OpenSAF and VMware from the Perspective of High Availability

Ali Nikzad

A Thesis

in

The Department of Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Software Engineering) at

Concordia University

Montreal, Quebec, Canada

November 2013

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: Ali Nikzad

Entitled: OpenSAF and VMware from the Perspective of High Availability

and submitted in partial fulfillment of the requirements for the degree of

_____

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. T. Popa _____ Chair

Dr. J. Rilling _____ Examiner

Dr. J. Paquet _____ Examiner

Dr. F. Khendek _____ Supervisor

Dr. M. Toeroe _____ Co-Supervisor

Approved by Dr. S. P. Mudur _____
Chair of Department

Dr. C. Trueman _____
Dean of Faculty

Date: November 2013

Abstract

OpenSAF and VMware from the Perspective of High Availability

Ali Nikzad

Cloud is becoming one of the most popular means of delivering computational services to users who demand services with higher availability. Virtualization is one of the key enablers of the cloud infrastructure. Availability of the virtual machines along with the availability of the hosted software components are the fundamental ingredients for achieving highly available services in the cloud. There are some availability solutions developed by virtualization vendors like VMware HA and VMware FT. At the same time the SAForum specifications and OpenSAF as a compliant implementation offer a standard based open solution for service high availability. Our work aims at comparing virtualization solutions, VMware, with OpenSAF from the high availability perspective, and proposes appropriate combinations to take advantage of the strengths of both solutions. To conduct our evaluations, we established metrics, selected a video streaming application and conducted experiments on different architectures covering OpenSAF in physical and virtual machines, the VMware HA and VMware FT. Based on the analysis of the initial measurements, we proposed other architectures that combine OpenSAF high availability and the virtualization provided by VMware. Our proposal included architectures targeting two types of hypervisors, non-bare-metal and bare-metal. In both of these proposed architectures we used OpenSAF to manage the availability of the VM

and the case study application running in the VM. The management of the availability of the VM is slightly different in these architectures because of the types of the hypervisors. In these architectures we used the libraries and the mechanisms which are available in many other hypervisors. Our work compared to other works on high availability in virtual environments has the important advantage of covering the application/service failure.

Acknowledgement

I would like to express the deepest appreciation to my supervisors Dr. Ferhat Khendek and Dr. Maria Toeroe who guided me through this work with their tireless dedication. This thesis would never have been possible without their immense knowledge and constructive comments.

My deepest heartfelt gratitude goes to my beautiful wife Sadaf, who has always been patient and supportive. I owe a very important debt to my parents Seddigheh and MohammadAli for their support, encouragement and a lot more which I would never be able to pay back. I would also like to particularly thank my sister Mahsa for her endless kindness.

I would like to offer my special thanks to my colleagues in the MAGIC lab for their pure friendship and creating a warm and pleasant environment.

This work has been partially supported by Natural Sciences and Engineering Research Council of Canada (NSERC) and Ericsson Research.

Table of Contents

List of Figures x

List of Tables xii

List of Equations xiii

List of Acronyms xiv

1. Introduction 1

1.1 Thesis motivations and contributions 3

1.2 Thesis organization 5

2. Background on AMF, OpenSAF and VMware 6

2.1 SAForum and high availability 6

2.1.1 Availability Management Framework (AMF) 7

2.1.2 OpenSAF 14

2.2 Virtualization and VMware 16

2.2.1 Virtualization 16

2.2.2 Cloud computing 18

2.2.3 VMware virtualization solutions 20

2.2.4 VMware solutions for availability 21

2.3 Related work 27

2.3.1 A Case for High Availability in a Virtualized Environment (HAVEN) [25] 28
2.3.2 Remus: High Availability via Asynchronous Virtual Machine Replication [27] 30
2.3.3 Other related works 31
2.3.4 Virtualization solutions supporting availability 32
3. The baseline architectures for availability, metrics and measurements 34
3.1 Hardware test-bed 34
3.2 Case study application 35
3.3 Deployed architectures 37
3.3.1 Architecture 1: OpenSAF on the physical nodes 37
3.3.2 Architecture 2: Availability using VMware HA 38
3.3.3 Architecture 3: Availability using VMware FT 38
3.3.4 Architecture 4: OpenSAF deployed in VMs 39
3.3.5 VMware HA settings on the cluster 40
3.4 Qualitative criteria 41
3.4.1 Complexity of using the high availability solution 41
3.4.2 Redundancy Models 42
3.4.3 Scope of failure 42
3.4.4 Service continuity 43
3.4.5 Supported operating systems: 44
3.4.6 Summary 44

3.5 Metrics 44
3.5.1 Reaction Time 45
3.5.2 Repair Time 45
3.5.3 Recovery time 45
3.5.4 Outage time 46
3.5.5 Memory consumption 46
3.6 Types of failures 47
3.7 Measurements in OpenSAF related architectures 48
3.8 Measurements in VMware based architectures 50
3.9 Measurements from our experiments 54
3.9.1 Component failure measurements 55
3.9.2 Virtual machine failure measurements 56
3.9.3 Physical node failure measurements 58
3.9.4 Memory overheads 60
3.10 Analysis 61
3.10.1 SA-Aware vs. Non-SA-Aware component 61
3.10.2 Overhead due to the VM 62
3.10.3 OpenSAF vs. VMware HA 62
3.10.4 Repair of the failed element 63
3.10.5 Combining VMware FT and OpenSAF 64
3.10.6 Fault propagation in VMware FT 65
3.10.7 Validity of the measurements 66

3.10.8 Conclusion 67
4. New architectures combining OpenSAF and virtualization 70
4.1 VM availability management in non-bare-metal hypervisor 71
4.1.1 Experiments with VM availability management in non-bare-metal hypervisor 73
4.2 VM availability management in bare-metal hypervisor 76
4.3 Conclusion 82
5. Conclusion 84
5.1 Future works 86
6. References 88

List of Figures

Figure 1 - Overview of AIS and HPI Services architecture [3] 7

Figure 2 - Defined redundancy models in AMF 12

Figure 3 - OpenSAF 4.0 architecture [5] 15

Figure 4 - Types of hypervisors 18

Figure 5 - Cloud service models 19

Figure 6 - VMware products [20] 21

Figure 7 - VMware HA 23

Figure 8 - VMware FT logging channel 26

Figure 9 - Synchronization in VMware FT 26

Figure 10 - VM life cycles defined in [25] 29

Figure 11 - Speculative execution and asynchronous replication in Remus[27] 30

Figure 12 - Remus high-level architecture [27] 31

Figure 13 - Availability management in Windows Azure 33

Figure 14 - The Magic Cluster (Ericsson's cluster in Concordia University) 35

Figure 15 - OpenSAF on the physical nodes 37

Figure 16 - Virtual machine running the original VLC managed by VMware HA 38

Figure 17- Virtual machine running the original VLC managed by VMware FT 39

Figure 18 - OpenSAF deployed in VMs 40

Figure 19 - Time related metrics 46

Figure 20- The sequence of actions done during the failure of a SA-Aware component. 49

Figure 21 - Sequence of actions after failure of a slave node in VMware HA 51

Figure 22 - Sequence of actions after failure of a Master node in VMware HA 51

Figure 23 - The sequence of actions during a node failure in VMware HA 52

Figure 24 - The sequence of actions during a VM failure in VMware HA 52

Figure 25 - The Sequence of actions after the failure of the primary VM in VMware FT53

Figure 26 - Outage due to VLC component failure 56

Figure 27 - Outage due to Virtual Machine failure..... 57

Figure 28 - Outage due to node failure 59

Figure 29 - Outage due to node failure without considering VMware HA 59

Figure 30 - Memory overhead (VMware FT is left out)..... 60

Figure 31 - VM availability management in non-bare-metal hypervisor 72

Figure 32 - VM availability management in bare-metal hypervisor..... 77

Figure 33 - VM failure in the proposed bare-metal architecture 79

Figure 34 - Sequence of actions for VM failure in the bare-metal based architecture 80

Figure 35 - Node failure in the proposed bare-metal architecture 80

Figure 36 - Sequence of actions for physical node failure in the bare-metal based architecture..... 81

List of Tables

Table 1 - Summary of the evaluation of the architectures with respect to qualitative criteria 44

Table 2 - Baseline architectures and applicable failure experiments 48

Table 3 - VLC component failure measurements 55

Table 4 - Virtual Machine Failure measurements 57

Table 5 - Node failure measurements 58

Table 6 - Repair of the failed element in different deployments 63

Table 7 - Comparison of measurements for VM failure in different architectures 75

Table 8 - Comparison of measurements for failure of the SA-Aware VLC component in different architectures 76

Table 9 - Timing expectations from our bare-metal hypervisor based proposal 82

List of Equations

Equation 1 - System Availability 1

List of Acronyms

AIS	Application Interface Specification
AMF	Availability Management Framework
API	Application Programming Interface
CLC	Component Life Cycle
CLI	Command Line Interface
CSI	Component Service Instance
FDM	Fault Domain Manager
FT	Fault Tolerance
HA	High Availability
IMM	Information Model Management
NFS	Network File System
Non-SA-Aware	Non Service Availability Aware
OMG	Object Management Group
OS	Operating System
SA-Aware	Service Availability Aware
SAForum	Service Availability Forum
SG	Service Group
SI	Service Instance
SU	Service Unit

TIPC	Transparent Inter-Process Communication
UDP	User Datagram Protocol
VLC	VideoLAN Client
VM	Virtual Machine
VMM	Virtual Machine Manager/Monitor

Chapter 1

Introduction

This chapter introduces topics discussed throughout this thesis, the motivations behind it and its contributions. We also introduce the organization of the thesis.

The world relies more and more on computers and the services that computer applications provide. This growth has led to higher demand and new requirements from the customers. They expect services to be available and in reach anytime, in other words, services that are highly available. Service or system availability is defined as a function of failure rate or Mean Time Between Failures (MTBF) and Mean Time To Repair [1] .

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

Equation 1 - System Availability

In many domains where downtime is catastrophic and not acceptable like air traffic control systems, life support, safety and security, service high availability is demanded. High availability is defined as at least 99.999% of availability [1]. This allows for a maximum of 5.26 minutes of downtime in a year. To tackle the problem a consortium of software and telecommunication companies, the Service Availability Forum (SAForum) [2] was created. This consortium has defined and standardized a set of middleware services.

AIS (Application Interface Specification) [3] is a set of middleware services defined by the SAForum to enable the development of highly available applications. The Availability Management Framework (AMF) [4] is one of the most important services defined in AIS. AMF manages the high availability of the services by managing the redundant software entities providing these services and shifting the workload from faulty entities to healthy ones at runtime. OpenSAF [5] is an open source SAForum compliant middleware implementation of some of the SAForum services like AMF, etc.

On the other hand, the computing world is moving toward cloud services and cloud computing, which mostly are based on virtual machines (VMs) and virtual resources. “The term virtualization broadly describes the separation of a resource or request for a service from the underlying physical delivery of that service” [6]. It is a mechanism for emulating hardware and software so that it would function exactly like a physical machine. The virtualization techniques can also be applied to other infrastructural layers like network, storage and memory.

Virtual machine is software implementation of a machine which is hosted by a software called hypervisor or Virtual Machine Manager/Monitor (VMM). The physical machine which hosts the hypervisor is the machine which provides the hardware resources to the VMs running in the hypervisor.

There are many advantages of using virtualization:

- Cost efficiency
- Testing and development purposes
- Increased resource utilization
- Green computing
- Increased scalability
- Portability of the VM images
- Legacy applications

1.1 Thesis motivations and contributions

Existing works related to availability in virtual environments suffer from the lack of the application level failure detection and recovery. These availability solutions in virtualized environments just address hardware and VM failures and cannot detect the application level failures. For example, consider a video streaming application running in a virtual machine that provides video service to clients. If such a video streaming application fails, current availability management solutions in virtualized environments will not detect the failure of this application and the provision of the video service to the clients will be stopped.

This lack of availability management described in the previous example provides the motivation for this thesis to introduce a solution supporting application failure detection and recovery within virtualized environments. As described earlier, SAForum specifications define a middleware for providing service high availability, and OpenSAF implements many of the services defined in SAForum specifications. VMware is one of the largest providers of virtualization environments supporting two different approaches for availability. One of our motivations in this research is to test and evaluate the qualitative and quantitative aspects of the existing availability solutions in VMware, and compare them to the comparable results from the OpenSAF based architectures. For this reason, first we evaluate OpenSAF high availability solution, VMware two availability solutions and a basic combination of both. For the purpose of our experiments we used VMware as an example of virtualization solutions and OpenSAF as a SAForum compliant middleware implementation. We compared the solutions qualitatively and defined some metrics so that we can have numeric analysis of different solutions. We set up experiments to evaluate the different solutions with respect to different types of failures. The experiments, measurements, comparisons and the analysis in the first step motivated us to propose new architectures supporting application level failures along with VM and physical node level failures in virtual environments.

We proposed two more architectures combining OpenSAF and virtualization for two different types of hypervisors. Thus, in these architectures we should benefit from both service high availability provided by OpenSAF and virtualization provided by VMware or other virtualization solutions. For measurement and comparison purposes, we

deployed one of the proposed architectures to evaluate the new method of handling the VM life cycle.

These architectures will certainly be useful in cloud environments for managing the availability of the VMs and the applications running within VMs. This work can lead to improve the availability management in virtual environments and cloud systems.

1.2 Thesis organization

The rest of the thesis is organized into four chapters. The second chapter is devoted to the background and the topics which make the building blocks of this manuscript. It will introduce AMF and OpenSAF as well as virtualization and the VMware solutions, specifically the availability related solutions. In chapter 3, we introduce and discuss our baseline architectures, test-bed, the case study and corresponding measurements. In this chapter we also analyze the measurements and consider the advantages and disadvantages of each solution. Chapter 4 elaborates our proposed architectures which are designed according the lessons learned from the analysis of the baseline architectures. These architectures benefit from the virtualization along with the high availability that OpenSAF provides. This chapter also discusses the deployment of one of the architectures with its measurements. In Chapter 5 we conclude our work and discuss some potential future investigations.

Chapter 2

Background on AMF, OpenSAF and VMware

In this chapter we will introduce the SAForum and its specifications, specifically AMF and one of its implementations, OpenSAF, which we have used in our deployments. We will also introduce VMware's solutions for virtualization. VMware has two approaches for providing availability in its VMware vSphere platform, VMware HA (High Availability) and VMware FT (Fault Tolerance) which we will also review in detail.

2.1 SAForum and high availability

There is a lot of work related to availability of systems. A summary of the concepts and definitions can be found in [7]. As mentioned before, SAForum is a consortium of telecommunication and computing companies developing standards for enabling highly available services and applications.

SAForum specifications are classified into two categories, Application Interface Specification (AIS) [4] and Hardware Platform Interface (HPI) [8]. AIS defines a set of

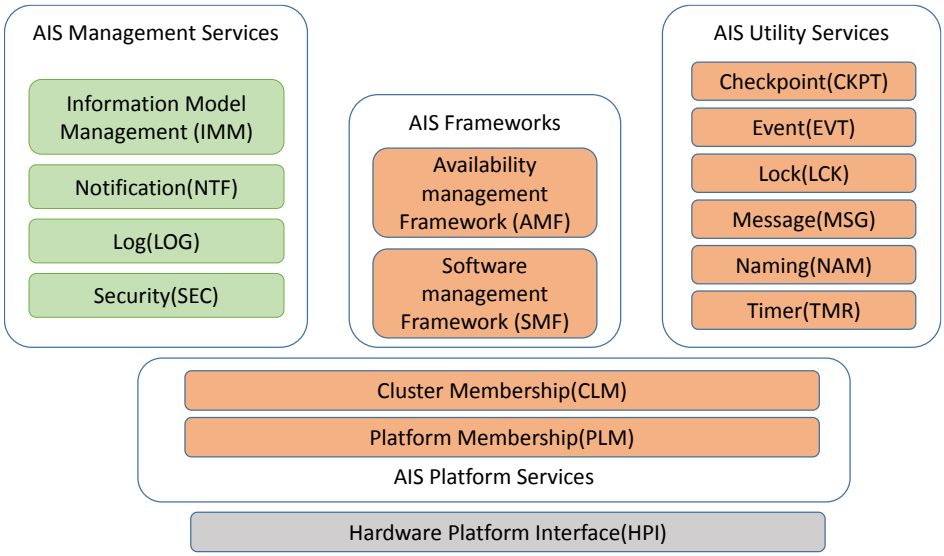

Figure 1 - Overview of AIS and HPI Services architecture [3]

interfaces that guide the application developers to create highly available software and services, where HPI provides application and middleware with accessing and managing hardware components via a standardized interface.

Figure 1 shows the services, frameworks and interfaces that the SAForum specifications offer.

In this thesis we will focus on the Availability Management Framework (AMF) since it is the AMF which enables the service availability within a cluster.

2.1.1 Availability Management Framework (AMF)

AMF plays the key role of keeping the services provided by an application highly available by managing and coordinating its redundant resources, and performing recovery/repair in case of a failure. For this purpose AMF detects the failure of application components and recovers their services according to the configuration

provided with the application. This configuration represents the application from the AMF perspective and describes the different entities composing it and their relations.

2.1.1.1 AMF Logical Entities

There is a set of logical entities defined in the AMF specification [4]. These logical entities are as follows:

- AMF Node

  The AMF node is a logical entity that represents a complete inventory of all AMF entities on a cluster node [4].

- Cluster Node

  A cluster node is the logical representation of all software running on a single operating system instance.

- AMF Cluster

  The set of AMF nodes in an AMF configuration is considered as the AMF cluster.

- Component

  The smallest entity which AMF can detect errors and carry out the appropriate recovery and repair actions. Components contain a set of software and/or hardware resources. There are two main component categories defined in AMF, Service Availability Aware (SA-Aware) and Non Service Availability Aware (Non-SA-Aware).

  - SA-Aware

    SA-Aware components are controlled directly by AMF and they are firmly integrated with the framework. The SA-Aware component registers itself

using the API function provided in AMF. The SA-Aware component also needs to implement the interfaces and callback functions which are the communications means between AMF and the components.

- Non-SA-Aware

  The Non-SA-Aware components are not directly registered with AMF. These types of components sometimes are indirectly linked to AMF through another component called a proxy component. By using the proxy, Non-SA-Aware component becomes a proxied component. The proxy itself is SA-Aware. When we do not have a proxy component to act as the mediator between the component and AMF, AMF is only able to control the life cycle of the component. The life cycle is controlled by the instantiation and termination of the Non-SA-Aware component.

As we mentioned before, AMF directly controls the life cycle of the non-proxied components through a set of command line interfaces. From the life cycle of the component's perspective there are two categories of components:

- pre-instantiable components:

  These types of components have the ability to stay idle after getting instantiated by AMF. This is for speeding up the recovery action so that the component would be ready to take over the responsibility of providing service whenever it is instructed by AMF. All of the SA-Aware components are pre-instantiable.

- non-pre-instantiable components:

These components provide the service as soon as they are instantiated. All non-proxied non-SA-Aware components are non-pre-instantiable. A proxied component can be pre-instantiable or non-pre-instantiable.

The life cycle of components which require environments different from the operating system accessible for AMF, cannot be controlled by AMF. The component inside the special environment is called contained and the environment itself is called container. In this case, the container acts as a mediator.

- Component Service Instance (CSI)

  CSI is the workload which AMF assigns to a component at runtime.

- Service Unit (SU)

  SU is composed of a set of components combining their individual functionalities to provide a higher level service. The SU is the smallest redundancy unit in AMF. SU can take the HA active, the HA standby or no HA state on behalf of a Service Instance (SI).

- Service Instance (SI)

  SI aggregates all CSIs which should be assigned to a SU in order to provide a higher level service.

- Service Group (SG)

  An SG is a group of one or more SUs protecting a set of SIs according to a redundancy model. All the components in the SUs should support the capabilities required by the redundancy model. AMF manages the service availability by assigning active and standby workloads to redundant SUs of each SG. The

redundancy models defined by the AMF specification differ on the number of SUs that can be active and standby for the SIs and on how these assignments are distributed among the SUs. There are five different redundancy models defined in AMF:

- **2N redundancy model**

  In the 2N redundancy model, there is at most one SU with active HA state and at most one SU with standby HA state. If there are other SUs present in the model, they will be considered as spares which have no SI assigned to them. For each SI, there is at most one active and at most one standby service unit (Figure 2.a).

- **N+M redundancy model**

  This redundancy model supports N service units with the active HA state and M service units with the standby HA state. The 2N redundancy model is a special type of N+M where N and M are 1 (1+1). For each SI, there will be at most one active service unit and at most one standby service unit (Figure 2.b).

- **N-Way redundancy model**

  Here the SG will contain N SUs and each SU can take the active HA state for some SIs and the standby HA state for some other. However, each SI can be assigned active to only one SU while it can be assigned standby to several SUs (Figure 2.c).

- **N-Way active redundancy model**

As the name implies, in this redundancy model, all N service units are assigned just as active and there is no SU with the standby HA state. Though, each SI can be assigned to more than one SU (Figure 2.d).

- **No-Redundancy redundancy model**

  Like the N-Way active redundancy model, this model does not support standby HA states assigned to the SUs. The difference is the one to one relationship between the SU and SI, meaning that in the No-Redundancy redundancy model, each SI is assigned to at most one SU and each SU protects at most one SI. There are no standby assignments in this redundancy model for the SIs (Figure 2.e).

The different redundancy models are displayed in Figure 2.

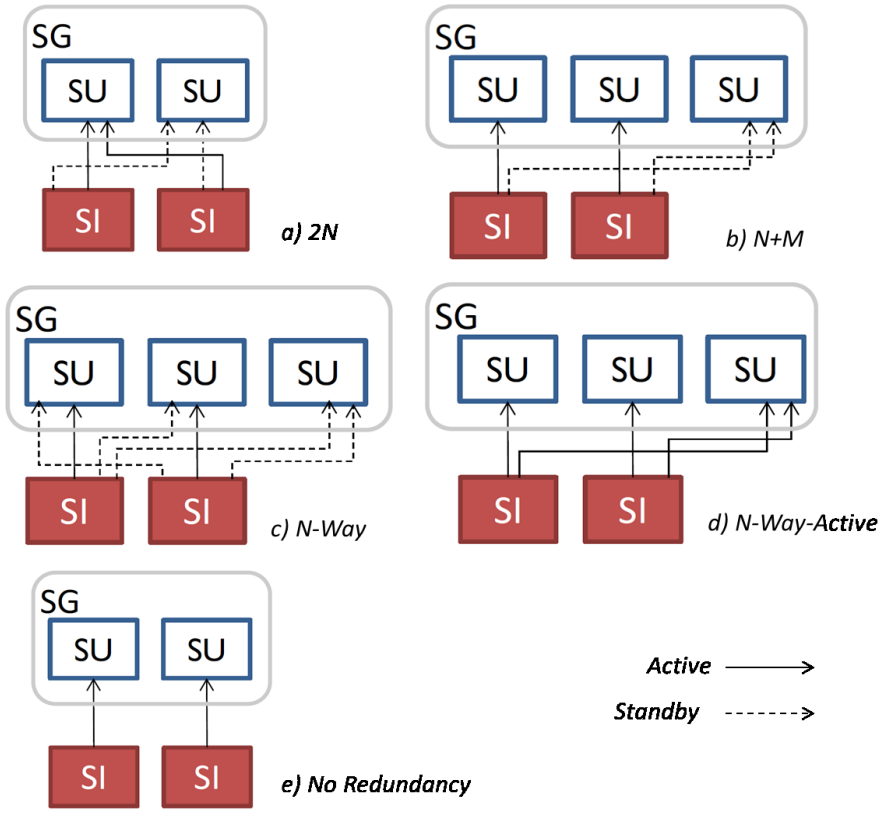

Figure 2 - Defined redundancy models in AMF

- Application

  An application is a logical entity comprising one or more SGs and SIs protected by those SGs. Each SG belongs to only one application. Application provides AMF with a higher scope for fault isolation and recovery.

2.1.1.2 Component Monitoring in AMF

For ensuring that the component is alive, AMF has defined three different methods for monitoring the component's health.

- Passive monitoring

  In this type of monitoring, the component is not involved and the operating system features are used to monitor the health of the component. One of these features is the processes of the components. For example the passive monitor checks the processes of the software component to see whether the process is still up and running or not.

- External active monitoring

  In this method, an external entity, called "monitor", is responsible for assessing the health of the target component by sending some service request and checking if it is responding in a timely manner.

- Internal active monitoring

  In internal active monitoring the component includes some code to monitor its own health. The health check is triggered by the component itself or by AMF.

2.1.2 OpenSAF

As mentioned before, OpenSAF is an open source project focused on Service Availability. It implements the SAForum service specifications [3], among others AMF [4], SMF [9] and IMM [10].

Figure 3 shows the overall architecture of OpenSAF middleware. OpenSAF has implemented the following SAForum services.

- AMF
- Checkpoint service (CKPT): provides a mechanism that can be used to replicate the state of application so that the service which application is providing, can continue after a failure.
- Cluster Membership Service (CLM): is the core for any clustered system. CLM decides whether the node is a member of the cluster or not. It monitors and provides the membership information of nodes in the cluster like joining or leaving the cluster. It also maintains a consistent view of the operational nodes in the cluster. Each cluster consists of a set of nodes with unique node name. CLM implements two logical entities: Cluster and Cluster node.
- Event service (EVT): provides standardized means to publish events and subscribe to events in the cluster.
- Lock service (LCK): provides a mean to manage access to shared cluster resources.
- Information Model Management (IMM) service: In this service, the different entities of the AIS services such as components, execution environments,

checkpoints and message queues are presented as managed objects in the information model (IM). IMM acts as the repository for these objects. It also provides an interface for the configuration and runtime management functions of the managed objects.

- Notification service (NTF): provides a mechanism to applications and services for notifying and explaining an incident or change of status to external entities. The notification service is based on publish/subscribe paradigm. The notification is done using three different interfaces, producer API, subscriber API and the reader interface.
- Log service (LOG): provides logging for alarms, notifications and system/application relevant log records
- Message service (MSG): provides service for inter-process communication

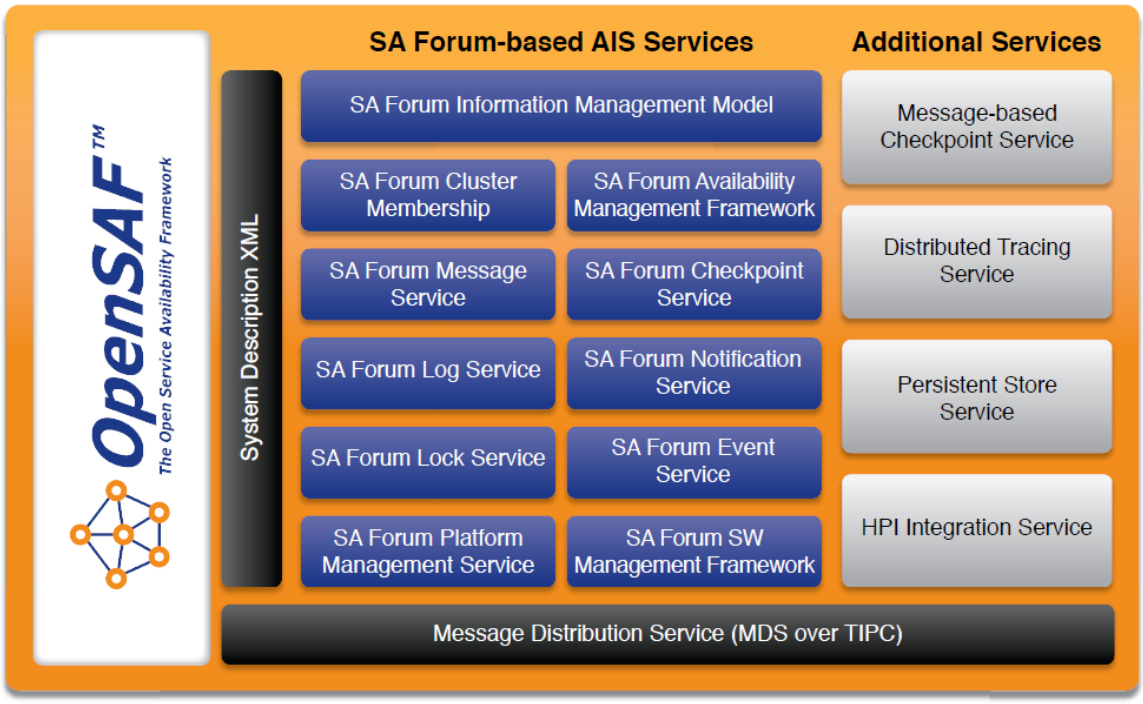

Figure 3 - OpenSAF 4.0 architecture [5]

system based on message queuing concept.

- Platform management service (PLM): The PLM Service provides a logical view of the hardware and low-level software of the system. Low-level software in this sense comprises the operating system and virtualization layers that provide execution environments for all kinds of software [11].
- Software Management Framework (SMF): complements AMF by providing a framework for delivering and upgrading software and hardware.

2.2 Virtualization and VMware

2.2.1 Virtualization

Virtualization separates resources or requests for a service from the underlying physical source of delivering the service and provides an abstraction of computing resources[6]. Virtualization is a term for the technology of creating a virtual or emulated version of software, hardware and resources. In this thesis we refer to virtual machine (VM) as a virtualized form of a physical node. A VM emulates the hardware resources needed for a node like RAM, CPU, etc., to function just like a physical machine. In virtualized systems, the physical resources are shared between the VMs running on the same physical node.

By using virtualization, the resource utilization can be increased because multiple VMs can run on just one host machine. It can also help the elasticity of the system by creating the VMs as needed. Virtualization is one of the key enablers of the cloud.

The advantages of virtualization include [12], [13]:

1. Cost efficiency:

   Having servers on the VMs instead of having a physical machine for each server

2. Testing and development purposes:

   The VMs can be used for trying and testing a new environment, software or OS.

3. Increased resource utilization:

   By providing a unified integrated operating platform for users and applications based on aggregation of heterogeneous and autonomous resources [12].

4. Green computing:

   Since we can have many VMs in a physical machine, the power consumption would be equal to one computer.

5. Increased scalability:

   VMs can be created according to the need and availability of the resources.

6. Portability of the VM images:

   VM images are actually file(s) on the storage device and these files can be ported and used on other machines with compatible software. These images can be kept as backups as well.

7. Legacy applications:

   Legacy applications can be kept within the VMs even if the organizations decide- to migrate to another platform.

The VMs are hosted on a specific software called **hypervisor** or **Virtual Machine Manager or Monitor (VMM)** and the machine which hosts the hypervisor and the VMs is called host machine where each VM is called guest machine [12].

There are two types of hypervisors:

- Native (bare metal)

  The hypervisor is installed and run directly on the host hardware. Examples of the bare metal hypervisors include: VMware ESX/ESXi [14], KVM [15], Microsoft Hyper-V [16] and Citrix XenServer [17].

- Hosted (non-bare metal)

  The hypervisor runs in a conventional operating system (the host) like Windows, Linux, FreeBSD, etc. Examples of this type of hypervisor include: VMware Workstation [18], VirtualBox [19], etc.

The two different types of hypervisors are illustrated in Figure 4.

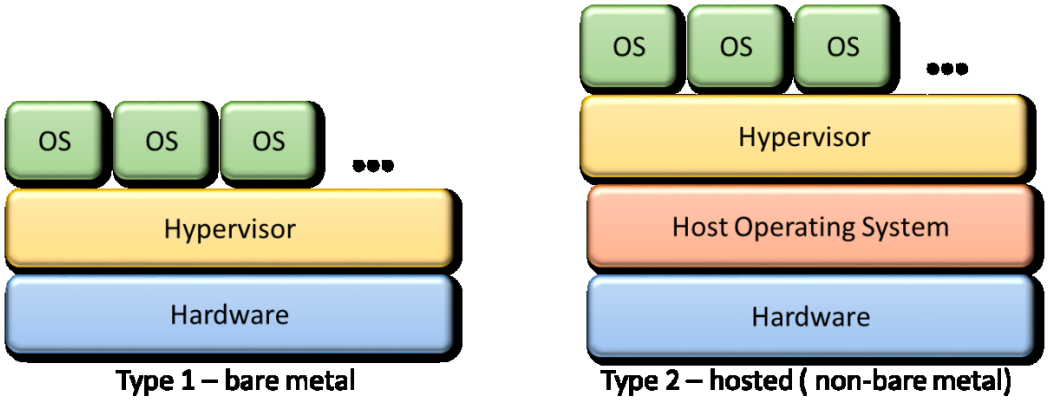

Figure 4 - Types of hypervisors

2.2.2 Cloud computing

Cloud computing is a new paradigm for enabling access to a shared pool of configurable computing resources. It provides on demand service to its customers by delivering different types of resources as service. Cloud systems most often use virtualization for sharing the resources and utilization of the available resources.

Cloud services are offered based on three basic models (Figure 5). These service models include:

- Infrastructure as a Service (IaaS), refers to providing VMs, network and storage resources as service to the end users.
- Platform as a Service (PaaS), the operating system, programming language execution environment and web server are provided as service to the end users.
- Software as a Service (SaaS), the end users are provided with software and applications as services.

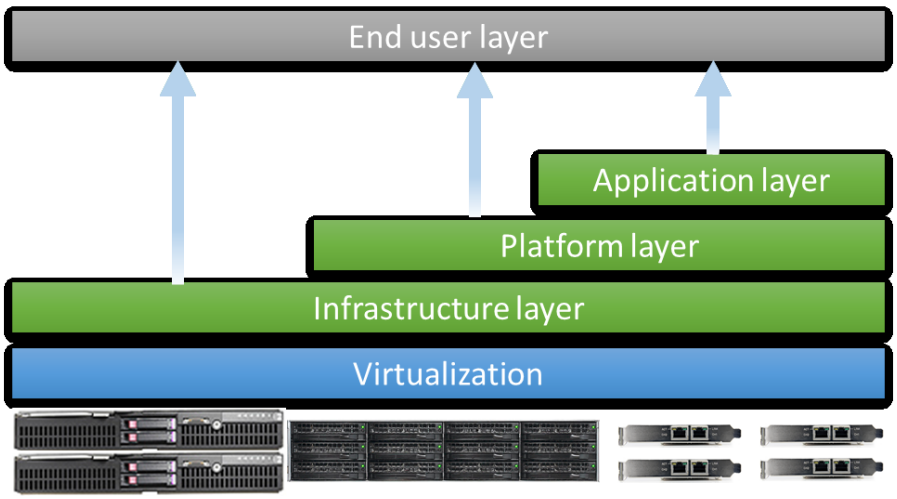

Figure 5 - Cloud service models

As mentioned before cloud computing is responsible for the delivery of the infrastructure, platform or software as service to the user. Virtualization forms the underlying infrastructure of the cloud as shown in Figure 5 and provides virtual computing, storage and networking for the cloud services.

Cloud computing offers different types of services to the users and then, availability of the services is an essential requirement. Each of the underlying layers of the cloud which

are shown in Figure 5, are subject to failure. The failure of any of these layers may cause the unavailability of the service to the user, emphasizing the importance of providing service high availability in cloud computing. From the redundancy models perspective all of the redundancy models can be deployed in the cloud. It is interesting to investigate how to achieve a service high availability management mechanism that provides availability in different layers.

2.2.3 VMware virtualization solutions

VMware [20] is one of the leading companies in providing virtualization solutions. VMware has a wide range of products including:

- **VMware workstation** is VMware's non-bare metal hypervisor.
- **VMware Player** is intended to run existing VM images. It also provides the basic means for creating a VM.
- **VMware ThinApp** is an application virtualization solution for Microsoft Windows which virtualizes the resources such as environment variables, files and Windows registry keys.
- **VMware View** provides remote desktop capabilities to users. It is mainly used for maintenance and administration purposes.
- **VMware vFabric** is an approach to building applications and running them on virtualized and cloud-based infrastructure. It includes the open source Spring Framework, VMware version of Tomcat, etc.
- **VMware vShield** includes security services for securing virtual environments and cloud environments at all levels -host, network, application, data and endpoint.

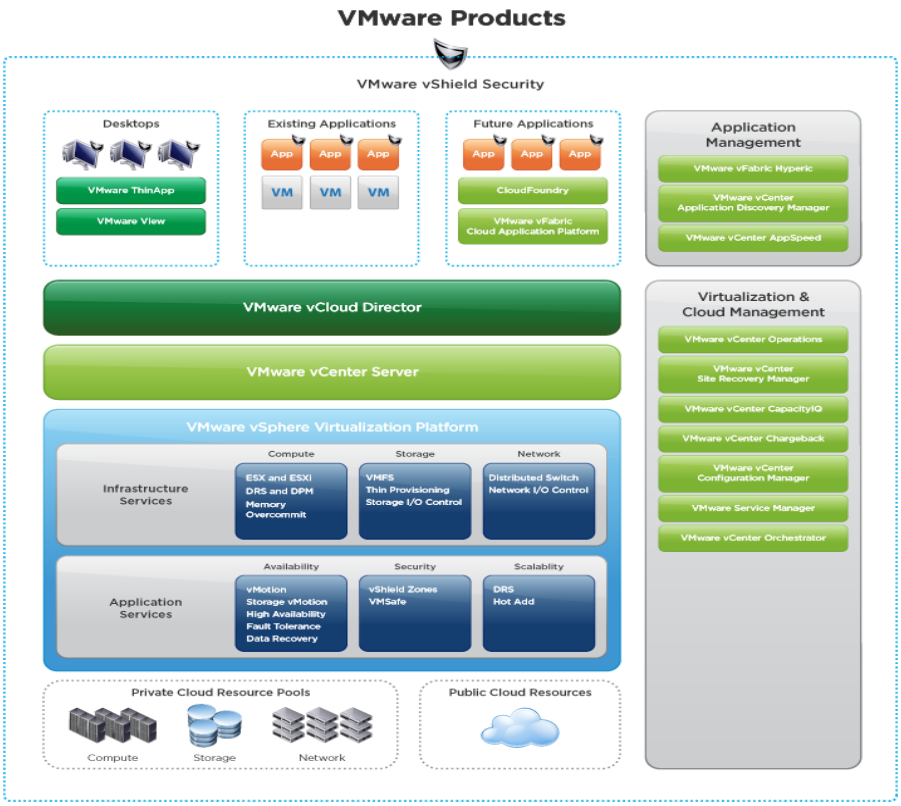

Figure 6 - VMware products [20]

- **VMware vSphere** is a cloud operating system. It virtualizes and aggregates the underlying physical hardware.
- **VMware ESX and VMware ESXi** are bare-metal hypervisors used in VMware vSphere.
- **VMware vCenter Server** provides unified management of all vSphere hosts and virtual machines in the datacenter from a single console.

Most of the VMware products are illustrated in Figure 6.

2.2.4 VMware solutions for availability

VMware offers two solutions for availability, VMware HA [21] and VMware FT [22]. Both solutions are part of the VMware vSphere.

2.2.4.1 VMware HA

VMware HA can be enabled in a cluster created in the VMware vCenter Server. VMware HA detects failed physical hosts or failed VMs. It uses a software agent deployed on each host, along with a network heartbeat to identify when a host is not responsive. If VMware HA detects that a host is not available, it restarts the VM(s) on another host in the cluster.

VMware HA uses the vCenter server to deploy the necessary software agent; but after hosts are enabled for VMware HA, the heartbeat and failure detection are completely independent from the vCenter server.

When hosts are added to a vSphere cluster with VMware HA enabled, two agents are activated on the host. These agents help detecting the failures in hosts by periodically sending information. Therefore, when VMware HA does not receive these information, it detects the failure. One of these agents is called the Fault Domain Manager (FDM). It is responsible for many tasks like communicating the host resource information, the VM states and HA properties with other hosts in the cluster. It also handles the heartbeat mechanisms, the virtual machine placements, the virtual machine restarts, the logging and much more. The other agent in the host is called 'hostd'. This agent is responsible for many tasks like powering on VMs and providing information about the VMs registered on the host to FDM. FDM talks directly to hostd and vCenter server [23]. Figure 7 shows a deployment example of VMware HA.

In a VMware HA enabled cluster, there are two types of hosts, master and slave. The master is selected through an election and the other nodes become the slaves. The election takes place when: VMware HA is enabled, the master host fails, becomes

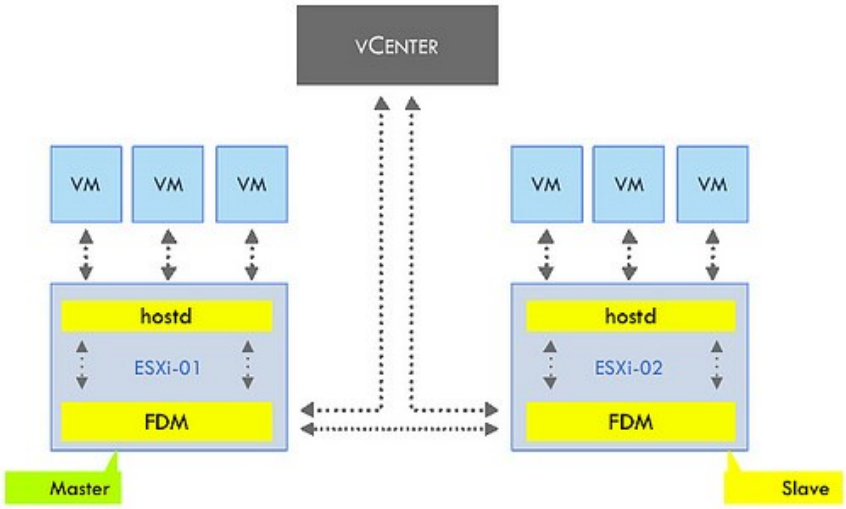

Figure 7 - VMware HA

network partitioned, disconnected from vCenter server or put into maintenance or standby mode. The election takes about 15 seconds. After the election, the slaves do not communicate with each other.

The master's main responsibility is to check the state of the VMs it is responsible for, and to take action when the VM fails. It is also responsible for exchanging the state information with vCenter server and initiating the restart of VMs when the host has failed.

A slave has fewer responsibilities compared to the master. A slave monitors the state of the virtual machines it is running and informs the master about any changes in this state. It also monitors the health of the master by monitoring the heartbeats. If the master is not available, the slaves initiate and participate in a new election.

Detecting the fault in VMware HA is done by heartbeat mechanisms. There are two types of heartbeating in vSphere: network heartbeating and datastore heartbeating.

In network heartbeating, each of the slaves sends a heartbeat to the master and the master sends a heartbeat to each of the slaves. This is done by default every second. When a slave does not receive any heartbeat from the master, it will try to determine whether it is isolated or not.

Datastore heartbeating helps to distinguish between isolation/network partitioning and failure of the host, if a host has lost its connectivity with the management network. If the master loses the network connectivity with the slaves, the "poweron" file, is updated by the isolated host while this does not happen when a host fails. Based on the results of the check, the master takes the appropriate action which is if the host has failed, then the master initiates a restart on the host. However, if the master determines that the slave is isolated, the action can be configured to power off, leave powered on or shut down through VMware tools installed on the VMs. By default, VMware HA picks two heartbeat datastores which are available for all hosts.

VMware HA can also detect VM failures using VM monitoring. It restarts the VM if its VMware tools heartbeat is not received within a set time. Occasionally, VMs or applications that are still functioning properly stop sending heartbeats. To avoid unnecessary resets, the VM Monitoring service also monitors a VM's I/O activity. If no heartbeats are received within the failure interval, the I/O stats interval is checked. The I/O stats interval is a cluster level attribute which determines if any disk or network activity has occurred for the virtual machine. If there is no activity the VM is reset. This interval is configurable between 30, 60 and 120 seconds.

VMware HA does not use any redundancy meaning there is no standby unit to take over when a failure happens; instead, as mentioned before, it restarts the VM in one of the hosts of the cluster. So the restart of the VM and other delays described in this section are included in the recovery time of the service. Also because of the VM's restart after the failure, the service is not necessarily continuous once the VM is recovered from the failure.

Application failures which are one of the aspects of this thesis are not fully covered in VMware HA. VMware introduced the application monitoring API in ESXi 5.0 for detecting the application failures but made it available to a limited number of vendors like Symantec. This API helps the detection of the failures, however for recovering from the failure, VMware HA restarts the whole VM which causes the time delays corresponding with the restart of an operating system.

2.2.4.2 VMware Fault Tolerance

VMware FT creates an exact replica of the protected VM and keeps this replica as secondary, ready to be brought into use upon a failure of the primary VM (similar to 2N redundancy model). VMware FT is built on the ESXi host platform using the vLockstep technology. The instruction executions of the primary VM are recorded and sent to the secondary through a network connection called logging channel and then replayed in the secondary (see Figure 8). Also, additional information is transmitted to ensure that secondary VM executes non-deterministic operations in the same way as the primary VM. Secondary replays all the operations except the outputs are dropped by the hypervisor [24].

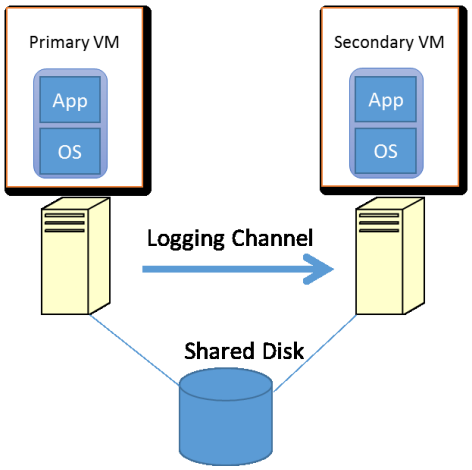

Figure 8 - VMware FT logging channel

In case of a failure the redundancy is restored by starting a new secondary VM in another host in the cluster. It only deals with fail-stop failures, which are server failures that can be detected before the failing server causes an incorrect externally visible action.

Figure 9 illustrates a timeline of events in VMware FT on the primary and secondary VMs. The arrows going from the primary line to the secondary line represent the transfer of log entries, and the arrows going from the secondary line to the primary line represent acknowledgements. As shown in the figure, an output to the outside world is delayed

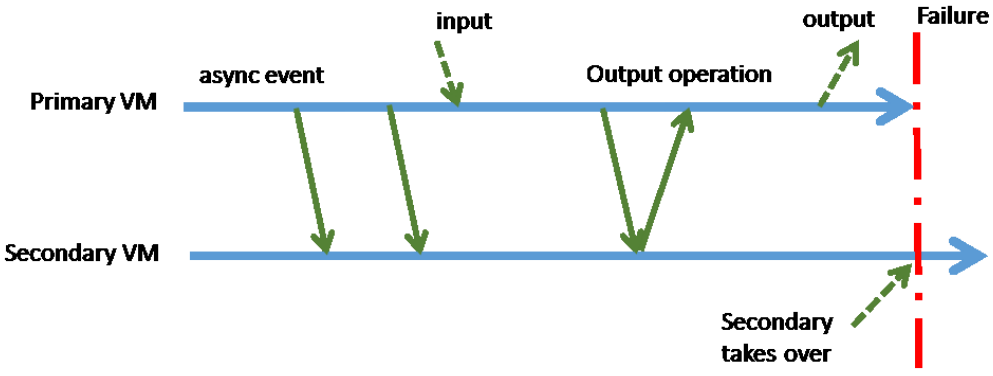

Figure 9 - Synchronization in VMware FT

until the primary VM has received an acknowledgement from the secondary VM.

For detecting failures, FT uses UDP heartbeating between hosts. A failure is declared if heartbeating or logging traffic has stopped for longer than a specific timeout.

Because there is one secondary VM along with the primary VM, we consider this mechanism as a 2N redundancy. In the event of Hardware failures or VM failures, the primary VM fails over to secondary, and secondary resumes the operating system where it stopped. So, if the failure can be detected by VMware FT, the service will be continuous after the recovery.

Unlike VMware HA, VMware FT does not have any kind of application failure detection mechanism. If the application fails VMware FT does not detect it and the VM continues as is.

2.2.4.3 VMware availability solutions summary

VMware, as one of the main providers of virtualization products, have introduced two solutions for availability. The previous sections focused on how these solutions function. As one of the motivations in this thesis we analyze their responsiveness to different kinds of faults in chapter 3. VMware also provides solutions for cloud management which is one of the targets and scopes of this thesis.

2.3 Related work

In the next sub-sections we will review research works on availability in virtual environments and also the other commercial virtual solutions supporting availability.

First we will discuss two of the works which are more related and then we will have a review on the other works.

2.3.1 A Case for High Availability in a Virtualized Environment (HAVEN) [25]

In this work, the states and the state transitions of the VM images are classified and extended. The original states defined by the Distributed Management Task Force in [26] include:

- Defined: The virtual system is defined but the virtual resources are not instantiated by the virtual platform.
- Active: The virtual system is instantiated and the virtualized resources are enabled to perform tasks.
- Paused: The virtual resources are instantiated like active but the virtual system and virtual resources are not enabled.
- Suspended: The state of the virtual system and virtual resources are stored on a non-volatile storage. Similar to Paused, the system and its resources are not enabled.

Accordingly for switching between these states, there are state transitions defined in [26] which include: define, activate, deactivate, pause, suspend, shut down, reboot, reset.

HAVEN (High Availability in a Virtualized Environment) [25] classifies the states of virtual machines into two more general states.

- Active: The active state is divided into two different states.
    - Operational: An operational virtual system provides service delivery

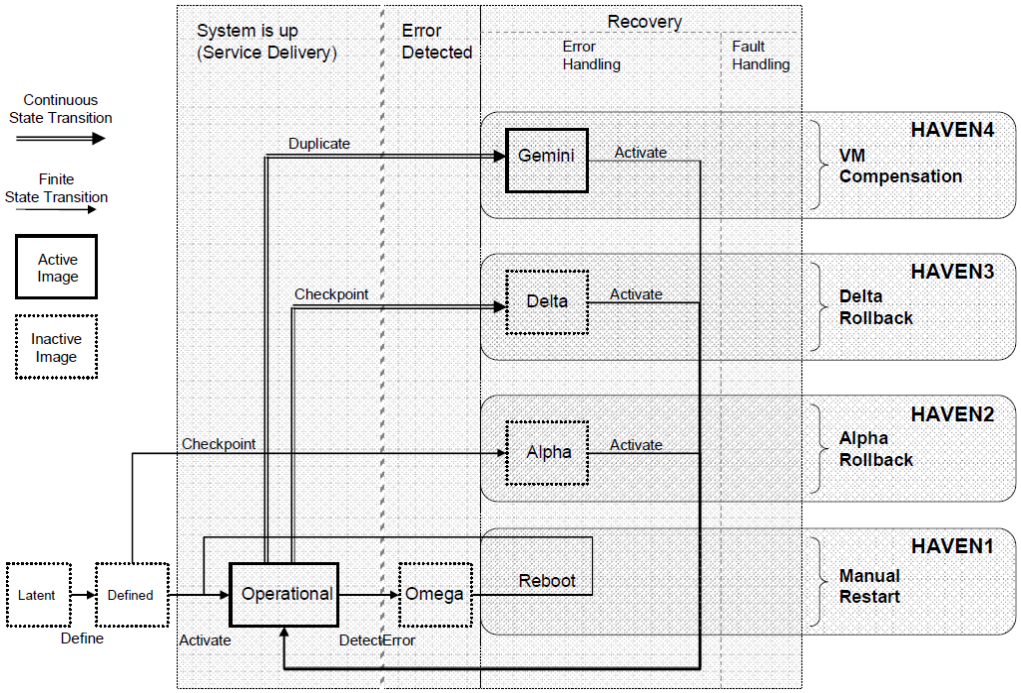

Figure 10 - VM life cycles defined in [25]

- Gemini: It is a clone of the operational virtual system. Gemini performs the same operations as Operational but manages the output and results similar to VMware FT.

- Inactive: The state that the VM is not able to perform tasks.
  - Planned: The VM is intentionally disabled or it is in an intermediate state to become operational. The planned inactive state is further categorized into: Latent, Defined, Alpha, Delta, Paused and Suspended. In which Latent, Alpha and Delta are the extensions to the original VM states. Latent is the image which does not exist yet. Alpha is created by taking checkpoint of a defined image. Delta is created by taking the checkpoint of an active image.

- Unplanned: An error has occurred causing service outage. When an error occurs and is detected while being in the active state, the image transitions to the Omega state.

They defined different levels of availability ranging HAVEN1 to HAVEN4.

However, in this work they just focused on the states and transitions when an error is detected. They did not discuss how they are detecting the hardware or software failures.

2.3.2 Remus: High Availability via Asynchronous Virtual Machine Replication [27]

This work which is implemented based on XEN hypervisor [17], is an attempt to provide high availability for virtual machines by replicating the host. Their goal is to recover from fail-stop failures of a single physical machine, "the failed process stops working and all data associated with the failed process are lost" [28]. The backup host, asynchronously updates its state slightly after the active replica. It can be classified as 2N redundancy model since there is a backup or standby replica taking over in the event of failure.

Remus used speculative execution for increasing the system performance; it means the

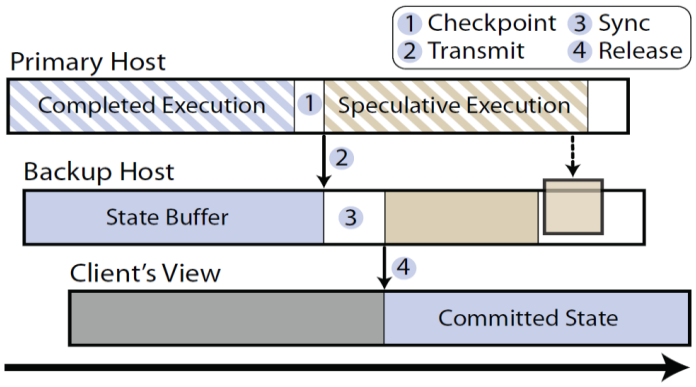

Figure 11 - Speculative execution and asynchronous replication in Remus[27]

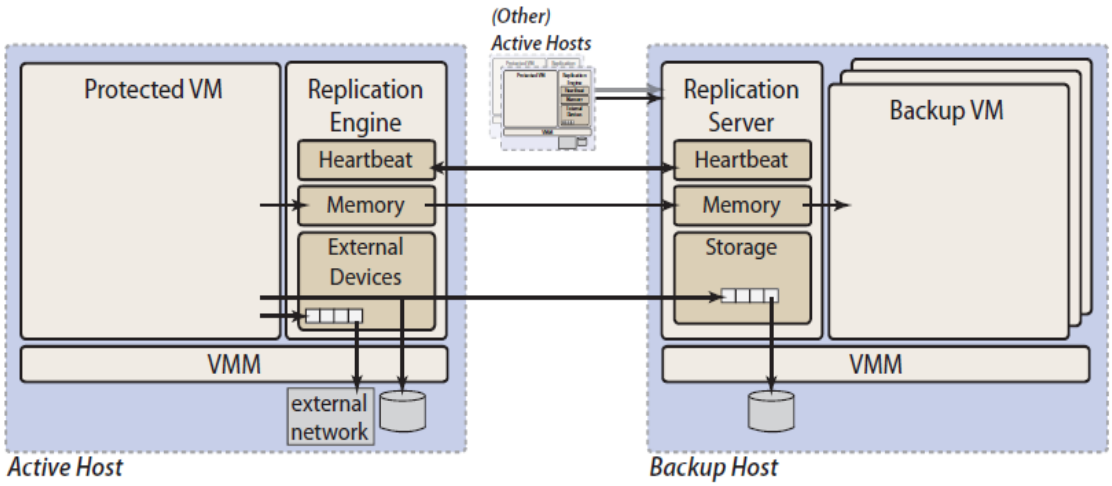

Figure 12 - Remus high-level architecture [27]

active host continues its execution and producing output while the replicated host is synchronized asynchronously by buffering output at the primary host as shown in Figure 11.

The high availability is achieved by propagating frequent pipelined checkpoints of the active VM to the backup physical host. Figure 12 shows the high level architecture of Remus illustrating the active and backup hosts and their communication.

2.3.3 Other related works

In nearly all the works the focus is on the host or VM failures and generally they use similar techniques like restarting the VM or using hot, warm and cold standbys [29] for replication of the VMs. Checkpointing is one of the common ways to maintain the state as discussed in HAVEN [25] and Remus [27].

In [30] an approach for providing high availability to the request of cloud clients is proposed. Similar to the previous works, the failover mechanism is done by checkpointing. There is a request manager defined who gets the requests from the clients.

It uses a global checkpoint to manage the status of sub-clouds where the jobs are distributed. To decrease the checkpointing overheads they used multilevel checkpointing. In the case of failure in one of the nodes, the jobs are migrated to the failed node's secondary node.

Loveland et al [31] used virtualization for cost reduction and resource consolidation of the traditional HA approaches like active/active active/cold-standby and active/passive. They used LPAR for logically partitioning the physical host and create the replicas on a logical partition. They considered three different types of failures: crash, hang and loop.

In [32], Braastad introduced a solution for providing high availability using virtualization. In this work, an add-on to Heartbeat – an open source software package used to create high availability clusters – is developed, allowing Heartbeat to be able to seamlessly migrate the VMs between the physical nodes, when shut down gracefully. As mentioned, the emphasis is on graceful failures meaning the solution does not support uncontrolled failures like power, hardware or network failures.

One of the big advantages of our work in comparison to the other works is focusing on the availability of the service rather than only covering the availability of the VMs.

2.3.4 Virtualization solutions supporting availability

With improvements in hypervisors and virtualization because of the increase in computer performance, and the demand for high availability, virtualization solution vendors started to support availability as part of their products. Hypervisors like VMware ESXi, embedded HA and FT techniques in their products which we already discussed. The other virtualization solution to embed the availability support is Windows Azure [33], which is

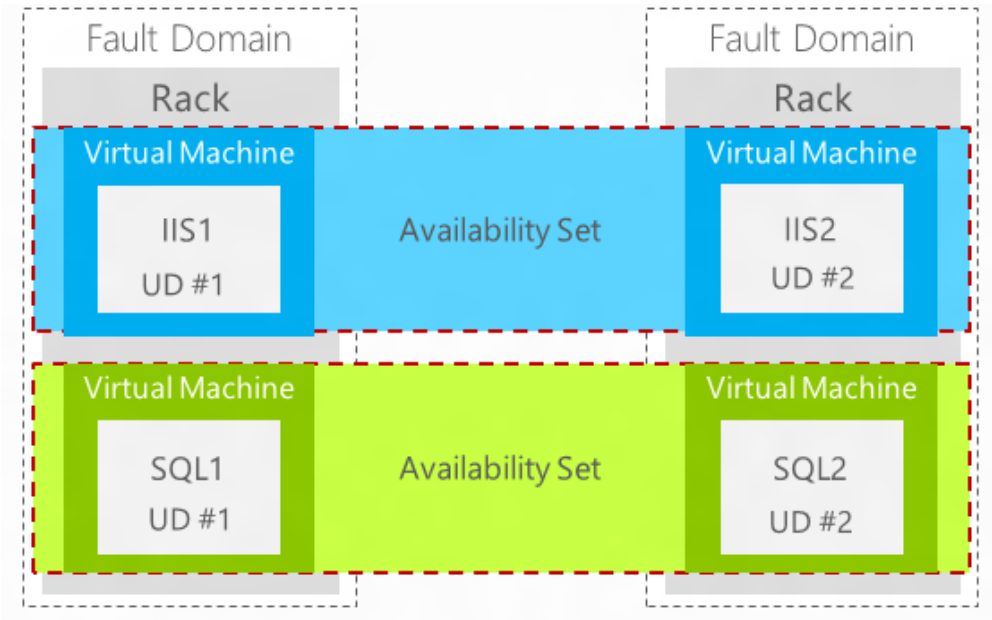

Figure 13 - Availability management in Windows Azure

the Microsoft's cloud platform. Windows Azure clustering solution also has the same approach of active/passive architecture but the replication is limited to the replication of the physical node.

They manage the availability of the application that uses multiple virtual machines by adding the machines to an availability set. Availability sets are directly related to fault domains and update domains. In fact, a fault domain is closely equivalent to a rack of physical servers. Figure 13 shows a sample configuration of availability management in Windows Azure.

Xen is another hypervisor which supports availability management. Xen uses Remus which is described in 2.3.2.

Chapter 3

The baseline architectures for availability, metrics and measurements

This chapter covers the baseline architectures for availability including OpenSAF and VMware solutions. We will introduce our qualitative criteria and the metrics. This chapter also covers the measurements for the defined metrics and finally we analyze these measurements and results. First we will describe the environment used and the infrastructure on which we ran our experiments followed by the application case study.

3.1 Hardware test-bed

For experimenting with VMware and OpenSAF we configured a cluster of 5 nodes with identical hardware (see Figure 14). All the nodes are Dell Power Edge 1950 [34]. There are also two switches of type Summit X450a-48t [35]. We also built a VLAN on the ports of the switch for our experiments. All the nodes are connected to both switches for redundancy. The first switch is connected to the outside world by an uplink connection. All the nodes are capable of connecting to internet when it is needed (i.e. installing software or upgrading), but not otherwise.

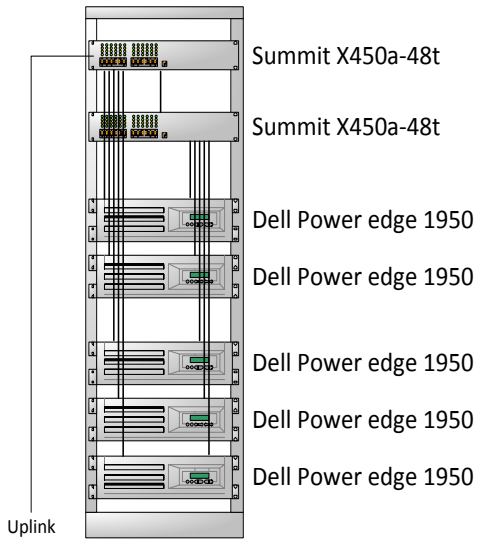

Figure 14 - The Magic Cluster (Ericsson's cluster in Concordia University)

3.2 Case study application

As the case study application, we selected the VLC media player [36]. There are many reasons for selecting VLC as our case study application. One of the reasons is that VLC has been already modified to work with OpenSAF as a SA-Aware component [37]. Streaming or playing multimedia helps us visually view the behavior of the HA solutions when facing failures. Service continuity after failure is an important feature in our experiments.

In our experiments we used the streaming functionality of VLC to stream a video across the network. The clients can view the video. We experimented with different failures on the streaming node to see the behavior and to measure different metrics defined in a following section.

To make VLC an SA-Aware component Kanso et al [37] made some changes to its code including checkpointing the media stream to enable service continuity. An IP component

is also added which is responsible for keeping the streaming IP address live. In the SA-Aware version, VLC component is a pre-instantiable component meaning that VLC is initialized before the SU takes the active assignment. On the other hand the IP component is a non-pre-instantiable meaning it is only instantiated when the SU gets the active assignment. This is described extensively and in more detail in [1] as well. In addition to the existing SA-Aware version of VLC, we used the original code as a Non-SA-Aware version where the VLC code was not changed. In both of the versions, we need a standby component because if any of the streaming component or node fails, the standby component takes over for having a better recovery time for the service. In our case study application, the SG is configured to be 2N redundancy model. The No-Redundancy redundancy model and N-Way active redundancy models are not applicable because they do not support the standby assignments. Our components do not support x_active_and_y_standby capability model which is the requirement of N-Way redundancy model, so N-Way is not a good match either. In our case study we have one SI with one active and one standby assignments. If the N+M is used with just one SI, it would be equivalent to the 2N redundancy model and could have been used as well. 2N is a special form of N+M redundancy model.

In our architectures, VLC can be used as an application in VMware HA, or as a non-SA-Aware or an SA-Aware component managed by OpenSAF.

3.3 Deployed architectures

We designed and configured three different architectures to experiment with. Following subsections are for the description of our architectures. In all of the architectures we used Ubuntu 10.04 as the operating system.

3.3.1 Architecture 1: OpenSAF on the physical nodes

In this architecture the operating system is installed directly on two of the physical nodes of our cluster. We installed OpenSAF on both of them and also compiled and installed our SA-aware version of VLC 1.1 on top of the operating system.

Figure 15 shows the SUs containing two components in the physical nodes. Since we are using a 2N redundancy model, a possible active assignment performed at runtime by OpenSAF is shown with solid line and the standby with the dashed line.

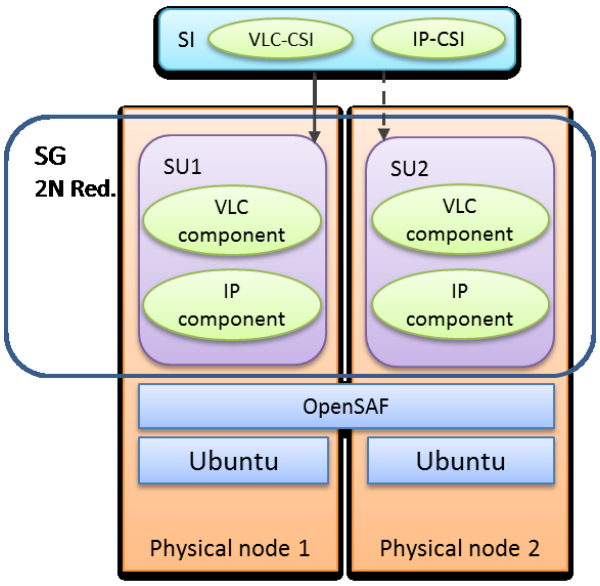

Figure 15 - OpenSAF on the physical nodes

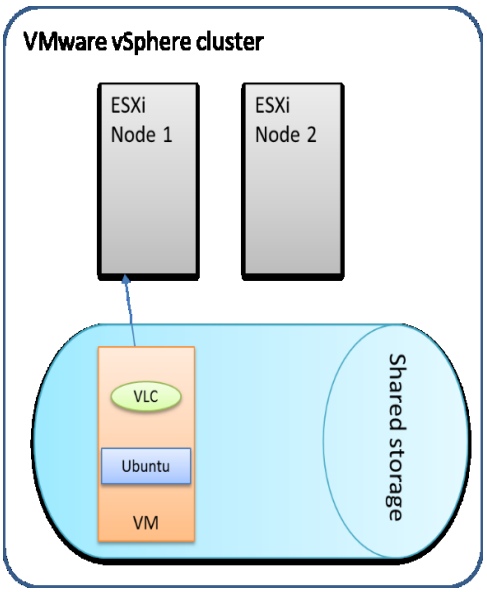

Figure 16 - Virtual machine running the original VLC managed by VMware HA

3.3.2 Architecture 2: Availability using VMware HA

In this architecture we created a vSphere cluster using two ESXi nodes and enabled VMware HA on the cluster using VMware vCenter. We added a VM with Ubuntu and installed the original VLC media player application. We put the VM image as shown in Figure 16 on a Network File System (NFS) shared storage so that it is accessible from all ESXi nodes in the cluster. The reason is that when a failure happens on the host or the VM, the VM can be restarted on the other ESXi node.

3.3.3 Architecture 3: Availability using VMware FT

VMware FT has stringent hardware requirements. Our test-bed at Concordia University does not meet these requirements. For this reason we used three blades in another test-bed from a provincial project called 'ÉcoloTIC'[38].

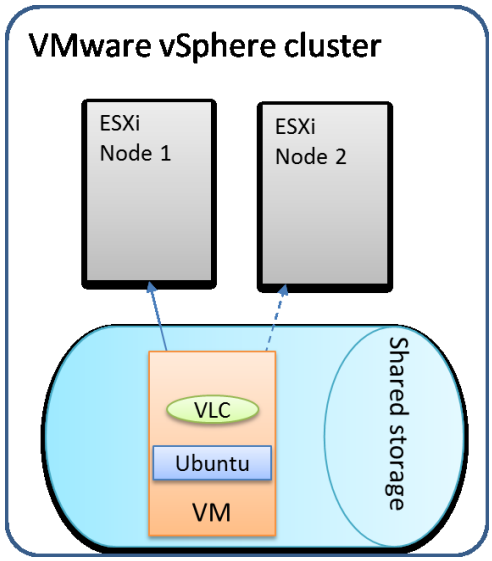

Figure 17- Virtual machine running the original VLC managed by VMware FT

The configuration is very similar to the configuration we had in VMware HA architecture, but the difference is that we enabled the FT on one of the VMs in our cluster. This architecture uses ESXi as its hypervisor and the VM is placed on a shared NFS storage. VMware FT creates a second VM running as secondary in another node (See Figure 17)

3.3.4 Architecture 4: OpenSAF deployed in VMs

VMware HA does not detect the application failures. It is a solution for recovering from hardware and VM failures. OpenSAF is designed for making application services highly available. To take advantage of the VMware virtualization and the service high availability management of OpenSAF we combined these two solutions.

In this first combination we deployed the OpenSAF cluster on virtual nodes rather than on physical nodes. We used this architecture with both VMware HA enabled and disabled.

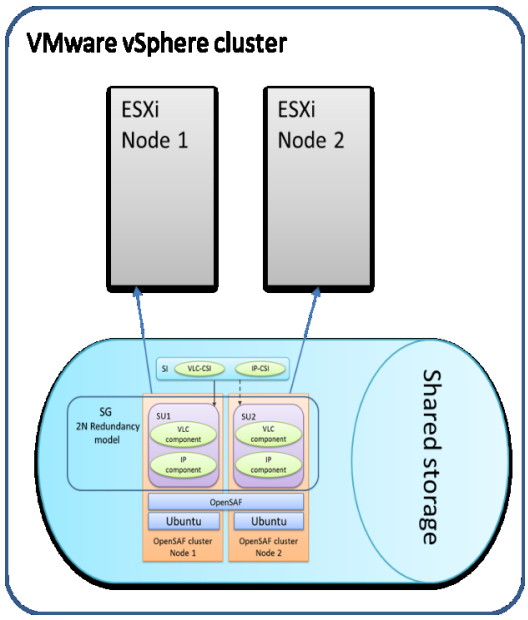

Figure 18 - OpenSAF deployed in VMs

Figure 18 shows this architecture where the OpenSAF configuration used in the physical nodes was applied in the VMs. We put the VM images on a shared storage so that they would be accessible for both ESXi nodes.

3.3.5 VMware HA settings on the cluster

Here we will describe the settings applied when creating the VMware cluster. During the establishment of the cluster we turned the vSphere HA on.

In VMware HA settings we also enabled the "host monitoring" so that the ESXi hosts would be able to exchange network heartbeats to make sure that the host is alive. We disabled admission control so that the cluster capacity would not be reserved for VM failovers. This feature reduces the number of VMs that can be run and might cause the failover not to happen after a failure. In the VM monitoring status we turned the VM

monitoring on and set the VMs restart priority to high so that VMware HA can detect the VM failure as soon as possible.

We also added two NFS shared storages to our cluster to host the VM images and to be used for datastore heartbeat.

3.4 Qualitative criteria

Our first step for comparing different solutions is comparing them from their general characteristics and their functionality. In this section we look at the criteria of the different solutions.

3.4.1 Complexity of using the high availability solution

This is related to how much effort and time are needed to make the availability solution functional.

- **OpenSAF** provides many solutions for providing different levels of high availability and some of them do not require making changes to the source code (Non-SA-Aware). If the application wants to support the middleware APIs and support some features like service continuity, the source code should be changed to use the advanced features of OpenSAF.

  OpenSAF installation needs some configurations to make it run correctly.
- **VMware HA & FT**: There is no need to change anything. Application needs to be installed in the VM and VMware HA takes care of hardware and VM failures. VMware monitors the health of the node and the VM as described before. So the only complexity we face is the setting up and configuring the environment which is straight forward. VMware HA in

vSphere 5.0 introduced the application monitoring API for detecting the application failures. Using these APIs also beg for altering the application source code.

3.4.2 Redundancy Models

This section discusses which redundancy models are supported in each of the technologies.

- **OpenSAF** supports five different types of redundancy models which gives the user more options. As discussed earlier, the redundancy models which OpenSAF supports are: No-Redundancy, 2N, N+M, N-Way, and N-Way active. The applications can use any of these redundancy models according to their specifications and requirements.
- **VMware HA & FT:** VMware has not defined any redundancy model specifically, but in comparison to the redundancy models defined in OpenSAF we can say that VMware HA is similar to the No-Redundancy redundancy model and VMware FT is similar to 2N, since in VMware FT there is a standby replica of the VM ready to take over if a failure happens.

3.4.3 Scope of failure

The range of failures that the availability solution can detect and recover. The failures can range from application level failures to hardware and VM failures.

- **OpenSAF** can detect a wide range of failures like hardware, operating system and application failures.

- **VMware HA & FT,** detect hardware level and VM failures with limited failure detection of the operating system. VMware FT does not support the application failures at all. In VMware vSphere 5.0, VMware released the application monitoring API, supporting application level failure detection, to a limited third party vendors like Symantec (Symantec ApplicationHA [39]).

3.4.4 Service continuity

Service continuity is the continuation of the service after a failure from the point where the failure occurred. For instance, in the case of the failure of the VLC service, the streaming of the video continues from the point where the failure happened.

- **OpenSAF:** we can achieve service continuity by using services like checkpointing in OpenSAF. Using checkpoint service if the active application component fails, the redundant component takes over and continues from the point where the failure happened.
- **VMware HA & FT,** act differently in this context.
  - **VMware HA** restarts the failed VM so the service would be available after the restart. There is no continuation of the service.
  - **VMware FT** creates an exact replica of the VM on another host. Primary and secondary nodes are synchronized as described before, so if there is a hardware or virtual machine failure on the primary VM, the secondary VM will take over and continue the

service. Problem arises when there is a software failure. VMware FT is not capable of detecting software failures.

3.4.5 Supported operating systems:

This section describes the types of the operating systems supported by these technologies.

- **OpenSAF** is implemented only on Linux so it can provide high availability only for Linux based applications.
- **VMware HA & FT** are independent of the operating system running in the VM. So these solutions are operating system independent.

3.4.6 Summary

Table 1 shows a summary of the qualitative criteria in different architectures.

	OpenSAF	VMware HA	VMware FT
Complexity of using the high availability Solution	Needs many initial configurations (More complex)	Not much configuration	Not much configuration
Redundancy models	Five different types of redundancy models	No redundancy	Primary-Backup (2N)
Scope of failure detection	Hardware - OS - application	Hardware - VM (very limited application)	Hardware - VM
Service continuity	Yes (using SA-Aware components and checkpointing)	No	Yes
Supported operating systems	Linux	Platform independent	Platform independent

Table 1 - Summary of the evaluation of the architectures with respect to qualitative criteria

3.5 Metrics

To evaluate and compare the two solutions and their combinations from an availability perspective, we defined a set of metrics. We selected these metrics in a way that they can be measured uniformly in all of the architectures.

In the following subsections, we will introduce the metrics and discuss how they can be measured in different solutions.

3.5.1 Reaction Time

Reaction time is the duration from the time when the failure happens until the time of the first reaction seen from the availability solution. The first reaction is different in different types of architectures. The details are described in sections 3.7 and 3.8. The failures are generated by a user command or executing a script. So in some cases from the execution of the command until the real failure there is a delay which is included in the reaction time. But since in the compared metrics the same delay is included, we consider them as comparable.

3.5.2 Repair Time

It is the duration of time that the availability solution repairs the failure. For example, in the 2N redundancy model, although the service is recovered by failing over to the standby unit, the faulty component needs to be repaired for acting as the new standby for the new active unit. So this metric shows how much time it takes for the faulty unit to become operational again.

3.5.3 Recovery time

Recovery time is the time of the first reaction until the service has been recovered. This time shows how much it takes for the availability solution to recover from the failure and resume/start over the service to the end user.

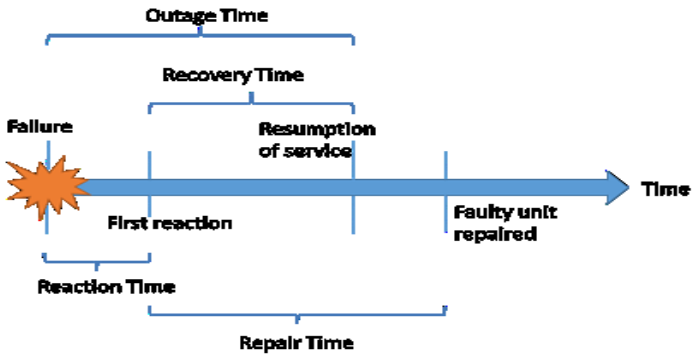

Figure 19 - Time related metrics

3.5.4 Outage time

It is the period of time the service is not available to the user. Basically, outage time is from the occurrence of the failure until the user gets the service again. So it can be considered as 'Reaction Time' + 'Recovery Time'. This is one of the most important metrics because minimizing the outage time is one of the factors to keep the service more available.

Figure 19 illustrates the defined metrics and their relations.

3.5.5 Memory consumption

With this metric we want to determine the amount of the memory overhead induced by each solution. Memory is one of the resources that impacts the performance of the service so the memory overhead of each solution plays an important role in the overall performance of the service.

Using Linux commands, 'vmstat' and 'pmap', we can get the amount of memory consumed. VMware has monitoring functionality that displays the memory consumption.

3.6 Types of failures

For the experiments we considered different failures.

- **VLC component failure**

  This failure is achieved by "killing" the VLC component process in the operating system.

  This kind of failure causes reaction only on architectures using OpenSAF because neither VMware HA nor VMware FT can detect application failures.

- **Physical node failure**:

  For simulating the node failure, we used force-rebooting the physical node. In the measurements associated with force-rebooting, there is some immeasurable time, from the reboot command until the real restart. But since this is the case for all node failure measurements, the results are comparable. So for our measurements, we used the time of issuing the force-rebooting command.

- **VM failure**:

  VMs have process IDs in the bare-metal hypervisors and in case of non-bare-metal hypervisors in the host OS. Killing this process can be used for simulating the VM failure in our experiments.

Some of these failures are not applicable on all architectures. For instance VM failure is not applicable when OpenSAF is installed on a physical machine. Also, component failure cannot be detected where the only availability solution is VMware HA or VMware FT. Table 2 summarizes the baseline architectures and the respective applicable failure types.

Architectures		Failures		
		VLC component failure	VM failure	Node Failure
OpenSAF on physical nodes	SA-Aware VLC component	√	Not applicable	√
	Non-SA-Aware VLC component	√	Not applicable	√
OpenSAF on virtual nodes	SA-Aware VLC component (with/without VMware HA enabled)	√	√	√
	Non-SA-Aware VLC component (with/without VMware HA enabled)	√	√	√
VMware HA		Not detectable	√	√
VMware FT		Not detectable	√	√

Table 2 - Baseline architectures and applicable failure experiments

3.7 Measurements in OpenSAF related architectures

When a failure happens, AMF detects the failure and tries to clean up, instantiate and register the related component(s) if they are pre-instantiable and SA-Aware. Meanwhile, the active assignment is failed over to the standby SU. If the assignment is successful, the new active component will provide the service.

In our case, as described before, there are two components VLC and IP where VLC is a SA-Aware component and is pre-instantiable, and IP is a non-SA-Aware component.

Applying our defined metrics to the sequence of actions taking place in the event of a failure, the reaction time is from the time of the failure until the component is cleaned up. Recovery time is from the time of the component clean-up until the active assignment is assigned to the standby SU. The clean-up time until the registration of the failed component is considered as the repair time and the time of the failure until the assignment of the active assignment would be the outage time.

The reaction time is measured differently when there is a node or VM failure. We considered the first reaction as the instantiation of the IP component on the second node

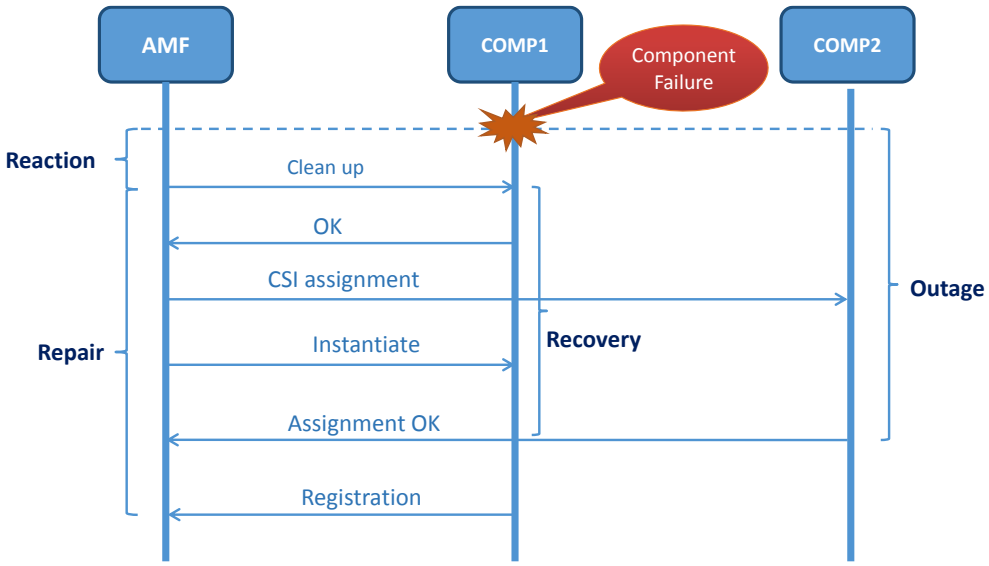

Figure 20- The sequence of actions done during the failure of a SA-Aware component

because AMF cannot clean-up the components and the first reaction to the failure is starting the IP component on the other node.

Figure 20 shows the sequence of actions when a component fails and the assignment fails over to the standby. It also shows the ranges of the times that are measured. For the measurements we added some timestamps in the Component Life Cycle Command Line Interface (CLC-CLI) scripts and in the source code of the SA-Aware VLC media player. This is done to extract the exact and appropriate times. The timestamps include the time of the registration of the VLC component to AMF and getting the active assignment.

The following lines are added to the shell scripts of the CLC-CLI commands to show and record the exact moment of executing the scripts:

```
myTime=$(date +%T+%N)
logger "X is being started|terminated|cleaned up at:$myTime"
```

Where X can be the VLC or IP components. Using this command we can keep track of the time the scripts are executed. "`date +%T+%N`" returns the exact time of timestamp in nanoseconds and `logger` command logs that time with the description to the log of the system so we would be able to retrieve it later.

Since the quality of our measurements depends highly on time synchronization, we used the Network Time Protocol (NTP) [40] and Precision Time Protocol daemon (PTPd) [41]. PTPd is an implementation of Precision Time Protocol (PTP) the IEEE 1588 standard [42].

3.8 Measurements in VMware based architectures

Since VMware is a proprietary software and we do not have the options to put the timestamps in the code, we used the times logged by VMware and scripts added to the guest operating system.

As described before, there are two types of nodes, slave and master. When a host fails, depending on the type of the node, the sequence of actions is different. When a slave node fails, after three seconds the master begins monitoring the datastore heartbeat for 15 seconds to make sure this is not a split brain (Split brain happens when a node is functional but has lost connection with the other nodes). On the 10^{th} second when no network or datastore heartbeats have been detected, the host is declared as "unreachable". The master will also start pinging the management network of the failed host at the 10^{th} second and will continue that for 5 seconds. The host is declared dead at 18^{th} second and the VMs on the failed host are restarted on other hosts. Figure 21 shows the described procedure.

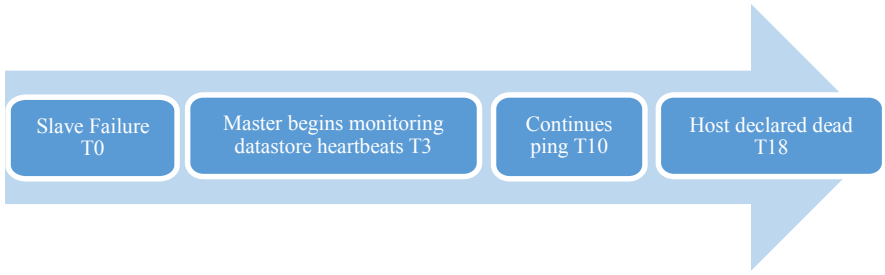

Figure 21 - Sequence of actions after failure of a slave node in VMware HA

The steps taken after the failure of the master node is slightly different because there needs to be a master before any restart can be initiated. It means there should be an election among the slaves to select a new master.

Since the slaves get the network heartbeat from the master, if the slaves stop receiving the network heartbeats the master is declared as unreachable. This happens in the 10th second after the master's failure, which is the time the election starts. The election takes 15 seconds. From 25th second to 35th, the new master reads the HA protected virtual machines. In the 35th second, master initiates the restart of all VMs which are not currently running [23]. This is shown in Figure 22.

Figure 23 shows the correlation of our defined metrics according to the VMware HA failure recovery procedure.

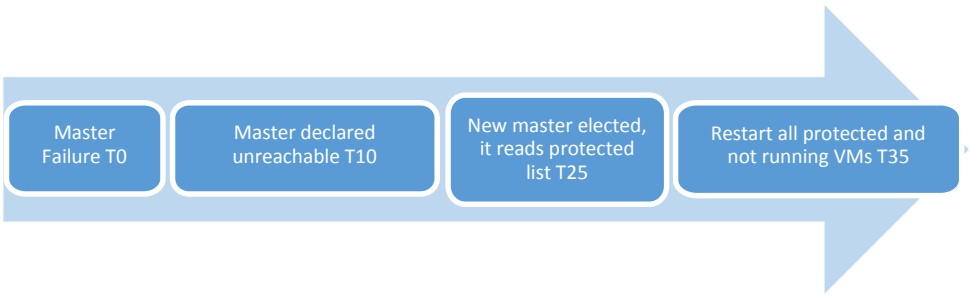

Figure 22 - Sequence of actions after failure of a Master node in VMware HA

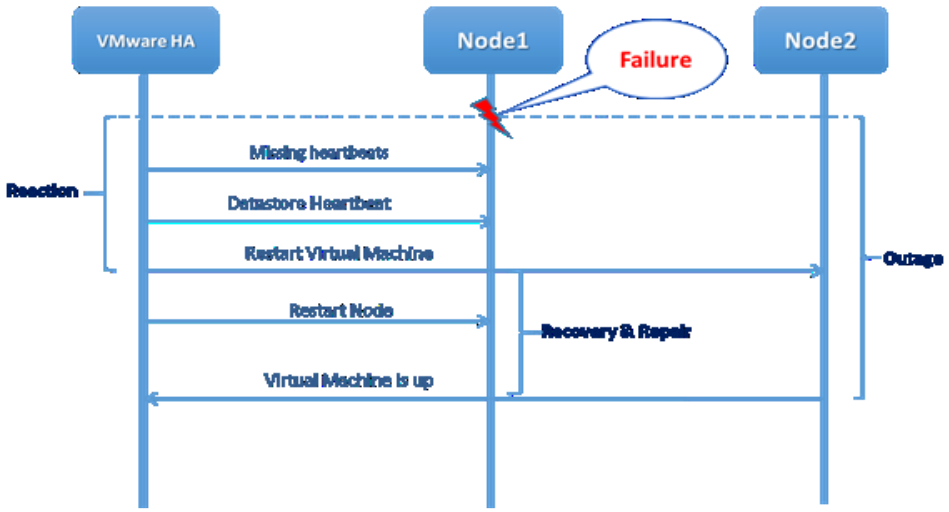

Figure 23 - The sequence of actions during a node failure in VMware HA

We considered the first reaction to the failure as the initiation of the restarts of the VM because before that there is no reaction seen from VMware HA side. So as illustrated, the reaction time would be from the time of failure until the time VMware initiates a restart on the failed VM. This time can be retrieved from the logs on the ESXi hypervisor.

In this architecture, there is no secondary or standby VM so the recovery would be achieved by repairing the failed VM. Repairing means restarting the failed VM in the

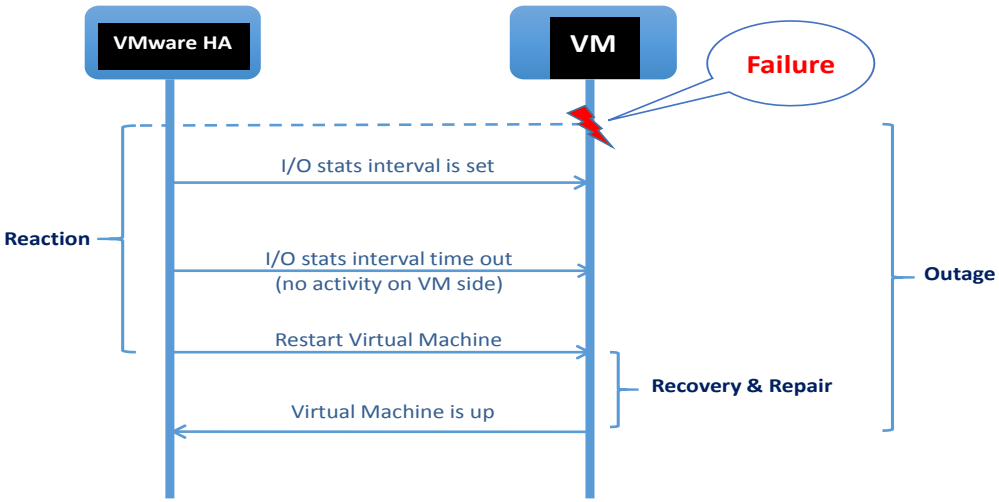

Figure 24 - The sequence of actions during a VM failure in VMware HA

same or another ESXi host. The outage of the service is calculated from the time of the failure until the failed VM is up and running.

Figure 23 and Figure 24 show the overall sequence of events in the VMware HA architecture for node and VM failures. Those figures also show the ranges of time measurements corresponding to the defined metrics.

For VMware HA, we used the VMware official documents to review what happens after a failure, but in the case of VMware FT, less documentation is available. For this reason we used the log files to follow the sequence of events. Also we used the log for all measurements except the outage time that we used an intrusive method. We simulate the failure of the VM by killing the process of the primary VM and then logging the time as the failure time. For the other times we dug in the logs of the ESXi hypervisors to find the notifications of the reaction and repair of the VM(s). The first log entry we found as the reaction to the failure was 'Destroyed hbResponseWorldlet'. So we considered the period from the failure time until this timestamp as the reaction time. The log entry

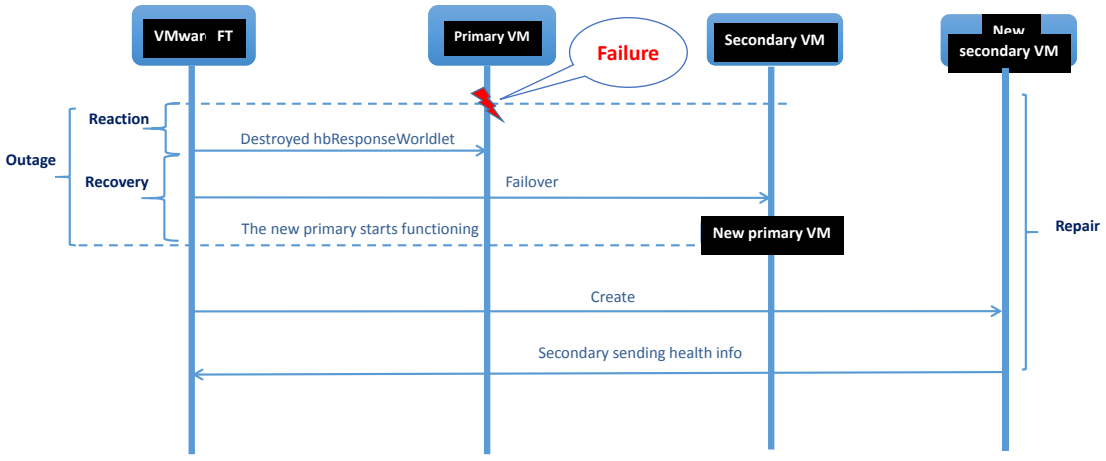

Figure 25 - The Sequence of actions after the failure of the primary VM in VMware FT

showing the repair was: ‘Secondary sending health info’. This log entry along with its timestamp shows the time that the secondary VM is created somewhere in the cluster and it is ready.

For the outage time we did not actually find the appropriate log record for the time the new primary is ready to continue the service. In the related log, the recorded time conflicted with our experiments in section 3.10.5. Because of this, we used an intrusive method to measure the outage time. We created a script to log a timestamp with the frequency of 10 milliseconds. We started the script in the VM and then triggered the failover by killing the VM process. The time difference in the log from the time of the failure until the time of the new timestamp after the failover would be the outage time in VMware FT. Because of using the intrusive method, the recovery time is assumed as the difference from reaction to the calculated outage time. The sequence of actions and events after the failure of the primary VM are shown in Figure 25.

3.9 Measurements from our experiments

In the previous section, we defined the details of how to measure the defined metrics in our baseline architectures. Here we will show the measurements of our defined metrics in different tables for each failure type. The measurements were taken in each architecture for each applicable failure type. We repeated each experiment 10 times and we determined the average. By repeating the experiments we make sure that the results can be replicated and stay the same after many repeats.

The first three rows in the following tables represent the measurements for the SA-Aware VLC component in the three different deployments. The next three rows represent the measurements for the Non-SA-Aware component in the same deployments.

Note that all the time measurements are in **seconds.**

3.9.1 Component failure measurements

As described before VMware HA does not detect component failures so the rows for VMware HA and VMware FT have no values. Also the Non-SA-Aware version of VLC does not support the repair of the failed component. We simulated the failure of the component by killing the VLC process in the OS and keeping the exact killing time for measurement purposes.

According to Table 3, we created a bar chart (Figure *26*) to show the differences in the outages. We selected outage because from the service availability point of view, this is the most important aspect.

	Reaction	Repair	Recovery	Outage
SA Aware - stand alone	0.009	0.136	0.046	0.055
SA Aware - VM	0.013	0.243	0.068	0.081
SA Aware - VM - HA enabled	0.013	0.243	0.068	0.081
non-SA Aware - stand alone	0.429		0.069	0.499
non-SA Aware - VM	0.489		0.103	0.592
non-SA Aware - VM - HA enabled	0.489		0.103	0.592
VMware FT				
VMware HA				

Table 3 - VLC component failure measurements

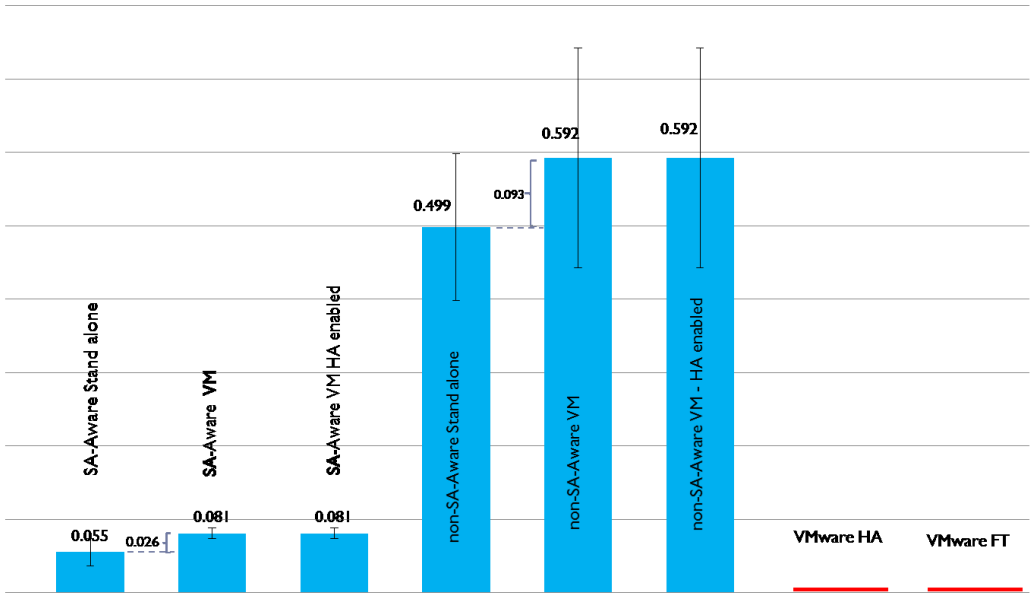

Figure 26 - Outage due to VLC component failure

The application and component failure detection and recovery is our focus throughout the thesis which VMware HA and VMware FT do not support on their own and this is the reason we used a baseline combined architecture.

3.9.2 Virtual machine failure measurements

This failure is only applicable where we have used VMs in the deployment. This type of failure is not applicable in OpenSAF on the stand-alone node architectures. The VM failure is simulated by killing the VM process on the hypervisor and recording the time of failure for the measurements. Table 4 shows the corresponding numerical values.

It should be noted that there is no repair when VMware HA is not enabled in VMware vSphere. This means that the VM is not restarted when a VM failure occurs and VMware HA is not enabled. VMware HA can detect VM failures and restarts the failed VM in the same host if the host is available or on another host in the cluster. As shown in the table

	Reaction	Repair	Recovery	Outage
SA Aware - stand alone				
SA Aware - VM	1.906		0.048	1.954
SA Aware - VM - HA enabled	1.906	107.9	0.048	1.954
non-SA Aware - stand alone				
non-SA Aware - VM	2.651		0.057	2.707
non-SA Aware - VM - HA enabled	2.651	104.2	0.057	2.707
VMware FT	0.024	10.63	1.29	1.314
VMware HA	72.166	27.166		99.332

Table 4 - Virtual Machine Failure measurements

when we have OpenSAF in the VM, the service is recovered much faster than just using the VMware HA. This is because when a VM fails, OpenSAF detects the failure of the cluster node and fails over the active assignment to the standby SU.

The VM failure measurements show that the VMware FT performs better in both outage

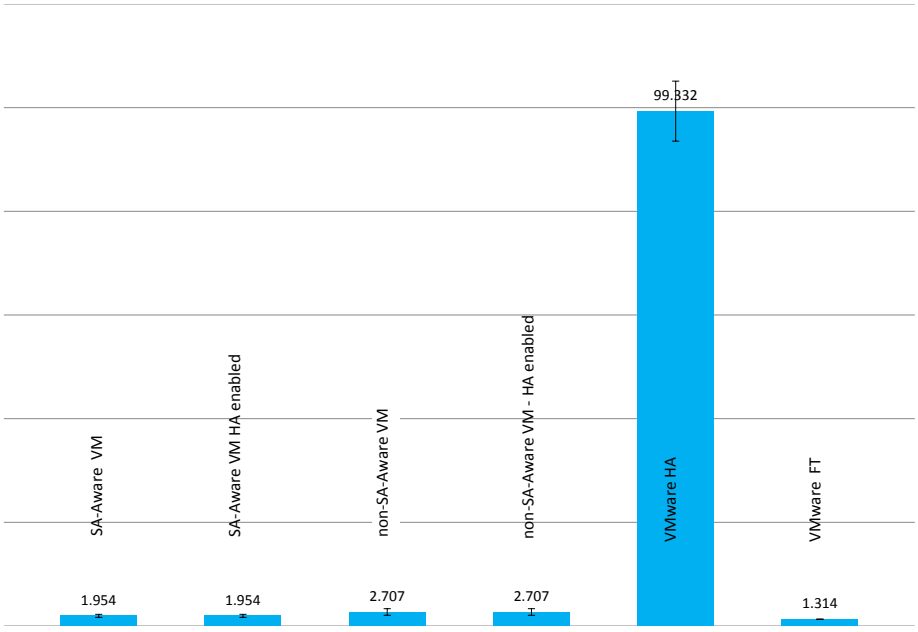

Figure 27 - Outage due to Virtual Machine failure

time and the repair time compared to other architectures.

Figure 27 shows the big difference between the VMware HA outage time and the other architectures. From high availability perspective it is a huge difference when we demand 99.999% availability, which is only about 5 minutes a year. So if our VMware HA solution fails more than 3 times a year we cannot call it highly available anymore.

3.9.3 Physical node failure measurements

Node failure is simulated by forcing the shutdown of the physical node. This was done through the command line interface of the architectures where we had Ubuntu installed directly on the physical node, or by using the SSH client command on the ESXi nodes.

As shown in Table 5, repair of the failed node is not supported in the OpenSAF based stand-alone architectures and VM based architectures without HA enabled. There is also a big difference in the outage time between VMware HA and the other architectures as shown in Figure 28.

Although we used the force shut down command, but still there was a delay from issuing

	Reaction	Repair	Recovery	outage
SA Aware - stand alone	2.046		0.019	2.065
SA Aware - VM	2.169		0.046	2.215
SA Aware - VM - HA enabled	2.169	105	0.046	2.215
non-SA Aware - stand alone	2.702		0.060	2.762
non-SA Aware - VM	2.874		0.051	2.926
non-SA Aware - VM - HA enabled	2.874	105	0.051	2.926
VMware FT	0.024	10.63	1.29	1.31
VMware HA	61.1	28.3		89.4

Table 5 - Node failure measurements

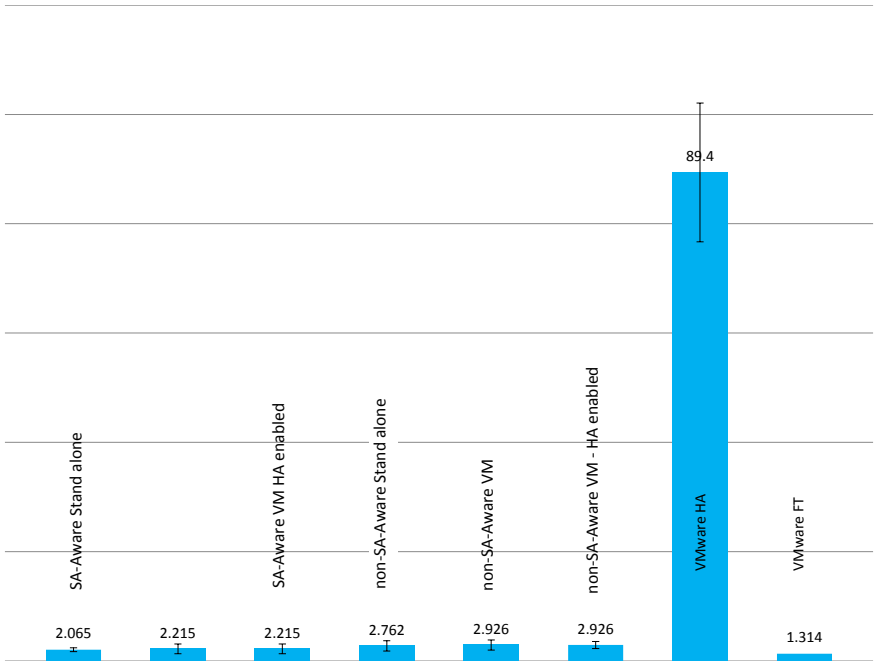

Figure 28 - Outage due to node failure

the command until the actual shutting down of the node. We could not measure this delay because we could not log the actual failure time. So the reaction and outage times of this measurement include this delay.

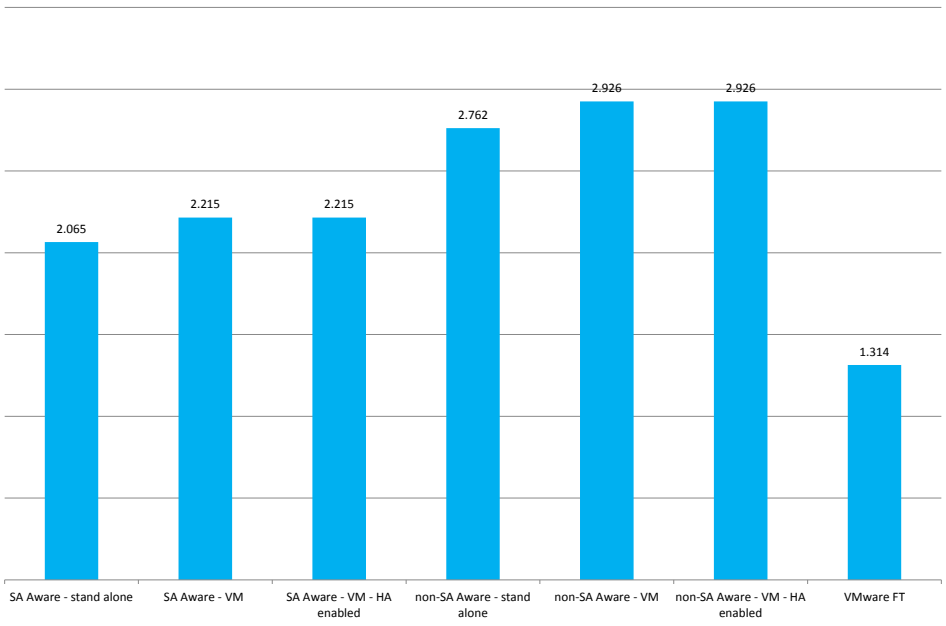

Figure 29 - Outage due to node failure without considering VMware HA

For a better comparison in the node failures, we created another chart, but this time without considering the VMware HA so that the differences of the other architectures can be more visible. The chart in Figure 29 shows that the VMware FT again has the least outage time comparing to other architectures.

3.9.4 Memory overheads

For measuring the memory overhead of different availability solutions, we recorded the total memory usage before and after enabling or running our availability solutions. For example in OpenSAF we started the middleware with the basic imm.xml file which contains the basic configuration of the nodes of the cluster. So we make sure that the memory consumption of the component would not be counted. In VMware we did the same experience with enabling and disabling VMware HA and calculating the difference. Since there are two types of nodes in VMware HA, we ran the experiment in both master and slave nodes.

As shown in Figure 30, the memory consumption of OpenSAF is approximately 15 MB

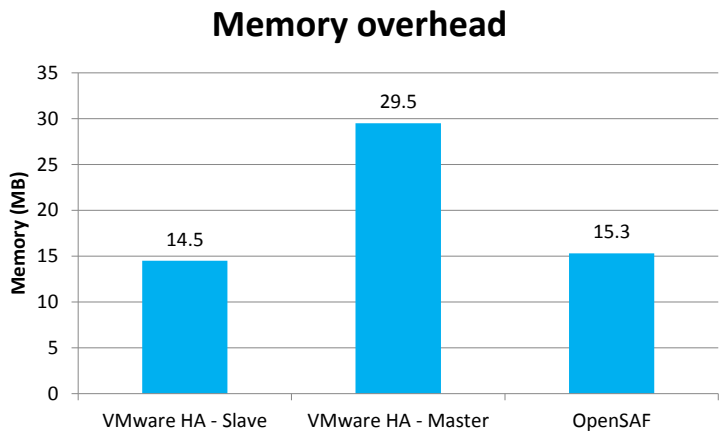

Figure 30 - Memory overhead (VMware FT is left out)

which is similar to the memory consumption of VMware HA in a slave node. The chart also shows that VMware HA in the master node uses almost twice the memory compared to the slave node in the same cluster. The overheads of the high availability mechanisms in the slave node of VMware HA and OpenSAF are close, but it should be considered that the hypervisor itself has a memory overhead which we did not measure.

The memory overhead in VMware FT is one of its weakest points. Since VMware FT runs a whole VM in parallel with the primary VM as the secondary VM, it doubles the memory consumption. It means if the VM is set to consume 1GB of memory, the overall consumption of the memory in the cluster for that VM is twice and is 2GB. There would be 1GB of overhead just for a single VM in this case. This amount of overhead is much higher than the memory overhead of the other solutions and that is why the VMware FT memory consumption is left out in Figure 30.

3.10 Analysis

After experimenting on the baseline architectures and trying different failure types, we learned a lot about the different solutions. The analysis on data can show us the advantages and disadvantages of each architecture. This analysis will lead to other combined architectures that we will introduce in the next chapter.

3.10.1 SA-Aware vs. Non-SA-Aware component

One of the clear differences in the charts (Figure 26 and Figure 29) is the difference of outage between SA-Aware and Non-SA-Aware components when the architecture is the same. This difference varies from 444 milliseconds to 711 milliseconds. The first reason of the difference is in the way the fault is detected. The SA-Aware components have a

very tight coupling with AMF; they link to the AMF library which results in faster failure detection. In the Non-SA-Aware component, there is only a passive monitor which is implemented to check the process ID of the VLC process in the OS. The second reason is related to the recovery, which is faster in the SA-Aware version. The standby SA-Aware VLC is pre-instantiated on the other node so that it only needs to take over the active assignment. For the Non-SA-Aware component, the instantiation is done when it is assigned as active.

3.10.2 Overhead due to the VM

The other variance in the charts is where we add a VM layer. For example in Figure 26, the SA-Aware component failure in the stand-alone physical machine has less outage than the same component failure in a VM. This applies to Non-SA-Aware component as well. It indicates a 15% to 30% overhead for the VM layer. Huber et al also report in [43] that the performance of the application running in VMs is worse compared to the application running directly on physical host. This supports the assumption that the delay we experienced is the result of the difference in the performance of a VM and a physical machine, and this is inevitable.

3.10.3 OpenSAF vs. VMware HA

The difference in service outage between the VMware HA and OpenSAF based deployments is quite obvious from Figure 27 and Figure 28. In VMware HA there is no standby and it can only restart the failed node on the same or another ESXi host. One of the important factors in VMware HA outage time is the way that it handles failures which is described in 2.2.4.1. The physical node failure detection and recovery times and

mechanisms are different in master and slave. One of the advantages of VMware HA over OpenSAF is the detection of network isolation which is covered in VMware HA. The datastore heartbeat can detect whether the host is unavailable because of the physical failure or network isolation. This mechanism is also one of the reasons that VMware HA recovery takes longer than OpenSAF related architectures take.

3.10.4 Repair of the failed element

One of the measurements that we did not cover in the previous charts is the repair time. The repair of the failed component/application is possible when we are using OpenSAF based architectures. The VM availability can be managed by VMware HA so in the architectures where VMware HA is enabled, the failed VM can be repaired. In the case of physical node failure, since VM images are placed on a shared storage, VMware HA can restart the VMs on the failed host, on other hosts.

Table 6 shows the differences in the repair capabilities of the different deployments. Accordingly only the combined architecture supports the repair of all failure types. The repair of a failed SA-Aware component on standalone and virtual nodes takes

	Repair of		
	Failed component	**Failed VM**	**Failed Node**
OpenSAF on Standalone machine	Yes	-	No
OpenSAF in VM	Yes	No	No
VMware HA	No	Yes	Yes(restarting the VM on another host)
OpenSAF in VM + HA	***Yes***	***Yes***	***Yes***(restarting the VM on another host)

Table 6 - Repair of the failed element in different deployments

respectively 0.136 and 0.243 seconds, while the repair of a VM or node takes around 100

seconds in the VMware HA enabled deployments. Repairing a failed element is important for high availability because during this time the service is not protected and there is a possibility of failure. Hence it would be nice to reduce the repair time as well. The table also indicates that the repair of the failed VM is not applicable in OpenSAF installed on a physical machine without virtualization.

3.10.5 Combining VMware FT and OpenSAF

Our case study application as described before uses a 2N redundancy model and for that we need two nodes (physical or virtual). On the other hand, VMware FT creates a secondary copy of a VM which runs in lockstep with the primary node. An architecture which combines the VMware FT and OpenSAF can be considered as enabling VMware FT for each of the VMs containing the OpenSAF nodes. In this combination OpenSAF takes care of the application failures and VMware FT does the recovery for VM and physical node failures. As experimented, VM/node failure does not trigger OpenSAF availability mechanism to react because the outage time of VMware FT is less than the timeout of the Transparent Inter-Process Communication (TIPC) protocol. TIPC is the protocol used in OpenSAF for the inter-cluster communications. The timeout of TIPC, which is a timer setting, causes OpenSAF to react as the node failure.

The advantage of this combination is lower repair and outage times for VM/node failures while taking advantage of fast application failure detection and recovery with OpenSAF.

As mentioned before, VMware FT has many hardware requirements and the limitation of using just one virtual CPU per each VM. The double usage of memory and CPU is

another VMware FT disadvantage. The two VMs with VMware FT enabled, will use as much as four VMs memory and CPU.

3.10.6 Fault propagation in VMware FT

We already mentioned that VMware FT has a secondary VM which uses lockstep to synchronize with the primary. This synchronization causes problems for VMware FT and the architectures or applications using VMware FT as their availability mechanism. The problem arises when the operating system, or application running in the VM crashes for any reason. In these cases, the failure of the operating system or failure of the application is propagated to the secondary VM.

To show this problem, we added a thread in the VLC code that halts the operating system after a random amount of time when the video started streaming. The thread waits between 10 to 20 seconds and then executes the 'halt' command which causes the operating system to halt. According to the hypervisor log, after the operating system inside the primary VM halts, there is an attempt to failover to the secondary VM "Waiting for ack from secondary" and then "Secondary didn't ack data in 8000 ms". The second log record mentioned here shows that the secondary VM is not available for failover. The reason is the propagation of the failure to the secondary VM. After the halt command in the primary, the same command is synchronized with the secondary VM and because it is running in sync with the primary VM, the secondary VM halts too. Fault propagation is one of the main problems of VMware FT.

3.10.7 Validity of the measurements

Throughout our experiments, one of our main concerns was the comparability of the architectures, measurements and the experimental environments. The CPU speed and amount of memory of the node play important roles in this context. The underlying hardware were the same in all of the experiments except for the experiments on VMware FT, which were done on the ÉcoloTIC nodes because of the hardware requirements of VMware FT. For minimizing the effect of hardware differences, we assigned the same amount of memory and the number of CPU cores to all VMs in all our experiments.

One of the key points that should be considered when comparing the virtual environment and the non-virtual environment is the inevitable overhead due to the virtualization layer, discussed in detail within section 3.10.2.

The repair in VMware HA is different from the repair in VMware FT. In VMware HA there is no backup VM so the repair is done by restarting a VM, which its image is located on a shared storage. But in VMware FT, instead of starting a VM from scratch, the runtime data of the new primary VM is copied to the new secondary VM. Usually data to be copied for runtime data is less so it will function faster. Reading and running a VM image on a shared storage or copying the runtime data of a VM, is done through the communication network. The bandwidth of the network infrastructure has a big impact on these network dependent operations. The links used between the hosts in our first experiments were 100 Mbps links and the links used for VMware FT experiments were gigabit Ethernet links. The faster link in VMware FT, causes a better performance in the startup of the VM and the synchronization of the primary VM with its secondary VM.

These links are used for connecting the hosts to each other and the hosts to the shared storage(s).

From the software environment perspective, all of our experiments were based on the Ubuntu 10.04 operating system. We used VLC version 1.1 and OpenSAF version 4.2.1 in all of our experiments. The virtualization environment was VMware vSphere 5.0 (VMware ESXi 5.0) in all of our experiments.

Like any other experimental research, our work is subject to validity threats. One of our concerns is the time synchronization between the nodes. While we used time synchronization protocols, given that our measurements are in the milliseconds range, our results might be affected by the inevitable small differences in system times among the nodes.

In calculating the outage time for VMware FT, we used an intrusive method. If a related log record was used, the outage time conflicted with the experiment in section 3.10.5. This conflict is the reason of selecting an intrusive way described in section 3.8 for measuring the outage time for VM failure in VMware FT.

3.10.8 Conclusion

According to the definition of high availability which allows for at most 5.26 minutes of down time per year, it is unlikely that the VMware HA solution can be considered highly available if the VM needs to be restarted at least four times a year, it would already use up this time budget. Furthermore, it cannot detect the application failures. On the other hand, VMware FT supports very limited hardware and it does not support VMs with multiple processors. According to our measurements, one can conclude that the VMware

HA alone cannot compete with the service high availability management provided by OpenSAF.

Our last architecture, which combined OpenSAF and VMware HA, has covered most of the shortcomings of each solution. For example by using OpenSAF on physical node we could not benefit from the advantages of the virtualization. Furthermore, VMware HA by its own does not cover the detection and recovery of the component failure.

Although this initial combination of OpenSAF and VMware HA was better than each solution by its own, it still suffered some problems. The first problem was the very long repair time of the failed VM. If the service provider (VM or component) fails again during those 100 seconds, there would be no service provider to fail over to. The second problem which is also the problem of VMware FT is, VMware HA and VMware FT are proprietary solutions and not all virtualization solution providers support VM availability management.

VMware FT in our measurements had lowest outage and repair time for VM and physical node failures. But VMware FT has its own disadvantages and problems. Considering that a secondary VM is created for each VM, the memory and CPU usages are doubled. The secondary VM is in continuous synchronization with the primary which consumes the network bandwidth as well. VMware FT does not cover any application failure support and in the case of OS or application failure, the failure is propagated to the secondary VM too. VMware FT also has many hardware requirements which might not necessarily be fulfilled with the hardware used in a cluster. As an instance our first test-bed cluster did not support these requirements. That being said, although VMware FT performs

better in some of the time measurements, it has many disadvantages. The combination of VMware FT and OpenSAF is not recommended for systems with many VMs because the duplication of memory and CPU will reduce the performance of the system. But in environments with limited number of VMs and the need for both application and VM failure detection/recovery support, this combination can be a good solution.

In the next chapter, we will propose two new architectures which address the problems from our previous architectures.

Chapter 4

New architectures combining OpenSAF and virtualization

Our previous virtualization based deployments used VMware HA to manage the availability of VMs. With respect to service outage we saw that it was outperformed by OpenSAF, but using OpenSAF alone in the VMs without VMware HA enabled, did not cover repair for the failed VMs. In addition not all virtualization solutions provide similar mechanism for managing the VM's life cycle. Hence, based on the results of our analysis, in this chapter we propose architectures, taking further advantage of the different solutions' strength and fix their weaknesses. For this we try to use tools and libraries available in other virtualization solutions as well, so that they can be used with other virtualization solutions. In these architectures the main goal is to manage the life cycle of the VM by OpenSAF to reduce not only the outage, but also the repair time in case of failures, while the services running in the VMs are protected by another OpenSAF cluster. The first architecture targets non-bare-metal hypervisors while the second one targets bare-metal hypervisors. Both of the proposed architectures include the same two

virtual machines running our case study application. The main difference is that we added a second OpenSAF cluster to manage the life cycle of these two VMs.

The contribution of these proposed architectures is improving the service availability in application level and improved repair time in the case of VM/node failures. The proposed architectures allow for the detection and recovery of failures at application, VM and physical node level.

This chapter also includes the experiments with the first proposed architecture, which has been deployed using VMware workstation. We measured the same defined metrics to see whether our new architecture improved the shortcomings of the previous combined architecture or not.

4.1 VM availability management in non-bare-metal hypervisor

Most of the hypervisor vendors provide APIs and CLI commands. They can be used to develop tools to control some of the functions the hypervisor provides like managing the VM life cycle. For example VMware provides the VIX APIs. Using these CLI commands and APIs we can start, stop, pause and resume the VMs in the hypervisor. Also, the non-bare-metal hypervisors (e.g. VMware Workstation) expose a process ID for each VM running in the host operating system. So, the main idea behind the first architecture is to make OpenSAF to use the hypervisor CLI commands to control the lifecycle of the VMs and to use OpenSAF passive monitoring to monitor the VM process in the operating system. The passive monitoring is started using the CLI script where OpenSAF uses to instantiate the VM component. Figure 31 illustrates this architecture.

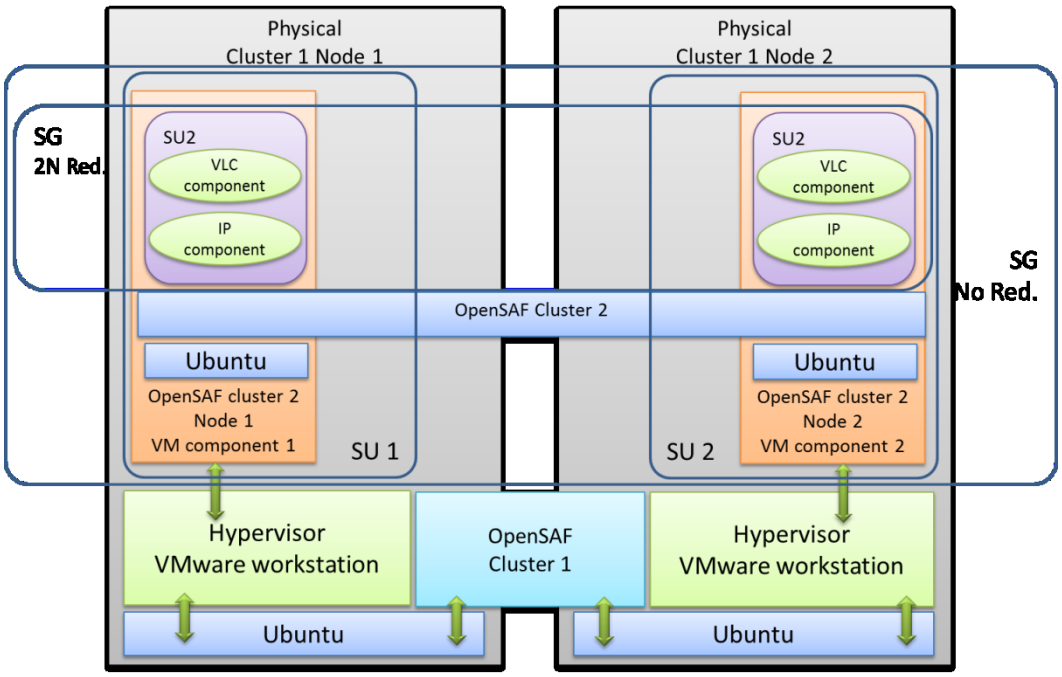

Figure 31 - VM availability management in non-bare-metal hypervisor

There are two different and independent OpenSAF clusters in this configuration. The VMs are considered as Non-SA-Aware non-proxied components in the first cluster. The started VMs form a second OpenSAF cluster for protecting the target application. When a VM fails, the VM process dies and OpenSAF of the first cluster detects the failure through the passive monitoring. When a failure is detected OpenSAF runs the cleanup CLI script associated with the VM component and if successful, it re-starts the VM. In the second cluster, OpenSAF detects the VM failure as a node failure, so it fails over any active assignment served by the VM to the standby.

From the redundancy model perspective, the SG in the first cluster where the components are the VMs, is configured with the No-Redundancy redundancy model and in the second cluster, which is the same configuration as we had in our previous architectures, it is configured with 2N redundancy model. The 2N redundancy model is the optimum model for our case study. The No-Redundancy redundancy model and N-Way active

redundancy models are not applicable because they do not support the standby assignments. Our components do not support x_active_and_y_standby capability model which is the requirement of N-Way redundancy model, so N-Way is not a good match either. 2N redundancy model is a special case of N+M where N and M are one, so N+M can be used too. The choice of No-Redundancy redundancy model for the first cluster is because the service provided from the second cluster where the 2N redundancy model is used, already has the redundant unit and the service will resume when there is a failure at any level.

This architecture can be configured with any number of VMs and any AMF configuration inside the VMs in the second cluster. Furthermore, since OpenSAF is using the CLI interface of the hypervisor, any hypervisor with the CLI support can be used. In this architecture we did not consider the migration of the VMs or OpenSAF failing over the VMs to other hosts. The VMs are stored on the local disk of the physical machines.

4.1.1 Experiments with VM availability management in non-bare-metal hypervisor

We deployed the VM availability management using VMware Workstation 9.1 as the non-bare-metal hypervisor and installed it on two of our test-bed cluster nodes running Ubuntu 10.04. We also installed OpenSAF 4.2.1 on the same nodes so that it could control the life cycle of the VMs running in the VMware Workstation on the nodes. Inside the VMs we installed the same Ubuntu 10.04 operating system along with OpenSAF 4.2.1 and the SA-Aware version of our case study application, the VLC media

player. We configured two different and independent OpenSAF clusters because of the easier manageability for the implementation, which were not aware of each other.

In this deployment, OpenSAF controls the life cycle of the VMs using the "vmrun" command included in the VIX API libraries [44]. We used the start and stop commands respectively for instantiating and terminating a VM, which is configured with these CLC-CLI commands as a Non-SA-Aware non-proxied component. When the cluster on the physical nodes is started, OpenSAF starts the VMs and when the VMs are up, OpenSAF of the second cluster inside the VMs starts the VLC components and assigns them the active and standby states. The CSI in the first cluster is configured with the address of the images of the VMs.

In this setup we experimented with component failure as described earlier by killing the active VLC component as well as with VM failure. In the latter case we killed the process of the VM which hosted the active VLC component. This was perceived as a component failure in the first cluster and resulted in the restart of the failed VM. On the other hand, in the second cluster it was detected as a node (VM) failure and OpenSAF failed over the service to the standby VLC component, which resumed the service. Once the failed node was restarted, AMF assigned the standby assignment to its VLC component.

We took the same measurements as we did in the baseline architectures.

Table 7 shows that the VM repair time is reduced drastically from more than 100 seconds with VMware HA to 3.73 seconds. This table shows that the delay is mainly because of the reaction time. Since the reaction time in VM failure is defined as the time when the VM process is killed until the time when the first CLC-CLI command is called on the

	Reaction	Repair	Recovery	Outage
Bare-metal hypervisor (ESXi) Only HA enabled	72.166	27.166		99.332
VMware FT	0.024	10.63	1.29	1.314
OpenSAF with VMware HA managing the VMs	1.905	107.90	0.047	1.953
OpenSAF and VMware workstation (non-bare-metal hypervisor) VMs managed by OpenSAF	3.449	3.73	0.056	3.505

Table 7 - Comparison of measurements for VM failure in different architectures

other node, the delay because of the underlying host operating system can be an important reason. Note that without OpenSAF it takes VMware HA about 60 seconds to detect the failure and 30 to repair the VM, while with OpenSAF, it is OpenSAF which reacts to the failure and hence reduces the outage. When OpenSAF performs the VM restart as well, the repair time is improved considerably. In this architecture we had tremendous improvement in the repair time in comparison to the previous combined architecture.

Reviewing the results from VMware FT, the outage time with VMware FT is less than the outage time of our new deployed architecture. The recovery and repair of the new deployment takes less time than VMware FT. Our new deployment suffers from the long reaction time. As mentioned in the previous paragraph, one of the reasons can be the delays caused by the underlying host operating system. In similar experiments with ESXi where we had the VM with OpenSAF installed and the same case study application settings, the reaction time was 1.905 seconds as shown in Table 7. So we can assume that using the bare-metal hypervisor can potentially reduce the reaction time as low as two seconds and ultimately cause the reduction of the outage time.

	Reaction	Repair	Recovery	Outage
VMware HA	Not covered			
VMware FT	Not covered			
OpenSAF with No virtualization	0.009	0.136	0.046	0.055
OpenSAF with VMware HA managing the VMs	0.013	0.243	0.068	0.081
OpenSAF and VMware workstation (non-bare-metal hypervisor) VMs managed by OpenSAF	0.012	0.848	0.580	0.592

Table 8 - Comparison of measurements for failure of the SA-Aware VLC component in different architectures

Table 8 shows the measurements of the failure of the SA-Aware VLC media player in the different architectures. In this table the same issue of the increased outage time because of the non-bare-metal architecture is visible. The next architecture addresses this issue by proposing a new architecture, using the ESXi as the bare-metal hypervisor.

The drawback of this architecture is the increased service outage which we attribute to the additional layer of the host operating system and also the difference between the hypervisors used (ESXi vs. Workstation). We assume that if we could use ESXi, the delay introduced in this deployment would disappear and we would have the same functionality and better repair time. Hence, we came up with the next architecture.

4.2 VM availability management in bare-metal hypervisor

In this architecture we have the same two VM images on a shared storage and the two ESXi hypervisors as we had in our baseline architecture of OpenSAF in virtual nodes (see Figure 18). The VMs provide the video streaming service so we will call them "Service VMs".

The previous combined architecture suffered longer delay in service outage for component failure than our baseline architectures. In this architecture we created two new VMs which we call “Manager VMs”, which run OpenSAF that manages the following configuration: We defined two SIs each with a CSI which corresponds to the one of the ”Service VMs”. So there are four components configured in two different SGs with the 2N redundancy model where each SG holds the active and standby components of one of the “Service VMs”. We selected the 2N redundancy model because if the hypervisor running the VM fails, its standby component can restart the VM on the other node instead of waiting for the hypervisor to be available again before starting the VM. This improves the repair time of the VM in the failed hypervisor.

Each component resides in a different SU meaning we have four SUs (see Figure 32). The SUs are ranked so that the SI assignments are distributed evenly between the two

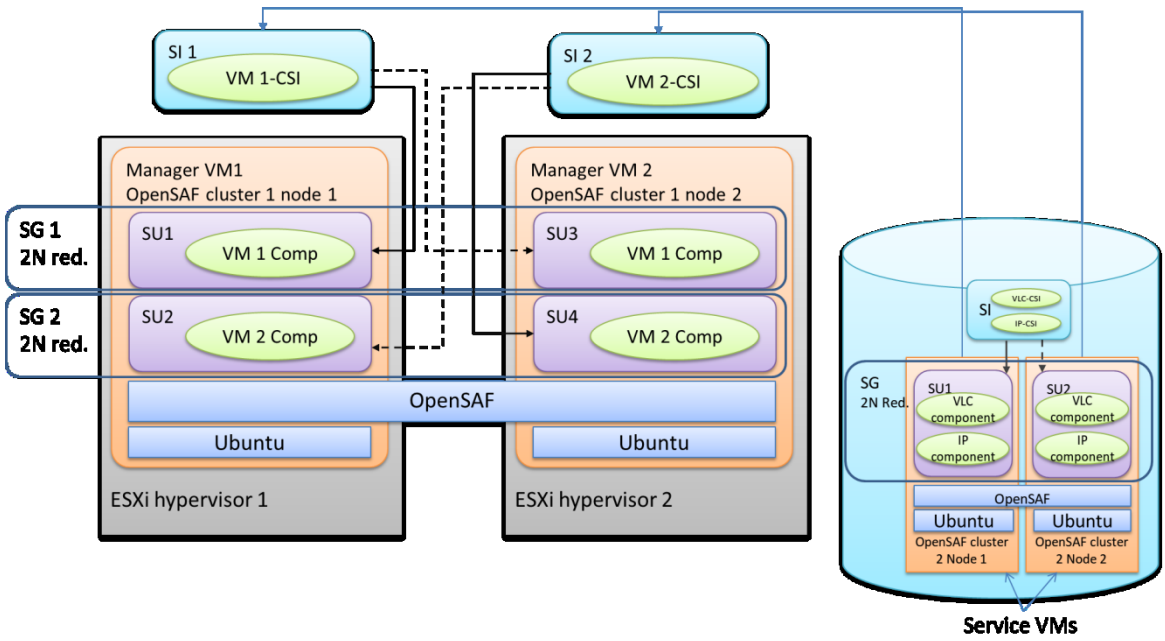

Figure 32 - VM availability management in bare-metal hypervisor

nodes. In AMF, a service unit with higher ranking standby for that service instance is preferred over a service unit with a lower ranking. We propose using ranking in our architecture so that the active assignments are balanced between the nodes. By setting the auto adjust attribute of the SG, the SI assignments to the service units in the service group are transferred back to the most preferred SI assignments in which the highest-ranked available service units are assigned. The components are sets of scripts for starting and stopping the "Service VMs" using the "virsh" command of the libvirt library. Hence, they are non-SA-aware-non-proxied components. The health of the "Service VMs" is checked by external active monitors running in the "Manager VM"s started with the VMs. They use libvirt library to get the health status of the VM. The "Manager VMs" form a separate cluster from the "Service VMs". In this architecture VMs are associated with CSIs so a VM failure, is a failure of the active SU containing the VM component. For the components we configure the component restart than fail-over in the event of failures which reduces the need for potential switch overs because of auto-adjusting. It means if just the VM fails VM is restarted on the same node instead of failing over to the other node. A possible runtime arrangement is shown in Figure 32. Manager VM1 contains SU1 with VM1 component having the active assignment for Service VM1 and SU2 with VM2 component having the standby assignment for Service VM2. Manager VM2 contains the SU3 with VM1 component with standby assignment for Service VM1 and SU4 with VM2 component having the active assignment for Service VM2. At the same time the VLC component in Service VM1 has the active assignment and the one in Service VM2 the standby. The Service VMs are started in the hypervisor, which also runs the Manager VMs.

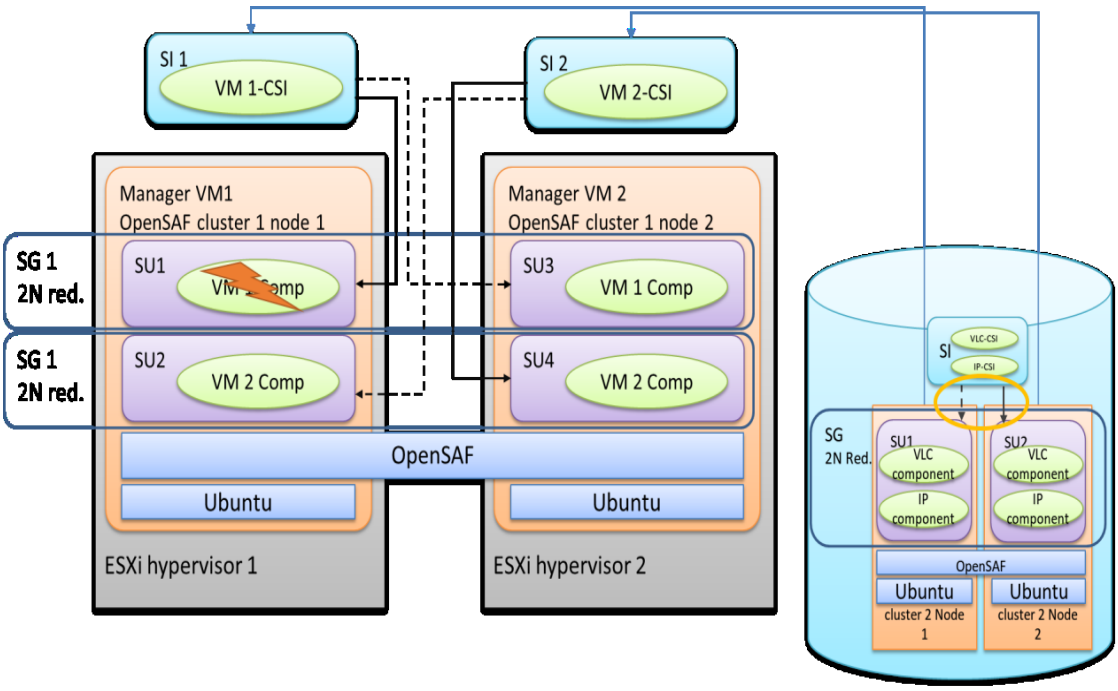

Figure 33 - VM failure in the proposed bare-metal architecture

Consider a scenario that the Service VM1 dies (Figure 33). The external active monitor reports it to AMF, where AMF first clean-ups the VM1 component in SU1 by calling its clean-up script, and then tries to restart the failed component. The same component gets restarted and instantiates the Service VM1 again by calling its instantiation script. Meanwhile since Service VM1 hosted the active VLC component (Figure 32) for the streaming service it is failed over to the VLC component in Service VM2 because the second OpenSAF cluster detects the node failure; so the service continues.

Figure 34 shows this scenario and illustrates the sequence of actions after the VM1 fails. The arrow pointing down shows the service.

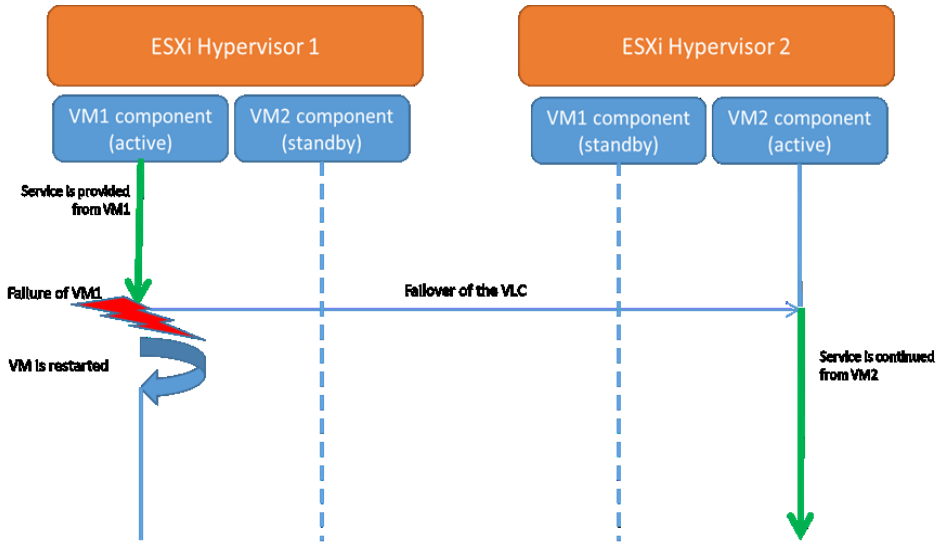

Figure 34 - Sequence of actions for VM failure in the bare-metal based architecture

Consider another scenario where the node ESXi hypervisor 1 node fails (Figure 35). In this case, AMF fails over its CSI to the standby VM in SU 3 which causes the restart of the failed VM on ESXi hypervisor 2. As a result both Service VMs are hosted in ESXi hypervisor 2. Once VM1 component becomes available OpenSAF will switch back the SI representing Service VM1 to its preferred node. This way we avoid the situation that one

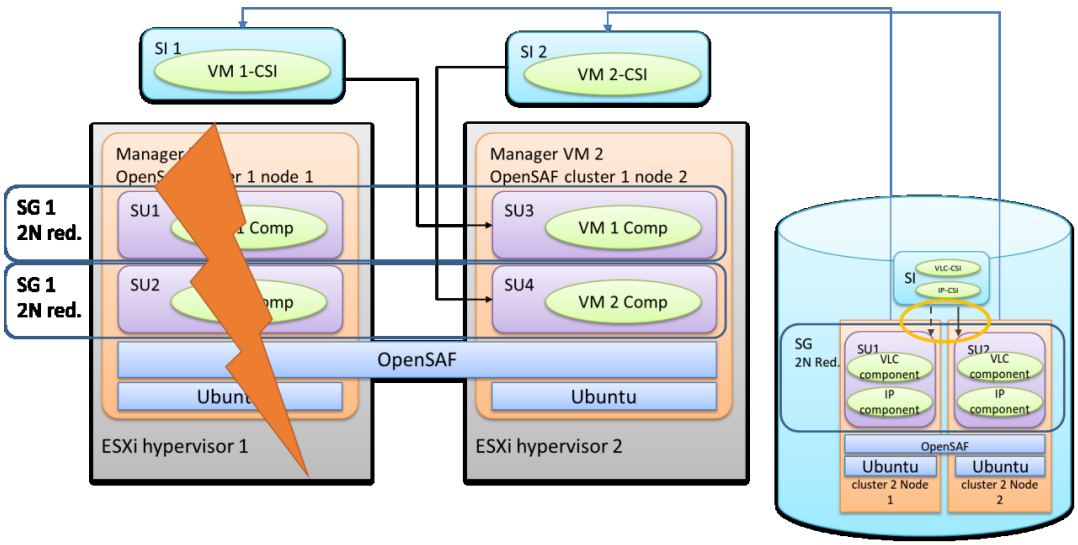

Figure 35 - Node failure in the proposed bare-metal architecture

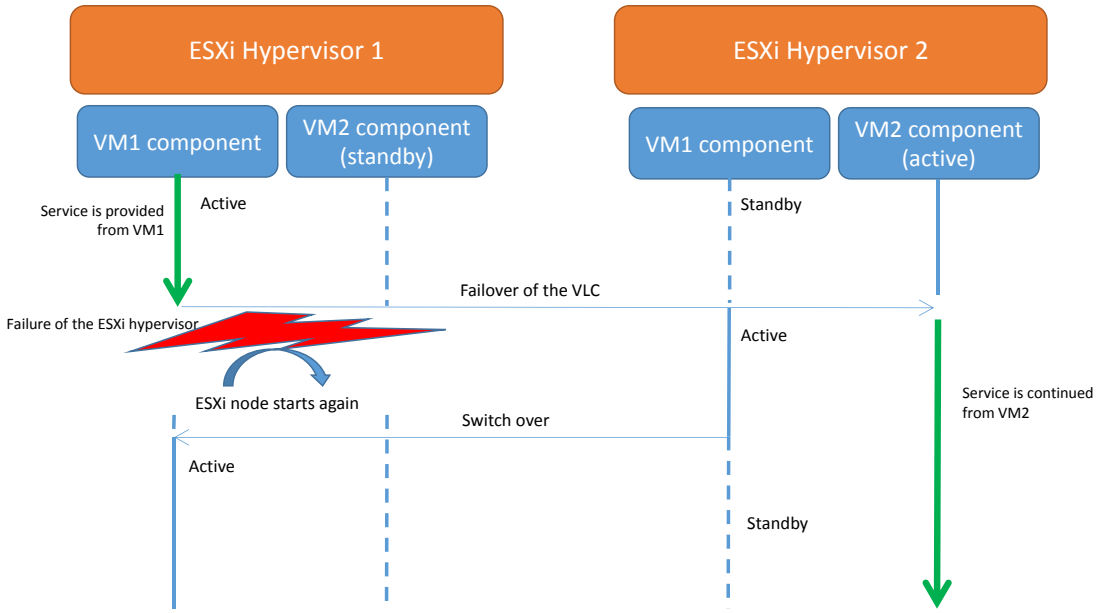

Figure 36 - Sequence of actions for physical node failure in the bare-metal based architecture

ESXi node holds both Service VMs for extended period of time. The balancing of the VMs on the VMware HA is not supported. Figure 36 shows the sequence of actions.

An added benefit of this architecture is that it is not limited to VMware ESXi and can be deployed in the hypervisors which support libvirt library like Linux KVM, Xen, etc.

In this architecture we expect that the long delays for reaction time in the VM failure for the previous architecture to be fixed. In Table 9, the last row shows our expectations from this architecture compared to other architectures. The service VMs are the same as the VMs in the first combined architecture (OpenSAF with VMware HA managing the VMs). In the case of component or VM failures, it is the OpenSAF cluster in the service VMs which takes care of the service availability and failover the service to the standby component. So that is why for the outage of component and VM failures we estimate the same time as the first combined architecture. But repair of the VM in the VM failure is done by the OpenSAF cluster in the manager VMs, so we estimated its repair time

according to the first proposal in this chapter for non-bare-metal hypervisor which is deployed.

Comparing the estimated values with our previous architectures shows that the new proposed architecture not only covers the application failure detection and recovery but also has a lower outage and repair time than most of other architectures. The only case which the new proposed architecture does not have the lowest value is the outage time in VM failure. Although VMware FT has lower outage time in that case, it does not support application failures which is one of our key motivations. It also has many disadvantages which was covered in 3.10.8.

4.3 Conclusion

In this chapter, we introduced two new architectures for bare-metal and non-bare-metal hypervisors. The aim of these architectures is to focus on service high availability along with enhancing the repair time of failed VM by managing the life cycle of the VMs using OpenSAF. Deployment of the first proposed architecture showed that the concept of managing the life cycle of VMs using OpenSAF is valid and the repair time enhanced

* Time are in seconds	Outage		Repair	
	Component failure	VM failure	Component failure	VM failure
VMware FT	--	1.314	--	10.63
VMware HA	--	99.332	--	27.166
OpenSAF with VMware HA managing the VMs	0.081	1.953	0.243	100
OpenSAF and VMware workstation (non-bare-metal hypervisor) VMs managed by OpenSAF	0.592	3.505	0.848	3.73
OpenSAF and ESXi (bare-metal hypervisor) VMs managed by OpenSAF	~0.1	~2	~0.250	~3

Table 9 - Timing expectations from our bare-metal hypervisor based proposal

greatly compared to our baseline combined architecture. Comparing the results also showed that the outage in VMware FT for VM and node failures is less than our first proposed architecture. Assuming the delay was caused by the non-bare-metal hypervisor used, the second proposed architecture aims at fixing this shortcoming by using a bare-metal hypervisor. One of the limitation we faced in proposing the bare-metal approach was that the bare-metal hypervisor does not have an operating system so we could not have OpenSAF directly on the hypervisor. This is the reason we considered two other VMs as the manager VMs. Also for the same reason we could not use the hypervisor as a container, to contain the VMs.

Because virtualization is one of the key components in cloud computing, this study potentially opens the doors for improving high availability in cloud computing by using OpenSAF in virtualized environments. Supporting application failure detection and recovery is the key advantage of our approach compared to other existing solutions.

From the scalability perspective, both proposed architectures are limited by the scalability of VMware and OpenSAF.

One of the shortcomings of our second proposed architecture is that we do not consider the availability of the manager VMs. This means if the manager VM fails there is no mechanism defined for repairing it. This does not cause any problem as long as the second manager VM continues working. As a solution we can use the virtualization availability mechanisms like VMware HA to manage the availability of the manager VMs.

Chapter 5

Conclusion

The overall goal of this thesis was to find a solution for availability in virtual environments focusing on application failures. As mentioned before, virtual environments are becoming more popular particularly in cloud services where the availability of the provided services is important. The aim of this work was to evaluate OpenSAF high availability solution and VMware virtualization considering VMware HA and VMware FT. We also compared the different solutions and architectures and proposed new architectures taking advantage of the strength of experimented solutions and architectures. VMware HA which was the focus of our study is a VMware solution for managing the availability of the virtual machines. We started the experiments on four baseline architectures to study their behaviors qualitatively. The metrics were the other form of investigating and studying these architectures. The measurements were conducted based on the defined metrics. The results from the measurements and specifically the analysis of the results helped us to recognize the weaknesses and strength of those architectures.

Our initial experiments showed that VMware HA was not as responsive to failures as OpenSAF, which is essential for service high availability. Also, the two solutions handle different sets of failures.

In our investigations we used an initial combination of these solutions which benefited from the advantages of both solutions. This first combined architecture had some shortcomings i.e. the long repair time of the failed unit and being dependent on the virtualization product which was VMware HA. To combine their features, take advantage of their strengths and to fix these problems, we devised two new architectures that used non-bare-metal and bare-metal hypervisors. In these architectures we used OpenSAF for two purposes: to manage the availability of the VMs and to make the service running in the VMs highly available.

We implemented the first proposed architecture and had some experiments on it. The results showed some improvement in comparison to the baseline architectures, but other characteristics suffered from using the non-bare-metal hypervisor. Believing that the shortcomings of the first proposed architectures can be fixed using the bare-metal hypervisor, we proposed the second architecture. To make these architectures more open to other virtualization solutions we proposed the use of the libvirt library which is supported in some other virtualization solutions like VMware, XEN and Linux KVM.

The biggest advantage of our proposal is covering the application failure. This type of failure is not supported in other related architectures. Our work also takes advantage of the fast failure detection of OpenSAF which implements one of the most advanced high availability solutions. High availability provided by OpenSAF plus the presence of

virtualization in our proposed architectures can be a very good fit for the next high availability solutions in the cloud.

Beside the advantages, our research also has limitations. Our proposed architectures rely on the commands which VMware supports. These commands are also supported by some other virtualization solutions but not all. So these proposals cannot be used in the hypervisors which do not support libvirt library or vmrun command like 'Virtual PC' from Microsoft.

5.1 Future works

From our two proposed architectures we deployed and experimented with the first architecture which was based on non-bare-metal hypervisors. As the next step, the second proposed architecture can be deployed and experimented using the defined metrics. The measurements of our bare-metal hypervisor solution will result in a better comparison with the other architectures.

Moreover, as a future work, the proposed architectures especially the second one can be generalized to adapt to the cloud. This generalization should include the study of scalability of the work. Scalability can be discussed in two directions, the provided services and the number of physical machines/VMs. In the 2N redundancy model, the number of machines/VMs is always two, one can investigate the scalability in terms of provided services. As for the No-redundancy redundancy model, one can investigate variation in terms of services and machines/VMs.

Other redundancy models, if required, can be investigated for scalability in a similar manner. That being said, the overall scalability of the system will certainly be dependent of VMware and OpenSAF scalabilities.

References

[1] M. Toeroe and T. Francis, *Service Availability: Principles and Practice*. Wiley, 2012.

[2] Service Availability™ Forum, "Official homepage." [Online]. Available: http://www.saforum.org.

[3] Service Availability™ Forum, "Service Availability Interface, Overview, SAI-Overview-B.05.02."

[4] Service Availability™ Forum, "Application Interface Specification, Availability Management Framework, SAI-AIS-AMF-B.04.01."

[5] The Open Service availability Framework, "Official homepage." [Online]. Available: http://www.opensaf.org.

[6] VMware Inc., "Virtualization Overview," 2006. [Online]. Available: http://www.vmware.com/pdf/virtualization.pdf.

[7] J. Laprie, B. Randell, C. Landwehr, and S. Member, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[8] Service Availability™ Forum, "Hardware Platform Interface, SAI-HPI-B.03.02," 2009.

[9] Service Availability™ Forum, "Application Interface Specification, Software Management Framework SAI-AIS-SMF-A.01.02," pp. 1–174.

[10] Service Availability™ Forum, "Application Interface Specification, Information Model Management Service SAI-AIS-IMM-A.03.01," pp. 1–202.

[11] Service Availability™ Forum, "Platform Management Service, SAI-AIS-PLM-A.01.02."

[12] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A Survey on Concepts, Taxonomy and Associated Security Issues," *2010 Second International Conference on Computer and Network Technology*, pp. 222–226, 2010.

[13] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, "Virtualization for High-Performance Computing," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 2, pp. 8–11, 2006.

[14] VMWare Inc., “VMware vSphere hypervisor (ESXi).” [Online]. Available: http://www.vmware.com/products/vsphere-hypervisor.

[15] “Linux KVM.” [Online]. Available: http://www.linux-kvm.org/.

[16] Microsoft, “Microsoft Hyper-V Server.” [Online]. Available: http://www.microsoft.com/hyper-v-server.

[17] Linux Foundation Collaborative Projects, “The Xen project homepage,” 2013. [Online]. Available: http://www.xenproject.org/.

[18] VMWare Inc., “VMware Workstation.” [Online]. Available: http://www.vmware.com/products/workstation/.

[19] Oracle, “VirtualBox.” [Online]. Available: https://www.virtualbox.org/.

[20] VMWare Inc., “Official homepage.” [Online]. Available: http://www.vmware.com.

[21] VMWare Inc., “VMware High Availability.” [Online]. Available: http://www.vmware.com/files/pdf/VMware-High-Availability-DS-EN.pdf.

[22] VMWare Inc., “VMware Fault Tolerance.” [Online]. Available: http://www.vmware.com/files/pdf/VMware-Fault-Tolerance-FT-DS-EN.pdf.

[23] D. Epping and F. Denneman, *VMware vSphere 5 Clustering technical deepdive*. 2011.

[24] D. J. Scales, M. Nelson, G. Venkitachalam, and J. Daniel, “The Design of a The Design of a Practical System for Practical Virtual System for Machines Fault-Tolerant Virtual Machines,” *SIGOPS Operating Systems Review*, vol. 44, no. 4, pp. 30–39, 2010.

[25] E. Farr, R. Harper, L. Spainhower, and J. Xenidis, “A Case for High Availability in a Virtualized Environment (HAVEN),” *2008 Third International Conference on Availability, Reliability and Security*, pp. 675–682, Mar. 2008.

[26] DMTF, “Virtual System Profile - DSP1057- version 1.0.0,” 2010. [Online]. Available: http://www.dmtf.org/sites/default/files/standards/documents/DSP1057_1.0.0_0.pdf .

[27] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, “Remus : High Availability via Asynchronous Virtual Machine Replication,” in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, 2008.

[28] J. Dongarra, “Algorithm-Based Fault Tolerance for Fail-Stop Failures,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 12, pp. 1628–1641, Dec. 2008.

[29] P.-Y. Chung, Y. Huang, D. Liang, C.-Y. Shih, and S. Yajnik, “Method and apparatus for providing failure detection and recovery with predetermined replication style for distributed applications in a network,” US6266781 B1.

[30] D. Singh, J. Singh, and A. Chhabra, “High Availability of Clouds: Failover Strategies for Cloud Computing Using Integrated Checkpointing Algorithms,” *2012 International Conference on Communication Systems and Network Technologies*, pp. 698–703, May 2012.

[31] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, and P. F. Chan, “Leveraging virtualization to optimize high-availability system configurations,” *IBM Systems Journal*, vol. 47, no. 4, pp. 591–604, 2008.

[32] E. Braastad, “Management of high availability services using virtualization,” University of Oslo, 2006.

[33] Microsoft, “Windows Azure, Microsoft’s cloud platform.” [Online]. Available: http://www.windowsazure.com/.

[34] “PowerEdge 1950 Server Product Details.” [Online]. Available: http://www.dell.com/us/dfb/p/poweredge-1950/pd.

[35] “Summit X450a Series,” *Extreme Networks*. [Online]. Available: http://www.extremenetworks.com/products/summit-x450a.aspx.

[36] “VideoLAN - Official page for VLC media player, the Open Source video framework.” [Online]. Available: http://www.videolan.org/vlc/.

[37] A. Kanso, F. Khendek, A. Mishra, and M. Toeroe, “Integrating Legacy Applications for High Availability : a Case Study,” in *High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on*, 2011, pp. 83–90.

[38] “Projet mobilisateur en ÉcoloTIC.” [Online]. Available: http://equationtic.com/.

[39] “Symantec ApplicationHA.” [Online]. Available: http://www.symantec.com/application-ha.

[40] “NTP: The Network Time Protocol.” [Online]. Available: http://www.ntp.org/.

[41] “Precision Time Protocol daemon (PTPd).” [Online]. Available: http://ptpd.sourceforge.net/.

[42] IEEE, "IEEE SA 1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems." [Online]. Available: http://standards.ieee.org/findstds/standard/1588-2008.html.

[43] N. Huber, M. Von Quast, M. Hauck, and S. Kounev, "Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments," in *International Conference on Cloud Computing and Service Science (CLOSER 2011)*, 2011.

[44] VMware Inc., "Using vmrun to Control Virtual Machines," 2009. [Online]. Available: www.vmware.com/pdf/vix180_vmrun_command.pdf.