with no single points of failure - the result is a fault tolerant computer.

Another approach is to have duplicates of entire physical computers. The result is that we end up with two or more physical computers which can act as backup computers which can failover[2] services from each other. This approach is often used in high availability solutions [8,12,14,15].

2.2 High availability clusters

High availability clusters consist of connected nodes that run specialized software that ties the nodes together and makes them cooperate [12]. The nodes in the cluster act as redundancy for each other so that the critical services are still available even though some of the nodes have failed. There are different open source approaches to this:

The Linux Virtual Server (LVS) project is using an "intelligent" load balancing scheduler at the gateway which forwards requests to the nodes which are known to be healthy. Instead of using distributed high availability software on the nodes in between, the nodes report directly to the front-end gateway to report their status. The gateway keeps track of which nodes that are healthy and which that have failed. Since the gateway is the load balancer that forwards data, it can choose not to forward data to nodes that it knows are not healthy. The motivation of the project is to provide tools to create scalable, highly available, easy manageable and cost-effective clusters. Scalability is achieved by making it possible to transparently add or remove physical nodes in the cluster to decrease or increase hardware resources to cope with demand. Availability is achieved by monitoring at the gateway that keeps track of which nodes that are healthy, and exclude the failed nodes from the resource sharing. Cost-effectiveness is achieved because it is possible to use cheap computers as nodes [16].

[1]Redundant Array of Independent Disks

[2]Failover of a service is that another computer starts hosting the service when the primary computer fails.

Heartbeat is another open source solution to high availability clustering. The basic idea of Heartbeat is that the nodes in the cluster broadcast their status as heartbeats with information about which services are running on that node (for example each second). This way, the other nodes in the cluster are able to know which nodes and services which are up and available and which is not. It is developed by the High-Availability Linux project[3]. The project started as a mailing list in 1997 about how one could proceed if one was to write a piece of high availability software. At this time there was no such thing available. In 1998, Alan Robertson started to implement the essential parts from the mailing list, and the name of the software became *heartbeat* [8]. It featured UDP broadcasts and execution of scripts when failure was detected. Simplicity was, and is still, a part of the system design, where the idea is that simplicity and robustness are required to achieve reliability [8]. Since the first version, Heartbeat has been improved and the list of features has grown.

The nodes in the cluster act as failover nodes for each other. If one of the nodes in the cluster fails, the other nodes collaborate in hosting the affected services to reestablish availability [7]. Heartbeat is further discussed in section 2.5.

Other high availability open source projects are Keepalived, Ultra-monkey and Red Hat Cluster manager. Hewlett Packard provides one of the commercial solutions, initially developed for HP-UX, but later ported to Linux and Windows. It is called HP Serviceguard [12, 17]. Another commercial actor is VMware which provides tools to create high availability clusters with their ESX virtualization software [18].

2.3 Computer virtualization

Virtualization is often used when we want to do something that is not possible in reality, or something which we do not have the necessary equipment for. Most people are familiar with Virtual Reality, where an environment gets simulated on a computer, often for visual experiences.

IBM was the first company to make use of virtualization in server environments. Their motivation was to utilize the powerful hardware in the mainframes as much as possible and to add flexibility. The Virtual Machine Monitor (VMM) was introduced in the late 1960s when IBM wanted to split the resources on their mainframes for multitasking purposes. Today, virtualization is the very basis of their high-end zSeries mainframes [19].

Over the years, consumer hardware has become cheaper per performance which makes it reasonable to use virtualization to fully utilize consumer computers as well. The virtual computers running on a physical computer are

[3]www.linux-ha.org

completely cut off from each other [3], and to communicate with each other, they typically have to use the IP protocol. One can say that the virtual computers are jailed from each other. The advantage is that if one of the virtual computers is compromised by an attacker, the others are unaffected. By running only one service per physical machine, system administrators have minimized the risk of affecting other services if one gets compromised.

Doing this without virtualization can introduce significant costs due to the demand for more hardware and maintaining costs, but using virtualization the services can be hosted on virtual machines running together on one physical host. The physical host is still a single point of failure, but with a minimal and stable operating system that is able to run no more than virtual machines it will probably be more reliable than today's operating systems with complex installations.

Virtualization can be done in many ways, but the most common is to decouple the applications from the hardware by adding a virtualization layer between the hardware and the operating systems (See figure 2.1) [19].

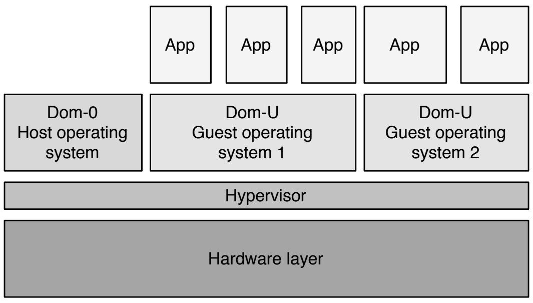


Figure 2.1: The layers in para-virtualization. One physical computer is running three operating systems.

Figure 2.1 shows an example where one single physical computer is running three operating systems simultaneously. Two of them are virtual machines (Dom-U) and one is the host operating system (Dom-0). The abstraction layer (hypervisor) provides encapsulation upward in the layers. The result is that the complete software package running inside a virtual machine is strongly encapsulated, including connection states, CPU states and memory states [3, 19]. If a virtual machine is compromised, it does neither affect the host operating system nor the other virtual machines (figure 2.2).

2.3.1 Virtualization technologies

There are many different virtualization techniques available today, and which to choose depends on the goals of the system. The main differences between them are the degree of flexibility and performance. Flexibility in virtualization reflects the OS dependence and hardware dependence which both are results

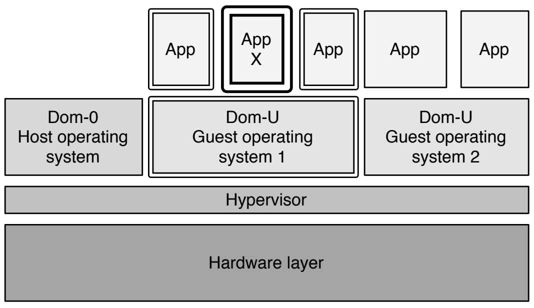

App
App X
App
App
App
Dom-0 Host operating system
Dom-U Guest operating system 1
Dom-U Guest operating system 2
Hypervisor
Hardware layer

Figure 2.2: Application X on *operating system 1* has been compromised, which affects only operating system 1 and its applications. The host operating system and guest operating system 2 are unaffected.

of how decoupled the virtual machine really is from the hardware. The decoupling (abstraction) layer (figure 2.1) adds flexibility, but also overhead. A complex abstraction layer gives high flexibility but lower performance. The different virtualization techniques are:

Full virtualization makes virtual machines (VMs) that are very flexible. The host OS emulates the complete hardware package which are visible for the virtualized operating systems (See figure 2.3), this makes it possible to for example emulate x86 architecture to run Windows on a Macintosh with PPC architecture. The emulator software (virtualization software) creates a layer that smoothes out differences in hardware architectures, and makes it possible to run the same virtual machine on different hosts with different architectures without problems. This gives the flexibility to move entire virtual machines from host to host very easily, but for the cost of performance due to the overhead added by the emulator layer. Tools that use this method are Virtual PC and VMware Workstation.

Full virtualization can be compared to the Java programming language regarding compromising between flexibility and performance. Java programs are cross platform programs, which make the language very flexible. The flexibility is added by the JVM (Java Virtual Machine), which results in overhead. Figure 2.3 can be applied to Java also, where ”Guest operating system” should be replaced with ”Java program” and ”Host operating system” should be replaced with ”Operating system running JVM”.

OS Virtualization is virtualization on the host OS level (see figure 2.4). This method gives very high performance, for the cost of flexibility. The performance of a virtualized operating system is exact the same as the performance of the physical host. This method requires the virtualized operating systems to directly support the hardware used and to use the same

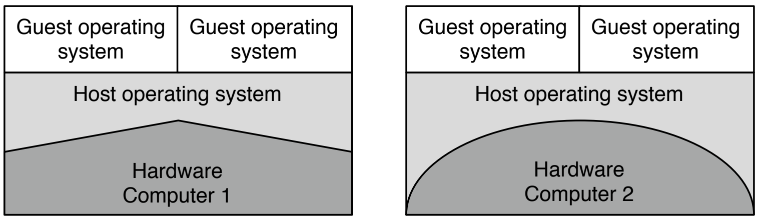


Figure 2.3: Full virtualization technology. The abstraction layer provided by the host operating system smoothes out the differences in the hardware layer, making full virtualized computers independent on hardware architectures. Migration is possible.

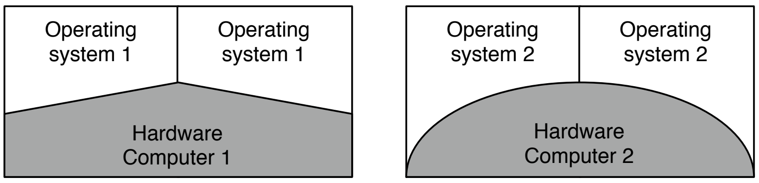


Figure 2.4: Virtualization on the operating system level. Operating system 1 does not support the hardware on computer 2, and migration is impossible.

operating system as the physical host. A tool that does OS virtualization is OpenVZ[4].

Para Virtualization is a method used by Xen [3, 20], Denali [21] and VMware ESX, which can be placed performance wise between full virtualization and OS virtualization (see figure 2.5). The guest operating systems are running in Domain U in parallel with the host operating system that is running in Domain 0. Domain 0 is the privileged domain, which means that the operating system running in it can administer the other operating systems and issue commands to them, e.g. boot, reboot and shut down.

The black layer in figure 2.5 is called the hypervisor software, and it provides little overhead compared to the thicker layers provided by full virtualization. Para virtualization requires the virtualized operating system itself to be ported, but inside the virtualized OS is it possible to run standard binaries and applications for that OS (not ported) [19]. Para virtualization is different from full virtualization because it provides the virtualized operating system to see both real and virtual resources. This increases the performance greatly, and at the same time keeps some of the flexibility (figure 2.6). A physical host running Ubuntu Linux can host other linux distributions with other kernels, or even BSD or Win-

[4]More information at http://openvz.org/

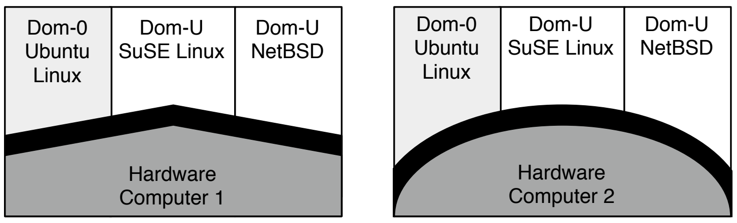


Figure 2.5: Paravirtualization with Ubuntu Linux as host operating system and Suse Linux and NetBSD in two virtual machines. The operating systems running as virtual machines have to be modified to be able to run on the hypervisor [3,19].

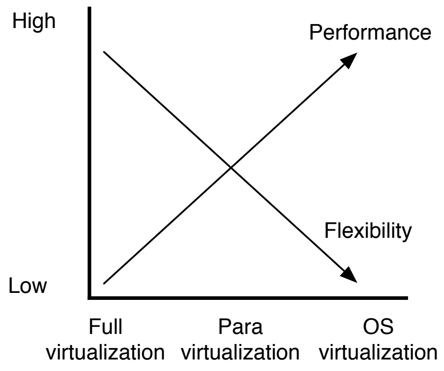


Figure 2.6: Relative performance and flexibility in full virtualization, para virtualization and OS virtualization.

dows operating systems, assuming they are ported. Windows is not yet ported, but BSD is. NetBSD/Xen has been available since 11th of March 2004.

2.3.2 Virtualization advantages

Virtualization has several advantages that can be utilized in high availability clusters. This section aims to elaborate on them.

Flexibility is given in several ways. It is added because one can run more than one instance of an operating system one a single computer, it is possible to migrate a virtualized instance to another physical computer and the virtual instances can be graceful from the host operating system with features like 'pause', 'resume', 'shutdown' and 'boot' [19].

It is also possible to change the specifications of virtual computers while they are running, for example the amount of ram, hard disc size and more [22].

Availability is added because one can keep the virtualized instances running even though the physical node has to be shut down, i.e. for hardware upgrade or maintenance. This is done by temporarily migrating the virtual instances to another computer, and migrate them back when the maintenance is finished and the primary computer is ready to serve. Hardware can be changed, upgraded, maintained and repaired without downtime in the services [2].

Scalability is added because is very easy to add or remove nodes. If the demand for capacity increases over time, it is very easy to insert a physical node with the basic cluster installation, and it will contribute in running the existing virtual machines that run services. This way, the cluster will scale with the company as it expands.

Hardware utilization is most likely increased if more than one operating system is hosted simultaneously. This is because virtual machines utilize hardware resources that are left idle by the host operating system [3,19].

Security is added because greater separation of services is introduced. Using multiple virtual machines, it is possible to separate services by running one service on each virtual machine. This approach is also called jailing of services. If one service is compromised, the other services are unaffected [19,22].

Because break ins are usually done through software bugs in running software, it is important to only run software that are required for the company. Using virtualization, the server would contain a minimal install that could host several virtual machines. Each virtual machine consists of a minimal operating system install and one service, for example the web server. Let us say that the web server is being compromised. The web pages hosted will be unreliable, but the break in does not affect the remaining running services - the database server, mail server and the file server.

2.3.3 Virtualization disadvantages

Virtualization obviously has many advantages, but it also has disadvantages:

Overhead causing decreased performance has been the biggest con with virtualization. Performance is often being compromised due to flexibility, or contradictory, as figure 2.6 shows. As the developers have worked hard to decrease the overhead, the amount of overhead has been reduced. Xen, which is para virtualized, has maximum a few percent overhead [3], bringing it very close to the performance of the standalone physical computer.

SPOF (Single point of failure) in the hardware is still an issue. Even though the virtual machine is decoupled from the hardware, it is still dependent on the hardware working. Failure in the hardware will most likely lead to failure in the virtual machine, which will force a reboot.

The management interface is closely linked to the virtualization platform. This can be a problem as it encumbers consolidation of several platforms into the same environment. There is an ongoing effort to create an abstract management interface at OUC[5] [23].

2.3.4 Using virtualization to create high availability clusters

Traditionally the server software is dependent on the hardware it runs on. This adds very little flexibility when it comes to duplicating services or moving them to other hardware. Moving services can be useful in a heterogeneous cluster where not all of the nodes perform equally well. A low prioritized serviced today might have higher priority tomorrow, and hence is it convenient to have the flexibility to move services to better hardware without too much hassle. Virtualization makes migration of complete operating systems possible because of the abstraction between the hardware and software, with minimal loss of availability [19] (see figure 2.7). Migration of complete virtual machines is relatively simple because the virtual machine and the user applications are treated as a whole with "no loose ends". Process migration, on the other hand, is far more complicated [4].

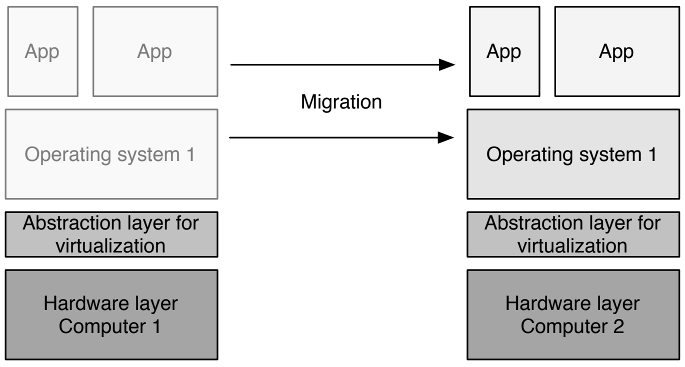

App
App
Operating system 1
Abstraction layer for virtualization
Hardware layer Computer 1
Migration
App
App
Operating system 1
Abstraction layer for virtualization
Hardware layer Computer 2

Figure 2.7: A complete operating system is migrated from computer 1 to computer 2.

Cluster management of a pool consisting of virtual machines running critical services is discussed in [2]. The interesting most part is that the main

[5]Oslo University College

problem is said to be the element that has an overview of the complete cluster and makes the decisions on where the different virtual machines should be hosted. This problem is apparently already solved very well in Heartbeat, previously discussed in 2.2, but for regular services (typically apache, mysql, etc). According to [2], the developer team behind Xen is currently working on a cluster manager that is specialized for Xen, but it has not been released yet.

2.4 Measuring availability

The degree of availability is an important factor in high availability systems. To be able to formalize the degree of availability in the service level agreement, we need a numerical value that represents our demand for the degree of availability. Hence, the service provider needs a method for calculating the availability in the active systems.

Based on historical data, we can calculate how available the system has been, the last week, month or year for example. This can be used as an expectation value of how available we expect the system to be in the future [12]. Knowledge about how often failures occur (MTBF[6]) and the time to repair the failures (MTTR[7]) can be used in this formula for calculating availability [13,24]:

$$Availability = \frac{MTBF}{MTBF + MTTR}$$

The availability of the system is dependent on the failure rate of the equipments. These failure rates are often measured in the mean time before failure (MTBF) for each of the components in the system. The MTBF for the complete system is much lower than the MTBF for single components. It is usual to take the lowest MTBF in the system and divide it on the number of components. This means that failures are more expected to occur in a larger and more complex system [12, 24].

The availability of a system is measured in the famous ″nines″, given in percent of the total time where the system should be available (See table 2.1) [9, 15, 24].

2.5 Heartbeat for high availability

The first versions of heartbeat supported monitoring between two servers only, where one of them (the active one) was used regularly for serving. The passive one would serve as a backup server and all the time check the status on the active server by monitoring the heartbeats. If the active server stopped sending

[6]Mean time before failure

[7]Mean time to repair

Availability:	Total downtime:
99%	3,7 days
99,9%	8,76 hours
99,99%	52,55 min
99,999%	5,25 min
99,9999%	32 sec

Table 2.1: Availability and allowed downtime per year

heartbeats, the passive backup server would take over the IP address of the active one and continue to serve the same content. The passive server now became the active one. If the failed active server came back online, heartbeat would negotiate a failback of the service, so that the initial passive server let back the control to the active one [8] (see figure 2.8).

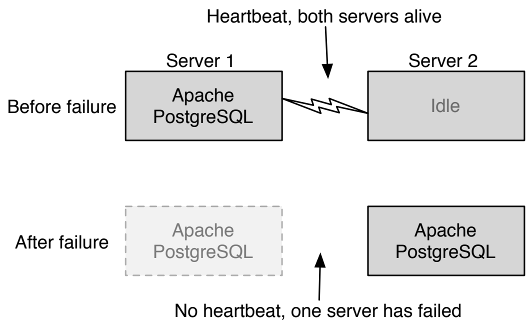


Figure 2.8: Failover in an active/passive configuration with two servers. Only one server serves at all times.

A more advanced usage is having two active servers, each serving different content or services. A very typical example here is one server (a) that is running an SQL server, and one server (b) that is running an apache web server. They are both active since they both are serving. In case of failure of one of them, i.e. if server (a) dies, server (b) would start mysql in addition to apache so that it serves both services at the same time. The necessary changes to the configuration happen automatically. If server (a) resurrects, server (b) would stop its mysql server and server (a) would start it once again. This feature is called autofailback [8] (see figure 2.9).

For service failovers as in these two examples to be reasonable, the IP address should follow the service over to the ”new” server. Otherwise, the service would get a new IP address and be unresolved by the users of the service. IP address failover is done by IPAT (IP address takeover) which is using a secondary IP address or IP alias that is failed over together with the service [14]. In our setup with virtual machines as services, the IP addresses follow the

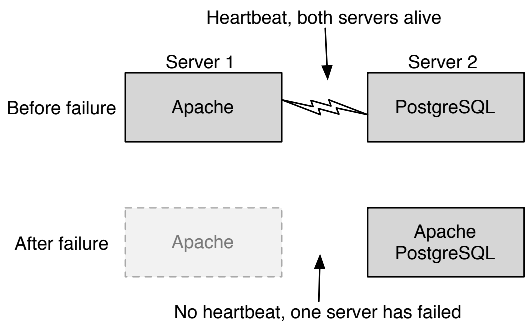


Figure 2.9: Failover in an active/active configuration with two servers. Both servers serve initially.

virtual machines internally without involving heartbeat.

Heartbeat is as of this writing in version 2.0.2 stable, and has gone through lots of changes. The biggest and most important change is that it now supports n nodes in a cluster, from only 2 nodes in version 1. Heartbeat on the nodes in the cluster send broadcast heartbeats to diminish the load of the network traffic, and thus being able to scale better ($O(N)$). Messages sent from every machine to every machine do not scale very well ($O(N^2)$) [7].

For increased security, redundant communication paths for Heartbeat are recommended [7]. In large clusters, this can be solved by adding a dedicated network for Heartbeat to use.

2.6 Xen virtualization

Xen is a virtualization tool for the x86 architecture that is developed at the University of Cambridge Computer Laboratory. It is released under the GNU General Public License (GPL), which means that it is open source and free to use and modify for everyone [5]. An overview of Xen is given in the paper ”Xen and the art of virtualization” [3].

The x86 architecture supports 4 different privileges in hardware that is called *rings* (figure 2.10), numbered from 0 to 3. Ring 0 is the most privileged, and the operating systems take for granted that they can execute code in this ring. However, in virtualization this is not possible. For secure separation between the virtual machines, they have to run in ring 1 [3,20]. The hypervisor is the piece of software that runs in ring 0 and lets the virtual machines validate and execute privileged code through Xen. This hypervisor requires the guest operating systems to be slightly modified, and the reward is a very efficient virtualization platform, in fact it performs very close to the native host [3].

The goal of Xen is to scale up to 100 virtual machines, where all of them run services, on one single physical computer [3]. However, independent research shows that this number of concurrent VMs may be too high. Experiments

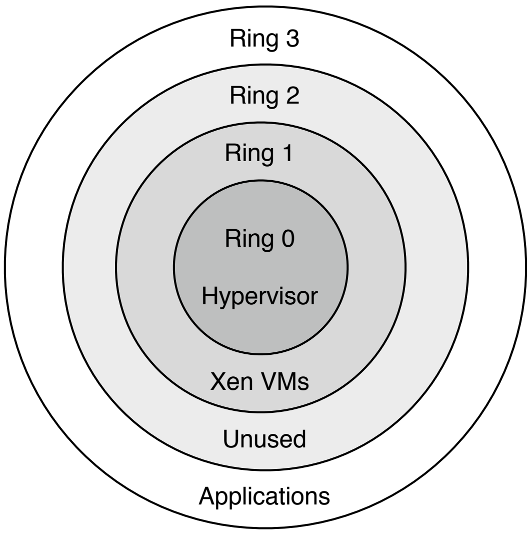


Figure 2.10: The rings of privilege in x86 architecture, as used by Xen.

with web servers reveal decrease in performance with as few as 16 concurrent VMs [5]. Also, due to the memory management in Xen, each of the VMs has dedicated memory allocation, that is not dynamically shared. If we dedicate 128MB ram to each of the virtual machines, 100 VMs will require over 12GB of memory [5].

As figure 2.11 shows, one instance of Xen performs very well compared to running Linux natively, and also compared to competing virtualization tools (VMware workstation and UML). VMware ESX is using para-virtualization and is expected to perform better than VMware workstation, but no research papers exist to our knowledge that is really proving this. VMware has prevented research on performance to be published with to this sentence in the ESX EULA[8]:

> ”You may not disclose the results of any benchmark test of the Software to any third party without VMware’s prior written approval.”

It is indicated, however without any published scientific results, that Xen is noticeably outperforming the VMware ESX server [3]. Our project aims to run few virtual machines on each physical host, approximately 3 concurrent VMs, which can be hosted by Xen with minimal decrease in performance. Xen is well suited for the task as long as the number of VMs running on each host is below 16 [5]. Third party experiments show that Xen is actually 10 times more efficient than UML performing OS-intensive tasks [6].

[8]From the end user license agreement for VMware®ESXserver(tm) and VMware®Virtual SMP(tm) software products

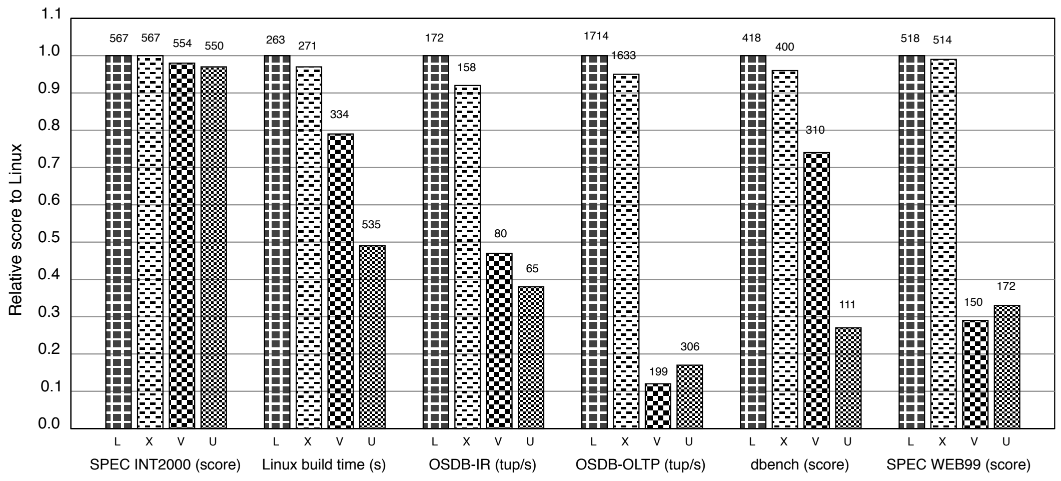


Figure 2.11: Relative performance of native Linux (L), XenoLinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U). OSDB is Open Source Database Benchmark suite, tested with multi-user information retrieval (IR) and online transactional processing (OLTP). *dbench* is a file system benchmark that emulates load on a file server by Windows 95 clients. SPECint is a CPU integer processor power benchmark and WEB99 is a benchmark for evaluating web servers. These are results from measurements in [3]

Migration is a term used when moving something to another place. *Migration of virtual machines* means to move the virtual machine from one physical computer to another as illustrated in figure 2.7. This can be done close to seamlessly to generate as little outage to the users as possible by running the VM while copying memory, and just do a quick "pause - resume" to keep the VM available as possible. This is called *live migration* [3]. The two main requirements for being able to perform live migrations are that:

1. The disk images used by the virtual machines must be accessible from all of the physical computers. This is called shared storage [3].

2. The physical computers must be homogenous, which means that the CPUs must support the same features [9].

Clark et al. [2] have measured the efficiency of live migration in Xen and the results are overwhelming. They managed to live migrate VM running a server of the popular multi-player game quake 3 with 6 players connected with only 60 ms downtime. According to the paper, this downtime was not noticeable by the players.

[9]Several people have reported problems on the Xen mailing lists when trying to live migrate VMs between computers with different CPUs

2.7 Summary of previous research

The advantages in combining virtualization and high availability software are discussed and supported in previous research. Xen provides an exceptional platform of virtualization. It is stable, easy to use, has very high performance and several virtual machines can run simultaneously on one physical computer without severely affect each other due to resource guarantees.

By utilizing the advantages gained by Xen virtual machines in a server environment, the node layout that is shown in figure 2.12 can be achieved. Each physical computer hosts several critical services in the top layer: the illustrated example shows 4 services. These services can have a dedicated virtual computer each (as critical service 1 and 2), or share one virtual computer (as critical service 3 and 4). The advantage of having *one critical service per virtual computer* is the separation between services. If one gets compromised, the other services remain intact. The VMs that have been compromised may potentially be used to perform denial of service attacks to the remaining VMs to affect them, but at least - they are not compromised.

VMware provides a high availability solution based on virtualization, but the drawbacks are high licensing costs, closed source and that it hinders independent research due to limitations in the end user license.

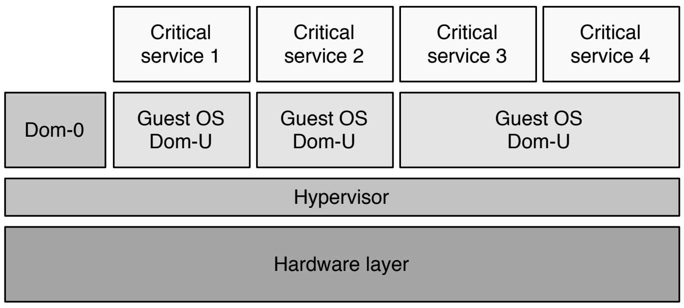


Figure 2.12: Virtualization used for secure separation of critical services. Each virtual machine run as few services as possible to avoid the "domino effect" if one service gets compromised. Dom-0 is running the host operating system that is privileged over the guest operating systems.

An open source solution is needed, but has yet not been created. Cluster management using Xen virtualization is discussed in [2] but it is not yet implemented and published. Heartbeat provides the ability to monitor and control resources, but not virtual machines including the functionality of live migration. Heartbeat is developed through many years

This part is solved by Heartbeat, but for regular services as *apache* and *mysql*. As of this writing, Heartbeat does not support virtual machines. The functionality required in order to make Heartbeat able to live migrate virtual machines is rather advanced compared with the already supported *start* and *stop* functionality, and most importantly, it is lacking today.

Chapter 3

Methodology

The main motivation of the project is to implement a proof of concept high availability cluster using Xen virtual machines and Heartbeat for high availability. Since Heartbeat is lacking the necessary functionality, an add-on package should be developed. The resulting software, Xen and Heartbeat should be used to configure a high availability cluster.

This chapter will introduce the reader to the solution, and go in detail on the equipment and tools used. The validation methodology is discussed towards the end.

3.1 Objectives

The objectives were first formalized in section 1.2. In chapter 2, the objectives got more specified:

1. *Review previous work on high available clustering using virtualization and open source tools.*

   - In previous research, no-one has configured a high availability cluster using open source tools and virtualization, and there exist no documentation on how to do it.
   - Previous research support virtualization used in high availability solutions [2].

2. *Implement an experimental high availability cluster using Heartbeat and Xen.*

   - For successfully configuring a high availability cluster using Heartbeat and Xen, an add-on to Heartbeat which makes Heartbeat able to live migrate virtual machines must be developed.

3. *Do an assessment with a scientific approach to validate the setup and use statistical methods to analyze the results if objective 2 is completed successfully.*

   - The most important property of a high availability setup is its availability, which makes it reasonable to measure the degree of availability to validate the usefulness of the approach. Scientific methods should be used in the measurements.

Task 1 was completed in chapter 2. Task 2 and 3 remain for the next chapters. An add-on to Heartbeat must be developed and implemented, and scientific measurements should be performed to validate the setup.

This chapter discusses the equipment and tools used and will introduce the reader to the methodology used for validation and measurements.

3.2 System model

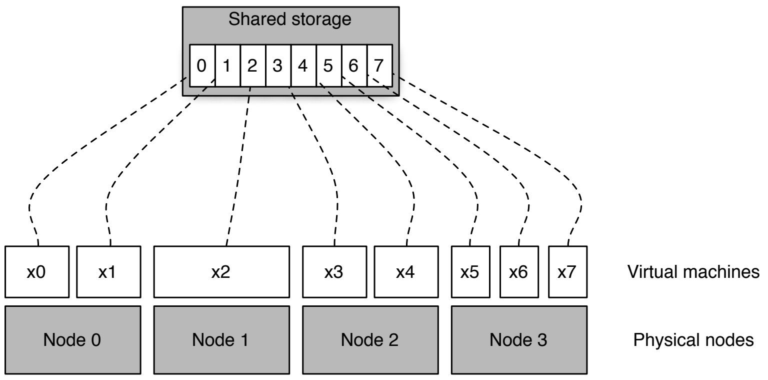


Figure 3.1: The disk images of the virtual machines are stored in *shared storage* that is available for all of the physical nodes. This makes the physical nodes able to run all of the virtual machines, and live migration possible. This example: 8 virtual machines are running on 4 physical nodes.

The key element in the design is that the physical computers create a layer for the virtual machines to run on, so that it is possible to live migrate virtual machines between the physical nodes. As figure 3.1 shows, the nodes (node0 .. node3) are collaborating on hosting 8 VMs (x0 .. x7). One requirement for *live migration* of virtual machines is that the physical computers have access to the same storage area where the VMs have their root and data partitions, making it possible to boot all VMs on all nodes. A common method to do this is to mount the shared storage on all of the physical nodes. To install a new physical node, only a basic installation is required, and it can quickly contribute in hosting the virtual machines. This basic software setup is described in detail in section 3.3.2.

The VMs are decoupled from the physical computers, and even though one physical computer fails, the other computers can collaborate on hosting all 8 VMs (see figure 3.2).

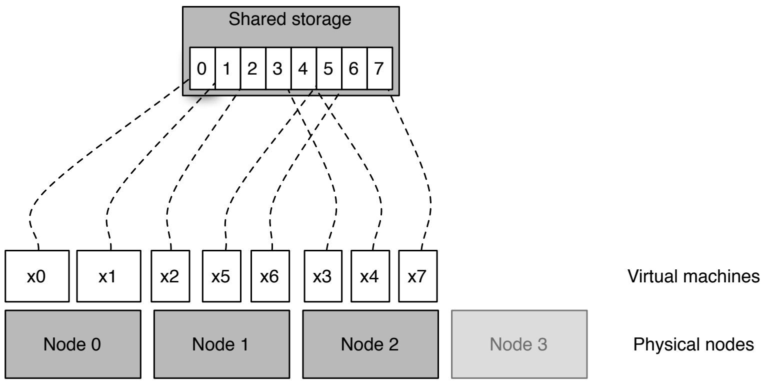


Figure 3.2: Follow-up from figure 3.1. If a physical node fails, the affected virtual computers are *failed over* (live migrated to or booted at) the other physical nodes.

By automating this failover process, we hope to develop a server hosting solution that is superior compared to traditional server hosting solutions when it comes to availability, scalability, ability to change, cost saving and lessened workload on the system administrators. A framework for an assessment of these properties needs to be developed as well.

Heartbeat is designed to control services, not more advanced applications like virtual machines. The functionality required by traditional services are simply just starting and stopping the service, and checking whether it is running or not. Virtual computers require additional functionality when it comes to migration. Heartbeat needs functionality to migrate the running virtual machines over to other nodes in case of failure of the active node. This functionality is illustrated in terms of the missing "link" in figure 3.3. This "link" has to be developed for the system design to be fully functional.

3.3 Tools and equipment

3.3.1 Hardware

We use standard commodity x86 computers to do this experiment. This is because they are cheap, available and make the experiment easy to reproduce. The hardware is installed in the test laboratory on Linpro.

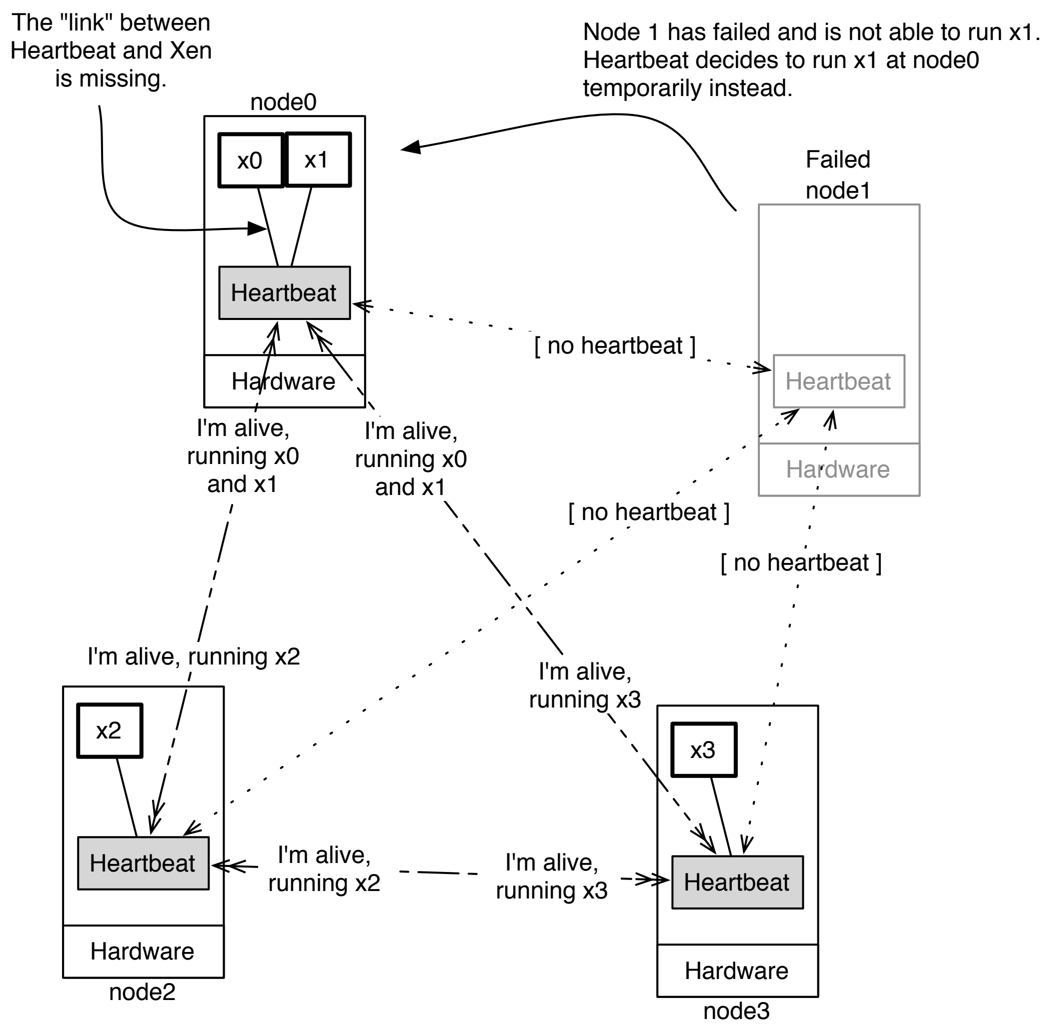


Figure 3.3: Heartbeat and Xen in a fully functional setup where four physical computers are collaborating on hosting four virtual machines. *node1* has failed and *node0* has taken over *x1* to keep it available.

```
The gateway:
  Intel(R) Pentium(R) 4 CPU 3.00GHz
  2GB memory
  Barracuda 7200.7 SATA 80GB in RAID-1

Four nodes:
  AMD Sempron(tm) Processor 3100+ (1800MHz)
  1GB memory
  Samsung HD040GJ SATA 40GB

1 Gbps internal network between the nodes and the gateway
```

The nodes are connected to an APC power distribution unit (no. AP7921) which makes it possible to control the power to the nodes remotely. If a node freezes during the experiment, it is still possible to shut it down and then boot without physical access to the server room. It is also being used to test the high availability failover using real power failure.

Equipment in production environments

The equipment used in this test is not suitable for a high availability solution in production environments. Equipment in production should have more redundancy and higher fault tolerance, i.e. RAID mirror root disks on the nodes, redundant internal network, SAN[1] and redundant gateways with failover capability.

The motivation for our setup is to create a test environment that is suitable for experiment and analyze the topology we propose. Other and more expensive equipment is needed if the solution is going to be put in production mode.

3.3.2 Software setup on the physical nodes

The physical nodes have a total of 1024MB memory, where 128MB is dedicated for Domain-0, the host operating system. The rest is for Xen to use exclusively. The virtual machines are configured to require 256MB of memory each, which means that each node can host maximum 3 virtual machines simultaneously.

The nodes are configured equally, no matter which virtual machines they should host. The configuration is made as simple as possible and consists of as few software packages as possible. The software packages are:

Ubuntu Linux

Ubuntu Breezy Badger GNU/Linux is the operating system used on all of the nodes (in Domain-0), virtual machines (Domain-U) and at the gateway. The installation is default and includes GNBD:

[1]Storage Area Network

GNBD (Global Network Block Device)

A requirement for live migration is that each of the nodes has access to the same data. This is done by using shared storage. GNBD provides block level storage over the network which can be shared between multiple hosts. Our gateway acts as a GNBD server that exports a LVM[2] partition where all of the virtual machines have dedicated smaller partitions for exclusive usage. The root filesystem of the virtual machines are stored in these partitions on the gateway (see figure 3.4). The nodes run GFS[3] to import the shared storage as block devices. A more comprehensive illustration can be found in figure A.1 in the appendix.

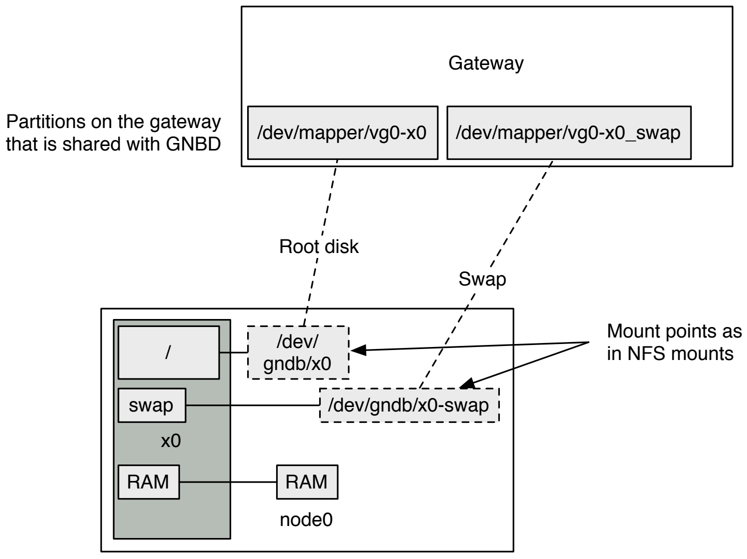


Figure 3.4: Shared storage using GNBD (A more complete illustration in A.1)

Since the nodes have access to the same data storage, the only necessary components to copy during a migration is the memory, CPU states and connection states.

Xen

The very latest version of Xen in time of writing is used, version 3.0.1 unstable. Xen is installed on each of the nodes in the cluster. Each virtual machine is configured by one configuration file, making it a total of 8 configuration files for 8 VMs. These 8 files specify the amount of RAM, network interface setup and which disk images to use. The files are distributed to each of the nodes using

[2]LVM is a Logical Volume Manager for Linux that provides more flexibility than traditional disk partitioning

[3]Global File System

symbolic links from the shared storage to make it easy to perform consistent changes in the future.

The configuration file for one of the VMs can be found in appendix A.2.

Heartbeat

The latest stable version of Heartbeat in time of writing is used, version 2.0.2. Heartbeat is configured using 2 configuration files[4], and it is required that all the nodes use the very same files. The configuration files used for controlling used throughout this project can be found in appendix A.3.

To get familiar with heartbeat and test different configurations, heartbeat was installed in a test environment, separated from our experiment environment. It was necessary to get to know how it worked, before trying to implement it with Xen. A cluster of 5 nodes was built using MLN[5] and UML[6], where the nodes cooperated in hosting *apache*, *named*, *mysql*, *exim4* and *vsftpd*:

```
 -> node1 : apache is running
 -> node2 : named is running
 -> node3 : mysql is running
 -> node4 : exim4 is running
 -> node5 : vsftpd is running
```

Each node hosted one service. When failure occurred to node 2 and 4, heartbeat started the affected services on the other available nodes:

```
 -> node1 : apache is running
 -> node1 : vsftpd is running
 -> node3 : named is running
 -> node5 : mysql is running
 -> node5 : exim4 is running
```

Named and *exim4* were previously hosted at node2 and node4, but when those nodes failed, Heartbeat rearranged the cluster to run all of the applications on the remaining nodes. *Named* was started at node3 and *exim4* was started at node5. This way, only three nodes were running - but all five services were still available. This is the standard usage of heartbeat.

3.3.3 Software setup on the virtual machines

The virtual machines should represent fully functional virtual machines as they could have been configured in a production environment with a realistic service running as a critical service. The services chosen for this purpose are:

[4]Configuration files for the Heartbeat daemon: */etc/ha.d/ha.cf* and the cluster resource manager: */var/lib/heartbeat/crm/cib.xml*

[5]http://mln.sourceforge.net/

[6]http://user-mode-linux.sourceforge.net/

Internet radio station

IceS 2.0.1 and Icecast 2.3.1 are installed at the virtual machine *x0*. Icecast is a streaming media server which supports Ogg Vorbis and MP3 audio streams. IceS is the component that provides the audio content to Icecast which is the serving component that provides the audio stream to the listeners (see figure 3.5).

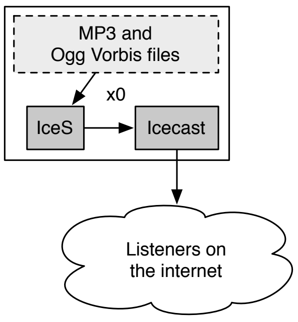


Figure 3.5: Components in internet streaming radio using IceS and Icecast

An audio stream is a continuous stream of data that requires continuous connectivity between the server and the client. However, failures can be hard to detect with a internet streaming radio because client side buffering may conceal the outages using a large buffer. If the outage is short, the buffer might be sufficiently large so that the failure is not detected at all.

IceS is configured to play audio content in ogg format and output it to IceCast in 160kbps bit rate with a sample rate of 44,1KHz in 2 channels. IceCast is streaming the audio feed in ogg format from port 2002 using SHOUTcast to stream audio over the TCP protocol.

Online multiplayer game server

BZFlag[7] is a free open source multiplayer multi-platform game. A BZFlag server is installed at the virtual machine *x3*. Online multiplayer games are very sensitive to failures and outages because so much data are sent back and forth to update the gameplay on all clients real time.

This service should be more sensitive to outages than radio streaming since multiplayer games do not use buffering. The server is configured to use the UDP protocol on port 5154.

[7]http://www.bzflag.org

MySQL server

Databases are natural components in business environments, and are often parts of business critical computer systems - for example in transactional systems. MySQL 14.7 is installed at *x2*. Mysqld is configured to listen for remote requests on port 3306 and is using the TCP protocol.

Apache and PHP

Apache 2.0.54 with PHP 5.0.5 is installed at *x1*. Traditional browsing on a web server is not very sensitive to short outages compared to real time multiplayer games and radio streaming since the data transfers are not continuous. If a web site is using a couple of seconds to load, few users will notice.

Apache is configured to listen for request at port 80. The mysql module for php is installed, making it possible to use the mysql server installed at *x2* in php scripts at *x1*.

3.3.4 Development software

The integration between Heartbeat and Xen will be implemented using Perl, a very powerful scripting language that supports network communication through the IP protocol.

3.4 Topology in production environment

In production environments it is important to analyze dependencies (single points of failure) and make them fault tolerant by redundancy. As figure 3.6 shows, the complete internal network, network components and internet connectivity are redundant and the nodes have mirrored root disks. The only single point of failure here is the shared storage.

Since migration of virtual machines depends on shared storage (as of this writing), it is not redundant in the topology diagram. However, many shared storage solutions are very secure, and have built in redundancy in terms of redundant controllers, disks and cables. The connection to the nodes can also be made redundant, even though only a single storage solution is used. Note that this storage solution implies significant increase in cost compared to most other alternatives. It is possible to use dual SANs as well, and implement synchronization between them, but whether the added reliability justifies the increased cost needs to be further analyzed.

In this topology, it is easy to add or remove physical nodes and it is very scalable. The only configuration files needed on a new physical node are the heartbeat configuration files, the virtual machine configuration files and the

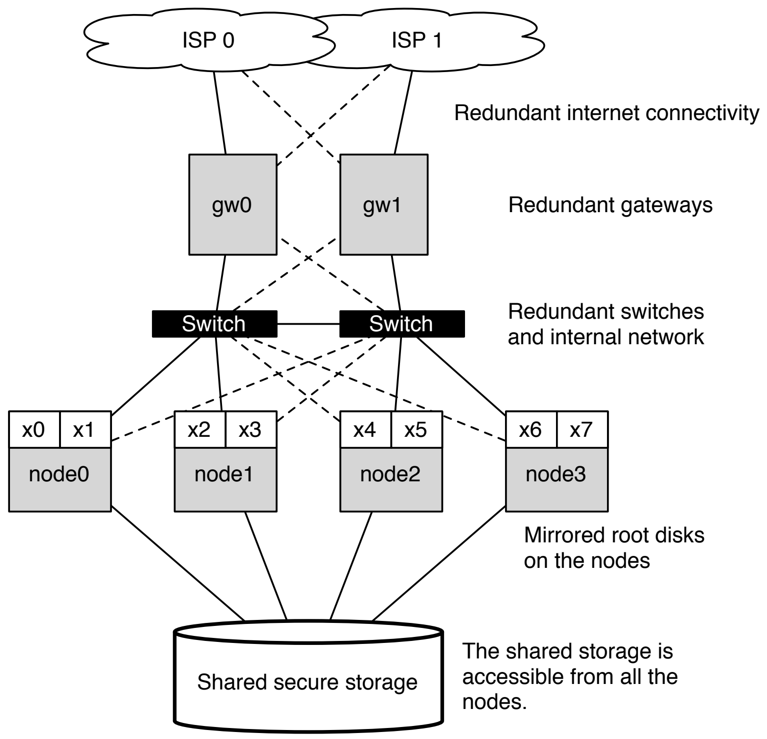

ISP 0
ISP 1
Redundant internet connectivity
gw0
gw1
Redundant gateways
Switch
Switch
Redundant switches and internal network
x0
x1
x2
x3
x4
x5
x6
x7
node0
node1
node2
node3
Mirrored root disks on the nodes
Shared secure storage
The shared storage is accessible from all the nodes.

Figure 3.6: Topology in production environment

shared storage configuration files. These can be pulled to the nodes using a configuration management tool, like *cfengine*[8]. Virtual machines can be added, or increase in capacity (memory, CPU prioritization and disk space) , within seconds. Disks can be added, removed or exchanged on the fly in the shared storage without affecting availability.

The operating system on the physical nodes can be hardened to drop all incoming and outgoing network traffic, if that is desirable. This makes it impossible to modify it without physical access to the server room.

3.5 Topology used in the experiment

Due to this being a experimental setup and because there are no experience on this field using these tools in this approach previously, it was not reasonable to buy the kind of high-end equipment that should be used in production environments (figure 3.6). Linpro supplied brand new commodity computers for this project exclusively to use.

The topology used in this experiment is showed in figure 3.7. The figure reveals many single points of failure, but this is of no consequence because the

[8]www.cfengine.org

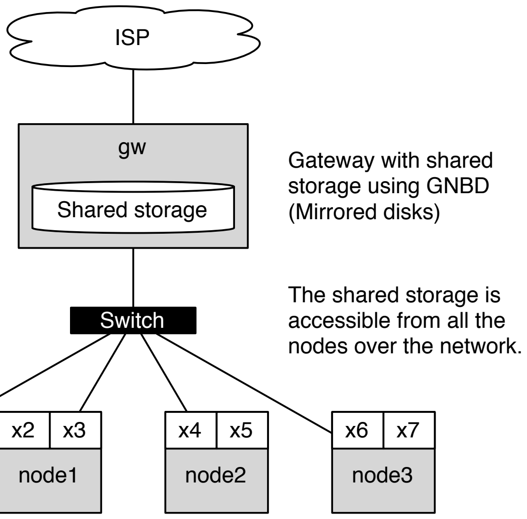

ISP
gw
Shared storage
Gateway with shared storage using GNBD (Mirrored disks)
Switch
The shared storage is accessible from all the nodes over the network.
x0
x1
node0
x2
x3
node1
x4
x5
node2
x6
x7
node3

Figure 3.7: Topology in experimental environment, where *node0..node3* are the physical computers and *x0..x7* are the virtual machines.

experiments will be about convergence towards the stable state when unavailability has occurred, and not measuring fault tolerance in regards to redundant hardware.

3.6 Definition of failures

The key challenge in this project is to integrate Xen with Heartbeat, so that Heartbeat can control the virtual machines automatically and execute migrations. Heartbeat is designed for purely stopping and starting services, which would lead to unnecessary high downtime in some failover scenarios. By analyzing different kinds of computer failures in regard to heartbeat, all could be fitted into one of two categories:

Uncontrolled failure. These failures are typical critical failures as for example power failure, hardware failure causing the system to freeze and network failure. An uncontrolled failure is instant, and affects the system immediately. When an uncontrolled failure occurs on a node, volatile memory like RAM is lost. The preferred method for repair is to simply cold boot[9] the affected virtual machines on the other nodes.

Graceful failure. Scenarios in this category are not necessarily failures, because they can be controlled. An example may be when the system administrator shuts down the node or reboots it, or simply if the UPS[10]

[9]Cold boot is booting the virtual machine from a powered-off state.

[10]Uninterruptible power supply

shuts down a node gracefully due to power failure. When a graceful failure occurs on a node, the preferred method for repair is to seamlessly failover the virtual machines running on that node over to other nodes in the cluster by live migration.

These two types of failures are illustrated in section 4.2.2.

3.7 Validation methodology

3.7.1 Validation through scientific measurements

A high availability service should be available according to the service level agreement, as stated in section 2.1. It depends on the service which kinds of outages are allowed and which is not. If a web server has an outage of 1 second, it is rarely noticeable. However, if a VoIP PBX gets unavailable for 1 second, it is annoying to the users. As such, the experiments are done by measuring the outages scientifically and discussing the results in respect to the service hosted by the virtual machine that had the outage. If the users do not notice an outage, it is harmless and does not affect the service level provided, although it may be measurable by scientific methods.

The availability of a system is measured using MTBF and MTTR (section 2.4). This methodology is difficult to use when verifying a topology, because MTBF and MTTR is component dependent. In our setup, we use commodity hardware for testing only. Production environments will have more reliable hardware and redundant setup, causing the MTBF to increase and MTTR to decrease.

The advantage in the proposed topology is the introduction of *live migration* as means to repair failures very quickly. Hence, measurements of the time to repair (MTTR) failures are interesting. If MTTR approaches 0, the availability will approach 100% even though MTBF is relatively low:

$$Availability = \frac{MTBF}{MTBF + MTTR}$$

Results from [2] show that an outage caused by one live migration in Xen is causing less than 1 second. If we set MTBF to 24 hours, and uses 1 second for MTTR, the availability will still be 99,999% which is very good considering one failure, although graceful, each day. Table 2.1 shows that 99,999% availability translates to 5:25 minutes total outage in one year. It should be noted that in many scenarios, several outages of one second is preferred over one outage of 5:25 minutes. Depending on the service, outages of 1 second can be difficult to notice for the users. If the users do not notice the outages, the experienced availability becomes 100%.

3.7.2 Selected measurements

The measurements in this project are limited to graceful failures due to the time limitation of this project. A graceful failure is for example when a physical computer is being shut down for maintenance. In traditional setups, a shutdown can result in long outages. The proposed topology is expected to be superior in terms of availability and MTTR in these kinds of failures, and such it should be scientifically measured. Future research should be done on uncontrolled failures. The selected measurements are:

Outage in time caused by a graceful failure. For how long time a service is unavailable when a graceful failure occurs. The mean value will indicate the MTTR from a graceful failure.

Outage in packet loss caused by a graceful failure. How many packets that are lost in the outage when a graceful failure occurs. This is important for UDP traffic, and can be used to calculate outages in time based on the mean number of packets per second that are sent and the number of packets lost per graceful failure.

The impact on network performance caused by a failure. A graceful failure leads to live migration of a virtual machine. Since the memory has to be copied in the migration process, the network performance might be decreased during migration.

The impact on VoIP services caused by a graceful failure. If a failure occurs in a traditional setup, the conversions will be broken. It is interesting to see how VoIP services are affected by graceful failures in our high availability cluster.

The impact on media streaming services caused by a graceful failure. If a failure occurs in a traditional setup, internet radios and online games will be aborted. When the failure is repaired, the users have to re-connect to the services. It is interesting to see how the users of internet streaming radios and online multiplayer games are affected by a graceful failure in our high availability cluster.

The impact on file transfers caused by a graceful failure. There are lots of file transfers taking place at all times, and a failure usually results in the file transfer being aborted. How are file transfers affected by graceful failures using the proposed topology?

The impact on web services caused by a graceful failure. How a web server is affected when a graceful failure occurs. Response times from the clients give a good indication on the performance of web servers.

These measurements are described in detail in section 5.

3.7.3 Future measurements

The following measurements are interesting, but are categorized as future work due to the time constraints in this project.

The impact on other VMs when a failure occurs. To measuring the performance on one VM when a failure occurs to another physical node that is running another VM. The network performance might be temporarily decreased and also the computation power if the VM is migrated to the same physical node as the VM that is measured.

The impact on outages from graceful failures caused by ARP updates. When a VM is migrated to another physical machine, the ARP cache table on the switch is likely to be updated. How does this affect the outage?

The outage caused by uncontrolled failures. Measure the outage caused by a instant power failure to one of the nodes in the cluster. The results can be used to optimize the Heartbeat configuration.

Availability with two layers of HA software. A two layer high availability topology is proposed in section 6.3.1. Outages caused by uncontrolled and graceful failures should be measured. They are expected to be equally short.

3.7.4 Tools used in the measurements

Due to the packet switched networks in computer environments, it is not possible to continuously measure the status of a connection like in circuit switched networks. We can monitor the connection by transferring packets and checking whether the packets arrive.

Pktgen [25, 26] is a kernel module to the linux kernel which is a highly effective UDP packet generator. It is used to measure the packet loss during failover. The installation and configuration process of pktgen is described in appendix A.4 and A.5.

Siege[11] is a benchmark testing utility that is suitable for generating traffic towards http servers. It calculates valuable output in regards to the server performance.

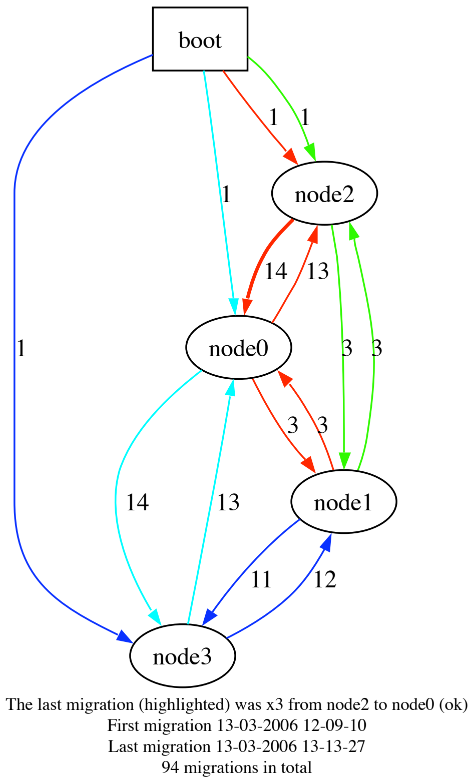
3.7.5 The Virtual Machine Migration Monitor

A virtual machine migration monitor was implemented in PHP to make it easier to monitor the failover history in the cluster. A state diagram is auto-

[11]http://www.joedog.org/

matically updated real-time on a web page to provide the user with a quick and easy way to get an overview of the last events.

Each failover is illustrated by an arrow between two physical nodes, shaped as ellipses. The color of the arrows illustrates which virtual machine that the arrow represents. There is a number at each arrow that illustrates how many times the specific failover has occurred. Each failover is also registered by the timestamp of the event, and it is possible to browse between the migrations to get a better overview of what has happened previously: Which virtual machines that got migrated from where, to where and at what time (see figure 3.8)



```
The last migration (highlighted) was x3 from node2 to node0 (ok)
First migration 13-03-2006 12-09-10
Last migration 13-03-2006 13-13-27
94 migrations in total
```

Figure 3.8: Output from the *Virtual Machine Migration Monitor*. The ellipses illustrate physical nodes and the arrows illustrates migrations, where the different colors represent the different virtual machines: x0 is green, x1 is blue, x2 is light blue and x3 is red.

Node ranking is a method for calculating the importance of each node in a network by looking at the transitions. In our case, the transitions illustrate migrations. The methodology for calculating the node ranking is explained in [13].

Heartbeat is deciding which nodes that should run the different virtual machines based on the amount of available hardware resources on each physical node. Using knowledge about node ranking, we can locate the nodes that are not currently preferred by Heartbeat and perform direct hardware upgrades on these nodes exclusively to make them more attractive for Heartbeat to use. This will maximize the profit of new hardware investments.

A feature in the *Virtual Machine Migration Monitor* that automatically calculates the node ranking is not currently implemented, but is straightforward to implement in the future. Our experimental setup is homogenous and the physical nodes are configured to be equally important, which makes this feature extraneous in this first stage of the research on this topology.

Chapter 4

System design analysis

The reader will be presented for the development process of the proposed add-on to Heartbeat. This add-on will provide the failover functionality of *live migrating* virtual machines in addition to the simple *stop* and *start* functionality. This chapter will complete objective 2 from section 3.1.

4.1 Prerequisite knowledge

Heartbeat was chosen because it is open source and is a well tested solution for high availability in Linux. It has been around for many years, and is used by many companies in production environments. One problem is that it is not very well documented. Writing a piece of production environment software like Heartbeat from scratch involves serious testing for stability over time, and therefore a stable solution like Heartbeat was chosen over the alternative of ”reinventing the wheel”.

4.1.1 Heartbeat decision making

The heartbeat daemons running in the cluster communicate with each other at all times. The logging is very verbose, and one can learn a lot about the behavior of Heartbeat by looking at the log files. A log entry is added as soon as something abnormal is happening, e.g. if a single heartbeat is delayed.

If the Heartbeat daemons have quorum (see section 4.1.5), they elect one node to be the cluster master. The cluster master is called the DC. If the DC for some reason disappears[1], a new DC will be re-elected at once by the remaining nodes. The DC keeps control over the resources (virtual machines) and the physical nodes, and it decides which physical hosts that should host which virtual machines.

[1]For example if a power failure occurs on the node that is currently DC

The Heartbeat daemons report their hardware usage to the DC, which sorts the list of nodes by resource usage. If a new service is about to be started, the DC will select the physical node with the most available hardware resources to host the service.

It should be noted that this decision making is dependent on the configuration files. The configuration files are very flexible, and one can specify the degree of preference on which nodes that the services should run on. The configuration is influencing the decisions greatly.

In our setup, each node are equal (homogenous network) and the virtual machines are not preferred to run on any particular node. However, in a heterogeneous network, preference can be used to further utilize the hardware resources or to reserve physical nodes to specific customers. If one customer is paying extra for more hardware resources, it is fair that the customer should be prioritized to use that extra hardware. This can be valuable if e.g. one customer needs hardware that is better suited to run databases than other customers.

Heartbeat can be configured to reserve hardware to specific virtual machines in policy state, but share the hardware to avoid outages when failures occur.

4.1.2 Heartbeat functionality

The simplest Heartbeat configuration (using Linux Standard Base, LSB) is controlling regular services by using *init.d scripts* using the arguments *start*, *stop* and *status*. These are for starting and stopping the services respectively, and for checking if the services are running or not. Heartbeat uses the output of the command to check if the command was successfully issued. The *init.d* scripts should follow the standardization given by the Free Standards Group[2]. Using the existing functionality, it is expected that it easy to configure the cluster with *start* and *stop* functionality on the virtual machines. In that case, *start* means boot the virtual machine, and *stop* means shutdown. However, it is expected that it will be a challenge to expand this functionality (*start*, *stop* and *status*) of Heartbeat to support *live migration* of virtual machines.

In Xen, the *start* and *stop* commands are respectively:

```
xm create x0
xm shutdown x0
```

4.1.3 Heartbeat internal messaging

It is difficult to extract information from Heartbeat about its future behavior at runtime because documentation how to extract messages does not exist. One

[2]http://www.freestandards.org/en/LSB

example of an internal message which is important for our project is *which of the physical nodes the virtual machines are going to be failed over to during a graceful failure*. It is expected that this might be an issue when trying to get Heartbeat perform live migrations with the built in migration command in Xen:

```
xm migrate x0 node0 --live
```

For this command to be used, the script that executes the command needs to know where the virtual machines should be migrated to. The command in the example will live migrate the virtual machine *x0* to the physical node *node0* from the node where the command is issued using PUSH[3] methodology.

4.1.4 Shared storage integrity

The disk images of the virtual machines are stored in the shared storage, making them able to be booted on all of the physical nodes. A risk that is introduced is that the same virtual machine can be booted simultaneously on more than one physical node. This means a severe risk of the disk integrity for that virtual machine to be broken. Various tools exist for coping with this problem. Two techniques that are supported by Heartbeat are called *node fencing* and *STONITH (Shoot The Other Node In The Head)*.

In our experimental setup, fencing and STONITH are not implemented as the project could be carried through without since the ”critical data” on the virtual machines are not really critical. This makes it possible to boot the same virtual machine from more than one physical computer concurrently - using the same disk image. Our virtual machines are fairly idle when it comes to changing data on disk. It is not considered as a problem since this is a experimental setup, but it is important to understand if corruption of data on the virtual machines occur.

Fencing is using I/O blocking to block out all the nodes by default from the shared storage and only allow access to the single physical computer that should have access to the disk images of the VMs that it hosts.

STONITH sounds brutal, and it is. When a node stops reporting that it is alive, Heartbeat makes sure that the node is dead by ”shooting it in it’s head”. One example of STONITH is power management. The scenario can be that the remaining Heartbeat nodes make sure that the supposedly dead node really is dead by cutting its power or network connection.

In production environments, these must be configured in order to ensure that the integrity of the disk images of the virtual machines stays intact.

[3]PUSH is a method for content delivery. The virtual machine is delivered from the existing physical node to node0.

4.1.5 Split brain

Split brain is the term used when two equally dominant groups disagree on which decision to make. This can occur if a cluster of even size is being split in half and then repaired. Both parties (the halves) are unable to decide which of the parties that should be the decision making part.

In a larger production environment with many physical nodes, many nodes have to fail for split brain to occur - and it will probably not be a problem at all. In the cluster used in this work, however, we have 4 physical nodes. If two of them fail, a split brain will occur. It is not considered as a problem since this is a experimental setup, but it is important to understand split brain when it occurs.

Heartbeat requires a quorum where over 50% of the nodes are available to be able to make decisions as of this writing. In our scenario, if 2 of the nodes fail, Heartbeat will be unable to make any decisions. This is implemented as a security feature to avoid split brains.

4.1.6 Heartbeat configuration and migration intensity

The behavior of Heartbeat is pretty much given in the configuration files (*cib.xml* and *ha.cf*). The Heartbeat configuration has a great impact on the properties of the cluster, since Heartbeat is the software package that controls all the critical services (in our case, the virtual machines).

When a node fails, the *failover* process is initiated to migrate the VMs to other nodes. When the failed node is repaired and restored to life, the *failback* process is initiated to migrate the VMs back to the repaired node. The result is at least two migrations per node failure, often more to balance the VMs between the nodes that are still alive. The result, after some failures and many migrations, will be a *balanced cluster* (Square no. 2 in figure 4.1). An illustration showing the amount of migrations is given in figure 4.2. The cluster is *balanced* when the hardware resources available are reasonable allocated to the virtual machines that are running.

The *failback* process is not required, and can be skipped in the configuration if the number of migrations should be as low as possible. The result, after some failures, will be a *unbalanced cluster* (Square no. 3 in figure 4.1).

Figure 4.2 and figure 4.3 differ from each other clearly in the amount of migrations carried through. Which of the configuration models that suits the system best, depends on what is prioritized, stability or performance. Optimizing the heartbeat configuration is not a part of this experiment. Throughout this project, the balanced cluster model is consequently used (square no. 2).

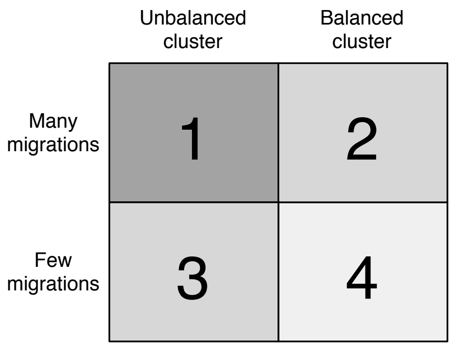


Figure 4.1: The ultimate goal is to achieve a balanced cluster with few migrations (4).

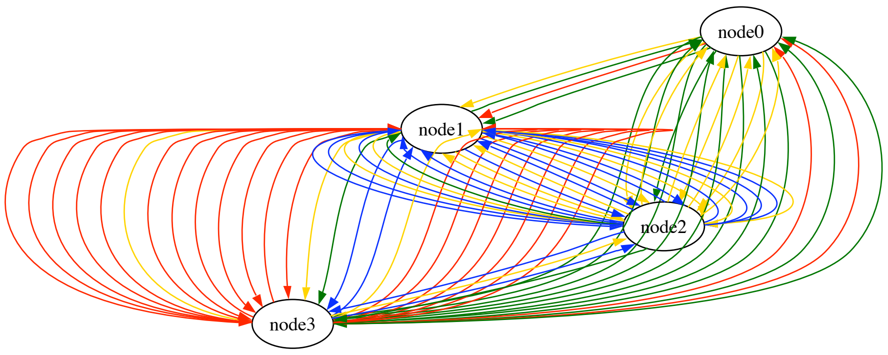


Figure 4.2: 30 failures are simulated in our network that consists of 4 nodes and 4 VMs. Heartbeat is configured to keep the cluster balanced, and carries through 77 migrations to do so. One arrow illustrates one migration, and the different colors illustrate the different virtual machines.

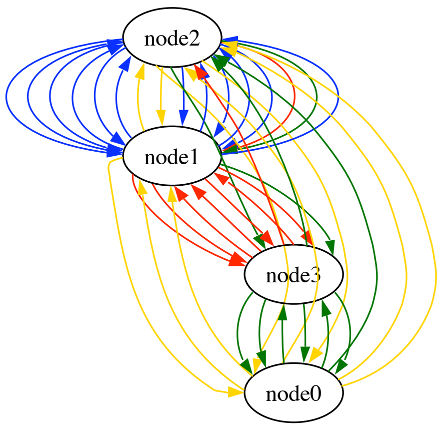


Figure 4.3: 30 failures are simulated in our network that consists of 4 nodes and 4 VMs. Heartbeat is configured to keep the VMs running on the nodes until failure of the node, resulting in a total of 42 migrations. One arrow illustrates one migration, and the different colors illustrate the different virtual machines.

4.2 Xen integration in Heartbeat, a novel approach

As uncontrolled failures are simply about *shutting down* and *starting* resources [4] (see section 3.6), it is straight forward to implement it with Heartbeat (discussed in section 4.1.2). In a failover process of the virtual machines *x0* and *x1* from the physical node node0 to node1 triggered by an uncontrolled failure, the virtual machines running on node0 would simply be shut down if possible, and afterwards booted on node1. The virtual machines would be unavailable in the interval between *shut down* was initiated on node0 to the *boot process* finished on node1. In static production environments, with little interference from the system administrators, this functionality is probably good enough. The only failures that occur are uncontrolled anyway.

In dynamic environments, however, where the nodes in the clusters are constantly maintained, upgraded, removed or added, a failover solution that results in less downtime is required. Better technology is constantly being developed, providing better performance per money than ever before and it is natural to adapt the servers to cope with higher demand, for example by upgrading the hardware. Hardware upgrades are probably done several times each year, where the nodes have to be shut down resulting in many *graceful failures* as shown in the following real-life example.

A company expands with its customers. The company has in the beginning of the year 20 customers hosted in an environment using our topology. 5 servers are used to host 20 virtual machines running critical services plus 1 server that is kept as a hardware backup server.

Throughout the year, the customers are expanding, resulting in increased demand on the servers. Our company acquires 2 new servers and upgrades the existing servers to the newest technology. The result is 7 top-of-the-line servers that serve 20 customers with high demand of capacity.

The adaptation from 5 low-end servers to 7 top-of-the-line servers went seamlessly and without any downtime to the critical services.

For this example to become reality, we need to utilize the live migration functionality included in Xen. It is proved to provide seamless migrations of the virtual machines between the nodes very effectively [2].

To be able to integrate Xen with Heartbeat, we need to understand the procedure of a failover processes in Heartbeat triggered by a graceful failure. An example is used to give a good overview:

[4]No migration involved since they occur instantly

Our network consists of 4 physical nodes collaborating on hosting 8 virtual machines. The physical nodes are currently hosting 2 virtual machines each. We want to shut down node0, and the virtual machines that it is hosting (*x0* and *x1*) should be migrated to other nodes.

1. We initiate shut down on node0.
2. When Heartbeat on node0 shuts down, it notifies the Heartbeat instances on the other physical nodes in the network that it is going down and which virtual machines that x0 and x1 are currently running on node0.
3. Heartbeat on node0 is initiating */etc/init.d/x0 stop* and */etc/init.d/x1 stop* to *stop* the virtual machines.
4. Heartbeat on the other nodes decides[5] which of the nodes that should failover *x0* and *x1*.
5. In this example, node1 was chosen to host *x0* and node2 was chosen to host *x1*.
6. Heartbeat on node1 performs */etc/init.d/x0 start* and heartbeat on node2 performs */etc/init.d/x1 start*.

Heartbeat gives little flexibility in manipulating these steps. In future releases of Heartbeat higher flexibility might be added, but as of now, the only supported action to take on a service is */etc/init.d/service start* and */etc/init.d/service stop*. The scripts used for *start* and *stop* are, however, fully configurable. Implementing the Xen command for migration can be done in either the *stop* section or *start* section of the scripts. Note that, as the sequential list shows, that the failover nodes initiates *start* after the failed node initiates *stop*:

stop The most evident way of implementing migration is to do it in the *stop* script, by performing a PUSH of the virtual machine to the failover node. Instead of shutting down the virtual machine, it would then perform a migration. The problem is that we need to know the hostname of the failover node, and as the sequential list shows, heartbeat decides after the *stop* script has finished which node that should act as the failover node.

start When heartbeat is executing the *start* script on a node, it has already chosen that node as the failover node. As the sequential list shows, the failing node has already executed the *stop* script to shut down the virtual machine.

[5]How decisions are made are explained in section **??**.

However, if the *stop* script simply leaves the virtual machine unaffected, and that it is still running when the failover node executes the *start* script. The *start* script can then send a message on the network to the failed node asking it to migrate the virtual machine to the failover node. This method is using PULL (the reverse of PUSH). See section 6.3.2 for a brief discussion on voluntary cooperation that relies on PULL methodology.

4.2.1 Our solution, the algorithm

The solution uses a "hack" in the *init.d stop* scripts that gives Heartbeat the impression that the virtual machines are being stopped, while they are still running. This is necessary out of two things:

- The virtual machines should never be stopped. If a VM is stopped it gets unavailable. The virtual machines should be live migrated during failover if possible, not stopped.

- Heartbeat on the failing physical node runs *stop* on both of the virtual machines and waits for them to shut down completely. It checks the status of the command by checking the exit code of the *init.d stop* script. If the virtual machines were shut down successfully (from the exit code of the stop script), Heartbeat proceeds with shutting down itself. If not, Heartbeat will continue to try running the stop script to shut down the virtual machines.

The "hack" is that the *stop* scripts return the exit code for successfully shutting down the virtual machines, while the virtual machines are not being shut down. This way, Heartbeat believes that the virtual machines are stopped, but they are not. Heartbeat on the other nodes decides which new physical nodes that should start the virtual machines that supposedly were just shut down. When the physical nodes issue the *start script*, the virtual machines get live migrated from the first physical node instead of being cold booted.

In a cluster where all the virtual machines are running, the algorithm in figure 4.4 is used.

Initially, no virtual machines are running, and we add support for cold booting virtual machines if they are not hosted on other nodes in the network. This extension is added in figure 4.5.

The *start* script is implemented in Perl using the following algorithm. The virtual machine *x0* is used as an example:

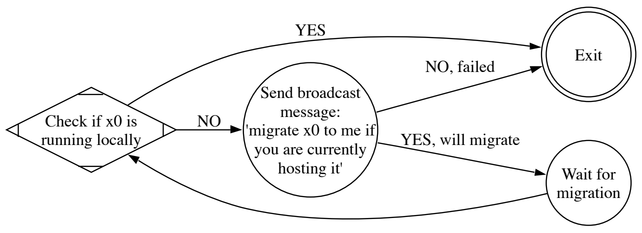

YES
Exit
NO, failed
Send broadcast message: 'migrate x0 to me if you are currently hosting it'
Check if x0 is running locally
NO
YES, will migrate
Wait for migration

Figure 4.4: The algorithm for */etc/initd.d/x0 start*

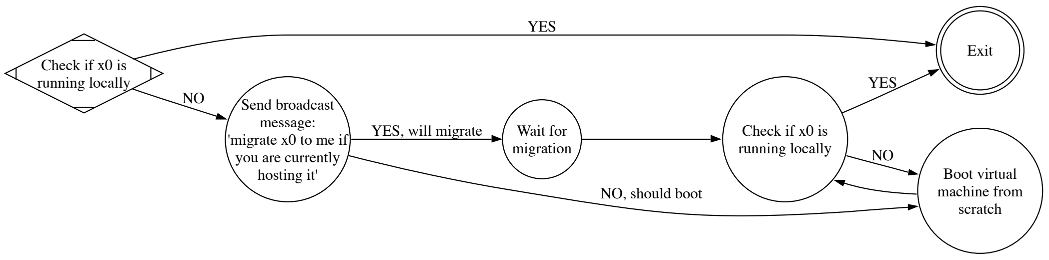

YES
Exit
Check if x0 is running locally
NO
Send broadcast message: 'migrate x0 to me if you are currently hosting it'
YES, will migrate
Wait for migration
Check if x0 is running locally
YES
NO
Boot virtual machine from scratch
NO, should boot

Figure 4.5: The extended algorithm for */etc/initd.d/x0 start*

```
Check if x0 is running locally
 Yes, OK. Exit.
 No, we have to get it running locally
  Send broadcast message to the other nodes - Migrate x0 to me!
  Waiting for migration to complete/fail
  Check if x0 is running locally
   Yes, migration succeeded. Exit.
   No - no nodes could or wanted to migrate x0 to me (or migration failed)
    Cold booting x0
    Check if x0 is running locally
     Yes, OK - cold booted successfully. Exit.
     No, failed to boot. Might be fenced out. Exit.
```

The daemon is very simple (see figure 4.6). It listens for connections to a port, and when a connection is established, it waits for a migration request. The migration request is a string with the following syntax:

```
<command> <argument>
e.g.:
migrateVirtualMachineToMe x0
```

The current version of *migrated* supports only one command (migrateVirtualMachineToMe), but more commands can be added in the future. It is made as simple as possible to make it stable and easy to use in the testing. The network communication is done using the reliable TCP protocol, and non-blocking communication is ensured by the Perl module IO::Select[6].

[6]http://perldoc.perl.org/IO/Select.html

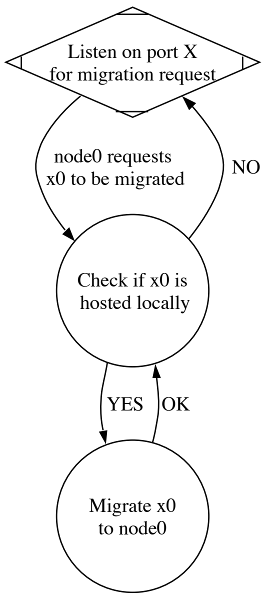


Figure 4.6: The algorithm for the daemon, called *migrated*.

Future optimization should be done before putting it into production environments. The main area of improvement is adding control data to increase reliability and to decrease the time required to execute the script due to the sleep statements. The algorithms are illustrated in two real-life examples in section 4.2.2.

4.2.2 Uncontrolled and graceful failures illustrated

The two types of failure in the topology are *uncontrolled failures* and *graceful failures* and the aim is to minimize the mean time to repair from these. A typical uncontrolled failure is *power failure*, and a typical graceful failure is *rebooting* of a physical node. The main difference is that in graceful failures, the heartbeat instance running on the physical node that is about to fail detects that it is on the way down - and notifies the other nodes about it going down and that they need to failover the virtual machines.

The difference between *uncontrolled failures* and *graceful failures* is illustrated by two scenarios in figure 4.7 and 4.8.

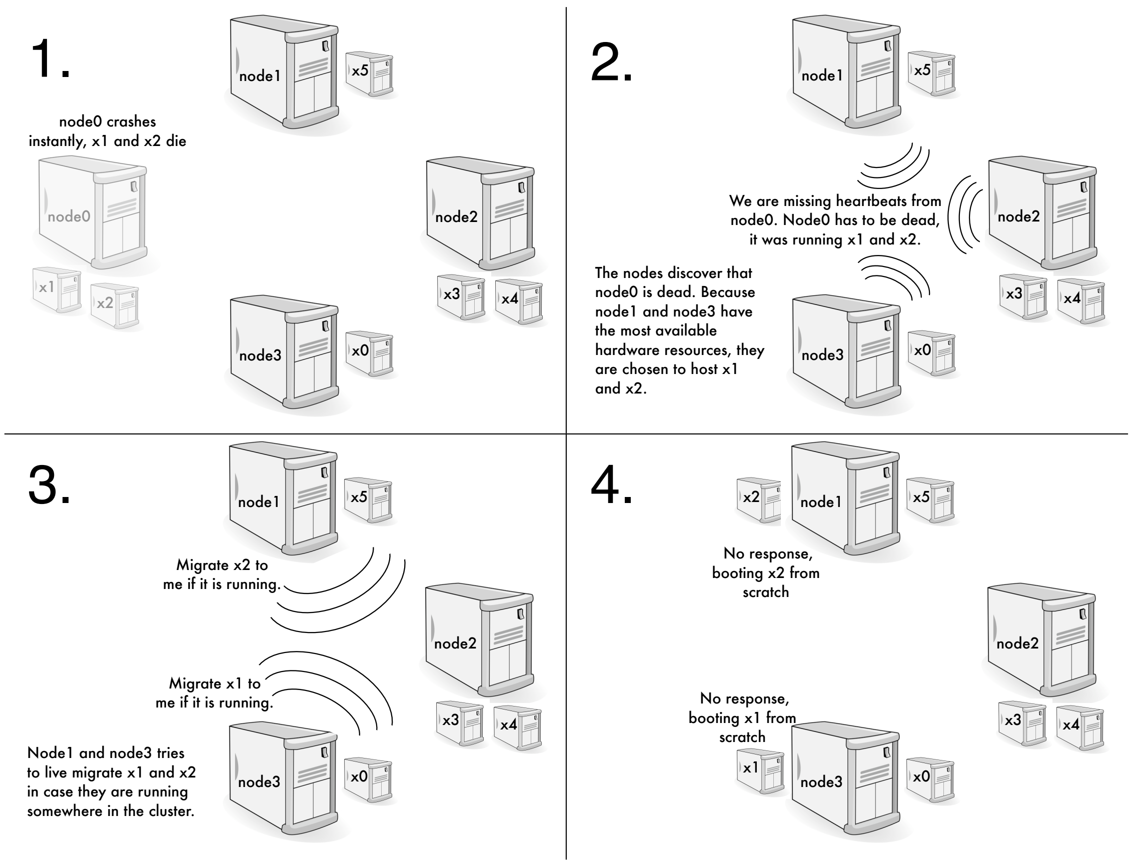

1.
node0 crashes instantly, x1 and x2 die
node1
x5
node0
x1
x2
node2
x3
x4
node3
x0
2.
node1
x5
We are missing heartbeats from node0. Node0 has to be dead, it was running x1 and x2.
node2
The nodes discover that node0 is dead. Because node1 and node3 have the most available hardware resources, they are chosen to host x1 and x2.
x3
x4
node3
x0
3.
node1
x5
Migrate x2 to me if it is running.
node2
Migrate x1 to me if it is running.
x3
x4
Node1 and node3 tries to live migrate x1 and x2 in case they are running somewhere in the cluster.
node3
x0
4.
x2
node1
x5
No response, booting x2 from scratch
node2
No response, booting x1 from scratch
x3
x4
x1
node3
x0

Figure 4.7: Uncontrolled failure. Setup with node0..node3 as physical nodes and x0..x5 as virtual machines. Node0 becomes unavailable immediately, and the virtual machines running on it become unavailable too. x1 and x2 must be cold booted.

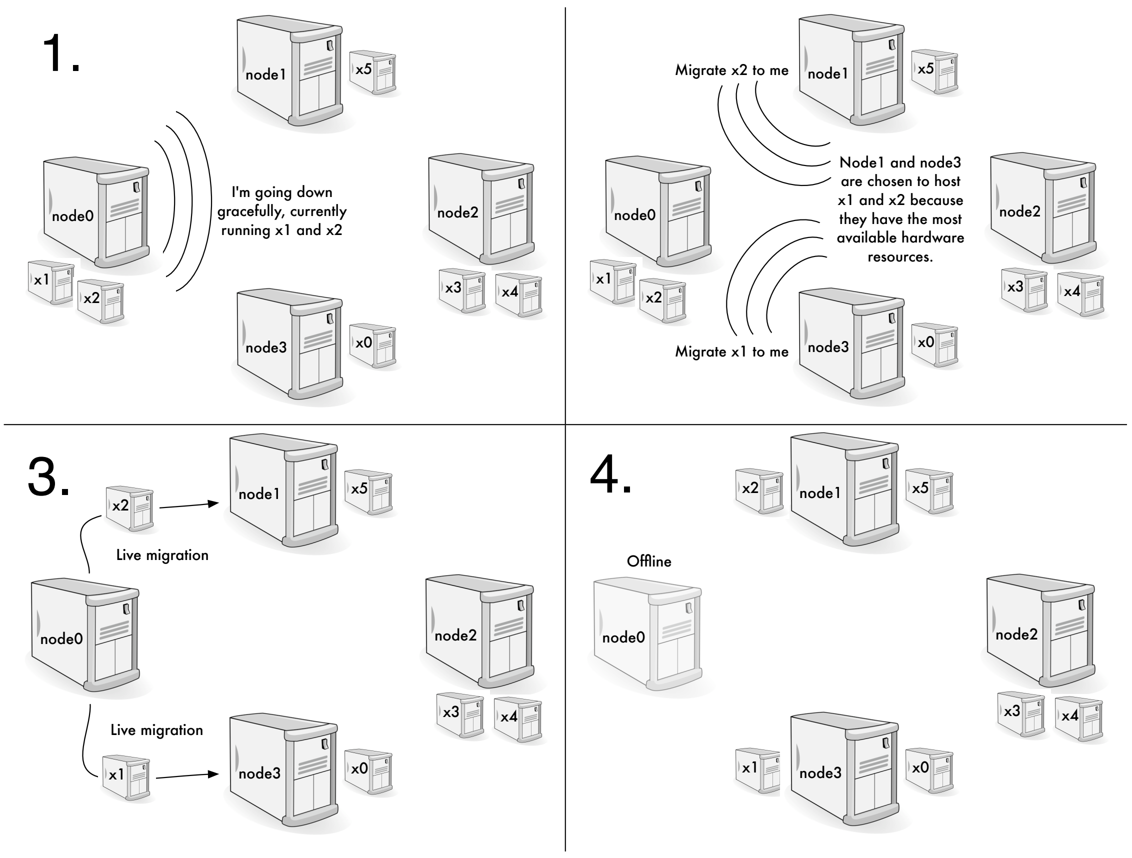


Figure 4.8: Controlled failure. Setup with node0..node3 as physical nodes and x0..x5 as virtual machines. Node0 is going down gracefully. On the way down, it notifies the other nodes in the cluster and waits for them to failover x1 and x2 using live migration.

Chapter 5

Measurements and results

An add-on to Heartbeat has been developed, which makes Heartbeat capable of controlling Xen instances automatically including the functionality of *"live migration"*. This chapter will explain the methodology and results used to validate the usefulness of it through scientific measurements of performance and convergence times (time to repair). Task 3 from the introduction in chapter 3 will be completed.

5.1 Introduction

An add-on to Heartbeat has been developed, which makes Heartbeat capable of controlling Xen instances automatically including the functionality of *"live migration"*. If a physical node fails gracefully, the virtual machines affected will successfully migrate to other physical nodes in the cluster. If the physical node fails instantly, the virtual machines affected will be cold booted on the other nodes.

The experimens in this project are limited[1] on the outage caused by graceful failures[2], and questions that are likely to be asked are:

- Are the critical services highly available during failover?
- How are the users of different service affected if the physical node running is shut down for maintenance?

As for network services, the experiments will concentrate on the TCP and UDP protocol, both heavily used in the Internet today . The most important and fundamental difference between these protocols is that TCP is *connection oriented* while UDP is *connection less*. Table 5.1 shows a quick comparison between TCP and UDP [27,28].

[1]The limitation is explained in section 3.7.2.

[2]Graceful failure is defined in section 3.6.

TCP:	**UDP:**
Reliable	Unreliable
Connection-oriented	Connectionless
Slow	Fast

Table 5.1: Showing the most important differences in TCP and UDP

Connection oriented means that the protocol is to be reliable for the applications that use it. The underlying software, as in the software controlling the protocol, is verifying that the packets have been received and if not, the packets are retransmitted. The receiver is checking the sequence numbers of the packets, and controlling that the ordering of the packets is correct before the packets can ascend up the TCP/IP stack on the receiver side [27, 28]. Figure 5.1 shows a TCP stream with an outage. The protocol software is responsible for carrying all packets through the link, and does it by re-transmitting the packets lost in the outage. The sequence numbers show that all packets are transmitted.

Connection less means that the protocol is not reliable and that the connections are not persistent. Reliability is achieved in connection oriented protocols by the cost of performance. Applications that prioritize real time updates and performance over reliability might be better suited with a *connection less protocol* [27, 28]. Figure 5.2 shows an UDP stream with an outage. The packets lost in the outage are lost forever, but when the connection once again is available, the most recent packets are sent. There are no sequence numbers as there are in TCP.

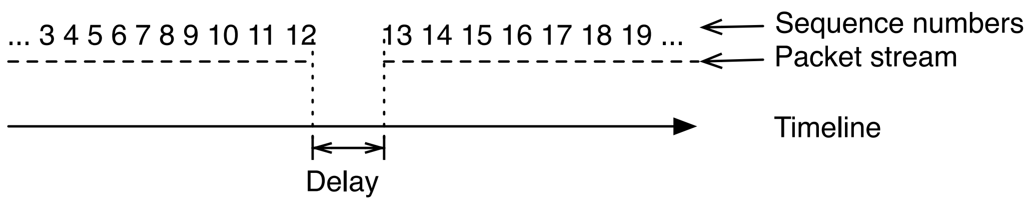


Figure 5.1: Expected behaviour of the TCP protocol. The arrangement of the sequence numbers are precise, and the protocol is responsible for the receiver to receive all packets.

The TCP protocol is expected to not suffer from packet loss because of the built in re-transmission in case of missing acknowledge packets (ACK), rather than an expected delay in data transfers (i.e. file transfers). As for UDP, packet loss is expected to result in problems in form of gaps in packet streams used

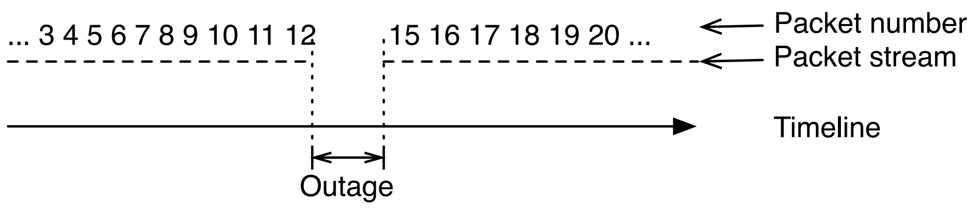


Figure 5.2: Expected behaviour of the UDP protocol. Performance is prioritized over reliability. Packet number 13 and 14 are lost. The packet numbering is added for illustrative purposes, the UDP protocol does not include sequence numbering.

for media transfers and other real time data transfers. It is therefore reasonable to perform different tests on TCP and UDP traffic.

The ICMP protocol is a fairly simple protocol that differentiates from TCP and UDP in several ways. It does not use ports in the communication and it is not based on a server and client model. It is designed for sending messages from one host to another, including routers and switches. It is robust since it is so simple, but it does not include delivery guarantees [27]. If the receiver of an ICMP packet is unavailable, the sender of the packet will not get any reply back. In many occasions, what is not returned is valuable information - e.g ICMP echo request packets that do not get any ICMP echo replies back indicates that a host is unavailable.

The main aim of the measurement part of the thesis is to map out the cost of one graceful failure in different scenarios, and the measurements are chosen and carried through with that in mind. The measurements are briefly described below to introduce the reader to the more detailed explanations in the next sections.

CPU performance in Xen virtual machines is measured to verify previous research and to validate Xen as a good candidate for high performance server virtualization (section 5.2).

The footprint of a graceful failure is measured and illustrated to show how the network latency is affected by a migration measured in a timeline. This is important because the cost of live migrating (copying the memory) from one node to another can affect critical services that demands low latency and constantly high network performance. The result will be used to validate and explain the results in in the other measurements (section 5.3).

Measuring the outage caused by live migrations is an attempt to measure the packet loss caused by a migration on a general basis. The result will be used to validate and explain the results in other measurements (section 5.4.1).

Impact on VoIP conversations. VoIP conversations are simulated using *pkt-gen* to generate similar network traffic and the packet loss caused by migrations is measured. The expected outage in terms of seconds caused by a graceful failure will be calculated (section 5.4.2).

Impact on file transfers using SCP. Measuring the time to copy a file with and without migration and mapping out how file transfers are affected by graceful failures. The expected result is that the time to copy the file increases during a graceful failure. The worst case scenarios are if the file transfer is aborted or file integrity errors occur (section 5.5.1).

Benchmark testing Apache2 and MySQL to illustrate the cost of migrating the web server versus migrating the database server with focus on reproducibility (section 5.5.2).

User activity simulation testing Apache2 and MySQL to illustrate the cost of migrating the web server versus migrating the database server with focus on realism (section 5.5.3).

Benchmark testing Apache2 with up to 300 users to see how the scalability of a web server is affected by migration (section 5.5.4).

User activity simulation testing Apache2 and MySQL with up to 200 users is done to see how the scalability is affected by migration, and the cost of migrating the web server versus migrating the database server when different amounts of connected users. The cost of hosting the two VMs versus hosting them on separate hardware is also illustrated (section 5.5.5).

Each of the experiments consists of three parts: introduction, methodology and result. The results will be briefly formalized, and further discussed in the discussion later (section 6).

5.2 CPU performance in Xen virtual machines

Introduction

As an introductory experiment, CPU utilization is measured to confirm previous research and to validate Xen as a good candidate in our topology.

Xen performance results have been published previously [3, 5]. These results are primarily focused towards CPU utilization, and not as much file transfer performance. File transfer performance is highly important in some services, i.e. on centralized storage solutions or backup servers. We know, from previous research, that the CPU utilization on a virtual machine (Dom-U) is very close to native performance. File transfer performance have many dependencies, and are difficult to measure on a general basis. In our experimental setup, we have commodity hardware that is not representative for the file transfer performance that will be achieved with production environment equipment. Therefore, only a simple CPU measurement to directly compare performance will be measured. For more elaborative measurements on performance, refer to [3,5,6,29].

Methodology

A dual AMD Opteron 242 1600MHz computer with 2GB ram is used. It is configured with two sets of working kernels: One native kernel without Xen support and the same kernel patched with Xen support. A script is created to perform the measurement 100 times repeatedly and to calculate averages and standard deviations. The test that is used to measure CPU performance is to measure the total time to calculate 123456^{123456} using the tool *bc*.

The difference between the native host and the virtual machine without migration is expected to be similar to the leftmost measurement in figure 2.11. The performance is expected to decrease during migration.

1. Boot the computer without Xen support. Run the CPU performance measurement script. Save the results.
2. Reboot the computer into a kernel with Xen support. Run the script in dom-0, save the results.
3. Boot one virtual machine and log onto it. Run the same script (in dom-U), save the results.
4. The results is plotted in a bar diagram with averages and standard deviations.

Results

The bars in figure 5.3 show that the virtual machine running in Dom-U is as fast as the native host when calculating. The average is 0.14 seconds slower, but it is insignificant due to the standard deviation. It is surprising that Dom-0 performs better than the native kernel without Xen support, 0.52 seconds on the average. The measurements are repeated 100 times, so the impact of random error is expected to be small.

Figure 2.11 shows that the native computer and the Xen virtual machine performs equally well in the benchmark test with SPECint. SPECint is a benchmark specification for CPU's integer processor power and is comparable to this measurement and result. As the graphs show, the results are equal - the integer processor power is equal in a Xen virtual machine to the native host itself.

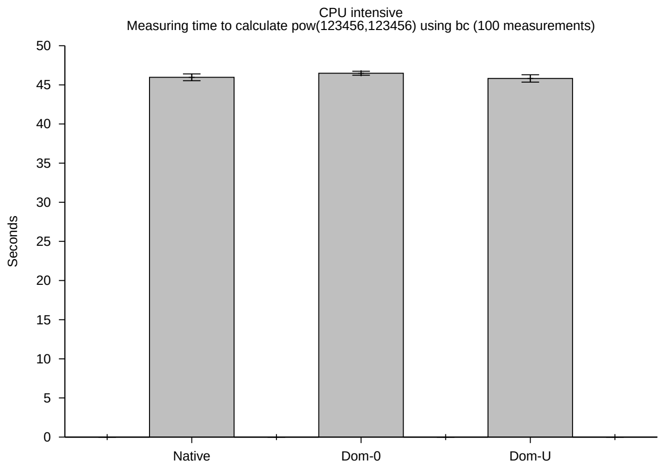


Figure 5.3: CPU performance comparison between the native host, the host with Xen support (dom-0) and a virtual machine (dom-U)

5.3 The footprint of a graceful failure

Introduction

Xen is using an optimized algorithm when live migrating a virtual machine between two physical nodes to minimize the outage. The process can be divided into three phases [2]:

1. The PUSH phase to copy memory pages while the VM is still running.
2. the STOP-AND-COPY phase where the VM is stopped on the old host and resumed on the new host.
3. the PULL phase to fetch faulty memory pages. This is in case some memory pages in 1 have been updated after they were copied to the new node, and the VM tries to access them.

The VM is available while copying memory in the first and the last phase (in 1. and 3.). In these phases, the network performance will most likely be slightly decreased due to the bandwidth usage in the migration process. In the middle phase, the VM is paused on the first physical node, and resumed on the new node. In this short gap, the VM is unavailable [2]. The PULL phase is initiated after the VM is started on the new node.

Methodology

Round trip times is measured and plotted in a graph to illustrate the impact on network latency given by one live migration. The equipment in use in this experiment is the gateway, one virtual machine (x3) and two physical nodes. Figure 5.4 shows the setup and how the measurement is performed.

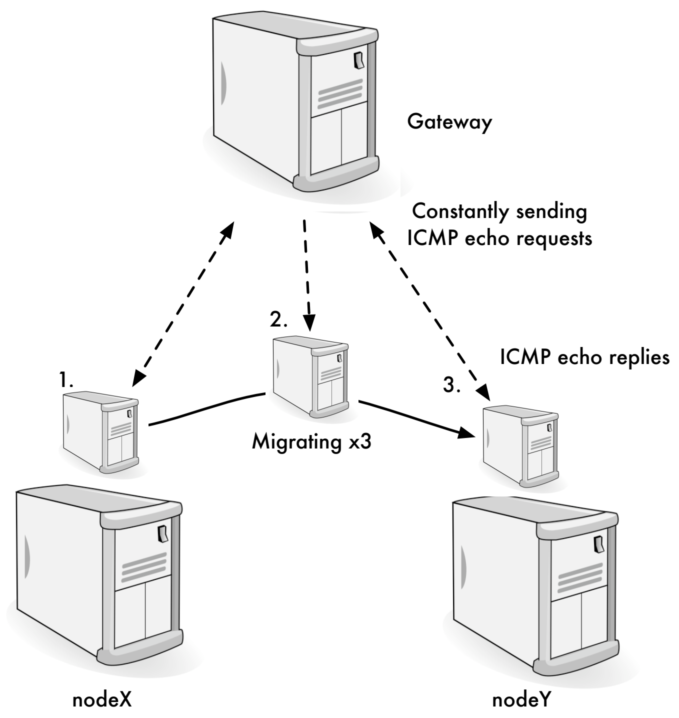


Figure 5.4: Experimental setup

2000 packets are sent with 100 packets per second from the gateway to the virtual machine. This adds up to a timeline of 20 seconds which is plotted on the x-axis. This command is issued on the gateway, which is sending ICMP echo-request packets to the virtual machine *x3*:

```
ping -c 2000 -i 0.01 x3
```

The output from ping is the round trip time of each packet. These round trip times are plotted on the y-axis. One graceful failure is simulated during these 20 seconds. Which of the remaining nodes that will be selected to failover the virtual machines (*nodeY*) will then be chosen by Heartbeat before the live migration starts. Stopping Heartbeat by */etc/init.d/heartbeat stop* is used to simulate a graceful failure (reboot or shut down).

Results

The graph (figure 5.5) shows that the migration process increased the round trip time from 6 to 10 seconds at the timeline. The graph indicates that the

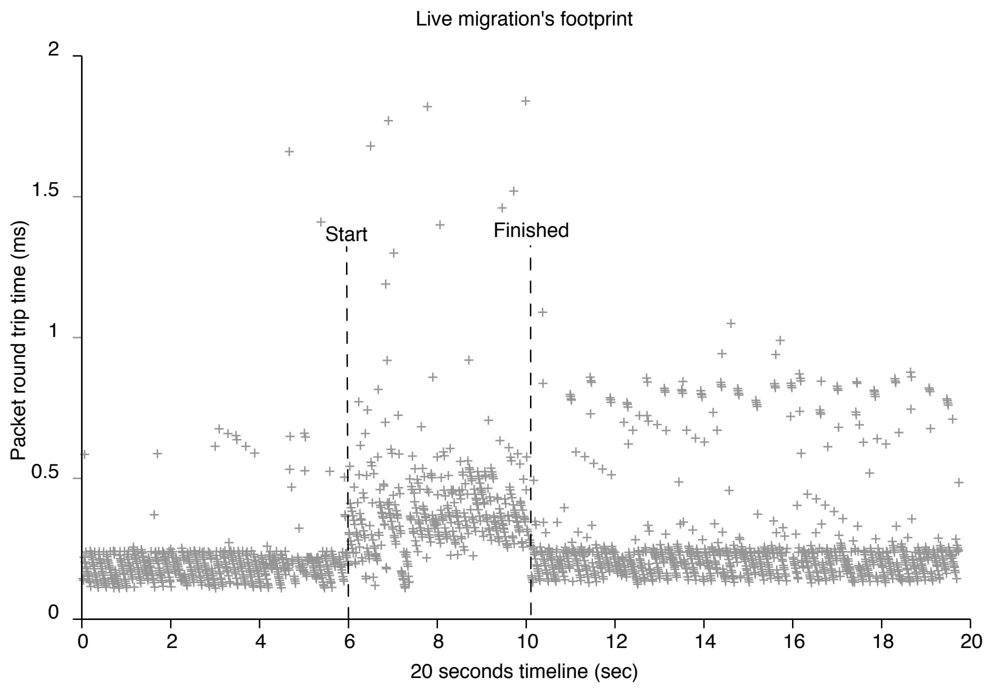


Figure 5.5: A graceful failure occurs while measuring the round trip times to a virtual machine. 2000 samples in a 20 seconds timeline.

outage caused by live migration is very small otherwise it would have been visible at the graph as a gap on the x-axis. The reported packet loss is between 10 and 20 packets for one failure. Other methods are needed for measuring outage times (see section 5.4.1).

5.4 Impact on UDP services

The UDP protocol is a very simple protocol that is prioritizing performance and simplicity over reliability. It is used for media streaming, video and voice conferences, etc. In these applications, the matter of receiving the most recent packets is more important than loosing a couple. I.e. if talking to a friend in a video conference, it is clearly more important to get the feeling of real time conferencing although with some gaps, than having what appears as an unsynchronized conversation without gaps.

It is difficult to measure exact times in computer networks, and especially short times like an outage caused by live migration. Due to the nature of packet switched networks, it is necessary to send packets to check if the link is available at the time the packets are sent and received. To measure availability over time, it is important to send packets often (giving high granularity) and at accurate intervals. Figure 5.6 shows an illustrative scenario where we try to measure an outage. If the units of the timeline is seconds, the outage is measured to be 3 seconds ($5 - 2 = 3$), which is not very accurate. The figure shows that the outage is approximately 2 seconds, but since we are measuring every second, it is interpreted as 3 seconds.

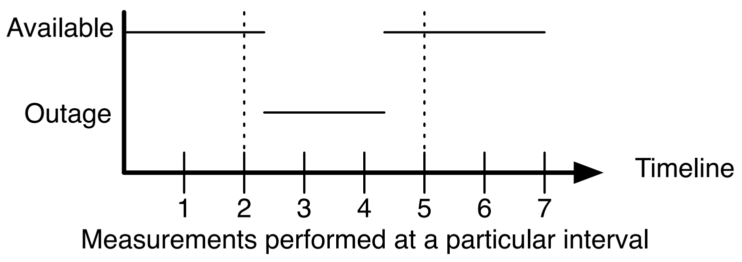


Figure 5.6: Measuring outages in packet switched networks. Due to the large intervals, higher uncertainty is unavoidable.

The measurements have to be done *discrete*, but often enough approach continuous behavior. The goal is to create a set of fine grained measurements that can be used to calculate outages within as little uncertainty as possible (figure 5.7). However, if the measurements have a too high granularity, the network traffic caused by the measurement can effect the results of the measurement negatively by putting too much load on the network and thus creating a bottleneck for the element we try to measure.

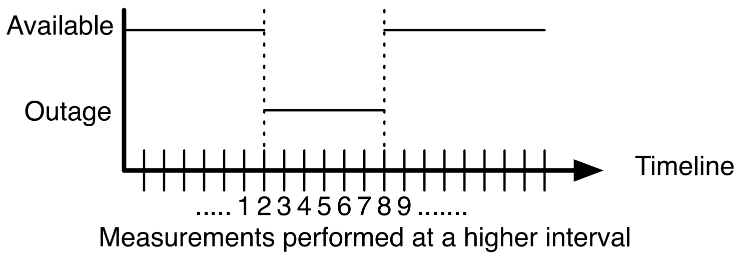


Figure 5.7: Measuring outages in packet switched networks. Smaller intervals pays off in terms of higher accuracy caused by smaller uncertainty.

The *Linux kernel packet generator* [25] is a kernel module in Linux that is able to generate UDP packets very effectively due to running in kernel space. It has therefore effectively direct access to the NIC. The installation procedure is shown in Appendix A.4 and the configuration is in Appendix A.5. The payload of each packet contains a sequence number, the time stamp in seconds[3] and the epoch time (in usec) from the time when the packet was generated.

[3]Unix time - seconds since January 1, 1970.

5.4.1 Measuring the outage caused by graceful failures

Introduction

Heartbeat[4] uses live migrations to repair unavailability caused by graceful failures. To measure the outage in time caused by this process on a general basis, a large amount of small packets are generated every second. Bigger packets are likely to affect the network more than small packets. We need to capture the packets received to see how many packets that are lost and to extract the inter-arrival times to see at what times the packets got lost in the timeline.

Methodology

Pktgen is used to send a number of packets per second in 45 seconds. The number of packets per second and the IPG (Inter-packet gap) is decided by the granularity used. Each measurement is repeated 50 times without migration to find the reference values and then 50 times with migration to see the difference in number of packets lost.

The equipment in use in this experiment is the virtual machine *x3* and the gateway. The algorithm for each measurement is:

1. Initiate tcpdump packet capture on x3.
2. Start sending packets with pktgen from the gateway to x3 (lasts for 45 seconds).
3. After 15 seconds, fail the physical node running x3 after 15 seconds by gracefully stop heartbeat.
4. Stop tcpdump packet capture on x3.
5. Fetch the amount of packets captured in the packet dump on x3.
6. Repair the cluster and regain availability before next iteration (50 iterations in total to minimize random errors).

For fully automating the measurements, four scripts are used:

Main script running on the gateway. Used for looping, starting packet generation, calculating averages and standard deviations. This script uses the other scripts to carry out commands. Implemented in Perl.

Client script running on the gateway. Used for sending commands to x3. Implemented in Perl.

[4]With the add-on developed during this work.

Packets sent	Packets lost Reference	Packets lost One failure	Packets lost Xm, one failure
450 (10pps)	0	2 ± 4	5 ± 8
4500 (100pps)	4 ± 4	55 ± 13	15 ± 5
45000 (1000pps)	177 ± 90	1071 ± 91	578 ± 81

Table 5.2: Packet loss in 45 seconds UDP transfers, measurements repeated 50 times each.

Server script running on x3. Used for executing commands given from the client (starting tcpdump packet capture, stopping packet capture and counting packets). Implemented in Perl.

Failure simulation script running on the gateway. Used for remotely executing failure to the node running a specified virtual machine. It is used to force through a migration by simulating reboot on one of the physical nodes. Implemented in Bash.

The experiment is repeated using different granularities of packets per second - 10 packets per second, 100 pps and 1000 pps. The payload of the packets is set to be 0 bytes.

In addition to using the automatic repair approach with Heartbeat, the measurements are repeated using the built in Xen commands for live migration to avoid the network traffic caused by Heartbeat. It is interesting to see if Heartbeat affects the packet loss in a live migration.

One of the measurements is chosen and plotted in a graph to visualize the inter-arrival times and where the packets are lost.

It is difficult to predict any results, but according to previous research [2] approximately 60ms downtime is reasonable to expect.

Results

Table 5.2 shows the results from each of the measurements with averages and standard deviations. Assuming that the length of the outages in time can be calculated by multiplying the number of packets lost with the packet gap, it is clear that the outage increases in length when the amount of packets per second increase.

The column in the middle shows the measurements where Heartbeat was running and used the add-on to repair unavailability by live migration. 10 packets per second measured an outage of $2 \times 0.1 = 0.2$ seconds while the measurement with 1000 packets per second measured an outage of $1071 \times 0.001 = 1$ second.

Codec	Bytes per packet	Packets per sec	Bytes per sec
g711 ulaw/alaw	160	50	8000
g726 adpcm-32	80	50	4000
g729	20	46	920
gsm-fast	33	50	1650
gsm-slow	66	25	1650

Table 5.3: Codecs used in voice over IP with the amount of packets per second.

The rightmost column shows the corresponding packet loss when the built in xen commands are used manually instead of using Heartbeat, the developed heartbeat add-on and simulation of failures resulting in automatic repair. The packet loss is generally lower when using the xen commands manually. This may be because the Heartbeat instances are communicating quite intensively, and when the network already is quite loaded, further packet loss is provoked by this communication.

5.4.2 Impact on VoIP conversations

Introduction

VoIP (Voice over IP) conversations require low latency and reliable packet transfer. It is a service that is very vulnerable to outages, and therefore it is suitable for using a reference service in regards to being able to handle small outages. If VoIP services are unaffected by the outages caused by live migration, it is reasonable to believe that few other services are affected.

The reader is referred to M. Wornhard [30] for more measurements on VoIP failover.

Methodology

In order to decide the granularity of the measurements, knowledge about VoIP compression algorithms are used to find a reasonable amount of packets to send per second, and the size of each packet. According to table 5.3, the least efficient compression algorithm requires 50 packets per second with a packet size of 160 bytes. To simulate one conversation, pktgen is configured to send 50 packets per second, giving a total of 2250 packets in 45 seconds. The IPG (Inter-packet gap) is set to 20ms. 10 simultaneous conversations require 500 packets per second and a total of 22500 packets in 45 seconds. IPG is set to 2ms. The packet size is configured to be 214 bytes in total (160 bytes payload plus headers). The lease efficient compression algorithm is chosen to achieve the worst case scenario.

1 conversation	Packets received	Packets lost	Calculated outage
Reference	2250	0	0ms
One graceful failure	2237 ± 5	13 ± 5	254 ± 94ms

Table 5.4: Simulating one VoIP conversation using the g711 ulaw/alow compression algorithm. The graceful failure results in one migration of the receiver of the packets.

10 conversations	Packets received	Packets lost	Calculated outage
Reference	22426 ± 44	74 ± 44	148 ± 88ms
One graceful failure	21898 ± 98	602 ± 98	1205 ± 195ms

Table 5.5: Simulating 10 concurrent VoIP conversations using the g711 ulaw/alow compression algorithm. The graceful failure results in one migration of the receiver of the packets.

The same scripts as in the previous experiment are used to generate and capture packets. The measurements are done for 1 simulated conversation and for 10 concurrent simulated conversations.

Results

Table 5.4 shows that the outage caused by a graceful failure to the physical node that is hosting the VM that routes one VoIP conversation will cause an outage of 0.25 ± 0.09 sec.

Table 5.5 shows that 10 concurrent conversations will experience small outages even when no failures occur. This result is interesting in itself, but it's the difference in the amount of packets lost that are the most interesting in this experiment. With one failure during the 45 seconds the 10 conversations lasted, a total of 602 ± 98 packets were lost. If we multiply the amount of packets lost with the gap between each generated packet we can calculate the total time of outage, although not exact - the packets might be lost on different times in the timeline. The total outage is calculated to be 1.20 ± 0.20 seconds.

Figure 5.8 illustrates the packet loss from one VoIP conversation in a timeline. In total, 11 packets got lost by one graceful failure. The packet loss are concentrated within a frame of 10 seconds, which is the time from the failure occurs to it is completely repaired. The user will only notice a packet loss of 11 packets, which is distributed over 10 seconds. In VoIP, this packet loss is insignificant considering that 50 packets are sent each second. The calculated total outage is 0.22 seconds, and as the figure shows, it is distributed over 10 seconds. The peaks in the inter-arrival times that appear each second are believed to be a systematic error generated by *pktgen*, the source was never found. The value in the results are still present since the peaks are constant.

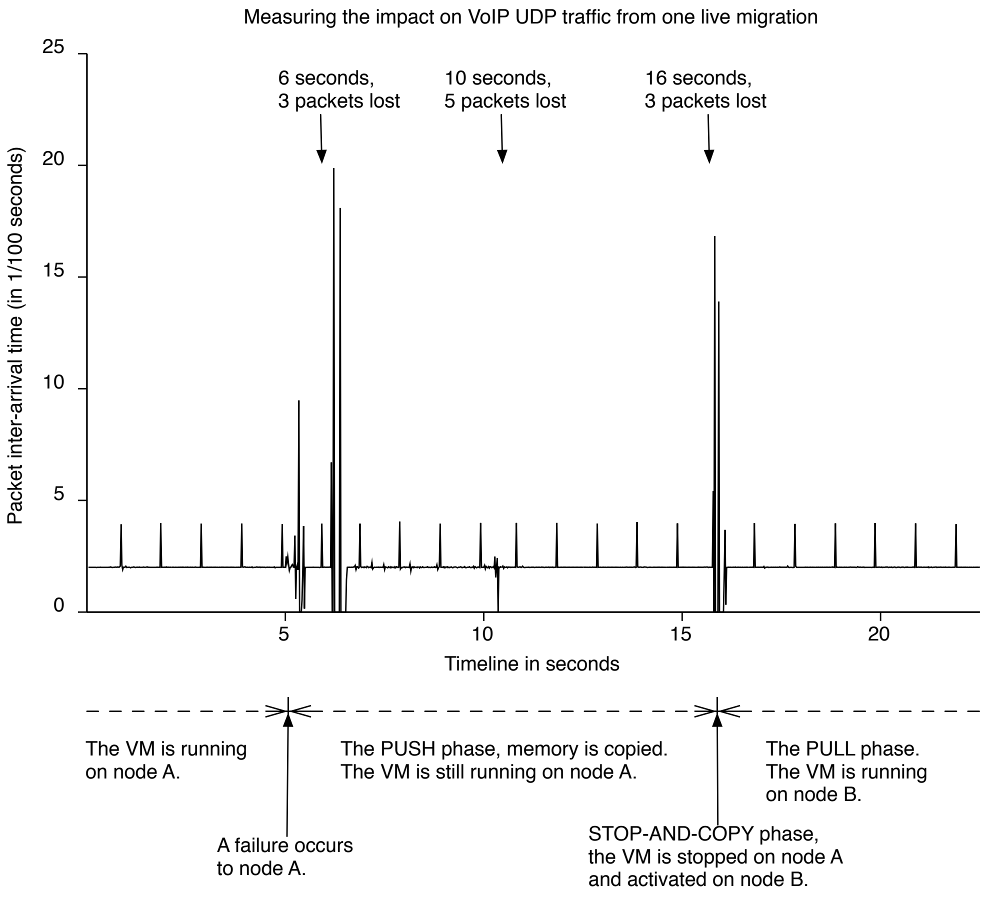


Figure 5.8: Inter-arrival times for packets simulating a VoIP conference with g711 ulaw/alaw codec. 160 bytes per packet, 50 packets per second. Showing a timeline of 20 seconds. A failure occurs at 6 seconds, causing a total packet loss of 11 packets. 11 packets translates into 0.22 seconds.

Refer to section 5.3 for a more detailed description of the three phases in the migration process.

5.5 Impact on TCP services

The expected impact caused by a short outage in TCP traffic differs from the expected impact on UDP traffic. This is because the TCP protocol is much more complex, and, most importantly, it features sequence numbers and packet retransmission. If a packet is lost, it the sender will re-send the packet until the receiver verifies that the packet is received. The expected result in TCP traffic is that transmissions are requiring longer time to complete if a migration occurs in the middle of the transmission.

The experiments performed on TCP services are performed at HTTP server performance measurements, database connectivity and file transfer using SCP.

5.5.1 Impact on file transfers using SCP

Introduction

According to the introduction in this section, the TCP protocol is expected to not drop any packets even though a small outage has occurred. An interesting experiment is to copy a large file to one VM while migrating it. The expected result is that due to retransmission of the packets, the total time for transfer will be negatively affected by migration but that the file integrity is kept intact since no packets are lost. To verify the integrity after the experiment, md5 checksums are compared.

If a failure occurs in the middle of a file transfer in a traditional setup, it is very likely to be aborted. This is because the IP stack is not preserved in a failover in traditional high availability setups, while it is preserved in a virtual machine that is being migrated.

The virtual machines are expected to perform poorer than the physical machines due to increased cost of transfer. The underlying path from the storage and to the destination is longer for the virtual machines mainly due to shared storage.

Methodology

An ISO image of the Ubuntu 5.10 i386 distribution is used as the file to transfer in the measurements. The size is 632316959 bytes, $\approx$ 603MB.

Three measurements are performed:

1. The time to copy the file from one physical node to another physical node.

2. The time to copy the file from a virtual machine to a physical node (not the physical node the virtual machine is running on).

3. The time to copy the file from a virtual machine to a physical node. One graceful failure is simulated in the middle of the file transfer. The virtual machine is hosted at node X and is copying the file to node Y while it is migrated to node Z.

Each measurement is repeated 100 times and plotted in a bar diagram with average values and standard deviations.

Results

As expected, figure 5.9 shows that a file transfer speed between two physical machines is superior compared with a file transfer between a virtual machine and a physical machine. The poor results for the virtual machines are probably negatively affected by the type of shared storage solution used in the experimental setup. In a production environment, the file transfer performance from virtual machines will better due to better hardware equipment (i.e. fiber channel).

The difference in time to copy a file from a virtual machine that is migrating is very small - just 4 seconds added to a total time of 70 seconds. This is probably not even noticed by any user that is using file transfers to and from virtual machines.

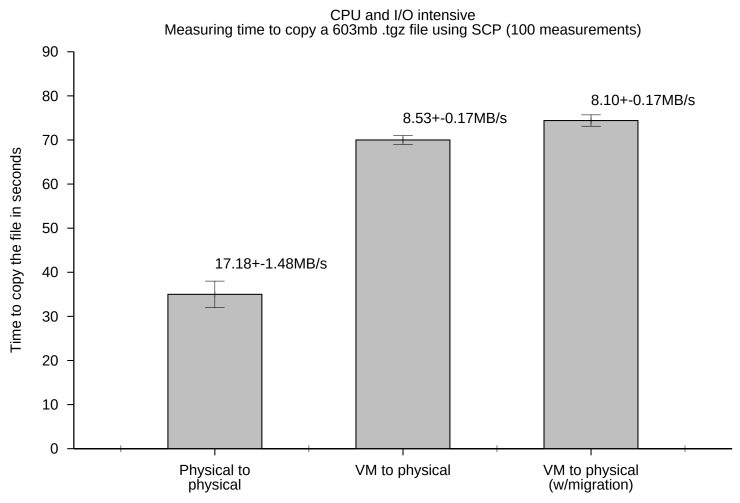


Figure 5.9: Comparing file transfer performance during migration of virtual machines using SCP

5.5.2 Benchmark testing Apache2 and MySQL

Introduction

A typical scenario is to run the Apache server and MySQL server on different physical servers due to increased performance caused by twice the amount of hardware and security caused by strict separation of services. Using the virtualization approach, the same positive effects can be achieved by hosting Apache and MySQL on two different virtual machines.

This experiment aims to measure the response times and number of successful transactions when apache and mysql run on different virtual machines and the load is generated by a fixed amount of users. The measurements are repeated in four different scenarios, making it possible to compare the cost of migration and the cost of hosting the two virtual machines on the same physical node.

The purpose of these measurements is to see how the performance of a web server using database queries to another server, from the users point of view, is affected by these four different scenarios. All of the four scenarios are realistic scenarios for this topology if it is implemented in production environments. The results are plotted in bar diagrams to show the average response times and the total number of successful transactions with standard deviations. It is expected that the web and database server performance is degraded during migration, and therefore the average response times will increase, causing the number of successful transactions to decrease.

Methodology

Siege is a benchmark testing utility that is suitable for generating traffic towards http servers. It has an option called *benchmark* that sets the delay between each request to 0 seconds. This means that the next request is sent immediately when a response is received from the server. This makes all the measurements have the very same circumstances, even with fewer repetitions.

A script is implemented to automate the measurements and calculate the results (averages and standard deviations) in the end. Each set of results calculated by this script comes from 45 seconds traffic generation using Siege, repeated 50 times to make accurate results with minimal amount of random error.

The script is used to measure the response times and the number of successful transactions in each of the scenarios. These four scenarios will be used in more measurements later, for the value of comparison and because they are realistic scenarios that are likely to occur in production environments using virtualization in high availability clusters:

- The web server and database server is running on *different physical nodes*, no failures occur during the measurements. This scenario is expected to have the highest performance, and is the preferred scenario in production environments.

- The web server and database server is running on *the same physical node*, no failures occur during the measurements. Due to the hardware resource sharing, this scenario is expected to perform poorer than the previous one on high loads.
- The web server and database server is running on different physical nodes, one failure of the physical node hosting the *web server* occur during each test, forcing a migration. This scenario is measured to make it easy to compare the cost of web server migration to the reference (no migration).
- The web server and database server is running on different physical nodes, one failure of the physical node hosting the *database server* occur during each test, forcing a migration. This scenario is measured to make it easy to compare the cost of database server migration to the reference (no migration).

In Siege, the *benchmark* option sets the delay between each request to 0 seconds. This makes the experiment easily reproducible, because there are no random values used. However, 0 seconds delay between each transaction is not very realistic. It is the worst possible case of traffic load with a fixed amount of concurrent users. In this experiment, 10 concurrent users with the benchmark option set are used to generate traffic. Due to the unrealistically low delay between request, this corresponds to many times the number of real concurrent users.

The algorithm used in the script for automating the measurements is:

```
$connections=10
$time=45S
loop 50 times {
 If test includes failure, initialize failure in 10 sec, put in background
 siege -b -c $connections -u http://x1.xen.linpro.no/phpinfo.php -t $time
 sleep 30 sec
 If test includes failure, repair all failures and sleep 20 sec
}
```

This algorithm is used for each of the four scenarios. Each single 45 seconds measurement in the algorithm is followed by a break (>30 seconds) to lower the load on the nodes and to repair the failures caused in the scenarios with migrations.

The failures are simulated by invoking the script */etc/init.d/heartbeat stop* on the physical node that is hosting the virtual machine at the moment of the failure. This forces Heartbeat to migrate the VM to another physical node.

The web page used in the measurements is served in 0.15 ± 0.08 seconds to one client.

Results

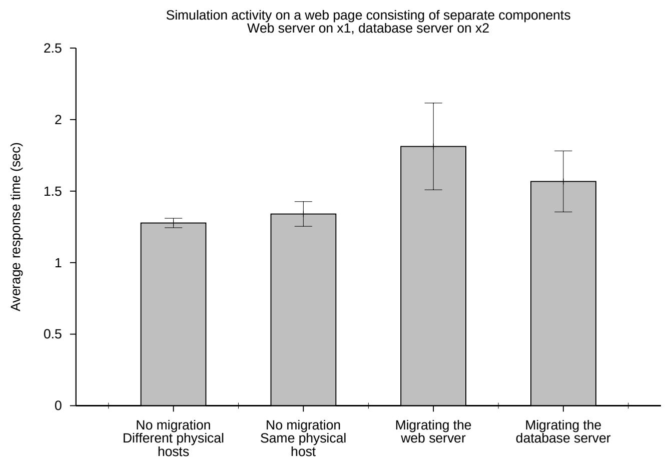


Figure 5.10: The average response time from a web server and a database server over 45 seconds. Tested with no delay between each transaction (benchmarking).

As figure 5.10 shows, the average response time is, as expected, lowest in the two measurements without migration. It is interesting to see, as expected, that the response time is lower if the web server and the database server are running on different physical nodes than if they are running on the same node. The difference is, however, expected to be higher if the server load increases.

The standard deviations are much larger in the migrating scenarios, but the difference between scenario 3 and 4 is not significant. It is not possible to say which of the servers that is most expensive to migrate because it depends completely upon on the web page complexity and the database queries that are used. The larger standard deviations are caused by that the migration process lasts for a shorter amount of time than 45 seconds - and therefore the remaining requests are served at lower response times. The results in the measurements with migration are mixes of low and high response times, giving larger standard deviations.

As figure 5.11 shows, lower response times signify that more transactions can be carried through. In these measurements, no transactions failed - the only negative effect caused by the migrations was that the response times increased, and therefore the average number of requests from each user decreased. This is due to the 45 seconds time limit in each measurement.

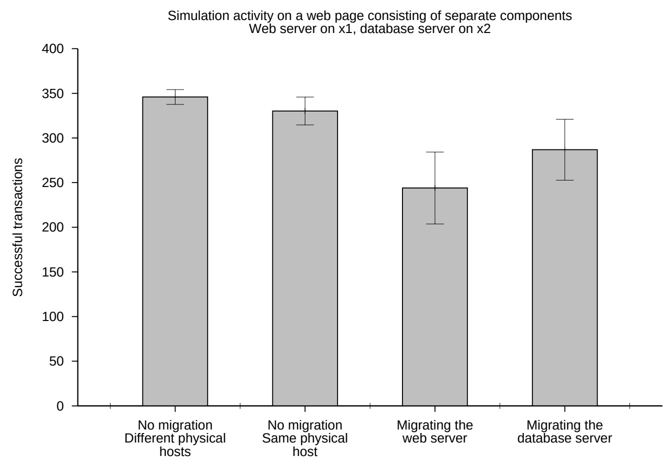


Figure 5.11: The number of successful transactions in each test of 45 seconds. Tested with no delay between each transaction (benchmarking).

5.5.3 User activity simulation testing Apache2 and MySQL

Introduction

According to the Siege v2.61 manual, is benchmark testing not recommended while load testing. Therefore, the measurements in the previous section are done twice - with and without benchmarking.

This experiment is the same as above (section 5.5.2), but with random delay between each transaction to increase the realism. The accuracy is decreased due to randomness, but since the measurements are repeated 50 times each, the averages are fairly equal. This experiment is more realistic than 5.5.2, but introduces more uncertainty in the performance difference between the scenarios.

Methodology

This experiment without benchmarking (simulation tests) introduces a random delay from 0 to 10 seconds between each transaction to simulate active users. The simulation tests have apart from the delay equal methodology as the benchmarking tests in the previous section.

The results are predicted to have the same trends as the benchmarking tests, although with better response times and a smaller amount of total successful transactions due to fewer requests. This experiment is expected to better reflect a real life scenario using equal hardware as in this project. Better performing hardware will undoubtedly get better results.

Results

The results were expected to follow the same pattern as the previous experiment (benchmarking). As figure 5.12 shows, the pattern are similar. Naturally, the response time in all of the scenarios are smaller due to the lessened amount of load (amount of user requests) put on the servers.

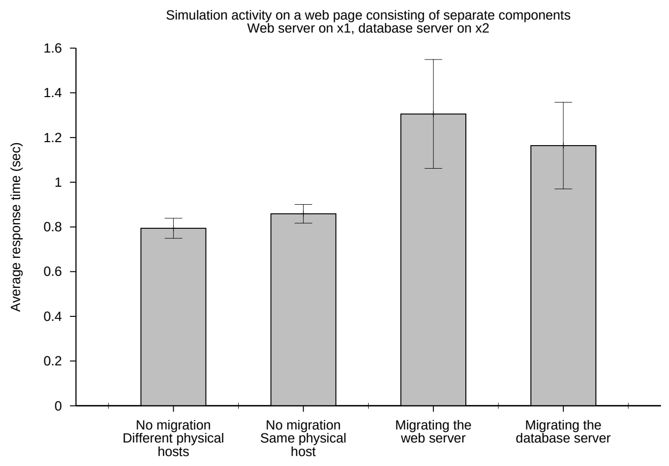


Figure 5.12: The average response time from a web server and a database server over 45 seconds. Tested with 0 to 10 seconds delay between each transaction (simulation)

5.5.4 Benchmark testing Apache2 with up to 300 users

Introduction

This experiment aims to measure the impact on response times when the server is loaded with increasing amount of concurrent users. A graph is plotted to illustrate the trends in performance with different amounts of users, during migration and without migration. As single web page without database content is used to avoid complexity. More complex measurements will follow in the next sections.

Methodology

Reference values are measured while the virtual machine is not migrated. The apache2 web server is installed on the virtual machine *x1*. To measure the impact on failure, migration is forced while benchmarking the web server. The phpinfo() page is used, and the response time for 1 user loading the page once is 0.0102 ± 0.0276 sec. The measurements are performed using this algorithm:

```
$connections=0
$time=45S
loop 100 times {
 $connections+=5
 loop 5 times {
  If test includes failure, initialize failure in 10 sec, put in background
  siege -b -c $connections -u http://x1.xen.linpro.no/phpinfo.php -t $time
  sleep 30
  If test includes failure, repair all failures and sleep 20 sec
 }
}
```

where $connections are incremented by 5 from 5 to 300, and each set of $connections are measured 5 times. In total this adds up to 300 measurements. Each measurement is done in 45 seconds (-t). -b means benchmarking, which means that Siege does not introduce delay in the testing, as would be preferred in simulation tests. This series of measurements is done both without migration (reference) and with one migration to visualize the difference in response times and number of successful transactions related to the number of users connected to the web server.

Preferably, each test should be done more than 5 times, i.e. 10 or 20 times. Due to time constraints, this was not possible. These measurements are very time consuming, taking more than 10 hours each ($\frac{((45s+30s)\times5)\times100)}{60\times60} = 10h$, best case). Even though the result could have been more concise with more repetitions causing less affect by random errors, the value of the results is still present; the main purpose of these measurements is to see if the performance differs significantly between a migrating web server and a non migrating web server.

The results of this experiment are plotted in a graph with number of users on the x-axis, and the response time and the number of successful transactions on the y-axis.

The expected results are that the tests during migration have higher response times than the reference tests without migration. Since this experiment is carried out using the benchmarking option, this is a worst case scenario that will perform many times poorer than the same amount of real users. The main focus is to see the difference in performance with few and many users with and without migration.

Results

The measurements (reference and the test with one migration) required approximately 10 hours each. The response times are plotted in figure 5.13 and the numbers of successful transactions are plotted in figure 5.14.

Figure 5.13 shows that the differences in average response times are very small, even if the amount of concurrent users increases. The standard deviation is higher in the tests with migration, naturally, due to a few requests that

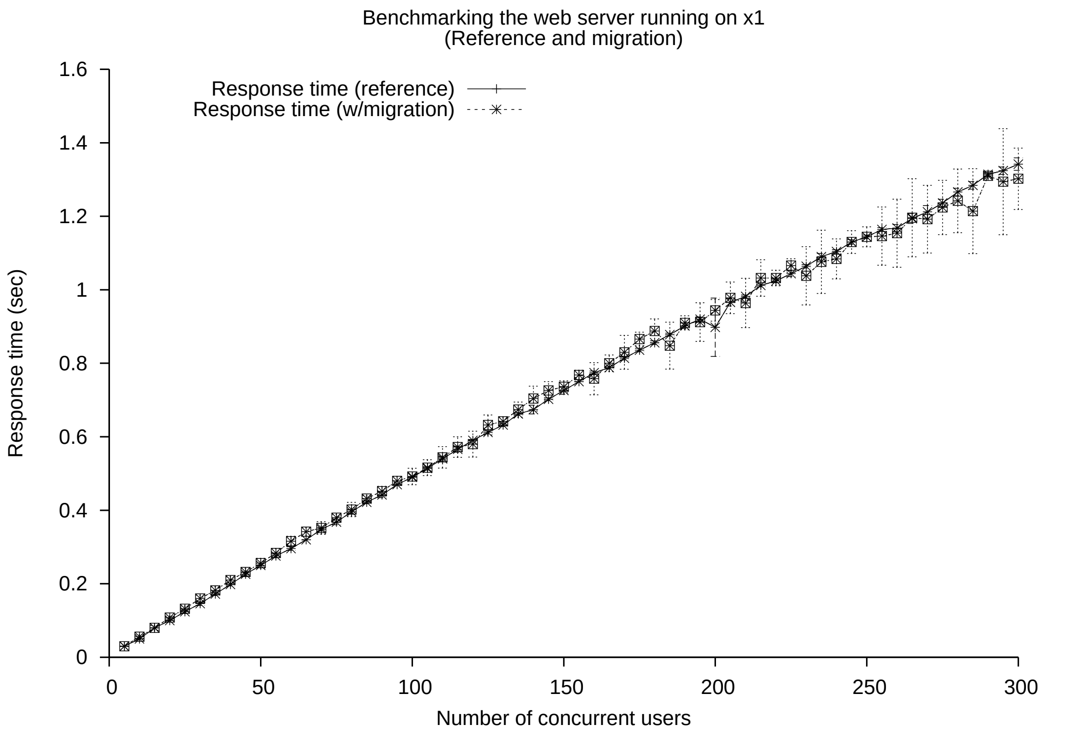


Figure 5.13: Response time on the y axis and number of users on the x axis. The reference measurement and the measurement with migration plotted in the same graph.

have higher response times than the others. However, the average over 45 seconds does not show significant poorer performance on the apache server that is migrated. Figure 5.14 is giving us more information about the measurements. One migration decreased the average number of successful transactions with approximately 500-1000 per each 45 seconds test. The standard deviation is still high, but the difference is significant.

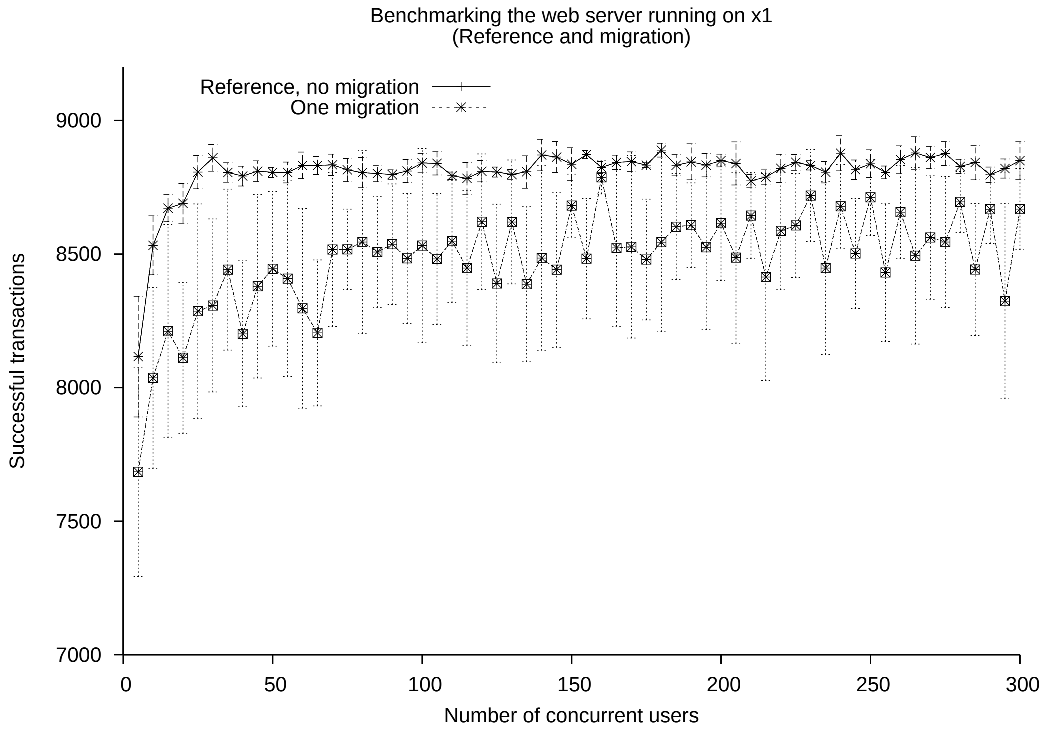


Figure 5.14: The number of successful transactions and the longest transaction in seconds on the y axis, number of users on the x axis. The reference measurement and the measurement with migration plotted in the same graph.

At approximately 50 concurrent users performing requests with 0 seconds delay (benchmarking), the graph in figure 5.14 reveals that the server is fully utilized. As the number of users is increasing, the number of successful transactions flattens out at approximately 8500-8750 transactions per 45 seconds. This is possible because the response time in figure 5.13 is increasing almost linearly as more users connect.

5.5.5 User activity simulation testing Apache2 and MySQL with up to 200 users

Introduction

This experiment combines last three experiments (5.5.2, 5.5.3 and 5.5.4): Increasing amount of users, using both apache and mysql (dynamic content) on different VMs and with delay (random 0 to 10 seconds) between the transactions to simulate users.

Reference values are measured while the virtual machines are not migrated. The apache2 web server is installed on the virtual machine *x1* and mysql server is installed on the virtual machine *x2*. To measure the impact on failures, migrations are forced while simulating load on the web server.

Methodology

The four measurement scenarios are the same scenarios as in section 5.5.2. The web page used in this experiment has a response time of 0.1522 ± 0.0794 seconds to one single request. The measurements are performed using this algorithm:

```
$connections=0
$time=45S
loop 40 times {
 $connections+=5
 loop 10 times {
  If test includes failure, initialize failure in 10 sec, put in background.
  siege -b -c $connections -u -d 10 \
  http://x1.xen.linpro.no/select.php?select=3 -t $time
  sleep 30
  If test includes failure, repair all failures and sleep 30 sec.
 }
}
```

Again, the response times are interesting. This measurement is done because most web pages are supported by a database server, and this scenario is more realistic than static web pages. It is interesting to see how the response times differ between the scenarios where the VMs are running on the same node and different nodes, and the cost of migrating the database server compared to the cost of migrating the web server. The results will depend upon the web page used, and the amount of web content (html, php, images and other web elements) and database content.

The results are plotted in two graphs illustrating the response times and the corresponding amounts of data that are being transferred.

Results

Figure 5.15 shows an increase in average response times at approximately 60 concurrent users. Loads with less than 60 users are handled without any no-

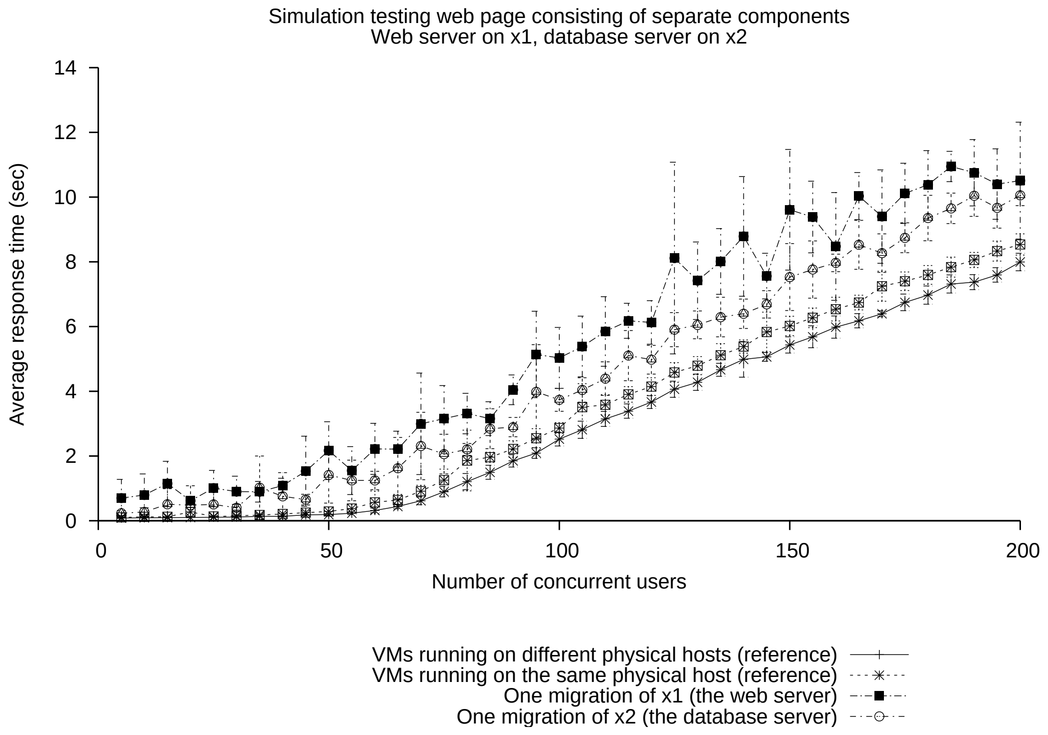


Figure 5.15: The average response time for each requests in 45 seconds tests. Simulating user activity and measured the response time for up to 200 users. Each user has a delay between 0 and 10 seconds (random) between each request. The web server and database server are separated on different virtual machines.

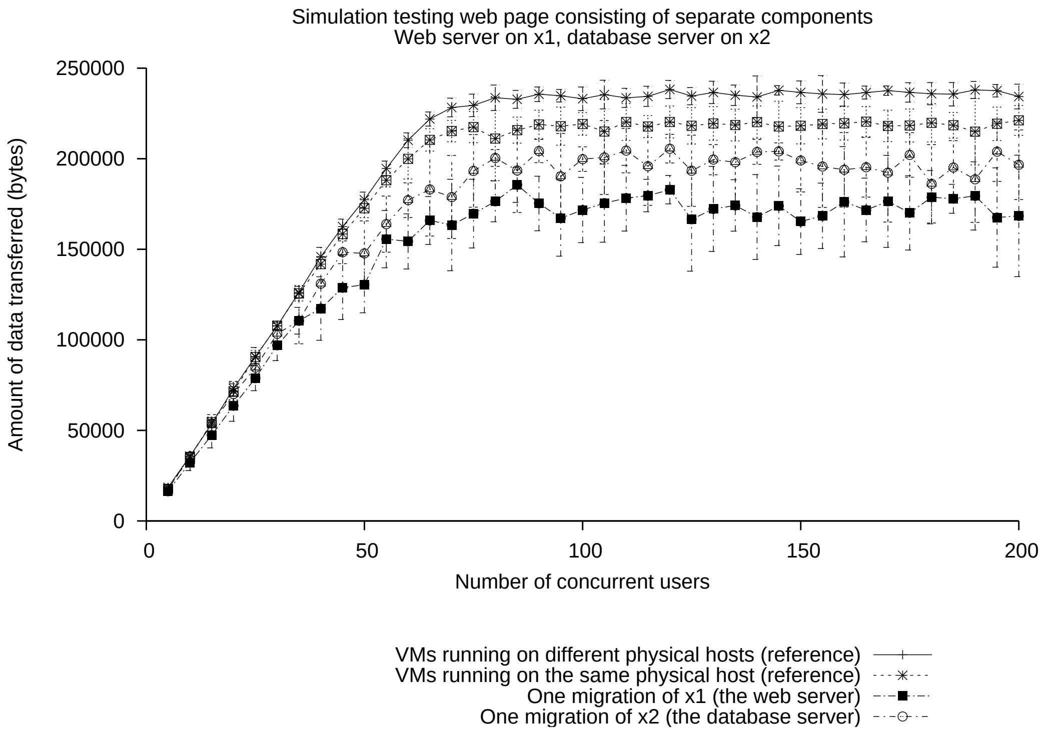


Figure 5.16: The total amount of data transferred in 45 seconds tests. Simulating user activity and measured the response time from 5 to 200 users. Each user has a delay between 0 and 10 seconds (random) between each request.

tably pressure to the servers. Figure 5.16 verifies this by showing that the total amount of data transferred flattens out at approximately 60 users. According to the graphs, the difference in response times increases when the servers are fully loaded (> 60 users). The cost of migrating the virtual machines at below 60 concurrent users is very low.

5.6 Impact on users of the services

It is difficult to scientifically measure the impact on services from the users point of view without a larger social study, but some personal qualitative observations have been made.

A script was implemented that simulated graceful failures to the physical host that hosted the virtual machine running a specific service every 30th second. Three observations were made, each briefly explained in the following sections.

5.6.1 Highly available internet radio streaming server

The internet radio streaming server that was installed on one of the virtual machines was commonly listened to, also with several concurrent listeners. Gaps in the audio stream were very rare, and by using reasonable settings on the buffer length (default settings are usually excellent) graceful failures were not noticed at all. Three presentations with demonstrations have been held, and none of the listeners were able to notice any outage in the audio stream at all.

5.6.2 Highly available web server

The experience on the web server was that graceful failures never were an issue. Since web traffic is bursted, delays up to at least one second are most likely ignored by the users.

5.6.3 Highly available multiplayer game server

A BZFlag[5] server was installed on *x3* and made public accessible. At max, 12 concurrent players were connected and in instant action since the world size of the map was very small (200 meters). The players were random players from around the world that had connected to the server through the public server list provided through the BZFlag clients.

[5]www.bzflag.org

The players were not aware of what happened, and they had only positive feedback when being asked about how the connection was. Nobody complained over lag or outages, not even when being asked about it directly. The conclusion is that small outages caused by graceful failures are unnoticeable for the players, and this corresponds to the more elaborative study on this topic by Clark et al. [2].

Chapter 6

Discussion

The main part of this project is to develop and implement an add-on to Heartbeat which makes it possible to configure a proof of concept working model of a high availability cluster using Xen virtual machines. This is a novel approach using Heartbeat and Xen in a high availability cluster based on virtualization and live migration. Below are the objectives of the thesis listed with a short summary on what have been done:

1. *Review previous work on high available clustering using virtualization and open source tools.*

    - *In previous research, no-one has configured a high availability cluster using open source tools and virtualization, and there exist no documentation on how to do it.*
    - *Previous research support virtualization used in clusters to create high availability solutions [2].*

2. *Implement an experimental high availability cluster using Heartbeat and Xen.*

    - *For successfully configuring a high availability cluster using Heartbeat and Xen, an add-on to Heartbeat which makes Heartbeat able to live migrate virtual machines must be developed.*
        - An add-on has been developed. This add-on adds the live migration functionality, such that if a VM is already running in the cluster at the time another physical node issues the *start* command for that VM, it gets live migrated to the new physical node instead of being cold booted.
        - The cluster is configured using Heartbeat, the live migration functionality add-on and Xen in a fully functional proof of concept environment.

3. *Do an assessment with a scientific approach to validate the setup and use statistical methods to analyze the results if objective 2 is completed successfully.*

   - *The most important property of a high availability setup is its availability, which makes it reasonable to measure the degree of availability to validate the usefulness of the approach. Scientific methods should be used in the measurements.*
     - A list of measurements is proposed in section 3.7.
     - A set of the proposed measurements were selected and performed.
     - The results from each performed measurement are separately discussed in chapter 5.
     - An overall discussion of the results follows in the next section (section 6.1).

The measurements and results show that the objectives have been successfully completed. The topology has shown to be highly available, very robust and extremely flexible. During the project, over 20000 migrations have occurred (see figure 6.1).

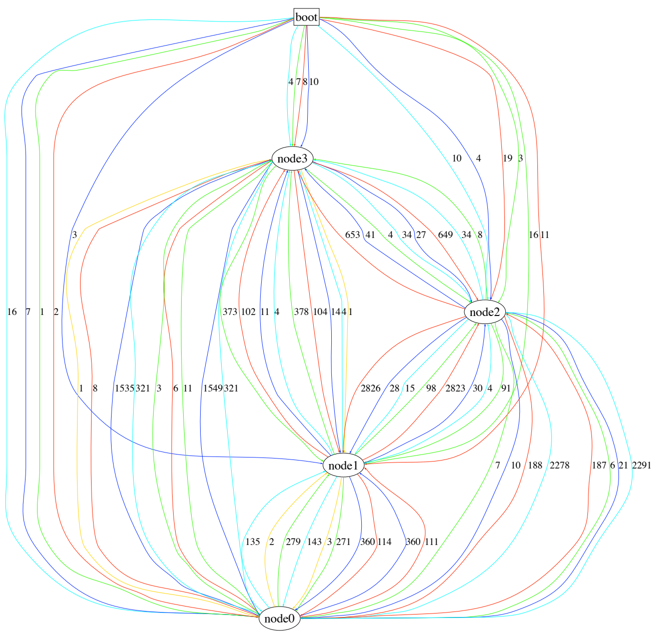


Figure 6.1: The Virtual Machine Migration Monitor shows that over 20000 migrations have occurred in total. See appendix A.6 for a larger figure.

6.1 Results

6.1.1 The outage of graceful failures in detail

It is difficult to measure the length of an outage exactly. This is partly because of the many uncertain elements[1], and also because the measurements are actively affecting the migration performance. *pktgen* were used to generate UDP packets used in the measurements. The results in 5.4.1 show that the packet loss increases when the granularity of the measurements is increased.

The packet loss when measuring with 1000 packets per second is significantly higher than the packet loss when measuring with 10 and 100 packets per second. Therefore, it is reasonable to believe that the measurements with 10 and 100 packets per second have the most correct results. Using these measurements only, the outage caused by a graceful failure is calculated to be between 0.2 and 0.55 seconds on average.

Figure 5.8 shows that the total time from a graceful failure occur to it is fully repaired is approximately 15 seconds. This includes the time Heartbeat use to notify the other nodes about the failure, the decision making, migration request from the new node, and the live migration itself. Figure 5.5 shows that a live migration is performed in totally 4 seconds on a VM which have 256MB of RAM.

TCP traffic is even less affected than the UDP traffic due to the re-transmission introduced by TCP. The few packets that are lost are automatically being resent by the protocol itself, causing the sender and receiver of the packets totally unaware of the packet loss - after all the packets that have been lost are re-sent anyway. This has been tested in section 5.5.1 by simulating a graceful failure when a large file is being transferred. The file transfer continues after the failure without any problems, and the only affection is that the total time to copy the 600MB file increased 3.6 seconds on the average, from a total of 70.8 seconds.

6.1.2 Migration decreases network performance slightly

According to [2], Xen is using so called "Dynamic Rate-Limiting" to limit the bandwidth usage for migration traffic if the network is heavily utilized. This is done because the migration traffic should affect the ongoing network traffic as little as possible. From the experiments in this project it has become clear that migrations do affect performance on services to some degree anyway, but this was to be expected (shown in figure 5.14). Without the dynamic rate-limiting, the negative impact would probably be higher.

[1]E.g. other network traffic, processes, behavior of the network protocols, performance of the network equipment (the NICs and the switch).

6.1.3 High load causes longer outages when a graceful failure occurs

Another finding was that the outages caused by graceful failures are longer if the servers and the network are heavily loaded. This is clearly visible in figure 5.16 and 5.15. Taking this into consideration, the results in table 5.2 makes more sense in the way that higher packet intensity causes longer outages. The footprint of a migration, figure 5.5, shows that the network gets loaded with packets in a longer time than the actual outage is experienced. Depending on the amount of memory in the virtual machine, this time will vary. In our experiment, the amount of memory was 256MB and the time to migrate was approximately 4 second (shown in the footprint). Due to the three phases (see section 5.3) in the migration process, it is reasonable to believe that the amount of memory and the load of the server should not affect the outage times. This finding shows that this might not be true.

6.1.4 The users of services are insignificantly affected by graceful failures

User satisfaction and user experiences are important factors in high availability systems running business critical services. The requirement of the system is that the data is available to the users and that the users are able to do their job. A system where the users experience outages frequently is not a adequate high availability system - even though the outages are small.

Two latency critical services were installed in our cluster to briefly test user satisfaction when graceful failures occurred frequently (every 30th second). These services were a online multiplayer game server and a internet streaming radio station (see section 5.6). At max, 12 concurrent users were connected to the game server and no-one experienced outages. The internet streaming radio station was used as a demonstration in three presentations, and none of the listeners noticed any outages or gap in the audio stream.

The conclusion is that even though certain packet loss is measurable by scientific methods, the users do not notice the outages. An outage that is not noticed by any user is considered as an insignificant outage.

6.2 SPOF analysis

We are benefiting from making the software able to run on all of the physical nodes available, which makes the hardware redundant even though we have less hardware than before available (in terms of number of physical nodes). This removes the physical nodes as the single point of failure automatically. However, new SPOFs are introduced:

The seemingly everlasting problem with single points of failures still exists in the software used for controlling and monitoring the virtual machines and in the virtualization software.

Heartbeat is given great responsibility and total control over the virtual machines. If Heartbeat for some reason malfunctions, the services in the entire network might become unavailable. Heartbeat is used by many companies for critical service hosting, and the bugs are usually fixed while the version is in unstable status. However, there are no guarantees and, even if Heartbeat is completely free from bugs, it still requires configuration files written by humans.

6.3 Possible future improvements

6.3.1 Adding a second layer of high availability

This thesis has focused on one layer of high availability software. This layer is located on the physical nodes, and is monitoring the health of the virtual machines. There are two disadvantages with this approach that can be coped with by adding another level of high availability:

- Because of the strong separation between the virtual machines and the physical machine, *heartbeat* on the physical machines can simply control whether the virtual machine is running or not. It has no direct access to the virtual machines to check that the critical services that the virtual machines are hosting are actually running and able to serve.

- The other disadvantage is that one layer high availability is less tolerant of instant and uncontrolled failures like power failures, because the virtual machines affected by the instant failures will have to be booted to repair availability. It is desirable to decrease the mean time to repair from an uncontrolled failure.

These two issues can be coped with by adding another layer of high availability. Figure 6.2 shows the first layer of high availability. This is *heartbeat* running on the physical nodes, controlling that the virtual machines are running. The second layer of high availability is illustrated in 6.3. Heartbeat is now also running on the virtual machines, directly controlling that the critical services are running. The virtual machines have to be coupled such that minimum two virtual machines are cooperating in hosting a critical service.

Only one of the two virtual machines are actively serving, while the other is passive and waiting for the active virtual machine to fail. If the physical node hosting the active virtual machine is failing instantly, the active virtual machine will also instantly fail. The passive virtual machine notices that the

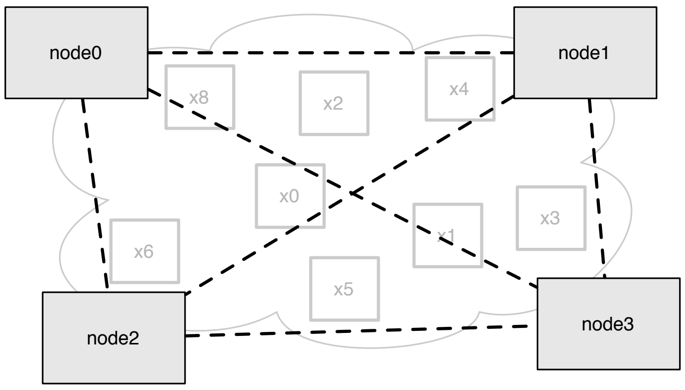


Figure 6.2: The first layer of high availability. Heartbeat is running on the physical nodes, controlling that the virtual machines are running. The dashed lines illustrate the heartbeats between the cooperating nodes.

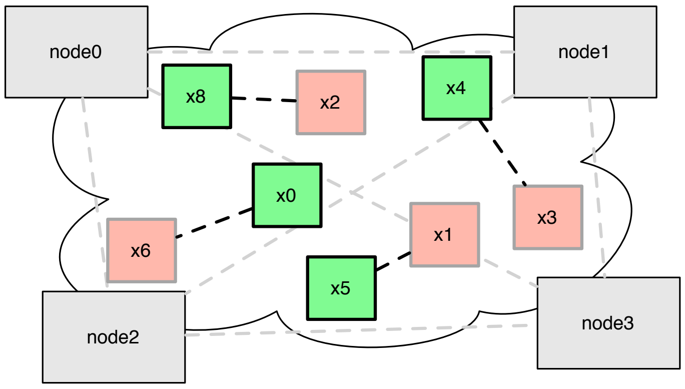


Figure 6.3: The second layer of high availability. Heartbeat is running on the virtual machines, controlling that the critical services (e.g. apache, mysql, etc) are running. The dashed lines illustrate the heartbeats between the cooperating VMs. The green VMs are currently active and serving content, and the red VMs are passive and waiting for the other VM to fail.

active one has died, claims the ownership of the IP address and starts to serve content in just a couple of seconds, depending on the configuration.

The basic requirement is of course that the virtual machines that are coupled never run on the same physical node. This rule is implemented in the heartbeat configuration on the physical nodes (the first layer). The disadvantages of two layers of high availability are increased complexity and that the number of virtual machines is doubled.

Adding a second layer of high availability is not a part of this thesis and has not been tried out in the test environment. It is, however, expected to be straight forward and easy to implement.

6.3.2 Voluntary cooperation

Using the PULL method by adding the request for migration in the *start* section, we can introduce voluntary cooperation into our solution. The failover node can choose whether it wants to receive the virtual machine by migration or cold boot it. In a clustered network like in our homogenous topology, this might sound like extraneous complexity. After all, the nodes should migrate the virtual machines no matter what.

In a heterogeneous network, however, with a variety of nodes with different processors, voluntary cooperation can be a necessity. Migration is a very complex process, and it has some requirements to be able to succeed. The most important requirement, along with shared storage, is that the CPU on the failover node is required to have many of the same features[2] as the CPU on the failed node. Using voluntary cooperation each of the nodes can have a list of supported nodes that are approved for migration. If the failed node is not in this list, the failover node can choose to cold boot the virtual machine instead of risking an unstable migration.

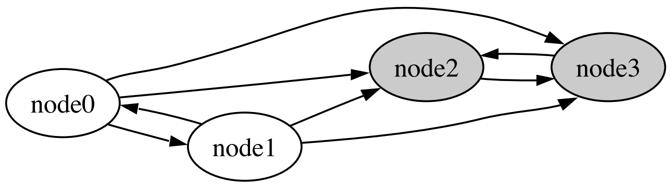


Figure 6.4: Showing migration possibilities in a heterogeneous network where node0 and node1 do not support migration from node2 and node3.

Voluntary cooperation is showed in a heterogeneous network in figure 6.4, where node0 and node1 are unable to retrieve virtual machines through migration from node2 and node3. The other way around is perfectly ok, node2

[2]The supported features (flags) of the processor are listed in */proc/cpuinfo*

and node3 can retrieve virtual machines through migration from node0 and node1.

Voluntary cooperation is not within the scope of this thesis, but should be kept it in mind for future development and improvement.

6.4 Production environment: A TO-DO list

The author has acquired knowledge and experience on the topic during this project, some that might be important to take into consideration when implementing the topology in a production environment.

- In order to increase flexibility, the physical nodes are required to have a large amount of RAM installed and for Xen to use. The ″amount of redundancy″ is relative to the amount of virtual machines each physical node is able to host, and to some degree it is valid to say that more RAM equals more redundancy.

- The Xen version used in this experiment was not successful in confirming that the receiving node had enough hardware resources available before the migration started. If the node were already running the maximum amount of virtual machines, the migration possibly failed - leaving the virtual machine in a migrating state.

- Heartbeat was initially developed for controlling services like apache and mysql that were running directly on physical hosts. Since Xen is ″hiding″ the total hardware resource usage, Heartbeat running in dom-0 was not able to calculate total hardware resource usage on the nodes. This way the decisions when choosing the best suited node for failover (failover node) were taken on incorrect information.

  Before deadline of this thesis, Heartbeat was shipped in a new version (version 2.0.5) that has a built in resource agent for Xen. This resource agent should be able to do these decisions based on correct information, and the topology would be further improved. Unfortunately, we did not have the time to update Heartbeat and test this feature.

- A dedicated network could be configured for Xen to migrate virtual machines on. This way, the business network would not be affected by migrations since they are running on separated networks. This would further decrease the impact on the critical services by live migrations.

- STONITH and fencing should be configured and installed to ensure data integrity.

6.5 Future work

In this thesis, the cost of migrating one virtual machine is measured in terms of time of outage and loss in performance during the migration process. It is reasonable to believe that the other virtual machines will be negatively affected by one of the virtual machines being migrated as well. This secondary cost of migration should be scientifically measured to take part in decision making regarding when to migrate the virtual machines.

Network performance is especially important in this topology where several services can be hosted on one physical node. The network performance of the virtual machines is as important as other benchmarks (i.e. CPU performance) in a server environment, and previous research on this topic has not come to the author's attention. Interesting areas of future research are network performance on a VM compared to the physical node, network performance between VMs on the same physical node and how network activity affects the latency on other virtual machines.

It would be interesting to do research on the cost of updating the ARP tables when a migration occurs. A switch used in the cluster is using an ARP table when IP addresses are translated to MAC addresses. When a virtual machine is migrated to another physical node, it is reasonable to believe that the ARP table must be updated before successfully transmitting the first package to the virtual machine on the "new" physical node after migration has occurred.

Chapter 7

Conclusion

In this work, an add-on for Heartbeat has been developed which makes Heartbeat capable of managing virtual machines in high availability clusters. Using the add-on, the preferred methodology for repair is *live migration* of the virtual machines between the physical nodes. This method minimizes the outage caused by a graceful failure[1], leaving the users of the high availability services completely unaware of the failures. Scientific experiments have shown that the solution is very robust, cause very short MTTR[2] and hardly affect the users if a graceful failure occur.

VMware has sold high availability cluster management as a supplementary software package for their ESX server for a while. Hence, this topology is not completely new. However, having only commercial solutions is not satisfactory. Solutions based on open source that prioritize quality over profit are needed, developed by the community itself.

The developed add-on consists of a LSB compliant[3] *init.d* script and a daemon. The daemon should run on all of the physical nodes and listen for migration requests. The *init.d* script is used when a virtual machine should be started, and if the virtual machine is already running on another physical node, the scripts will issue a live migration request of the VM instead of booting the VM from a powered-off state.

The traditional method for repair is first to *shut down* and then *cold boot* the VMs on the other nodes. This method is causing far longer outages than the proposed methodology which is using *live migration*. Other disadvantages with the traditional approach are that connections are lost, that the users need to re-connect to the services and that the services are restarted - which is severe in multiplayer games for example. To deal with some of these problems, a method by adding a second layer of high availability is proposed and dis-

[1]Graceful failure is defined as controlled failures to a physical node, e.g. rebooting or shutting down the node.

[2]Mean time to repair

[3]Linux Standard Base compliant, see section 4.1.2.

cussed as future work (section 6.3.1).

The consequences of graceful failures are measured using ICMP echo requests and replies, TCP and UDP traffic and real services (file transfers, web and database services). The obtained results show that the outages are extremely short and that the connection states are preserved during the migration. In fact, the outages are short enough to not be noticed by users connected to web servers, multiplayer game servers and audio streaming servers that run on hardware which is simulated to fail gracefully.

Bibliography

[1] Hewlett Packard. Virtualisation - it supply meets business demand. 5983-0462EEE. Rev. 1, September 2005.

[2] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA, May 2005.*, 2005.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[4] Dejan S. Miloji, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process migration. *ACM Comput. Surv.*, 32(3):241–299, 2000.

[5] Bryan Clark, Todd Deshane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Neefe Matthews. Xen and the art of repeated research. In *USENIX Annual Technical Conference, FREENIX Track*, pages 135–144. USENIX, 2004.

[6] Stephen Childs, Brian Coghlan, David O'Callaghan, Geoff Quigley, and John Walsh. A single-computer grid gateway using virtual machines. *aina*, 01:310–315, 2005.

[7] 4th Annual Linux Showcase and Conference. *Linux-HA Heartbeat System Design*, Atlanta, Georgia, USA, October 10-14 2000.

[8] UKUUG LISA/Winter Conference, High-Availability and Reliability. *The Evolution of the Linux-HA Project*, Bournemouth, February 25-26 2004.

[9] Alan Robertson. Highly-affordable high availability. *Linux Magazine*, November 2003.

[10] M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, **2**:333, 1994.

[11] D. Verma et al. Policy based sla management in enterprise networks. In *Policy Workshop 2001*. Springer Verlag, 2001.

[12] Peter S. Weygant. *Clusters for High Availability - A Primer of HP Solutions*. 0-13-089355-2. Hewlett-Packard Professional Books, second edition edition, 2001.

[13] Mark Burgess. *Analytical Network and System Administration. Managing Human-Computer Networks*. 0-470-86100-2. John Wiley & Sons, Ltd, 2004.

[14] Karl Kopper. *The Linux Enterprise Cluster*. 1-59327-036-4. No Starch Press, 2005.

[15] Budrean S., Yanhong Li, and Desai B.C. High availability solutions for transactional database systems. In *Database Engineering and Applications Symposium, 2003. Proceedings. Seventh International*, pages 347–355, 16-18 July 2003.

[16] Wensong Zhang and Wenzhuo Zhang. Linux virtual server clusters. *Linux Magazine*, November 2003.

[17] Dna Herington and Bryan Jacquot. *The HP Virtual Server Environment*. 0-13-185522-0. R. R. Donnelley & Sons, Inc., 2005.

[18] vmware. *ESX Server 2 - Mainframe-Class Virtual Machines for the Most Demanding Environments - Administration Guide, version 2.5.1*.

[19] Rosenblum M. and Garfinkel T. Virtual machine monitors: current technology and future trends. *Computer*, 38(5):39–47, May 2005.

[20] Håvard Bjerke. Hpc virtualization with xen on itanium. Master's thesis, NTNU, 2005.

[21] A. Whitaker, M. Shaw, and S. Gribble. Denali: Lightweight virtual machines for distributed and networked applications. In *In Proceedings of the USENIX Annual Technical Conference*, Monterey, CA, June 2002.

[22] Renato J. Figueiredo, Peter A. Dinda, and J. Fortes. A case for grid computing on virtual machines. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 550, Washington, DC, USA, 2003. IEEE Computer Society.

[23] Kyrre Begnum, Karst Koymans, Arjen Krap, and John Sechrest. Using virtual machines in system and network administration education. In *the USENIX/SANE Conference 2004*, 2004.

[24] Jim Gray and Daniel P. Siewiorek. High-availability computer systems. *IEEE Computer*, 24(9):39–48, 1991.

[25] Robert Olsson. pktgen the linux packet generator. In *Proceedings of the Linux Symposium*, pages 11–24, Ottawa, Ontario, Canada, July 20nd-23th 2005. The Linux Symposium.

[26] Fabian Schneider and Jörg Wallerich. Performance evaluation of packet capturing systems for high-speed networks. In *International Conference On Emerging Networking Experiments And Technologies, Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, pages 284 – 285, New York, NY, USA, 2005. ACM Press.

[27] Stephen Northcutt and Judy Novak. *Network Intrusion Detection, third edition. 0-7357-1265-4*. David Dwyer, New Riders Publishing, 2003.

[28] Dr. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. 0-201-63346-9. Addison-Wesley, 1994.

[29] Aravind Menon, JOse Renato Santos, Yoshio Turner, G. (John) Janakiraman, and Willy Zwaenepoel. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 13–23. ACM Press, 2005.

[30] Maurice David Wornhard. Scalability analysis of state-synchronized openbsd-firewalls for synthetic voip-traffic. Master's thesis, Oslo College University, 2006.

Appendix A

Appendix

A.1 GNBD usage

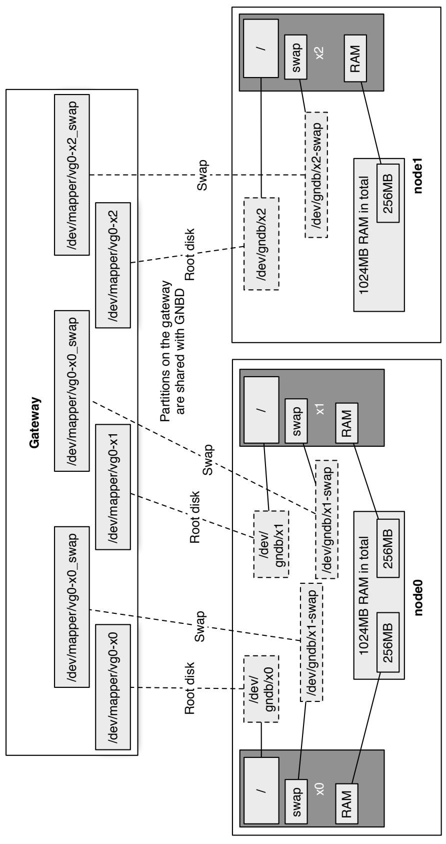


Figure A.1: Two physical nodes (node0 and node1) are hosting 3 virtual machines (x0, *x1* and *x2*). The root disks of the virtual machines are stored on the gateway (shared storage).

A.2 Virtual Machine configuration file

A.2.1 /etc/xen/x0

```
#  -*- mode: python; -*-
kernel = "/boot/vmlinuz-2.6.12.6-xenU"
memory = 256
name = "x0"
vif = [ 'bridge=xenbr0' ]
disk = [ 'phy:/dev/gnbd/x0,sda1,w', 'phy:/dev/gnbd/x0_swap,sda2,w' ]
root = "/dev/sda1 ro"
extra = "2"
on_poweroff = 'destroy'
on_reboot   = 'restart'
on_crash    = 'destroy'
```

A.3 Heartbeat configuration

A.3.1 /etc/ha.d/ha.cf

```
bcast eth0
debug 0
keepalive 1
warntime 10
use_logd on
initdead 60
udpport 694
auto_failback yes
deadtime 30
node node0.cluster node1.cluster node2.cluster node3.cluster
crm yes
```

A.3.2 /var/lib/heartbeat/crm/cib.xml

```
<cib>
 <configuration>
  <crm_config>
   <nvpair id="transition_idle_timeout" name="transition_idle_timeout"
   value="120s"/>
   <nvpair id="symmetric_cluster" name="symmetric_cluster" value="true"/>
   <nvpair id="stonith_enabled" name="stonith_enabled" value="false"/>
   <nvpair id="suppress_cib_writes" name="suppress_cib_writes"
   value="false"/>
   <nvpair id="default_resource_stickiness" name="default_node_stickiness"
   value="0"/>
   <nvpair id="require_quorum" name="require_quorum" value="true"/>
   <nvpair id="no_quorum_policy" name="no_quorum_policy" value="freeze"/>
  </crm_config>
  <nodes/>
  <resources>
   <primitive class="lsb" id="x0_id" type="x0"/>
   <primitive class="lsb" id="x1_id" type="x1"/>
   <primitive class="lsb" id="x2_id" type="x2"/>
   <primitive class="lsb" id="x3_id" type="x3"/>
   <primitive class="lsb" id="x4_id" type="x4"/>
   <primitive class="lsb" id="x5_id" type="x5"/>
   <primitive class="lsb" id="x6_id" type="x6"/>
   <primitive class="lsb" id="x7_id" type="x7"/>
  </resources>
  <constraints/>
  </configuration>
  <status/>
</cib>
```

A.4 Installation of Linux Packet Generator

The Linux packet generator module was already compiled in the running kernel (2.6.15). The pktgen.c was replaced with the pktgen.c modified by Fabian Schneider [26], and the modules compiled:

```
cd /usr/src/linux/net/core
make -C /usr/src/linux SUBDIRS=$PWD modules
```

When the modules were compiled, pktgen.ko was copied to /lib/modules/2.6.15/kernel/net/core/ to overwrite the previous version. This could also be done by make modules_install in the kernel directory.

A.5 Configuration of Linux Packet Generator

```
#!/bin/sh
modprobe pktgen -f

PGDEV=/proc/net/pktgen/pg0

function pgset(){
        local result
        echo $1 > $PGDEV

        result=`cat $PGDEV | fgrep "Result: OK:"`
        if [ "$result" = "" ]; then
                cat $PGDEV | fgrep Result:
        fi
}

function pg(){
        echo inject > $PGDEV
        cat $PGDEV
}

pgset "odev eth0"
pgset "pkt_size 214"
pgset "dst 80.232.38.241"
pgset "count 60000"
pgset "ipg 500000"om
pgset "start"
```

A.6 The Virtual Machie Migration Monitor

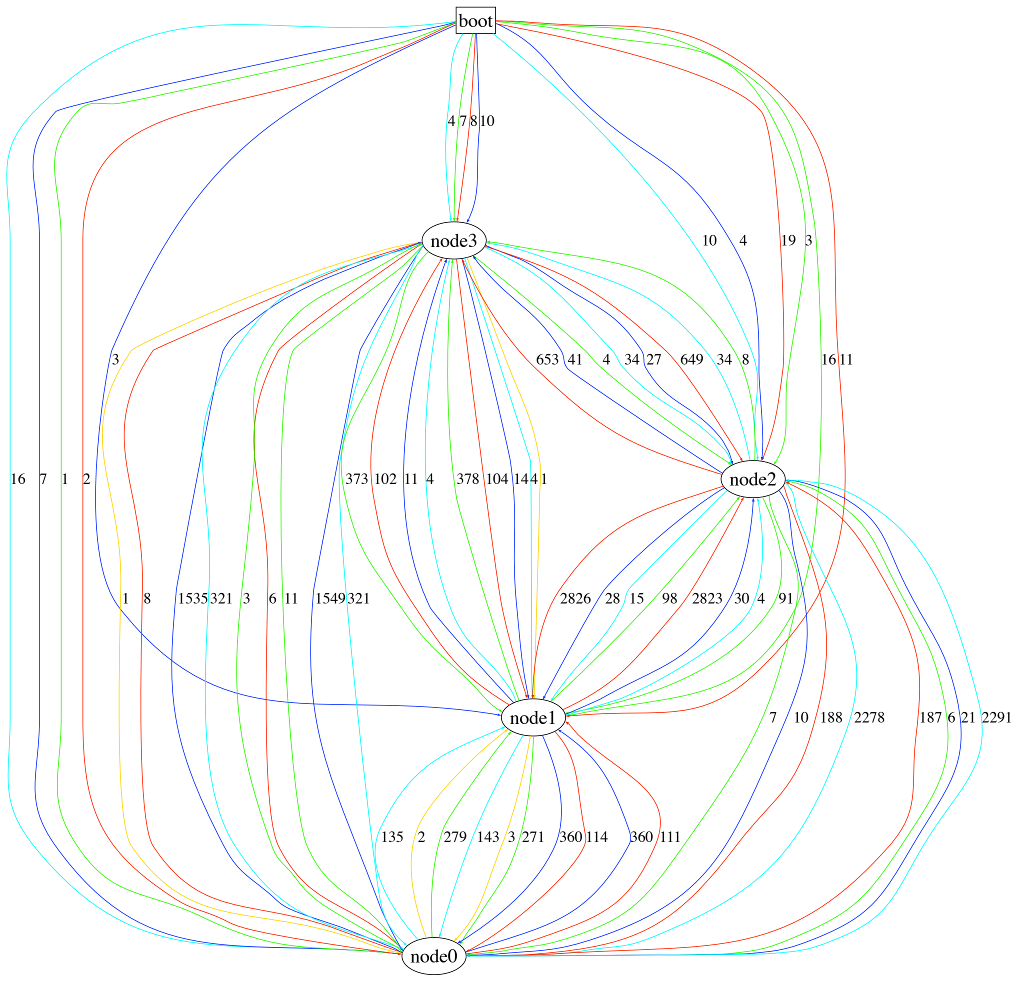


Figure A.2: The over 20000 migrations that have occurred during the work on this thesis illustrated. The physical nodes are illustrated by four ellipses. The migrations are illustrated by arrows, colored differently to represent the different virtual machines: x0 is green, x1 is blue, x2 is light blue and x3 is red. A number is attached to each arrow that represents the number of times that migration has occurred.