

Virtual machines: implementations

First Hadrian Popescu, Second Cosmin Natea, Third Calin Enachescu and Fourth Bogdan Crainicu

Abstract — Most forms of virtualization involve a computing design pattern that relate a consumer and provider. A consumer and provider interact using some interface. Virtualization places an intermediary between consumer and provider that acts on both sides of the interface, providing the interface for the actual consumer and consuming the interface of the actual provider.

Keywords — virtualization, high availability, paravirtualization.

I. INTRODUCTION

THE original sense of the term virtualization, dating from the 1960s, is in the creation of a virtual machine using a combination of hardware and software. For convenience, we will call this platform virtualization.

The creation and management of virtual machines has also been referred to as creating pseudo machines and server virtualization more recently. The terms virtualization and virtual machine have both also acquired additional meanings through the years.

Platform virtualization is performed on a given hardware platform by "host" software (a control program), which creates a simulated computer environment (a virtual machine) for its "guest" software. The "guest" software, which is often itself a complete operating system, runs just as if it were installed on a stand-alone hardware platform. Typically, many such virtual machines are simulated on a given physical machine. For the "guest" system to function, the simulation must be robust enough to support all the guest system's external interfaces, which (depending on the type of virtualization) may include hardware drivers.

There are several approaches to platform virtualization, listed below based on how complete a hardware simulation is implemented.

Emulation or simulation the virtual machine simulates the complete hardware, allowing an unmodified "guest" OS for a completely different CPU to be run. This approach has long been used to enable the creation of software for new processors before they were physically available.

Native virtualization and full virtualization the virtual machine simulates enough hardware to allow an unmodified "guest" OS (one designed for

the same CPU) to be run in isolation. Typically, many instances can be run at once.

Hardware enabled virtualization the virtual machine has its own hardware and allows a guest OS to be run in isolation. In many instances the virtual machine runs an operating system different than that of the host computer. Typically, the number of virtual machines installed on a primary OS is limited only by the host computer's hardware and memory resources.

Partial virtualization (and including "address space virtualization") the virtual machine simulates multiple instances of much (but not all) of an underlying hardware environment, particularly address spaces. Such an environment supports resource sharing and process isolation, but does not allow separate "guest" operating system instances.

Paravirtualization the virtual machine does not necessarily simulate hardware, but instead (or in addition) offers a special API that can only be used by modifying the "guest" OS.

Operating system-level virtualization virtualizing a physical server at the operating system level, enabling multiple isolated and secure virtualized servers to run on a single physical server.

Application Virtualization running a desktop or server application locally, using local resources, within an appropriate virtual machine; this is in contrast with running the application as conventional local software, i.e. software that has been 'installed' on the system.

II. BACKGROUND MATERIAL AND PREVIOUS WORK

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.

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.

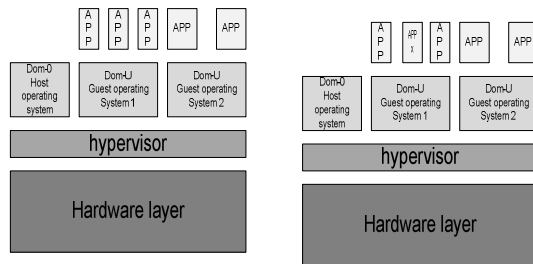


Figure 1

Figure 1 (left) shows an example where one single physical computer is running three operating systems simultaneously. Two of them are virtual machine (Doom-U) and one is the host operating system (Doom-0). The abstraction layer (hypervisor) provides encapsulation upward in the layers. The result is that the complete software package running inside virtual machine is strongly encapsulated, including connection states, CPU states and memory states. If a virtual machine is compromised, it does neither affect the host operating system nor the other virtual machine (figure 1 right).

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 dependences and hardware which both are results of how decoupled the virtual machine really is from the hardware. The different virtualization technique are:

Full virtualization makes virtual machines that are very flexible. The host OS emulates the complete hardware package which are visible for the virtualization operating (figure 2), this makes it possible to for example emulate x86 architecture (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).

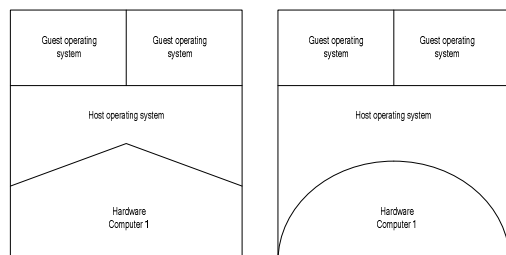


Figure 2

OS virtualization is virtualization on the host OS level (figure 3). This method gives very high performances, for the cost of flexibility. The performance of a virtualized operating system is exact the same as the performance of the physical host (operating system 1 does not support the hardware on computer 2, migration is impossible).

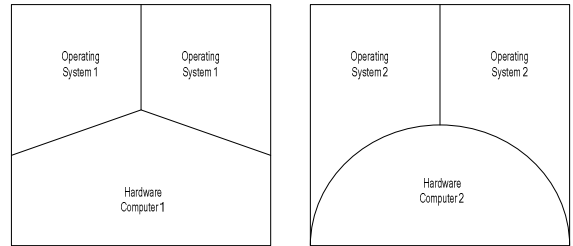


Figure 3

Para virtualization is a method used by Xen, Denali, Vmware ESX, which can be placed performances wise between full virtualization and OS virtualization (figure 4).

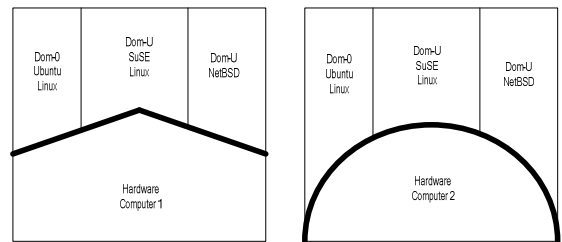


Figure 4

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 command to them, boot, reboot and shutdown. The black layer is called the hypervisor software, and it provides little overhead compared to the thicker layers provided by full virtualization.

Relative performance and flexibility in full virtualization, para virtualization and OS virtualization (figure 5).

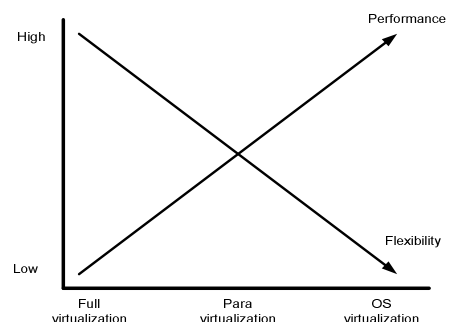


Figure 5

Virtualization advantages

Virtualization has several advantages that can be utilized in high availability clusters.

Flexibility is given in several ways. It is added because one can run more than one instance of an operating system on 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'.

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.

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.

Hardware utilization is most likely increased and efficiently used if more than one operating system is hosted simultaneously.

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.

Virtualization disadvantages

Overhead causing decreased performance has been the biggest con with virtualization. Performance is often being compromised due to flexibility, or contradictory.

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.

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 internet service providers, 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. It is important that planned and unplanned outages do not exceed this service level degree.

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.

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. Knowledge about how often failures occur (MTBF – mean time before failure) and the time to repair the failures (MTTR – mean time to repair) can be used in this formula for calculating availability:

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

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.

The availability of a system is measured in the famous "nines", given in percent of the total time where the system should be available:

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

III. 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 (figure 6). 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.

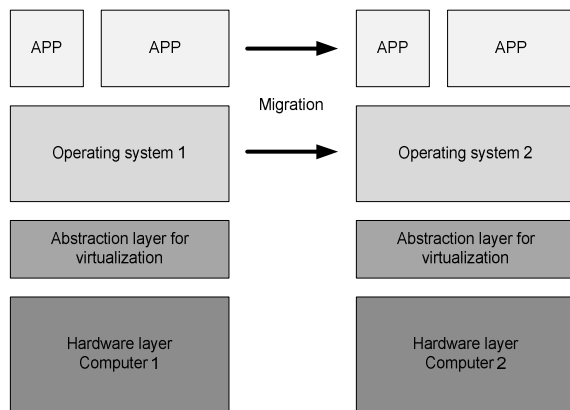


Figure 6

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 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 (figure 7).

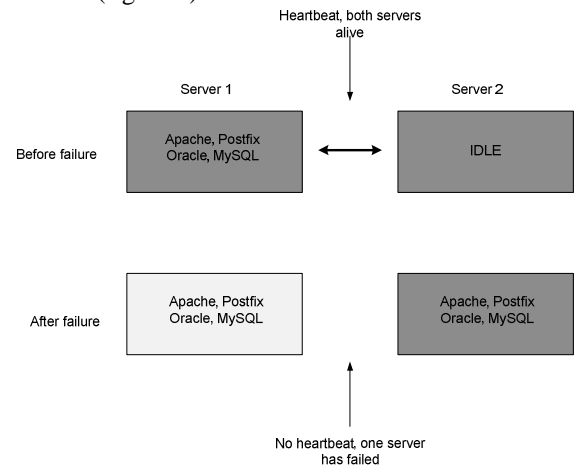


Figure 7

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 (figure 8). 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. In our setup with virtual machines as services, the IP addresses follow the virtual machines internally without involving heartbeat. Heartbeat is as of this writing in version 2.1.2 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(N²)). For increased security, redundant communication paths for Heartbeat are recommended. In large clusters, this can be solved by adding a dedicated network for Heartbeat to use.

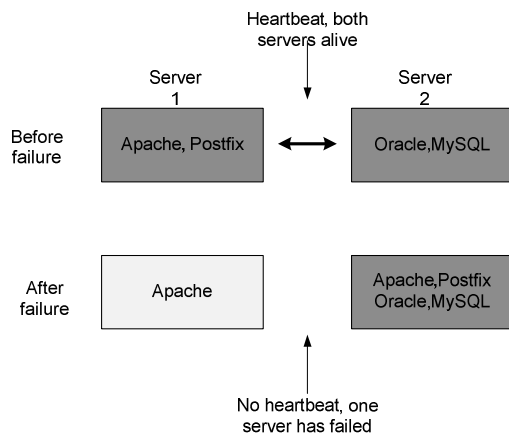


Figure 8

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. An overview of Xen is given in the paper "Xen and the art of virtualization". The x86 architecture supports 4 different privileges in hardware that is called rings (figure 9), 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. 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. The goal of Xen is to scale up to 100 virtual machines, where all of them run services, on one single physical computer. However, independent research shows that this number of concurrent VMs may be too high.

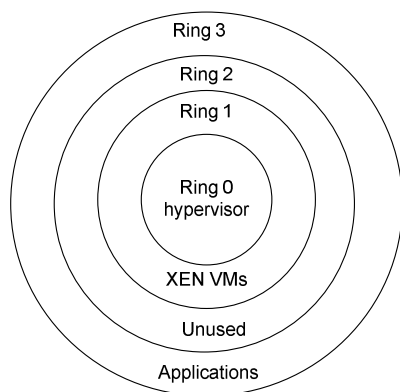


Figure 9

Experiments with web servers reveal decrease in

performance with as few as 16 concurrent VMs. Also, due to the memory management in Xen, each of the VMs has dedicated memory allocation, that is not dynamically shared. If we dedicate 256MB ram to each of the virtual machines, 50 VMs will require over 12GB of memory.

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 6. 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. 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.
2. The physical computers must be homogenous, which means that the CPUs must support the same features.

IV. OBJECTIVES AND SYSTEM MODEL

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. 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.

System model

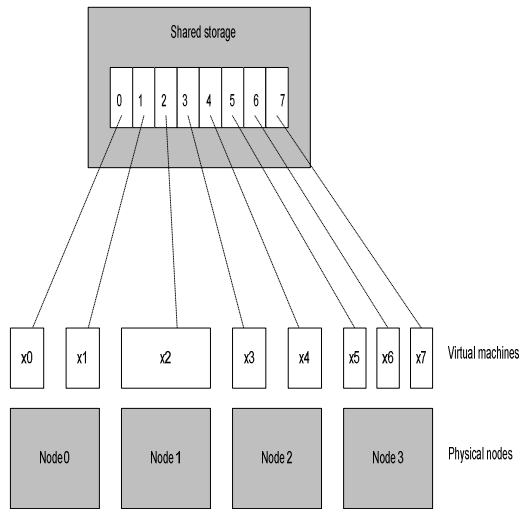


Figure 10

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 10 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.

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 11).

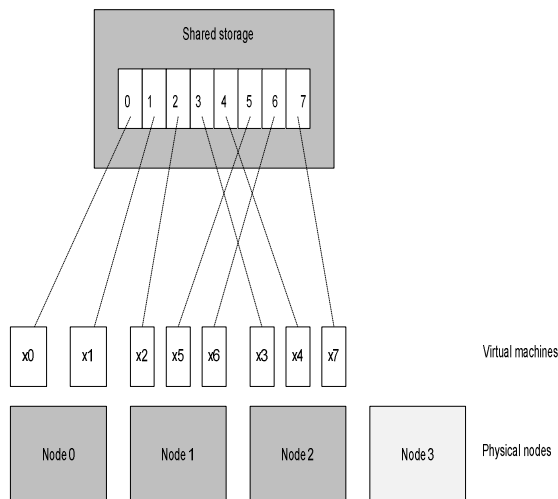


Figure 11

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 12. This "link" has to be developed for the system design to be fully functional.

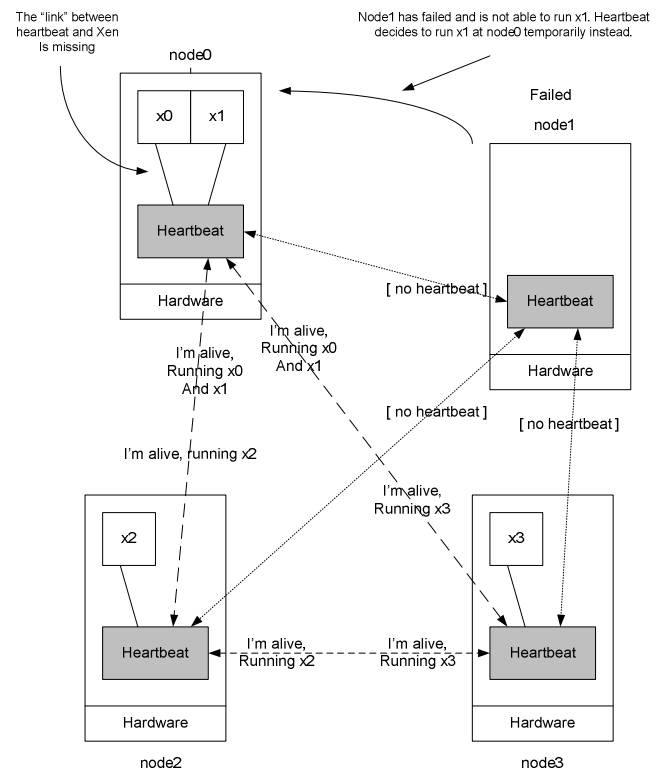


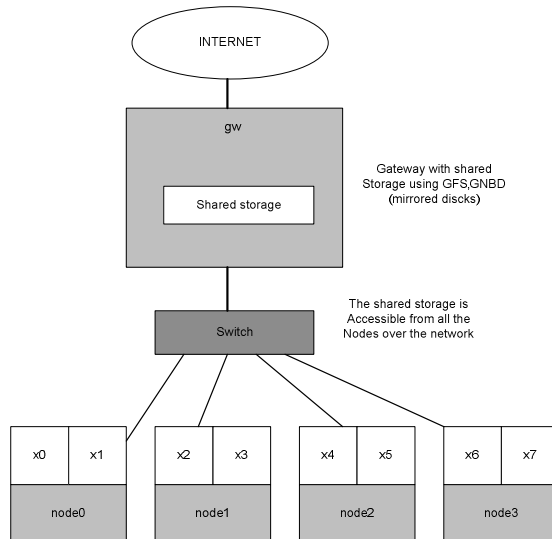
Figure 12

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.

Topology used in the experiment

Due to this being an 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 13). The topology used in this experiment is showed in figure 13. The figure reveals many single points of failure, but this is of no consequence because the experiments will be about convergence towards the stable state when unavailability has occurred, and not measuring fault tolerance in regards to redundant hardware.



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.

As uncontrolled failures are simply about shutting down and starting resources, it is straight forward to implement it with Heartbeat. 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.

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:

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 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.

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).

V. RESULTS AND FUTURE

It is difficult to measure the length of an outage exactly. This is partly because of the many uncertain elements¹, and also because the measurements are actively affecting the migration performance. pktgen were used to generate UDP packets used in the measurements.

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.

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.

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. Conclusion: migration decreases network performance slightly.

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).

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.

SPOF analysis, 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.

Bibliografie

- [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 *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] Bryan Clark, Todd Dethane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Neeffe Matthews. *Xen and the art of repeated research*. In *USENIX Annual Technical Conference, FREENIX Track*, pages 135–144. USENIX, 2004;
- [5] Stephen Childs, Brian Coghlan, David O'Callaghan, Geoff Quigley, and John Walsh. *A single-computer grid gateway using virtual machines*. 2005;
- [6] UKUUG LISA/Winter Conference, *High-Availability and Reliability. The Evolution of the Linux-HA Project*, Bournemouth, February 25-26 2004.
- [7] Alan Robertson. *Highly-affordable high availability*. *Linux Magazine*, November 2003;

- [8] Mark Burgess. *Analytical Network and System Administration. Managing Human-Computer Networks*. 0-470-86100-2. JohnWiley & Sons, Ltd, 2004;
- [9] Karl Kopper. *The Linux Enterprise Cluster*. 1-59327-036-4. No Starch Press, 2005;
- [10] Wensong Zhang and Wenzhuo Zhang. *Linux virtual server clusters*. *Linux Magazine*, November 2003;
- [11] Dna Herington and Bryan Jacquot. *The HP Virtual Server Environment*. 0-13-185522-0. R. R. Donnelley & Sons, Inc., 2005;
- [12] vmware. *ESX Server 3 - Mainframe-Class Virtual Machines for the Most Demanding Environments - Administration Guide*;
- [13] Rosenblum M. and Garfinkel T. *Virtual machine monitors: current technology and future trends*. *Computer*, 38(5):39-47, May 2005;
- [14] Jim Gray and Daniel P. Siewiorek. *High-availability computer systems*. *IEEE Computer*, 24(9):39-48, 1991.
- [15] Stephen Northcutt and Judy Novak. *Network Intrusion Detection*, third edition. 0-7357-1265-4. David Dwyer, New Riders Publishing, 2003.
- [16] Dr. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. 0-201- 63346-9. Addison-Wesley, 1994.
- [17] 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.