Dependable Systems 2010
Virtualization Fault Tolerance
Project Report

Matthias Richly and Christopher Schuster

Hasso Plattner Institute for Software Systems Engineering
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam
{matthias.richly,christopher.schuster}@student.hpi.uni-potsdam.de

Abstract. Server virtualization is a common trend in modern IT infrastructures. Virtualization of the x86 processor architecture brought this development a big step further, because this hardware architecture is widely spread and cheap compared to other platforms which supported virtualization for a long time, such as IBM mainframes. While virtualization has a lot of advantages, it introduces a considerable drawback in terms of dependability. If multiple physical machines are consolidated into virtual machines running on one single physical host, they all will fail, if the physical host breaks down. Vendors of virtualization solutions have recognized this problem and therefore developed solutions to increase fault tolerance in virtualized environments. In this report we will give a short survey of existing virtualization solutions and their reliability features and then deeply analyze one solution, namely VMware vSphere. In the end we will also say a few words about Kemari, an open source research project concerned with virtual machine synchronization for Xen and recently also for KVM. It uses a different synchronization approach as VMware vSphere.

Key words: Virtualization Fault Tolerance High Availability VMware

1 Introduction

1.1 Motivation

X86 server virtualization can bring a lot of benefits into business IT infrastructures. It permits to run several virtual machines with possibly different operating systems on one single physical host simultaneously. Thereby it can optimize the resource utilization on the physical machines and reduce the number of machines necessary to deliver all required services. Since virtual machines (VMs) abstract from the actual hardware, they are very flexible and can be easily (re-)deployed on different hosts, e.g. for host maintenance or depending on the current resource requirements of the VMs. Some virtualization solutions can even migrate running virtual machines between hosts. Virtualization also facilitates management, since administrators can usually configure, deploy and monitor all VMs

via a central management utility. All these features will ultimately save a lot of money for the business.

On the other hand, consolidation of many virtual machines on one single physical host requires this host to be very reliable, since if it fails, all VMs and thereby all services provided by these VMs will fail. But x86 hardware is not developed for maximal reliability, it is rather a trade off between few reliability features and an attractive sales price. To increase system availability despite and in presence of host failures, vendors of virtualization solutions integrated some reliability features directly into the virtualization software stack. Here they usually distinguish between two kinds of features: *high availability* and *fault tolerance*[1]. The characteristics of these two are described in the next section.

1.2 High Availability vs. Fault Tolerance

High availability (HA) tries to minimize the downtime of a service experienced by users in the presence of host failures. The main goal is to restart VMs, which became unresponsive due to a crash or network connection loss of the physical host. This is achieved by periodic monitoring of the hosts to detect failures, e.g. via heartbeats. When a failure is detected, all VMs currently registered to be running on the failed host will automatically be restarted on other available hosts with sufficient resources. There is no need of human interaction. The configuration is active/passive, remaining hosts act as spares and standby or even run other VMs. Although this features does not prevent failures of services, i.e. there is some downtime, it can significantly reduce the MTTR, and therefore truly provides high(er) availability, also for the general definition of availability ($A = MTTF/(MTTF + MTTR)$).

In contrast to HA, fault tolerance (FT) targets zero downtime for protected VMs by using a *hot standby* configuration. A secondary VM is running on another host and is synchronized with the primary VM by means of continuous replication. Therefore it can still be considered an *active/passive* configuration. In case of a host failure, the VM on the second host keeps running and becomes the primary VM. There is no data loss, i.e. the active state of the VM is maintained on the secondary host. This is contrary to the HA feature described above, where the state is lost and the VM is restarted from the current disk image.

1.3 Product Overview

Table 1 shows common virtualization products and which of the reliability features they provide.

The VMware enterprise solution ESX/vSphere provides both features natively. This product will be further introduced and analyzed in the section 2.

Citrix provides a competing solution called XenServer. Xen started as a research project at the University of Cambridge and was then distributed by the

[1] This terminology is used in conjunction with virtualization solution features.

hypervisor	ESX/vSphere	XenServer	Hyper-V
vendor	VMware	Citrix	Microsoft
HA	yes	yes	yes
FT	yes	yes	no

Table 1. Common virtualization products and the reliability features they provide

XenSource Inc. company. In 2007, Citrix acquired XenSource and renamed the product to its current brand XenServer. By itself, XenServer only supports high availability. Fault tolerance can be added via Marathon's EvenRun VM product, which was specifically developed for XenServer.

Microsoft calls its virtualization product Hyper-V. It is included as role in Windows Server 2008 (R2). There is also a free stand-alone version of Hyper-V, a Windows Server 2008 core with full Hyper-V functionality, but other Windows 2008 Server roles disabled. Microsoft supports high availability for VMs and applications via Microsoft Cluster Server (MSCS) resp. Microsoft Failover Clustering[2]. Fault tolerance is not supported (yet). In January 2009, Microsoft and Marathon Technologies announced an "expanded development and marketing agreement" [3]. This announcement was recently updated and clarified. The project now aims to provide fault tolerance for Windows Server 2003/2008 and fault tolerant virtual infrastructures in a future version of Hyper-V.

In section 4, we will introduce Kemari, another virtual machine synchronization mechanism currently implemented for Xen and KVM. The Kemari project is still in research and development and not a commercial product. That's why it is not mentioned in the previous table.

2 VMware vSphere details

2.1 VMware Virtual Infrastructure

Figure 1 shows the conceptual architecture of the VMware virtual infrastructure.

VMware ESX Server constitutes the foundation of the virtual environment. It is directly installed onto the physical machines without an operating system underneath. It is a so-called "bare-metal" hypervisor architecture and provides the virtualization layer which abstracts processor, memory, storage, and networking resources.

The second building block of the VMware Infrastructure is the VirtualCenter (vCenter) Server. This is a service installed in a native or virtualized Windows Server with network connection to the ESX hosts. It is used to manage all ESX hosts and virtual machines. Therefore it provides services such as:

– VM provisioning

[2] new brand since Windows Server 2008

[3] `http://www.marathontechnologies.com/microsoft_and_marathon_alliance.html`

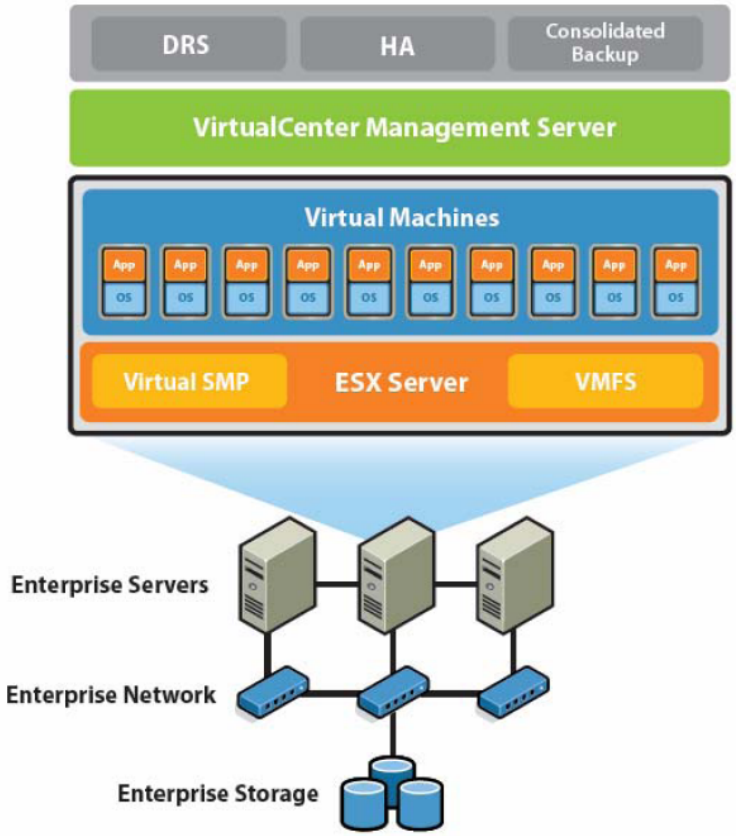

Fig. 1. Architecture of VMware Infrastructure

- Performance monitoring
- Automation
- Access control
- (Live) Migration of virtual machines (VMotion)

On top of that, it provides *cluster* services, such as:

- Distributed Resource Scheduler (DRS)
- High Availability (HA)
- Consolidated Backup
- Fault Tolerance (FT)[4]

A cluster is a concept which permits the combination of resources of multiple ESX hosts into one single manageable entity (compare figure 2). Virtual machines are then assigned to clusters instead of to hosts. The VM placement is

[4] Not shown in figure 1

done automatically by the cluster. DRS even allows to reallocate running VMs via VMotion to optimize resource utilization on the hosts. This concept of a cluster is the foundation for system level reliability features in VMware.

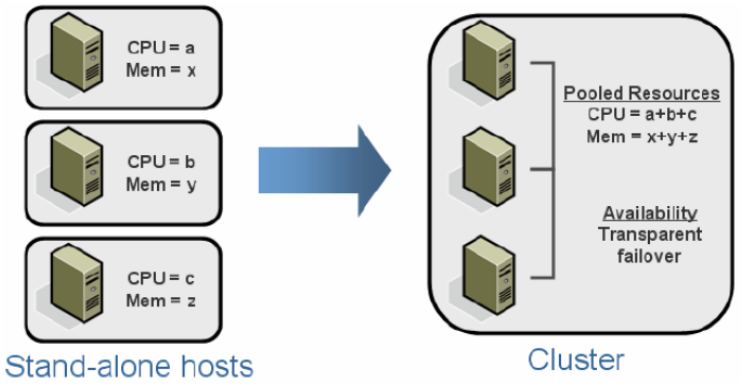

Fig. 2. Resource aggregation in VMware clusters

Access to the vCenter Server is provided either by the vSphere Client, a .NET based client for windows, or optionally via a web interface.

2.2 VMware Reliability Features Overview

Component-level features The VMware infrastructure offers two reliability features at component level.

The first one, *NIC teaming*, applies to network connectivity. NIC teaming allows to connect multiple physical Ethernet adapters to a single virtual switch (vSwitch). Such a team of NICs permits load balancing between physical and virtual networks and provides failover capabilities. Broken network connections are detected by checking the link status provided by the network adapters and optionally by so-called beacon probing. Beacon probing can detect more failures, such as switch ports being blocked by spanning tree, misconfigured VLAN or broken links on the other side of a physical switch. If a failure of a NIC is detected, network traffic is routed via another properly working NIC by the vSwitch.

The second feature is called *storage multipathing* and applies to storage link failures in SAN networks. Multipathing permits to configure multiple physical paths which transfer data between ESX hosts and storage devices. In case of a failure of any element in the SAN network, e.g. an adapter, switch, or cable, the host can switch to another physical path which does not use the failed component. In addition, multipathing can also be used for load balancing.

System-level features At the system level, there are the two already mentioned features VMware High Availability and VMware Fault Tolerance. These will be described in detail in the next two sections.

2.3 VMware High Availability

VMware HA is an implementation of the high availability feature introduced in section 1.2. Its goal is therefore to restart VMs located on hosts which appear to have failed. It requires a VMware Cluster setting. All disk images of the VMs in the cluster have to be located on shared storage. Shared storage can be implemented using one of the following options: fibre channel, iSCSI or NAS.

VMware HA works by continuously monitoring all ESX Server hosts in a cluster to detect failures. This process in shown in figure 3.

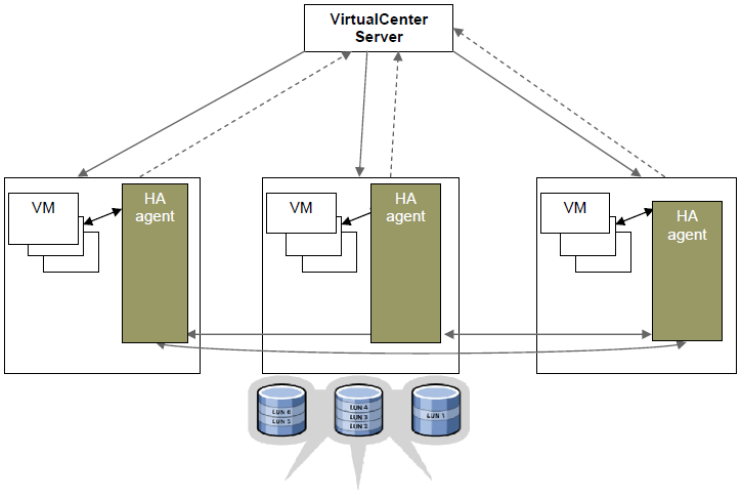

Fig. 3. Mutual monitoring of hosts via HA agents in VMware HA

The *HA agents* shown in the figure are installed and started on all hosts in the cluster, after HA is enabled for the cluster. They mutually monitor all other hosts in the cluster. To accomplish this they exchange heartbeat signals at a configured interval, i.e. every five seconds in the default configuration. A failover is initiated on loss of three consecutive heartbeats. The VMs previously running on the failed host are then distributed and restarted on other host in the cluster with sufficient available resources. This failover is fully automatic.

To be sure, that there are enough resources available in the cluster to restart all VMs in case of possibly multiple host failures, a so-called *failover capacity* can be configured. This number c is equal to the maximum number of host failures

that the cluster should be able tolerate. The vCenter Server automatically checks resource reservations of the VMs and assures, that no more VMs can be started in the cluster, if this would consume too much resources to tolerate a failure of the c hosts providing the most resources to the cluster (worst-case scenario).

It should be noted, that the HA agents act autonomous. The vCenter Server is **not** required for monitoring or failover. Hence, the vCenter Server does not constitute a single point of failure. However, the decision on which remaining hosts the VMs are restarted depends on the available extra resources on the hosts. This information is maintained by the vCenter Server. If it goes down, the failover strategy is based on the state of the cluster before the server went down.

Obviously, any host requires access to the shared storage, where the VM disk image is located, in order to run the VM. So this is a critical factor. If this access is lost, VMs cannot keep on running any more. Reliability of the shared storage should therefore be improved at component level, e.g. using RAID. The availability of the link to the shared storage can be improved using storage multipathing as described in section 2.2.

To summarize, we observe that HA can perceptibly increase the availability of services running in virtual machines. It is targeted to environments tolerating a brief interruption of services. This interruption has a magnitude of a few minutes. This is the time VMware HA needs to detect the failure and restart the VM(s). It is important to note that a restart is comparable with a hard reset of the VM. The running state of the VM is lost on failure. Applications running in the VM need to tolerate this. In addition, the VM/guest OS needs to be properly configured to autostart all required services on system reboot.

There is an additional feature called *VM Monitoring*. This is similar to HA for hosts, but monitors the VM guests. It needs VMware tools to be installed on the guest OS. Then it constantly exchanges heartbeats with the tools to recognize a failure of the guest OS, such as a Windows Bluescreen or Linux kernel panic. In that case, the VM is restarted.

2.4 VMware Fault Tolerance

VMware Fault Tolerance is a new feature in the recent version of the VMware infrastructure, VMware vSphere 4. It is an implementation of the fault tolerance feature introduced in section 1.2. It it based upon VMware HA, i.e. it requires a HA-enabled cluster. But in contrast to HA, it relies on redundancy in space. It uses a second VM running synchronized with the primary VM to prevent downtime and data loss even in the presence of ESX host failures. The synchronization is achieved by a feature called vLockstep, which will be described in the next paragraph.

vLockstep FT requires that primary and secondary VM are in identical state at each guest instruction boundary. vLockstep was developed to guarantee this

requirement. It is accomplished by having both VMs execute identical sequences of x86 processor instructions. Processors connected in this way are said to be running in lockstep.

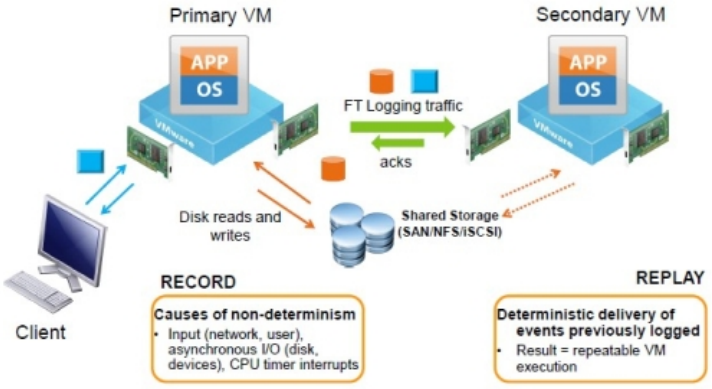

Fig. 4. vLockstep architecture

The challenge in getting two processors on different machines executing identical processor instructions is the non-determinism introduced within the processor and by external devices. All kinds of non-deterministic events need to be captured and replayed on the secondary VM at exactly the same position as they occurred on the primary VM. This technology is also known from the Record/Replay feature of VMware Workstation 6 and newer. The difference between Record/Replay and FT is, that Record/Replay logs events to a file and replays it later on demand, whereas FT transmits logging data over a dedicated logging network to the secondary VM and replays it right away. This process is shown in figure 4. The figure also states causes of non-determinism. These include timer events, user input, disk or network I/O and all other sources of processor interrupts.

Since primary and secondary VM execute exactly the same processor instructions, both initiate I/O operations. Of course, only the operations of the primary VM must take effect, e.g. network packets should not be sent more than once. This is implemented by the hypervisor of the secondary VM, which silently suppresses all output of the secondary VM. This can easily be achieved, because the guest never accesses hardware directly, but uses virtual devices supplied by the hypervisor. Due to the output suppression, the secondary VM is completely invisible to the external world, which always perceives the fault tolerant VM as one single unit.

Failover To detect if any of the hosts executing a protected VM has failed, the hypervisors establish a system of heartbeat signals and mutual monitoring

similar to HA. Figure 5 shows the failover in the case that the host with the primary VM fails.

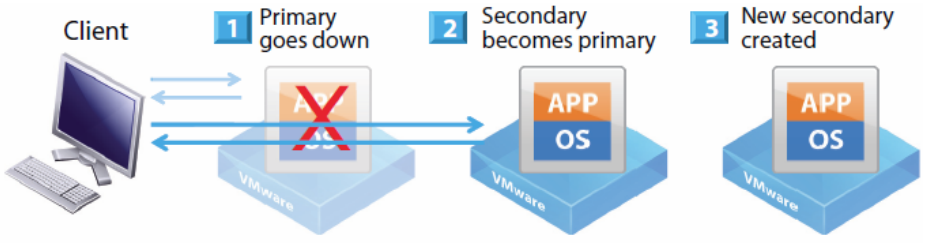

Fig. 5. Failover after host failure of the primary VM

As a first step, the hypervisor running the secondary VM recognizes the failure. It then disengages vLockstep and commits all pending I/O operations from the failed VM. Then it performs a "go live" operation which removes all dependencies from the failed host. Afterwards it will, for example, start to read network traffic designated to the fault tolerant VM directly from a physical NIC and commit disk writes. With this step, the secondary VM has become primary. The next step is to restore fault tolerance by choosing a new host for a new secondary VM, copy the primary VM to that host and re-engage vLockstep. The host selection is done automatically by HA and does not require communication with the vCenter Server, i.e. as for HA, it does not constitute a single point of failure.

If there is some kind of transient failure of the host executing the primary VM, it could happen that this host recovers after a failure was detected and the secondary took over. This could lead to a "split brain" situation, i.e. there are two active (primary) copies of the VM running. VMware FT prevents this by using a file locking mechanism on the shared storage. This guarantees that only one host can execute the VM at a time.

Requirements Lockstep synchronization of processors is a very difficult and sophisticated technology. Hence, a whole bunch of requirements need to be satisfied on hardware, ESX host and VM level to be able to use fault tolerance.

First of all, the hosts need a vLockstep capable processor. For this reason, VMware worked together with Intel and AMD and achieved some changes in the performance counter architecture and virtualization support processor features. These changes have been made to recent processor families from both vendors. Supported processors include AMD's 3rd generation Opteron based on the Barcelona, Budapest and Shanghai processor families and Intel Xeon processors based on the Penryn and Nehalem microarchitectures and their successors. Furthermore, lockstep is only possible between hosts having the same processor

family installed. The processors do not need to have exactly the same speed, a deviation of up to 400 MHz is tolerated.

In addition, a Gigabit-NIC is required for the FT logging network. VMware recommends to use separate NICs for VMotion, FT logging, service console and VM traffic. Therefore, a total number of at least four network cards is recommended. Further NICs could be used for NIC teaming to improve network reliability.

As mentioned before, FT requires a HA-enabled cluster with shared storage.

In vSphere 4.0, all hosts must be running exactly the same build of VMware ESX, too. This requirement has been relaxed in the most recent vSphere version 4.1.

On VM level, the most severe restriction is the limitation to one single virtual processor per VM, i.e. vSMP cannot be used for fault tolerant VMs. VMware may support vSMP in a future release, but as mentioned before, lockstep is hard to implement and multiple processors make it even harder.

Another requirement on VM level is to use "thick" disks, i.e. the virtual disks must be fully allocated on physical disks. There is also a class of devices, which VMware calls "non-repeatable", that cannot be attached to a fault tolerant VM. This class includes physical floppy and CD-ROM drives, sound and USB devices.

Besides, all guests need to be fully virtualized, i.e. so-called paravirtualization has to be turned off. Paravirtualization is an approach that does not completely hide the virtualization from the guest OS. Paravirtualized OSes interact with the underlying hardware via a special interface provided by the hypervisor in order to achieve less overhead and faster processing. This direct hardware access is problematic for FT, because it prohibits the record and replay functionality and suppression of outputs on the secondary VM.

Summary VMware fault tolerance provides a VM protection which ensures no downtime and no data loss in case of a ESX host failure. It is therefore useful for mission-critical applications that cannot tolerate disruptions or loss of transactional data. In contrast to many other clustering and fault tolerance techniques, it is fully transparent to the guest OS and the applications. Failure detection and failover is fully automatic.

On the other hand, VMware FT has lots of requirements and limitations. It can therefore not be used in many situations, e.g. VMs requiring more than one virtual CPU. In addition, it can only react on host failures. Serious events inside the VM, such as a Windows Bluescreen, happen on both, primary and secondary VM.

VMware FT certainly has a performance impact. The limiting factor is the FT logging bandwidth. VMware states the following formula to determine the necessary bandwidth:

$$bandwidth \approx \left(AvgDiskReads \left[\frac{MB}{s} \right] \cdot 8 + AvgNetworkInput \left[\frac{Mbit}{s} \right] \right) \cdot 1.2$$

A practical performance test showing the performance impact of VMware FT will be presented in section 3.2.

3 Experiments

To evaluate the characteristics of VMware vSphere, we performed two tests. In the first one, we tried to capture the performance impact of VMware HA and especially VMware FT. In the second test, we investigated the effectivity of the reliability features in terms of service interruption for the user in an audio streaming scenario.

3.1 Test Environment

We build our vSphere 4.0 environment on hardware provided by the *HPI Future SOC Lab*. The test system included two ESX 4.0 hosts on Fujitsu Primergy RX300 S5 servers with each a Intel Xeon E5540 4-core processor, 12 GB memory and 2 gigabit network interfaces. These two hosts formed a cluster as described in section 2.1, where we assigned the test VMs to. The vCenter Server was installed in a virtual machine on dedicated hardware, so it doesn't interfere with our tests.

3.2 Performance

To evaluate the performance impact, a simple test was performed with different reliability configurations. The run time of the Linux Kernel compilation was measured with the assumption that this task includes a fair amount of computation and memory consumption, as well as I/O reads and writes. The tests were performed on a single-core VM with a 64-bit Debian 5.0.4 installation. The results are shown in Table 2.

Configuration	Average time in seconds
no HA/FT	1639.03
HA only	1637.94
HA and FT	1963.37

Table 2. Linux Kernel compilation with different reliability configurations

The times measured with HA enabled were almost exactly the same as those without any reliability features. This shows that the run time overhead, which is introduced with the HA agent and heartbeat exchange, can be neglected. On the other side, the time necessary for compilation increased by approximately 20 percent with FT enabled. Regarding the load situation, this is also a presentable value.

3.3 Audio Stream Continuity

Setup For the test of the effectivity of VMware HA and Vmware FT, we set up a client-server application scenario. Thereby the server was located in the cluster of our vSphere installation, so it could be protected by the VMware features introduced before.

Rather than developing a custom application, an existing one based on the popular Hypertext Transfer Protocol (HTTP)[1] was chosen. HTTP works with a request-response scheme, but it is also possible to use it as transport layer for audio and video streams. This was done to analyze the continuous availability of the system since, in contrast to static websites, even short down-times of the audio stream are immediately recognizable by the user.

The unit under test was a virtual machine with one processor, 256 MB memory and *CentOS 5.5*[2] as guest operating system. *Icecast*[3] was used to stream audio over HTTP while the audio input of Icecast was delivered by the *Music Player Daemon*[4]. Icecast is a free multimedia streaming server. MPD is a music player server, i.e. it can play music from a playlist, but it does not include a user interface. The control MPD, e.g. to manipulate playback or the playlist, a client program is necessary.

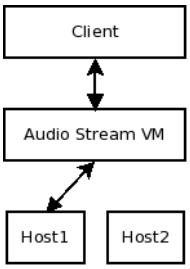

Fig. 6. Test Scenario Setup

Testing Procedure We tested three different configurations: no protection of the server, protection by VMware HA and protection by VMware FT (including VMware HA). For all these configurations, we adhered to the following testing procedure:

- Start of the VM with the audio streaming server. By configuration, icecast and mpd start automatically.
- Connect to the server with a MPD client and initiate the playback of the audio stream.
- Listen to the audio stream with a compatible player, such as Amarok or Winamp.
- Simulate a host failure and record, what happens.

– Recover the failed host and record, what happens.

Only host failures are considered in this scenario, because the VMware features to be inspected can only protect from such kind of failure. There are multiple possible reasons for those failures like software faults, power failures, lost network connectivity and so on. In fact, the exact reason for a host failure is irrelevant, so it is not necessary to investigate them all. To simulate a host failure in our scenario, we simply disabled the corresponding ports on the network switch. This renders the host and all its virtual machines unresponsive or unreachable from the clients and other hosts. At the same time, it permits to recover the host easily by re-enabling the ports.

Results: No Protection This test was done to serve as reference for the other tests. There will be no automatic recovery in this case, of course.

After the host running the VM has been disconnected from the network, the music stopped playing with a small delay. The MPD client also lost connection and tried to reconnect, but always encountered a timeout. The vSphere Client showed an alarm about "Host Connection and Power State" for the affected ESX host. After the re-establishment of the network connection on the switch, the MPD client reconnected automatically and showed, that mpd was still playing. This is comprehensible, because the host was only disconnected from the network, the VM could keep on running on the host. However, Winamp only resumed playing after pressing the stop and the play button.

Results: VMware HA After the simulated host failure, the behavior was first equal to the situation with no protection: Winamp stopped playing; the MPD client lost connection, tried to reconnect, but experienced timeouts; the "Host Connection and Power State" alarm appeared in the vSphere Client. But after about 90 seconds, the MPD client was able to reconnect to the streaming server on the virtual machine. However, as described in section 1.2, the server was rebooted and the state was lost. Thus, the server did not play and the playback needed to be re-initiated manually with the MPD client. Again, Winamp resumed playing only after pressing the stop and the play button. After the re-establishment of the network connection, the failed ESX host became available again in the vSphere Client. Nothing else happened, i.e. the remainder of the VM on the failed host did not interfere with the new VM and was silently removed. Altogether, the error recovery worked as expected.

Results: VMware HA & FT At first, we disconnected the host running the secondary VM. As expected, this had no effect on the streaming service or the clients. In the vSphere Client, a "Virtual Machine Fault Tolerance State Changed" alarm appeared and the VM was claimed to be not protected any more. Actually, HA should find another host and create a new secondary VM on this host in this situation. But in our scenario, we only had two ESX hosts, so there is no other host for this purpose, when one is disconnected. That's why

the unprotected state could not change until the failed host recovered. After re-connecting the failed host, it took about 36 seconds until the VM was FT-protected by the secondary VM again.

The last experiment was done by disabling the network connectivity to the host running the primary VM. After the simulated host failure, both, the music player and the MPD client experienced a short interruption of about one second. Then both continued without manual intervention. In the vSphere Client, the "Virtual Machine Fault Tolerance State Changed" alarm appeared and the VM was "not protected" any more, as in the previous case. After re-establishment of the network connection, the unprotected state warning disappeared after about 36 seconds, again.

So in general, the last test worked fine, too. However, we were a little bit astonished about the 1 second break, because VMware actually promised no service interruption at all. For that reason, we repeated the last test and this time, investigated the network traffic between the streaming server and MPD client with Wireshark[5]. The resulting network traffic is shown in Figure 7.

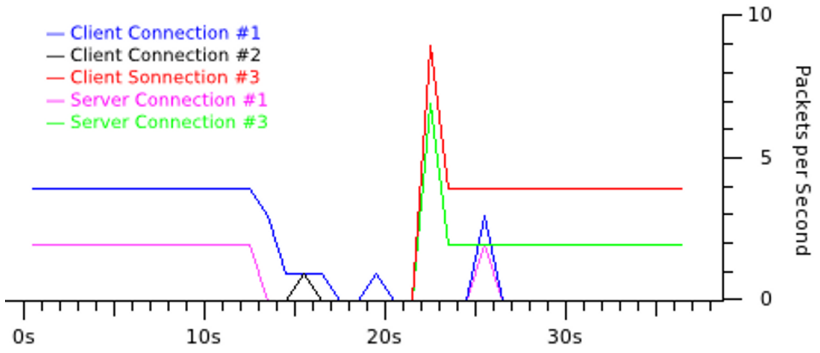

Fig. 7. Network Traffic Graph with Failover

MPD client and server communicate via a TCP connection. During normal operations, MPD sent a status request every 0.5 seconds, received a status information from the server and acknowledged the successful reception. This is why during the first 13 seconds, four IP packets were sent per second from the client and two from the server. Then the primary host of the virtual machine was disconnected and there were no more replies from the server. The client kept sending requests for retransmission, received timeouts and then tried to open another connection after overall 16 seconds, about three seconds after connection loss. It is important to notice that this second client connection was never established and even a client with a long timeout period would experience a service failure. After overall 22 seconds, the client tried to open a new TCP connection

[5] `http://www.wireshark.org/`

again and immediately received a reply from the server. This means that the virtual machine was running again and was able to handle requests. After this third connection was established, the MPD client used the new connection to communicate with the server in the same manner as before the host failure. The state of the server, i.e. the playlist, was not lost, as expected. Another interesting point is the network traffic at about 26 seconds. There was a TCP retransmission of the last status request from the client from approximately 11 seconds ago. The server now replied, however the reply had no value for the client as another connection had already been successfully established and therefore the client sent a TCP reset command that the server acknowledged with the effect that both sides gracefully shutdown the first connection and continued to use the third connection. If the timeout of the MPD client was much longer, then it could have used the first connection after the failover and would just experience an unexpected network delay, but no service failure. However, in this case the failover was indeed affecting the user, even if this was only recognizable for one second.

Conclusions VMware HA worked completely as expected. The host failure was recognized immediately and the VM was restarted on the other host. The service was available again after only one and a half minute in our scenario without any manual intervention. But in our scenario, the streaming did not start automatically and we had to do this by hand. This should be considered a configuration fault and cannot be charged to VMware. In a real-world application, the VMs protected by HA should be configured in a way, such that they resume the complete service automatically after reboot.

VMware FT did not work exactly as promised in the product description of VMware vSphere in our scenario. In fact, the clients experienced a short interruption of the service. Further investigations suggested, that the VM was not responsive for about nine seconds in our setting. Since we have only done this single test with audio streaming, we cannot conclude, that the failover of VMware FT always causes a short service interruption. In other scenarios without soft realtime requirements, such as ordinary web servers, application and database servers, VMware FT may keep its promise of uninterrupted service delivery, such that the host failure and the failover is not perceptible for the user. This would have to be further investigated. At least, the downtime with VMware FT was remarkably shorter and there was really no data loss compared to Vmware HA, even in our scenario.

4 Kemari

4.1 Introduction

Another approach to evaluate VMware FT is to look into related work and identify what parts of VMware FT are common to other virtualization fault tolerance solutions and what parts are unique and compare those differences. For this

comparison, we chose Kemari[5], because it pursues a different synchronization approach.

Kemari is an open source project that was started 2006. As the name suggests, its origin lies in Japan. Today, the project is funded by the Nippon Telegraph and Telephone Corporation (NTT) but adheres to an open development process that includes participation from outside by means of patches and community support. It is still in early stage of implementation and currently mainly for research purposes. Despite its very narrow and specialized use case there is still active development as of June 16, 2010.

4.2 Implementation

Virtualization Kemari was initially targeted at Xen[6] but rewritten from scratch in 2009 to support KVM[7] due to the rapidly growing KVM community and a declining interest in Xen[6]. In 2010, one of the first prototypes was released demonstrating the feasibility of the task while still being far from optimal in terms of performance.[7]

KVM consists of kernel modules as well as user-space libraries that are mostly inherited from Qemu. The first versions of Kemari for KVM focused on the user-space part and can still be used with a standard Linux kernel[8]. However, the following versions will add some performance optimizations implemented in the kernel.

Synchronization Similar to VMware FT, Kemari also uses an active-passive configuration where the state of the virtual machine is continuously replicated from the primary host to the secondary host. This enables hot standby, i.e. a failover does not result in a temporary service failure. In contrast to VMware FT, the secondary virtual machine in Kemari does not actually execute code. Rather than keeping two virtual processors in lockstep, a "live snapshot" or "continuous checkpointing" approach was implemented that transmits the state of the primary virtual machine to the secondary on each synchronization point. Of course, it is only necessary to send the differences to the previous synchronization point. This is done by transmitting the dirty pages[9] at each outgoing I/O operation[10]. It would not be safe for the primary virtual machine to execute code, especially disk writes, while the secondary virtual machine is still being synchronized. This

[6] See section 1.3 for details on Xen.

[7] Kernel-based Virtual Machine - the virtualization infrastructure of the Linux kernel

[8] Running completely with user-space emulation makes it relatively easy to alter the virtual machine behavior on a low level, e.g. it is possible to emulate hardware failures by inserting Machine Check Exceptions (MCEs) into the virtual machine.

[9] Memory pages that have been altered by the processor, but that are not yet written to the I/O device.

[10] On a side note, Kemari is able to use any kind of socket for transmitting memory, e.g. local pipes, TCP connections in the LAN or even encrypted SSH connections to hosts on the other side of the world.

is why the primary virtual machine is stopped until the passive virtual machine acknowledges a successful synchronization. Every code execution between I/O writes is assumed to be repeatable and therefore a failover will succeed even if it takes a considerable amount of time for the secondary virtual machine to catch up with the previous state of the erroneous primary virtual machine.

Failover detection Failover detection was successfully integrated with Heartbeat in the Xen version of Kemari. The next versions of Kemari for KVM will provide integration with other existing HA stacks like RHCS, Pacemaker and Corosync. It is also planned to detect hardware faults on new x86 hardware by reacting to MCEs[7].

4.3 Comparison with VMware FT

The objectives of Kemari and VMware FT are comparable, but the implementation approach is remarkably different as explained in section 4.2. A full evaluation of Kemari including performance and audio stream test in our test environment was out of the scope of this paper. Therefore we cannot present reliable numbers for a comparison with VMware FT. Instead we will try to analyze the consequences of the principle differences between both approaches, here.

At first, we notice that Kemari is a much simpler approach. It does not rely as heavily on the processor characteristics as VMware FT which results in Kemari being not very architecture-specific. In contrast to VMware FT, Kemari works with many different processor families, since it does not require customized microarchitectures. Since the secondary VM does not execute code, the host maintaining this VM does not waste as much CPU resources as with the VMware solution, too. But the most important advantage of Kemari is that it does support SMP in virtual machines. This is not possible with VMware FT, yet. With the limitation in processor clock rates and the recent trend to multi-core systems and parallel computing, this feature will become more and more important.

On the other hand, the checkpointing approach of Kemari raises doubts about its performance and applicability in real life. While the VMware FT performance impact is primarily defined by incoming and outgoing I/O, the number of dirty pages produced between synchronization points is the decisive factor for Kemari. Incoming and outgoing I/O depends on network bandwidth in most modern server landscapes, i.e. even storage is usually not located directly inside the server. Since FT logging also uses a network connection, the amount of incoming network traffic will probably be transferable over the logging network, too. So in that case, VMware FT does scale. On the other side, memory can be accessed and changed much faster and therefore there is a limit to the amount of written memory per second and the network connection has to be suited for the application. For example, if a specific application would produce 1000 megabytes of dirty pages in a second, an ordinary gigabit network connection would not be able to transmit this amount of data in time and the VM's performance would heavily decrease. Compared to the performance drop of 20 percent that

we measured with the Linux kernel compilation, which constitutes a high load situation for VMware FT, it shows that Kemari is heavily application dependent. Another problem also arises with Kemari in the case, that the amount of dirty pages becomes too big. Since we complained about the short unavailability of the VM during failover in our audio stream experiment with VMware FT, we should note, that this could happen even during normal operation with Kemari. This is due to the fact, that the primary VM is suspended during the synchronization phase. If the previous considerations were correct and such situations really occur, then enabling Kemari could render the protected VM unavailable from time to time.

To summarize, we observe, that Kemari has its advantages in its simplicity and, more important, the vSMP support. On the other side, the performance behavior with real workloads needs to be investigated, because the checkpointing approach raises doubts about its performance. VMware FT is a working solution, which showed good performance in our tests. Of course, VMware is years ahead in experience and engineering of virtualization products. But it might still be worthwhile to look at different solutions such as Kemari because there might be some special applications and use cases for which other FT approaches are also considerable.

References

1. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard) (June 1999) Updated by RFC 2817.
2. The CentOS Project, Red Hat, Inc: CentOS - The Community ENTerprise Operating System. http://centos.org/
3. Xiph.Org Foundation: Icecast. http://www.icecast.org
4. Community, T.M.: Music player daemon. http://mpd.wikia.com/wiki/Music_Player_Daemon_Wiki
5. Nippon Telegraph and Telephone Corporation: Kemari project. http://www.osrg.net/kemari/ (2009)
6. Cao, F.L.V.: KVM Fault Tolerance: Kemari for KVM. http://www.mail-archive.com/kvm@vger.kernel.org/msg25022.html (Nov 2009)
7. Tamura, Y.: Kemari for KVM: updates. http://www.archivum.info/qemu-devel@nongnu.org/2010-02/00234/(Qemu-devel)-(RFC)-Kemari-for-KVM-updates.html (Feb 2010)
8. VMware Inc.: VMware HA: Concepts and Best Practices. Technical report, VMware Inc. (January 2008) http://www.vmware.com/files/pdf/VMwareHA_twp.pdf.
9. VMware Inc.: Protecting Mission-Critical Workloads with VMware Fault Tolerance. Technical report, VMware Inc. (February 2009) http://www.vmware.com/files/pdf/resources/ft_virtualization_wp.pdf.
10. VMware Inc.: VMware Fault Tolerance Recommendations and Considerations on VMware vSphere 4. Technical report, VMware Inc. (July 2009) http://www.vmware.com/files/pdf/fault_tolerance_recommendations_considerations_on_vmw_vsphere4.pdf.

11. VMware Inc.: VMware Communities Roundtable #53 - VMware Fault Tolerance. http://recordings.talkshoe.com/TC-19367/TS-238167.mp3 (June 2009)
12. Tamura, Y.: KVM Fault Tolerance: Kemari for KVM Benchmark. http://www.mail-archive.com/kvm@vger.kernel.org/msg25328.html (Nov 2009)