Design and Deployment of AMF Configurations in the Cloud

Pradheba Chakrapani Rangarajan

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Electrical and Computer Engineering) at Concordia University

Montreal, Quebec, Canada

April, 2017

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Pradheba Chakrapani Rangarajan

Entitled: Design and Deployment of AMF Configurations in the Cloud

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

Master of Applied Science (Electrical and Computer Engineering)

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

Signed by the final examining committee:

_____ Chair
Dr. R. Raut

_____ External Examiner
Dr. R. Glitho

_____ Internal Examiner
Dr. A. Agarwal

_____ Co-Supervisor
Dr. F. Khendek

_____ Co-Supervisor
Dr. M. Toeroe

Approved by: _____
Dr. W.E. Lynch, Chair
Department of Electrical and Computer Engineering

_____ 2017___ _____
Dr. Amir Asif, Dean,
Faculty of Engineering and Computer Science

Abstract

Design and Deployment of AMF Configurations in the Cloud

Pradheba Chakrapani Rangarajan

With the ever growing popularity of cloud computing, the trend of deploying applications in the cloud is increasing more than ever. Cloud offers computing resources that can be provisioned as required and scaled according to the workload demand. This feature attracts service providers to deploy their applications in the cloud. As users continue to rely more on the services provided by these applications, it is essential to keep the applications running with minimal service outage. Service Availability Forum (SA Forum) has defined a framework called Availability Management Framework (AMF) which can be used to manage service availability. AMF is agnostic to the services provided by the applications. However, it manages the service availability of applications by orchestrating the redundant entities through a configuration called AMF configuration. The design of AMF configurations for a physical cluster based on the functional and non-functional requirements, such as minimum level of service availability, has been proposed in the literature. In these solutions, the number of physical hosts required to deploy an application is given as input and the resource utilization is not taken into consideration. However, for deploying applications in the cloud the number of physical hosts is not fixed and should vary depending on the workload. Therefore, the issue of minimizing the number of physical hosts while meeting the requested level of service availability arises. In particular, the service availability depends not only on the entities involved in providing the service but also on the interferences caused by the collocation of entities. To minimize these interferences, the collocated entities can be grouped into fault isolation units such as VMs. This in turn may increase the number of resources required.

In this thesis, an approach to generate AMF configuration for the cloud is proposed. In this approach, a novel method is used to calculate the number of AMF entities that meets the availability and resource utilization requirements. In addition, a method to estimate service availability is proposed. It aims to predict the availability of service by considering the potential factors that affect availability, including the interferences due to collocation. Furthermore, an approach to deploy AMF applications in the cloud is proposed. As a proof of concept, a prototype that demonstrates the generation and deployment of AMF configurations in an OpenStack cloud has been developed. This prototype includes the existing Monitoring and Elasticity Engine, previously developed in the MAGIC project.

Acknowledgements

Foremost, I would like to express my heartfelt thanks to my supervisor Dr. Ferhat Khendek for providing me the opportunity to do academic research under his supervision. His expertise, encouragement and continuous support helped me throughout the learning process of this thesis.

I would like to offer my profound gratitude to Dr. Maria Toeroe for her insightful comments, remarks and immense knowledge which steered me in the right direction during all time of the research.

I would like to acknowledge all my MAGIC colleagues for their friendship and assistance. Especially I would like to thank Mehran Khan for helping me during the final stages of this thesis.

I must also thank my family and friends for all their endless love and faithful support. I am especially grateful to my parents who encouraged me in all of my pursuits and inspiring me to follow my dreams.

This work is partially supported by Natural Sciences and Engineering Research Council of Canada (NSERC), Ericsson Research and Concordia University.

Table of Contents

List of Figures

List of Tables

List of abbreviations

Act cap per CST	Active capability of components per CST
Act proportion	Proportion of active SUs in a SG
AGM	Available guest memory
APM	Available physical memory
CF	Clean up failure
CFIWD	Clean up action failing while attempting to instantiate a component with delay
CFIWOD	Clean up action failing while attempting to instantiate a component without delay
Clt	Time required to perform clean up action for a component
CR	Configuration requirements
CSS	Time required to set HA assignment for a component
ct	Component type
EA	Estimated availability
ETF	Entity type file
GOSM	Memory used by guest OS from infrastructure file
HA	High availability
HM	Memory used by hypervisor from infrastructure file
HOSM	Memory used by host OS from infrastructure file
IF	Instantiation failure
K	Minimum number of redundant entities
Max	Maximum number of SIs per VM
Max no of comps per CT	Maximum number of components per CT based on the SU type
Max no of SIs per act SU	Maximum number of active SIs per SU
Max no of SIs per SG	Maximum number of SIs per SG
Max no of SIs per std SU	Maximum number of standby SIs per SU
Max no of VMs per PH	Maximum number of VMs per physical host based on capacity or SIs
Memory required per CSI	Memory required by one component to provide a CSI
Memory required per CST	Memory required by collaborating components to provide all CSIs of a CST

Memory required per SI	Memory required by all collaborating components to provide an active SI assignment
Min	Minimum number of SIs per VM
Min no of comps	Minimum number of components required per component type determined using [9]
MTTF	Mean time to failure
MTTR	Mean time to repair
NFV	Network function virtualization
NFVI	Network function virtualization infrastructure
NFV-MANO	Network function virtualization Management and Orchestration
nia	Number of instantiation attempts
NIWD	Number of instantiation attempts with delay
NIWOD	Number of instantiation attempts without delay
No of act SUs	Number of active SUs
No of coll comp per CT	Number of collocated components per component type in a SU
No of coll SUs	Number of collocated SUs in a VM
No of coll VMs	Number of collocated VMs in a PH
No of comps per CT	Number of components per CT in a SU
No of CSIs per CST	Number of CSIs per CST in a SI
No of PHs	Number of physical hosts
No of SGs	Number of SGs
No of SIs	Number of SIs from CR
No of SIs per PH	Number of SIs per physical host
No of SIs per SU	Number of SIs provided by a SU
No of SIs per VM	Number of SIs provided by a SUs hosted in a VM
No of std SUs	Number of standby SUs
No of SUs per SG	Number of SUs per SG
No of SUs per VM	Number of SUs hosted per VM
No of VM grps	Number of VM groups
No of VMs	Total number of VMs to deploy all SIs
No of VMs per PH	Number of VMs per physical host based on the capacity of a physical host
NS	Network service
NST	Node shut down time

OBF	Overbooking factor from the infrastructure provider
PCNS	Probability of clean up not successful
PCS	Probability of clean up successful
PINS	Probability of instantiation not successful without delay
PINSD	Probability of instantiation not successful with delay
PH	Physical host
RA	Requested availability from CR
SA	Service Availability
Set of actual rec of CTs	A set having actual recovery of components of a SU, determined using [11]
Set of SGs	A set having SGs determined for each CT
SOT	Switch over time
std cap per CST	Standby capability of components per CST
Std proportion	Proportion of standby SUs in a SG
SUT	Service unit type
TGM	Total guest memory
TPM	Total physical host's memory from infrastructure file
VM	Virtual machine
VNF	Virtualized network function
VNFC	Virtualized network function component
VNFCI	Virtualized network function component instance

Chapter 1

Introduction

In this chapter we will briefly introduce high availability and service continuity. We will also explain the motivations behind the thesis, its contributions and finally, its organization.

1.1 High Availability and Service Continuity

Service availability (SA) is defined as the percentage of time a system is ready to provide its service [1]. Highly available systems are those that can achieve at least 99.999% of service availability [1]. High availability (HA) is an essential and critical requirement for computer based systems that are expected to provide the service around the clock.

Cloud computing is a new and popular paradigm where compute, network and storage resources can be rented and managed in an on-demand fashion over internet [2]. Ensuring availability of application services in the cloud is a challenging task [3]. This is because, failures are inevitable regardless of the reliability of the software components or the underlying infrastructure. The impact of these failures could be catastrophic in some cases. For example in 2013, a major service outage occurred in Amazon's east coast data center had led to a loss of $66,240 per minute [4].

Highly available systems are designed to avoid single point of failure. This is ensured by incorporating proper redundancy mechanisms [1]. Resources such as software, hardware or communication elements are replicated in the system. The basic way of organizing redundancy is

to have an active and a standby resource. The active resource provides the service and shares its state with the standby resource. If the active resource fails, the standby resource takes over the service and thus the service remains provided with continuity [1].

1.2 Thesis motivations

The notion of ubiquitous resources is realized with the advent of cloud computing [5]. By leveraging existing technologies like virtualization [6], infrastructure providers offer computing resources as a service in a pay-as-you-go manner. Service providers rent these resources to build SaaS (Software as a Service) applications and offer them as a service to their customers or end users [5]. As opposed to the traditional computing, cloud computing allows service providers to provision resources as needed during initial deployment of application and scale resources according to the needs. As a result, there is a window of opportunity to optimize and efficiently use resources in the cloud [7]. With the aforementioned advantages, the cloud is attracting more and more service providers. As end users rely more on the services provided by these applications, any unplanned service outage could result in loss of revenue for the service providers. To avoid paying penalties to the customers or end users, service providers design the SaaS applications considering both functional and non-functional requirements including availability requirements [9].

The design of highly available applications is a challenging task. Using reliable application components does not guarantee high availability. One has to incorporate proper redundancy mechanisms depending on the type of the application and appropriate recovery mechanisms to minimize the service outage. More importantly, in the event of failure the coordination among the redundant entities is important to ensure service continuity. Traditionally, along with the application logics the availability mechanisms are also included by the application developers.

Service Availability Forum (SA Forum) [8] has defined the Availability Management Framework (AMF) [10], a middleware service that abstracts the availability mechanisms (i.e. managing the life cycle of application components, coordinating redundant entities and executing appropriate recovery mechanisms in the event of failure) into the framework [10]. To manage the availability of application services efficiently, AMF requires information about the application components and services in a configuration file called the AMF configuration [10].

Designing an AMF configuration [9] [11] [12] can be generically viewed as building and dimensioning an application that is intended to provide specific service functionalities. Designing AMF configurations for a cluster based on the functional and non-functional requirements has been proposed in the literature [9]. This approach starts with analyzing the functional requirements (i.e. the type of the services the applications are intended to provide) and identifies the software components that can provide the service. Based on the non-functional requirements (such as the requested level of service availability) and the number of physical hosts required to deploy an AMF application (given as input), an AMF configuration is generated [9]. This approach is not suitable for the cloud, because the number of physical hosts required should not be fixed and this can change according to the workload variations. Therefore, designing AMF applications that can be deployed using a minimum number of physical hosts and meets the requested level of service availability remains an open question. This is a challenging task because, to minimize the number of physical hosts, the software components are collocated in the same environment (example a VM). The repeated failure of software components hosted on the same VM indicates the fault in the VM and requires a VM reboot. This recovery action may affect the availability of all the services provided by the VM (called as interference). To reduce this interference, software

components can be grouped into fault isolation units with more VMs. This in turn will increase the number of physical hosts.

While deploying AMF applications (using the generated configurations) in the cloud, if the availability constraints (anti-collocation relation) between the entities is not respected, the service availability will be jeopardized. The anti-collocation relation between the software entities and the VMs are defined in the AMF configuration. However, the availability constraints at the physical host level is not defined in the configuration. To ensure service availability, this should also be taken into consideration.

In this thesis we aim to design, deploy and manage AMF applications in the cloud that meets both availability and resource utilization requirements. Using the existing Monitoring architecture [21] and Elasticity Engine [22], workload variations of the deployed applications can be effectively managed at run-time.

1.3 Thesis contributions

The contributions of this thesis are as follows:

1) An availability estimate method that predicts the availability of a service by considering the potential factors that affect the availability including the interferences due to collocations.
2) A method to calculate the number of AMF entities that satisfies both availability and resource utilization goals. From availability perspectives, the goal is to meet the requested availability and from resource utilization perspective, the goal is to deploy the applications using minimum number of physical hosts. This method is based on the aforementioned contribution on availability estimation taken into account interferences.

3) A method to deploy the generated configurations in the cloud has been devised. The purpose of this method is to define the anti-collocation constraints at physical host level based on the generated configurations and to deploy the applications in the cloud without jeopardizing availability.
4) A prototype tool that illustrates the generation and deployment of AMF applications in the cloud has been developed. Also, the deployed application is integrated with the existing Monitoring architecture [21] and Elasticity Engine [22].
5) An application for the AMF configuration generation process in the domain of Network Function Virtualization (NFV) [45] has been developed. Using this process, configurations can be generated for AMF managed Virtual Network Functions (VNFs [44]). For this purpose, the appropriate mapping between both fields has been proposed.

1.4 Thesis organizations

Besides the introduction, the thesis is divided into six chapters. Chapter 2 introduces the necessary background on SA Forum's AMF, cloud computing and also research works related to this thesis. Chapter 3 briefly explains the configuration generation process for a cluster [9] and highlights its limitations. Chapter 4 presents the generation and deployment of AMF configurations for the cloud. Chapter 5 discuss the prototype implementation of the aforementioned contributions and also the integration with the existing Elasticity Engine [22] and Monitoring architecture [21]. Chapter 6 depicts an application of the AMF configuration generation process for VNF [44]. Chapter 7 concludes this thesis and highlights potential future research directions.

Chapter 2

Background and Related Work

This chapter introduces SAF middleware specifications [13] and more importantly focuses on the main concepts involved in understanding the AMF [10]. Further, this chapter briefly introduces the general cloud computing concepts including its characteristics, service models and deployment models [14]. Finally, we review the research works related to this thesis.

2.1 SAF Middleware

Traditionally, telecommunication companies developed their own proprietary HA solutions. Applications built using this solution have limited portability and reusability [13]. In order to address this issue, leading telecommunication and computing companies joined together to develop an open standard for SA [13].

SA Forum [8] emerged to support and manage applications to provide highly available services. For this purpose, it defined two standardized interface specifications [13]: a) The Hardware Platform Interface (HPI) [13] and b) The Application Interface Specification (AIS) [13]. One of the main advantages of standardizing interfaces is that the applications can be ported easily and they can be deployed on any middleware that supports this interface [13].

As shown in Figure 2-1, AIS defined a set of services and management frameworks to support the development and management of highly available applications [13]. Among the frameworks, AMF [10] is used to manage the availability of application services. The next sub-

section (Section 2.2) focuses on the necessary concepts required to understand the AMF, as this is the main context of the thesis.

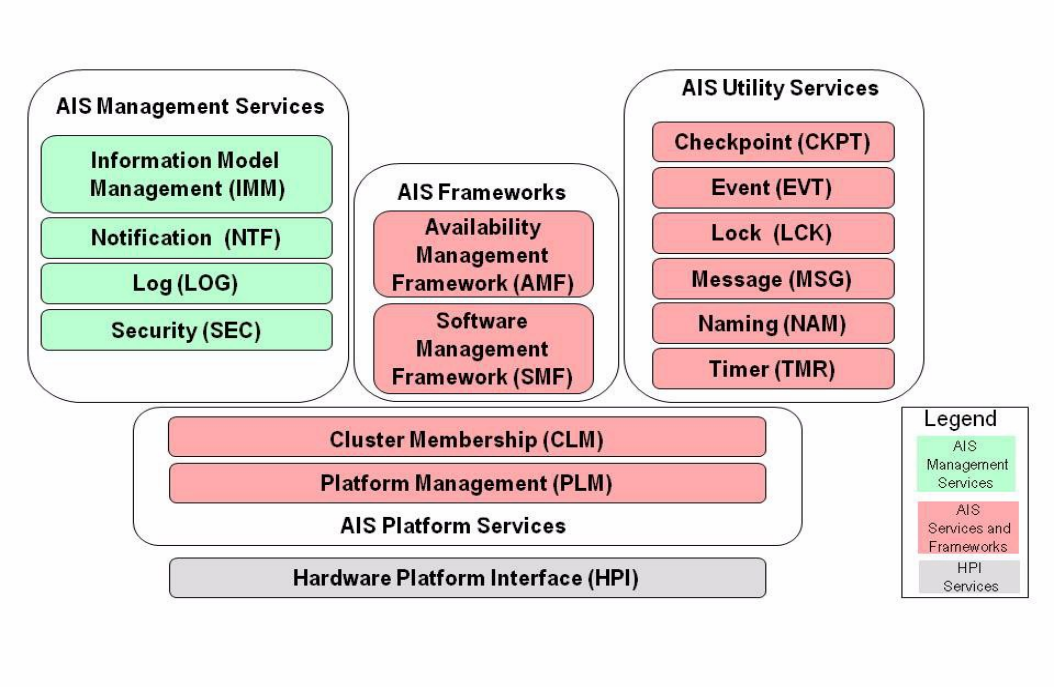

Figure 2-1 Overview of HPI and AIS services [13]

2.2 Availability Management Framework

AMF is responsible for managing the availability of the services provided by the applications by coordinating and managing its redundant entities [10]. To manage the availability of services provided by the applications, AMF requires information about the components, services provided by the components, dependencies and their logical groupings. This information is described as a configuration called AMF configuration [10]. At runtime AMF reads the configuration to know about the current state of a system, applicable redundancy mechanism, error detection and error recovery policies [10]. Using this information, AMF dynamically assigns active or standby roles to the service provisioning entities.

2.2.1 AMF Entities and AMF Entity Types

In an AMF configuration, there are two main entities such as service provider entities and the service entities [10]. Service provider entities includes components, service units (SUs), service groups (SGs), AMF nodes and AMF applications while service entities includes component service instances (CSIs) and service instances (SIs). The service provider entities and the service entities together called as AMF entities [10].

To be able to understand AMF entities and their organization into a hierarchy of logical entities, let us consider an example where a user wants to access a video through a web-interface. The web-server software and the software that plays the video represents the component [10]. It is the smallest service provider entity and also the smallest fault-zone within a system [10]. For the video component to be able to play the video service, workload should be assigned to it. Therefore, CSI represents a unit of service workload that a component is able to provide [10]. One may have noticed that a web-server component and a video player component collaborate to provide the video service. It also implies that due to tight collaboration, fault propagation can also occur [1]. For these reasons, the components that are collaborating to provide a service functionality are grouped logically into SUs [10]. This is the next fault-zone identified by the AMF that can be isolated and repaired on its own [1]. It should also be noted that, respective CSIs assigned to web-server components and video player components will compose a video SI [10]. AMF assigns SIs to SUs during run-time. To protect the service in spite of failures, redundant SUs work together and form a protection group called SG [10]. Typically, an AMF application consists of one or more SGs and also SIs that are protected by the SGs [10]. AMF nodes are logical entities that are used to deploy SGs [10]. This could be mapped to a physical hardware or a virtual machine (VM). SGs are deployed over a group of AMF nodes which forms the AMF cluster [10]. AMF entities are

typed, except for nodes and clusters. The common characteristics of AMF entities are captured in their respective types [10]. Figure 2-2 illustrates the AMF configuration for this example with service provider entities, service entities and their corresponding types.

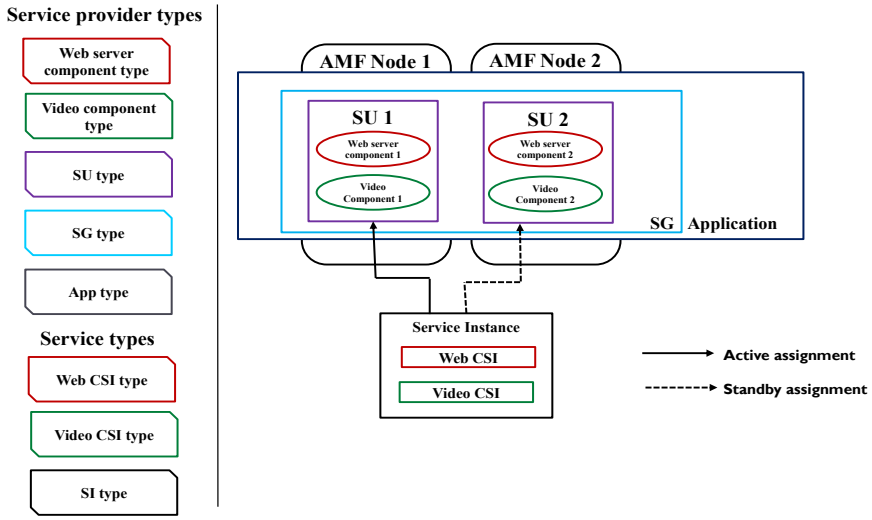

Figure 2-2 An example of AMF configuration

In the event of failure of a component, AMF detects the failure through its health monitoring or error reporting functionalities [10]. Depending on the recovery related attributes specified in the configuration, it then automatically recovers the service by performing recovery action either at the component level (component restart or component failover) or SU level (SU restart or SU failover) or AMF node level (node fail fast or node failover or node switchover) or application level (application restart) or cluster level (cluster reset) [10]. The actual recovery of the components in the context of the configuration can be determined using [11].

2.2.2 Redundancy models

A SG follow one of the following redundancy models; no-redundancy redundancy model; 2N-redundnacy model; N+M redundancy model; N-way active redundancy model; and N-way

redundancy model; Each SI is characterized with a number of active and standby assignments that varies according to the redundancy model [10].

- *No-redundancy redundancy model*

  In this case, each SI has at most one assignment and each SU can take at most one active assignment [10]. In other words, a SU in this redundancy model will not be assigned any standby assignments. Since each SI has only one assignment, from the service perspective this redundancy model provides "no-redundancy" [1]. However, from the service provider perspective there are other SUs in a SG that can take over the service in the case of failure. From this standpoint, this does provides redundancy [10]. Let us consider a SG with two in-service SUs (SU 1 and SU 2) as shown in Figure 2-3. In-service SUs are those that are instantiated and ready to take assignments [10]. This SG is configured to protect one SI (SI 1). For example, at-run time if AMF assigns active role to SU1, then SU2 will be the spare SU.

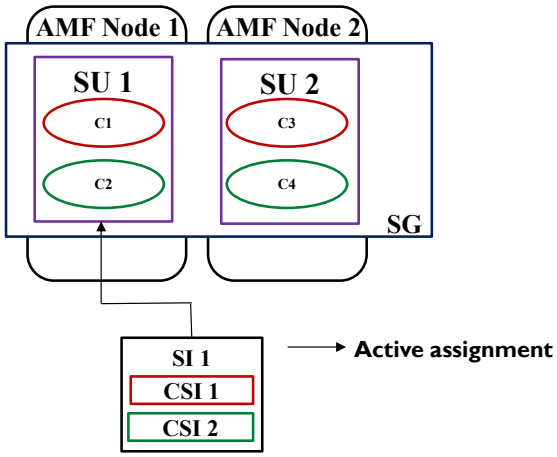

Figure 2-3 *An example for No-redundancy redundancy model*

- *2N redundancy model*

  It is also called as 1+1 or active-standby redundancy model [10]. From service side, each SI has one active assignment and one standby assignment [10]. From service provider

perspective, each SG is characterized by at most one active SU, one standby SU and spare SUs depending on the configuration. Let us consider a SG with three SUs (SU1, SU2 and SU 3). This SG is configured to protect 2 SIs (SI 1 and SI 2) as shown in Figure 2-4. For example, at run-time AMF may assign active role to SU 1 and standby role to SU 2 and SU 3 is considered as the spare SU. In the event of failure of SU 1, SU 2 will take over the active role and start providing SI 1 and SI 2. Also, SU 3 will be assigned standby role to protect against failures.

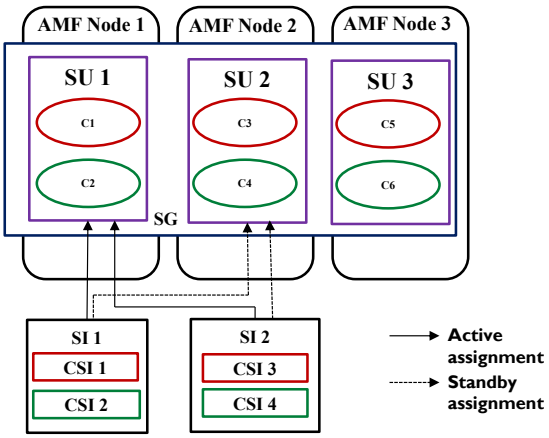

Figure 2-4 *An example for 2N-redundancy model*

- *N+M redundancy model*

From the service side, each SI has one active assignment and one standby assignment [10] and from the service provider side, a SG is characterized by N active SUs and M standby SUs [10]. Let us consider a SG with four SUs (SU 1, SU 2, SU 3 and SU 4) and this SG is configured to protect 3 SIs (SI 1, SI 2 and SI 3) as shown in Figure 2-5. At run-time, AMF assigns active role to SU 1 and SU 2 (N=2), standby role to SU 3 (M=1) and SU 4 is considered as the spare SU. For example, in the event of failure of SU 1, the active assignment of SI 1 and SI 2 is failed over to SU 3. Also, SU 4 will take the standby role to protect the SIs against failures.

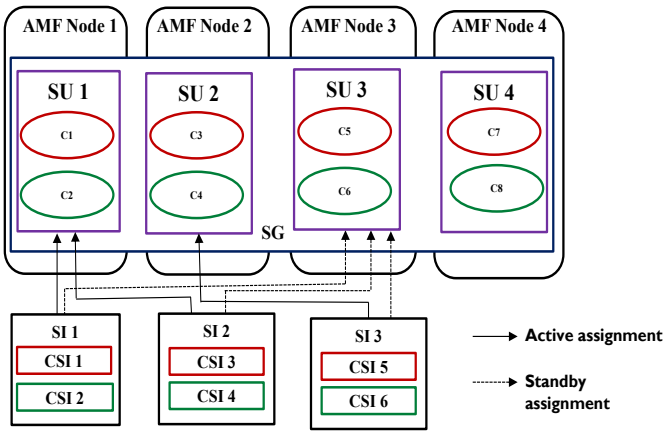

Figure 2-5 *An example for N+M redundancy model*

- *N-way redundancy model*

Each SI has one active and one or more standby assignments, depending on the configuration [10]. The SUs in a SG following N-way redundancy model can take active and/or standby role. The only constraint is that, a SU cannot be in the active and standby state for the same SI [10]. Let us consider a SG with four SUs (SU 1, SU 2, SU 3 and SU 4) and this SG is configured to protect 3 SIs (SI 1, SI 2 and SI 3) as shown in Figure 2-6. Also, the number of standby assignments per SI is configured to be 3. At run-time, SU 1 takes active assignment of SI 1 and standby assignments of SI 2 and SI 3. Similarly, SU 2 and SU 3 takes active assignments of SI 2 and SI 3 also standby assignments of other SIs respectively. SU 4 is considered as the spare SU. Note, that the standby assignments are ranked. In the event of failure of SU 3, SU (SU 1 or SU 2) that is assigned the highest ranked standby assignment will provide the SI 3.

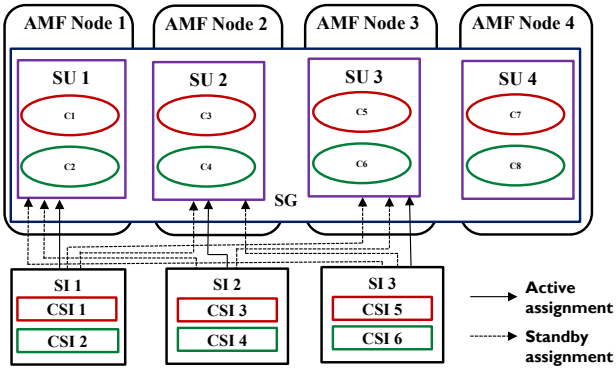

Figure 2-6 An example for N-way redundancy model

- *N-way active redundancy model*

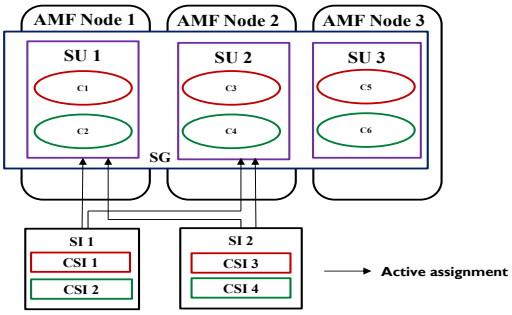

Figure 2-7 An example for N-way active redundancy model

Each SI is characterized by more than one active assignment [10]. This redundancy model does not support any standby assignments [10]. When active assignments are assigned to the SUs of a SG, it implies that all the service functionality is provided by all the active SUs. Let us consider a SG with three SUs (SU 1, SU 2 and SU 3) and this SG is configured to protect 2 SIs (SI 1 and SI 2) as shown in Figure 2-7. Each SI is configured to have two active assignments. At run-time, SU 1 and SU 2 are assigned active roles and SU 3 is considered as a spare SU. In the event of failure of SU 2, AMF will not consider this as a service interruption because, the service is provided by SU 1 [1]. However, AMF will failover the SIs provided by SU 2 to SU 3.

2.3 Entity Type File (ETF)

The software vendors describe the software entities in terms of prototypes in an XML file called ETF [15]. Every prototype mentioned in the ETF possess a name, version and its specific characteristics. For example, each component prototype in ETF specifies the name, version and the type of service the component is intended to provide. In addition, it also includes other characteristics like capabilities of the component prototypes and whether the components of this type can restart or not during service recovery actions [15]. For example, if active and standby capabilities of component prototype are 2 and 4 respectively, then it implies that the component prototype cannot take more than 2 active and 4 standby assignments.

It is mandatory to include the information about component prototypes and component service prototypes; however the information about the other AMF prototypes can be left optional in an ETF [15]. If the information about SU or App prototype is specified, then it implies that a vendor imposes restriction on how the component prototypes can be composed to collaborate.

2.4 Cloud Computing

With the recent advancement in the Internet technology, the need to rapidly deploy applications to meet the growing businesses has increased [5]. To support this growth, often systems are designed to handle maximum workloads. This overprovisioning of resources often results in underutilized server capacity and increases the total ownership cost [7]. Cloud computing reduces the upfront investment in purchasing and maintaining the resources and the total ownership is reduced considerably [5]. Resources can be provisioned as needed on a pay as you basis. This allows the systems to be designed to handle the minimum or average workload and the

application servers can be scaled up or down according to the workload demand. National Institute of Standards and Technology (NIST) [14] has proposed the following cloud computing definition:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"

Taken further, NIST has also defined five key characteristics, three service models and four deployment models for the cloud [14].

2.4.1 Key characteristics

The following are the important characteristics of a cloud system [14].

- ✓ *On-demand self-service* – Without the need for any significant assistance, the cloud resources (compute, storage and network) can be automatically provisioned.
- ✓ *Broad network access* – Heterogeneous client platforms such as mobile phones and laptops can be used to access the cloud resources.
- ✓ *Resource pooling* – Compute, storage and network resources are pooled by virtualizing them. This allows the flexibility to allocate and manage resources in the cloud computing paradigm.
- ✓ *Rapid elasticity* – Resource pooling gives the impression of infinite resources available to the customers. This allows rapidly allocating or reallocating resources to the customers based on their needs.

- ✓ *Measured service* – The resource pool is shared by multiple customers. Infrastructure provider monitors the resource usages for each customer and bills them according to their usage.

2.4.2 Service models

- ✓ *Infrastructure as a Service (IaaS)* - Typically a data center consists of heterogeneous physical servers, switches, storage elements etc. Infrastructure providers own and manage the physical and virtual resources in the data centers. These resources are virtualized and offered as a service to the customers. This allows the flexibility to choose the operating systems and other necessary software required to deploy the applications [5].
- ✓ *Platform as a Service (PaaS)* — PaaS provider like Google App Engine offers the necessary built-in services like databases and Application Programming Interface (APIs) to develop applications [16]. This provides the flexibility to develop and deploy applications without having to deal with directly with the infrastructure including network, operating systems or storage [5].
- ✓ *Software as a Service (SaaS)* — SaaS providers build SaaS applications (example Microsoft Office 365 [49]) and deploy it over the provider's infrastructure. Users are allowed to access the application without the need to install, run and maintain the underlying infrastructure, application [49]. Therefore, the up-front cost required to invest in the infrastructure and also in the software licensing is reduced [17].

2.4.3 Deployment models

- ✓ *Private cloud*

  A cloud is said to be a private cloud when the services are offered through a private network. Generally it is owed by a single organization that deals with more secure data and requires a more flexible and scalable platform [5].

- ✓ *Public cloud*

  A cloud is said to be public cloud when the services are offered through the internet (i.e. public network). This allows multiple organizations to share the resources and thereby reducing the total cost. [5]. Examples of public cloud includes, Google's web based e-mail and file storage system like Dropbox.

- ✓ *Hybrid cloud*

  A cloud is said to be hybrid cloud when it is a combination of both public and private cloud. The idea is to create a unified model employing both the clouds so that an organization can benefit from the best of both worlds [5].

2.5 OpenStack

OpenStack is a cloud operating system that dynamically manages the compute, network and storage resources in a data center [18]. It provides a cloud computing platform to build public or private cloud. OpenStack contains multiple components. Among these components, compute service is provided by nova [19]. It facilitates the provision of on-demand VM instances using the nova-scheduler service [20]. It is the responsibility of nova-scheduler to determine the mapping of VMs to physical hosts. For this purpose, the nova-scheduler uses the filtering and weighting mechanisms to determine the eligible physical hosts [20]. It allows the use of variety of filters including, but not limited to ram filter, core filter, disk filter, group affinity filter and group anti-

affinity filter. With the help of these filters, nova-scheduler eliminates the physical hosts that are not capable of hosting VMs. Further, the nova-scheduler orders the valid list of physical hosts by applying weights to them. Finally, it selects the physical host that is more weighted [20]. Figure 2-8 [20] illustrates the scheduler's filtering and weighting mechanism. Let us suppose that scheduler applies the ram filter to the list of available hosts (Host 1 to Host 6). It rejects Host 2 and Host 4 due to the presence of inadequate ram resource. It then applies weights to the Host 1, Host 3, Host 5 and Host 6 and ranks Host 5 as the most weighted host. Finally, Host 5 is selected to host a VM instance.

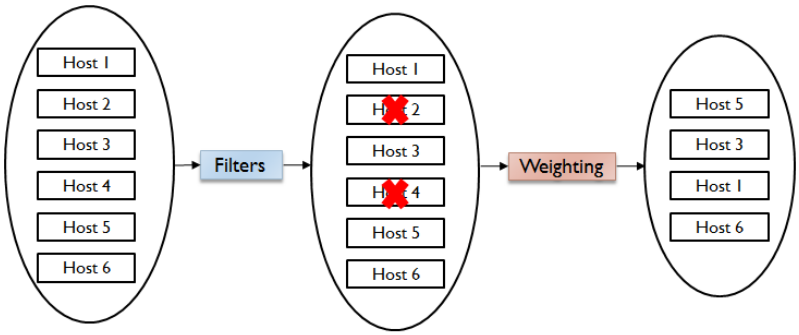

Figure 2-8 Scheduling in OpenStack [20]

2.6 Monitoring and Elasticity Engine

Recently a Monitoring architecture [21] and an Elasticity Engine [22] for AMF managed applications has been proposed in the literature. Unlike the existing monitoring tools in the cloud, this tool monitors the service level workload changes for the AMF managed applications [21]. It then triggers the Elasticity Engine [22] to allocate or reallocate resources based on the workload changes. The Monitoring Engine follows the client-server architecture. Each AMF node in the cluster, runs a monitoring client to measure the service level workload of the components residing on it. The monitoring clients sends the workload to the monitoring server at regular intervals. The monitoring server aggregates the workloads from each component and calculates the associated SI

workload [21]. Further, the monitoring server sends the SI workload to the workload analyzer to trigger workload increase or decrease to the Elasticity Engine. For this purpose, workload analyzer checks the received SI workload with the pre-configured threshold value and generate triggers accordingly [21] (Figure 2-9). Elasticity Engine then reads the current configuration and makes the appropriate changes at the SG level or at the cluster level through the Information Model Management (IMM) [23] service. AMF then reacts to the configuration change by adjusting the CSI assignments or SIs in accordance with the modified configuration. Thereby, the service providers are scaled-out or scaled-in based on the service level workload [21].

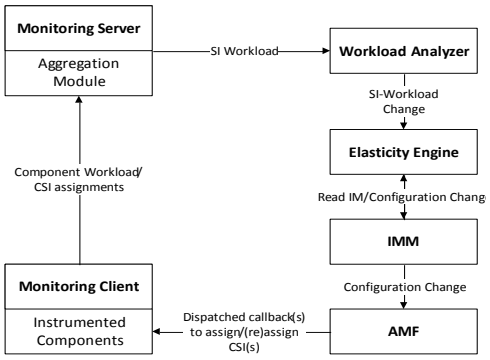

Figure 2-9 *Monitoring Engine and Elasticity Engine architecture integrated with AMF [21]*

2.7 Related Work

The following works [9] [11] [12] target the generation of AMF configurations. The author in [11] automated the AMF configuration generation process and generates multiple AMF configurations by taking into account various possible configuration options. The author in [12] takes into account the functional requirements and generates AMF configurations using a model driven approach [12]. Both the works [11] and [12] did not consider non-functional requirements such as availability requirements while generating AMF configurations. The author in [9] aims at generating configurations for a cluster that meets functional requirements and non-functional

availability requirements (i.e. requested level of service availability). More details on the AMF configuration generation process for a cluster [9] and its limitations are explained in Chapter 3.

There are numerous works that discuss about deploying applications in the cloud. Most of them focus on optimizing the placement of VMs on physical hosts based on multiple constraints like resource based, performance and availability [24] [25] [26] [27]. The authors in [24] aim at improving the availability and performance of services in the IaaS cloud while placing VMs on physical hosts. For this purpose, a structural constraint-aware VM placement technique is proposed. This is a hierarchical placement approach that considers demand, communication and availability constraints while mapping VMs to physical hosts. Another attempt [25] proposes a highly available optimal placement by considering interdependencies between the application components, communication delay tolerance and resource utilization. The authors in [27] proposed a VM placement method that generates a minimum redundant VM configuration that can survive any k-physical host failures. The above mentioned works considered mapping an application to a VM and optimally placing VMs on physical hosts. In our approach, this application could be mapped to an AMF component.

The authors in [28] presents a request aware VM placement approach to improve the availability of services by choosing the right deployment choices. This work is closely related because it not only considers the mapping application components to one or multiple VMs but also it considers the potential interference that may occur due to multi-tenancy of application components in a VM. However, this solution did not take the specificities of high availability middleware like AMF in to account.

Recently deploying applications in multi-cloud environments is becoming popular [29] [30]. For example the authors in [30] proposes a multi-objective scheduling technique that aims to

achieve high availability of applications and also it aims to minimize the application cost (i.e. by optimally scheduling or rescheduling the application components to a node based on the workload demand) and maximize the resource usage. In addition to this, scalability of applications across different clouds is considered.

Chapter 3

AMF Configuration Generation Process for a Cluster

3.1 Introduction

AMF configuration generation process for a cluster [9] requires two inputs; a) ETF model and b) configuration requirements (CR) as shown in Figure 3-1. As mentioned in Chapter 2, ETF is a software catalogue that is used to build AMF applications [15]. An ETF model may include ETFs from different software vendors [9]. CR captures the type of the service the application is intended to provide, the number of SIs of a service type, the number of CSIs of a component service type in each SI, optionally the redundancy model of a SG type, the number of active assignments (for N-way active redundancy model) and the number of standby assignments (for N-way redundancy model). Deployment details such as the number of AMF nodes in a cluster, cluster startup time, time required by an AMF node to shutdown is also included. In addition, maximum number of attempts required by the AMF to instantiate a component, maximum number of attempts required by the AMF to instantiate a component with a delay between the instantiation attempts and the delay between the instantiation attempts is specified. Finally, the non-functional requirement such as requested level of service availability is also included in the CR [9]. AMF configuration generation process [9] [11] has four main steps: a) ETF prototype selection; b) AMF type creation; c) AMF entities creation; d) Distribute AMF entities for deployment. The authors in [9] applies four design patterns and two methods to enhance the service availability.

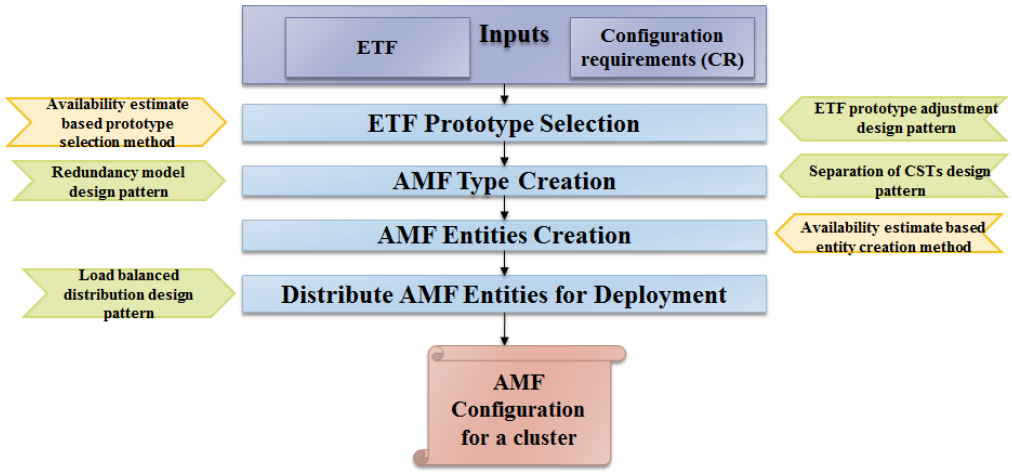

Figure 3-1 AMF Configuration Generation Process for a Cluster [9]

3.2 ETF Prototype Selection

In this step, ETFs from different vendors are analyzed and the prototypes that can provide the requested service are selected. It is possible to create hierarchy of prototypes (from app types to component types) called type stack from different prototypes therefore, each type stack may lead to different AMF configurations [9].

ETF prototypes from different software vendors are adjusted to improve the service availability. ETF prototypes specify multiple attributes available to configure the software from an availability perspective. Some of the recovery related attributes are altered to minimize the impact zone using the ETF prototype adjustment design pattern [9]. This step also intends to estimate the level of service availability using the availability estimate method [9] to check if a type stack can provide the requested level of service availability. If it is not met, then that type stack is discarded in the early stage of the configuration generation process [9]. Type stacks that meet the requested level of availability are considered for the next step.

3.3 AMF Type Creation

AMF types are software entity types that are defined for the AMF management purposes [15]. These AMF types are created from their corresponding ETF prototypes in the type stacks. However, if the information about SU prototype or SG prototype or App prototype is not found, then they are created in this step [9]. ETF prototypes specify a range of available options and this allows the possibility to create multiple AMF types from the same ETF prototype [9]. Using separation of CSTs design pattern, AMF types are created in such a way that the failure of a component will affect only minimum number of SIs [9]. Also, if the system designer has not requested the redundancy model for SG type(s), then this step determines the appropriate redundancy model for a service type based on the active and/or standby capability of component types [9].

3.4 AMF Entities Creation

This step aims to create the number of AMF entities from their corresponding AMF types based on the requested level of service availability. For this purpose, availability estimate-based entities creation method [9] is used. Considering the availability requirement and the number of SIs, this method calculates the number of components, the number of SUs and SGs required and they are configured. Note that these AMF entities are created according to the number of AMF nodes specified in the CR [9].

3.5 Distribute AMF Entities for Deployment

Once the AMF entities are created, then the next step is to distribute the SUs over the AMF nodes and to set the deployment related attributes [9]. In this step, different deployment options

are possible. Using load-balanced distribution design pattern, this step aims to distribute SUs in an even manner for 2N-redundnacy model [9].

3.6 Limitations

The above mentioned AMF configuration generation process for a cluster [9] is not suitable for the cloud because of the following issues.

1) This approach requires the system designer to specify the number of physical hosts (AMF nodes) as an input to generate the AMF configurations. These physical hosts are also considered to have infinite capacity. The issue is that the number of physical hosts specified by the system designer may not be minimum because the system designer may neither be aware of the effect of collocating entities nor be aware of the resource limitations. Furthermore, this number of physical hosts affects the AMF entities creation calculation.
2) The availability estimate method used in the AMF configuration generator [9] considers only the availability of the components. However, in any system the availability of the underlying infrastructure (both virtual and physical infrastructure) is also an important factor to consider.
3) This solution did not consider the effect of deploying these AMF entities together in a collocated manner. When entities are collocated in the same environment (e.g. VM), the collocated entities may fail and affect the availability of other SIs served by that environment. Note that collocated entities may interfere at the SU, VM and at the host level as well.

Chapter 4

AMF Configuration Generation, Deployment and Run-time Management in the Cloud

4.1 Introduction

This chapter presents an approach to generate, deploy and manage AMF applications in the cloud. For this purpose, this chapter is divided into three main sections. The first section (Section 4.2) describes the AMF configuration generation process for the cloud. This process overcomes the limitations highlighted in Chapter 3. This section includes the proposed AMF entities creation and availability estimate methods. The second section (Section 4.3) depicts the proposed method to deploy AMF applications in the cloud. The third section (Section 4.4) describes the run-time management of AMF applications using the existing Monitoring architecture [21] and Elasticity Engine [22]. Figure 4-1 illustrates the overall picture of generation, deployment and integration of Monitoring architecture and Elasticity Engine with the deployed applications.

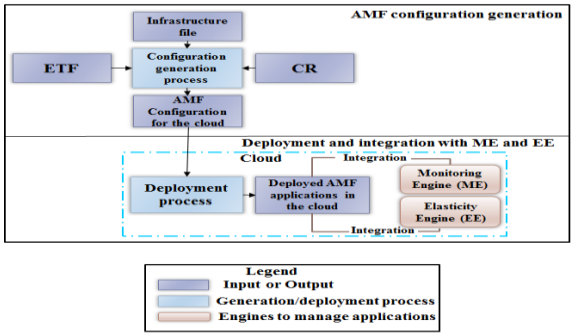

Figure 4-1 *AMF configuration generation, deployment and run-time management in the cloud*

4.2 AMF configuration generation process for the cloud

4.2.1 Introduction and overall view

To design AMF configurations, one has to be aware of the services AMF applications intend to provide. This is abstracted as workload units called SIs and CSIs [10]. The configuration requirements captures the type of the service and the number of SIs and CSIs to be provided by the application with a minimum level of service availability [9]. Our aim while generating AMF configurations is to build applications that can provide the specified number of SIs and guarantee the requested level of service availability. Since these AMF applications are intended to be deployed in the cloud, it is important to use minimum resources (physical hosts). This considerably reduces the upfront investment on infrastructure and by using the existing Elasticity Engine [22], the number of resources can be increased in the future as needed.

Our main goal is to design AMF applications/configurations that can:

- ✓ provide and protect all the SIs specified number in the CR
- ✓ meet the requested level of service availability
- ✓ use minimum resources (physical hosts) for deployment

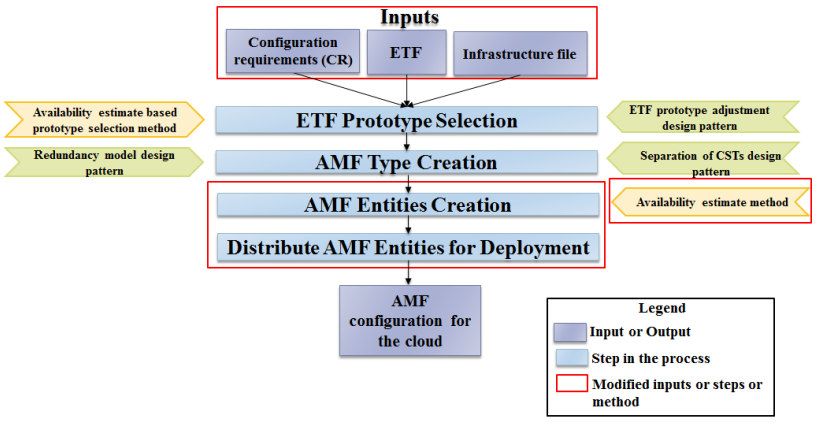

Figure 4-2 Modified AMF configuration generation process for the cloud

For this purpose, the existing AMF configuration generation process proposed for a cluster environment [9] (explained in Chapter 3) has been revisited and extended. As shown in Figure 4-2, inputs to the generation process, third and fourth steps of the generation process and availability estimate method used at the third step have been modified. The rationale behind the changes are as follows:

1) Inputs

- **ETF -** Software vendor describes the characteristics of components in an ETF file according to the ETF model defined in [9]. While determining the number of entities, the resource needed for a component to provide a service is required. For this purpose, part of the ETF model is extended as shown in Figure 4-3. Here, CT represents the component type class and CST represents the component service type class. The properties of a component type providing a CST are defined in CTCST association class. The memory usage required by a component to provide a CSI is added to the CTCST association class. Note that for simplicity only the memory resource is considered.

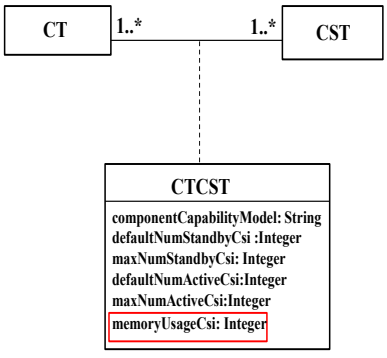

Figure 4-3 *Part of extended ETF domain model*

- **CR –** According to the CR model defined in [9], the system designer specifies the deployment information like the number of AMF nodes (i.e. physical hosts) along with the information about SIs (i.e. the number of SIs in a service type, the number of CSIs in each

SI and the requested level of service availability). As mentioned in Chapter 3, a system designer is not aware of the effect of collocation and resource limitations, therefore the number of physical hosts specified may not be accurate and minimum. For this purpose, the number of physical hosts is no longer specified as input and the CR model is modified accordingly.

- **Infrastructure file –** In addition to the ETF and CR, deployment information such as capacity of the physical host, the number of available VM flavors, the capacity of VM flavors, Mean Time to Fail (MTTF) of the infrastructure elements like physical hosts, VMs, guest OS, host OS and hypervisors are required while determining the number of entities. For this purpose, a third input called infrastructure file is added in the configuration generation process. This file is created according to the domain model shown in Figure 4-4.

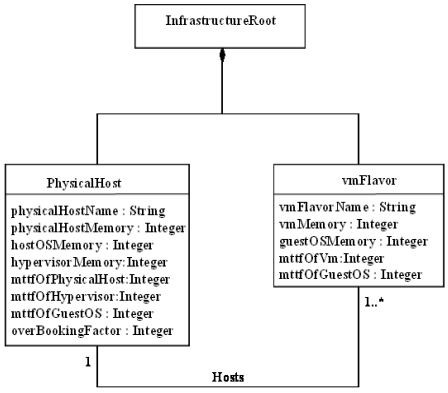

Figure 4-4 Infrastructure domain model

2) ***Modifications to the third and fourth steps –*** It is important to mention the reason behind changing only the third and fourth step as opposed to changing entire steps of the generation process. The first two steps of the generation process is about building the application types (i.e. starting from app types to all the way to component types) that can provide the desired service functionality [9]. Note that in these steps the applications are designed at the type level

independent of the deployment environment. However, the later steps are deployment specific. The third step of the configuration generation process [9] calculates the number of instances of each AMF entities (components, SUs, SGs) that forms an application according to the number of physical hosts in the physical cluster. In contrast, while designing applications for the cloud, only the minimum number of physical hosts needs to be considered. Therefore, the third and fourth steps are modified and they are explained in detail in Section 4.2.2 and 4.2.3 respectively.

3) ***Modifications to the availability estimate method*** **–** The availability estimate method [9] considers only the availability of components providing a service (SI). Collocating AMF entities into the same environment also affects service availability. This should also be taken into account while estimating the service availability. Also, in any systems the availability of multiple elements like physical infrastructure and virtual infrastructure should also be considered. For the above mentioned reasons, the availability estimate method [9] used in the third step of the generation process is modified.

4.2.2AMF Entities Creation step

4.2.2.1 Factors influencing the number of physical hosts

As shown in Figure 4-5, there are three factors that influence the number of physical hosts:

1) Redundancy; 2) Interferences due to the collocated entities and 3) Capacity of VMs and physical hosts.

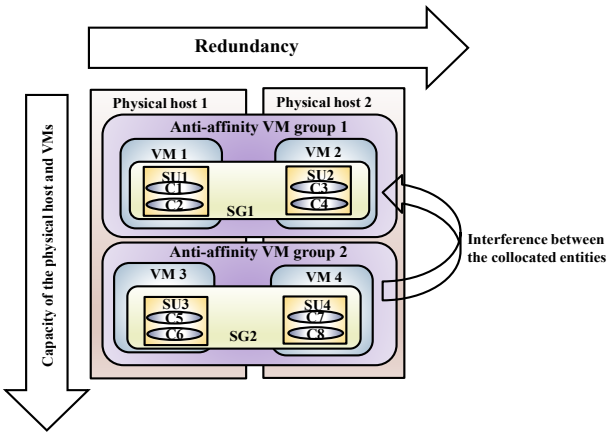

Figure 4-5 *Factors influencing number of physical hosts*

1) Redundancy

Redundancy requires additional resources to protect the service against failures [1]. The number of SUs per SG determines the minimum number of physical hosts required. This is because, SGs are deployed over VMs in an anti-affinity group and eventually these VMs are hosted over physical hosts. For example, in the case of 2N-redundancy model, a minimum of one active SU and one standby SU is required per SG. These SGs are deployed over a minimum of two VMs and these VMs are hosted in an anti-affinity group over a minimum of two physical hosts. Taking into account potential interferences between the collocated entities and the capacity of the physical host and the VM, the number of physical hosts required may be more than two.

2) Interference between the collocated entities

The rationale behind collocating SUs belonging to different SGs into a VM and collocating VMs protecting different services into a physical host is to minimize the number of physical hosts. However, when several entities are collocated they may fail and affect the other SIs hosted in that environment. To minimize the interference, components can be grouped into multiple fault isolation units such as VMs, but this may increase the number of physical hosts required.

3) Capacity of the resources

Another factor influencing the number of physical hosts is the capacity of the host and the capacity of the VMs hosted on it. Physical hosts have finite resources in terms of RAM, disk, number of cores and so on. VMs are available in various pre-defined flavors such as tiny, medium, large etc. A limited number of VMs, with different flavors, can be hosted on a physical host. It is well known that virtualization introduces some overhead due to the presence of the hypervisor in the host and the guest OS in each of the VMs, and this will be taken into account in the calculation of collocated instances.

4.2.2.2 Two phases of AMF entities creation step

We identified two phases in the AMF entities creation step.

1) Determining the number of SIs per VM flavor from the perspective of availability and resource utilization
2) Selection of the VM flavor

For each type stack, created in the step one and two of the configuration generation process, the above mentioned phases are repeated. The number of SIs per VM calculation is carried out for all the VM flavors in the infrastructure file. Next, in the second phase, an appropriate VM flavor that satisfies availability requirements and supports a minimum number of physical hosts is selected.

1) Determining the number of SIs per VM flavor from the perspective of availability and resource utilization for a type stack

Initially, the memory required by the components collaborating to provide an active SI assignment of the requested service type is calculated. It is the summation of the memory required

for all the CSIs of a SI as shown in the Equation (1). Here, *Memory required per CST* represents the memory required for all the CSIs per CST.

$$Memory\ required\ \ per\ SI = \sum_{k=1}^{k \le m} Memory\ required\ \ per\ CST_k \qquad (1)$$

k iterates through the *m* component service types in the service type. Equation (2) determines the *Memory required per CST* by multiplying the *Memory required per CSI* and the *No of CSIs per CST*. The *Memory required per CSI* of a CST and the *No of CSIs per CST* are obtained from the extended ETF model and the CR respectively.

$$Memory\ required\ \ per\ CST = Memory\ required\ \ per\ CSI\ *\ No\ of\ \ CSIs\ \ per\ CST \qquad (2)$$

The total guest memory (*TGM*) of a VM flavor is used by the guest OS and the components hosted by the VM. To determine the guest memory available (*AGM*) to host the components, the virtual memory required by the guest OS (*GOSM*) is excluded from the *TGM* as shown in Equation (3).

$$AGM = TGM - GOSM \qquad (3)$$

Next, based on the memory required to provide an active SI assignment and the *AGM*, the number of SIs per VM (*No of SIs per VM*) is determined using Equation (4).

$$No\ of\ \ SIs\ \ per\ VM\ = \ floor\left(\frac{AGM}{Memory\ required\ \ per\ SI} \right) \qquad (4)$$

Once the *No of SIs per VM* is calculated, then the next step is to determine the capacity of the SU in terms of SIs and the number of SGs (*No of SGs*) and evaluate the effect of collocating components in a SU, collocating SUs in a VM and collocating VMs in a physical host. These collocated entities are those that are hosted in the same environment as the components providing

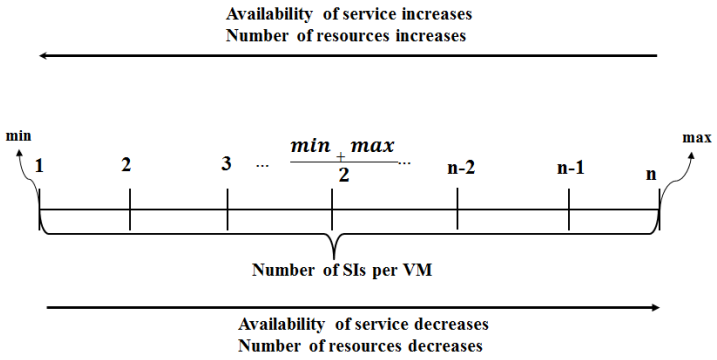

Figure 4-6 *Number of SIs per VM*

the SI, whose availability is being estimated using the availability estimate method in Section C. The failure of these collocated entities would require the recovery action to be performed in a bigger scope and this impacts the availability of the other SIs. For example, if the components in the collocated SUs are configured to recover with component restart fails repeatedly, then in order to capture the fault the recovery action may escalate from component level to the SU and VM levels thereby affecting the availability of other SIs.

Even though the capacity of the VM sets an upper limit for the number of SIs it can host, in reality it is limited due to the interference caused due to the collocated entities, the capacity of the SU and the *No of SGs*. As shown in Figure 4-6, *min* represents the minimum number of SI (which is actually one) a VM may provide while *max* represents the maximum number of SIs a VM can support based on its capacity, the SU capacity and the *No of SGs*. When the *No of SIs per VM* (i.e. the components providing these SIs) is increased gradually from *min* to *max*, the availability of the services decreases as the interference between the collocated components increases. On the other hand, the collocation of components in the VMs results in lesser number of VMs and physical hosts. Our aim is to determine the actual *No of SIs per VM* that meets the requested availability and results in minimum number of physical hosts.

A. Estimate availability for various scenarios

In order to determine the *No of SIs per VM* flavor for a type stack, Algorithm-1 is used. It uses the two methods described in Section B and C to calculate the number of entities and to estimate the service availability for various scenarios.

Algorithm-1 Calculate the number of SIs per VM flavor for a type stack
Input: No of SIs per VM from Equation (4), RA from CR
Output: No of SIs per VM

```
1   Begin
2     Initialize max to No of SIs per VM
3     Initialize min to one
4     Initialize RA from CR
5     Calculate number of entities and number of collocated entities using Algorithm-2 for min
6     Use availability estimate method from Section C to EA for the number of collocated entities
7     if (EA = RA) then
8       No of SIs per VM = min
9     else if (EA>RA)
10      No of SIs per VM = Call getNoOfSIsPerVM (max, min, RA)
11    else
12      Discard type stack for this current VM flavor
13      Break
14    end if
15    return No of SIs per VM
16  End
17  getNoOfSIsPerVM (max, min, RA)
18  Begin
19    Calculate number of entities, No of SIs per VM and number of collocated entities using Algorithm-2 for max
20    Update the max with No of SIs per VM
21    Use availability estimate method from Section C to EA for the number of collocated entities
22    if (EA < RA) then
23      Initialize mean to (min + max)/2
24      Calculate number of entities, No of SIs per VM and number of collocated entities using Algorithm-2 for
          mean
25      Update the mean with No of SIs per VM
26      Use availability estimate method from Section C to EA for the number of collocated entities
27      if (EA < RA) then
28        max = mean
29        Call getNoOfSIsPerVM (max, min, RA)
30      else if (EA > RA) then
31        min = mean
32        Call getNoOfSIsPerVM (max, min, RA)
33      else
34          return mean
35    end if
36    else
37        return max
38    end if
39  End
```

a. Best case scenario

Initially, Algorithm-1 determines the number of entities such as the *No of SGs*, the number of VMs (*No of VMs*), the number of physical hosts (*No of PHs*) and the number of collocated entities for *min* using the Algorithm-2 (line 5). Here, *min* represents the best case scenario from the perspective of availability estimation because there is only one SU in a VM that is providing a SI and there are no collocated SUs in a VM. As a result, it provides the maximum level of service availability. While interference due to the collocation of VMs exists, this is less than the interference due to the collocation of SUs in a VM. This is because VMs provide better fault isolation compared to SUs. We estimate the availability provided by this best case scenario using the method described in Section C (line 6). If the estimated availability is equal to the requested availability, then *min* becomes the *No of SIs per VM* (lines 7-8). On the other hand, if the estimated availability is greater than the requested availability, then the availability is estimated for the worst case scenario because with respect to resource utilization this is the worst case (lines 9-10).

If the estimated availability is less than the requested availability even for the best case scenario then this type stack is discarded for this VM flavor (lines 11-13). The rationale behind this is that, if the type stack is not able to meet the requested availability for the best case scenario then, there is no way the requested availability will be met for any case for a given VM flavor. However, it is possible that estimated availability for the best case scenario may meet for the other available VM flavors. This is due to the varying Mean Time to Fail (MTTF) and the number of collocated VMs for each flavor.

b. Worst case scenario

While determining the number of entities, the SIs are distributed vertically based on the capacity of the SU and horizontally to the SUs of the SG depending on the redundancy model. It is possible

that due to the limitation on the capacity of the SU and the *No of SGs*, the *No of SIs per VM* may be reduced. If reduced, this becomes the *max* and the number of collocated entities is calculated based on this (line 19-20). *Max* represents the worst case scenario from the perspective of availability estimation because, the interference between the collocated components is the maximum. However, this is the best case from the resource utilization perspective. If the estimated availability is greater than or equal to the requested availability then we select *max* to be the *No of SIs per VM* because the requested availability is met and also it infers the lowest number of VMs and physical hosts (line 36-37). If the worst case scenario is not satisfied but the best case scenario is satisfied then the solution lies between *min* and *max*. To converge faster to the solution, the number of entities is calculated next half way between them i.e. at *floor (min + max)/2*. Again, the *No of SIs per VM* may be reduced due to the distribution of SIs and this reduced SIs per VM, if any becomes the *mean* (lines 24-25).

When availability is estimated for this *mean,* three possible cases exist. They are:

i. If the estimated availability is equal to requested availability, then the *mean* becomes the *No of SIs per VM* (line 33-34).

ii. If the estimated availability is greater than requested availability, then the solution interval becomes [*mean, max*] and the value of *min* is updated to the *mean* (line 30-31).

iii. If the estimated availability is lesser than the requested availability, then the solution interval becomes [*min*, *mean*] and the value of *max* is updated to the *mean* (line 27-28).

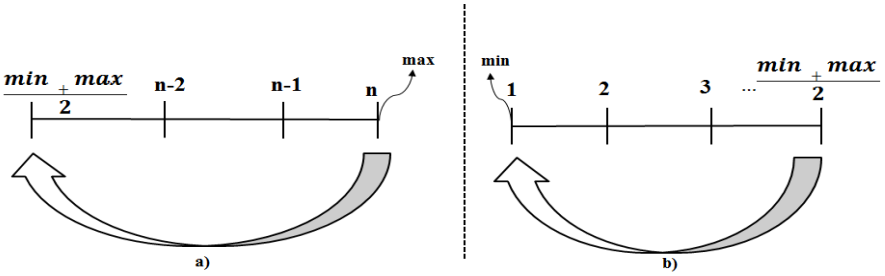

Figure 4-7 (a) Estimated availability is less than requested availability at max (b) Estimated availability is less than requested availability at (min + max)/2

When the case is either ii or iii (Figure 4-7 (a) & (b)), the above mentioned procedure is repeated until the estimated availability for the middle point of interval meets the requested availability.

B. Number of entities creation method

Alogorithm-2 Determine the number of entities and number of collocated entities for a type stack
Input: SUT, No of SIs per VM, Set of actual rec of CTs determined using [11], Set of min no of comps determined using [9] and Set of max no of comps from ETF
Output: No of SIs per VM, No of SGs, No of VMs, No of PHs, No of SIs per PH, No of coll comps per SU, No of coll SUs, No of coll VMs

```
1   Begin
2     Initialize Actual rec of CT to false
3     if (SUT is provided by the vendor and max no of comps is specified) then
4       if ( max no of comps in Set of max no of comps is lesser than min no of comps in
        Set of min no of comps )then
5         Discard type stack
6         Break
7       end if
8     end if
9     for each ct in Set of actual rec of CTs do
10      if (ct.Actual recovery is not equal to Component restart or Component failover) then
11        Actual rec of CT = true
12      end if
13    end for
14    if (Actual rec of CT is equal to true OR redundancy model is equal to No-redundancy ) then
15      No of SIs per SU = 1
16    else
17      Calculate No of SIs per SU using Equation (5)
18    end if
19    Calculate the No of comps per SU using Equation(6-7)
20    for each ct in a SUT do
21      Calculate ct.No of SGs using Equation (8-24) and add it to the Set of no of SGs
22    end for
23    Calculate No of SGs by considering the max (Set of no of SGs)
24    Calculate the No of SIs per VM, No of VMs,No of PHs, No of SIs per PH,
      No of coll comps per SU, No of coll SUs, No of coll VMs using Equation (25-41)
25    Return No of SIs per VM, No of SGs, No of VMs, No of PHs, No of SIs per PH,
      No of coll comps per SU and No of coll SUs, No of coll VMs
26  End
```

Depending on the *No of SI per VM*, the number of entities and the number of collocated entities is determined. Initially, Algorithm-2 checks if a SU is able to provide at least one SI or limited by the SU type (lines 3-8), if given. A vendor delivering a SU type may restrict the number of components per component type (*Max no of comps per CT*) that can be put together in a SU. *Min no of comps per CT* denotes the minimum number of components per component type required to provide a SI [9]. It is calculated based on the *No of CSIs per CST* and active and standby capability of the components. If the *Min no of comps per CT* is greater than the *Max no of comps per CT*, then a SU cannot be formed. Therefore, this type stack is discarded. Next, the *No of SIs per SU* and the number of components required to form a SU (*No of comps per SU*) is determined (lines 9-19). The *No of SGs* calculation is done per component type and the one that results in greater number of SGs is considered (lines 20-23) [9]. Finally, the *No of SIs per VM*, the number of VMs (*No of VMs*), the number of physical hosts (*No of PHs*) required to deploy the SGs and also the number of collocated entities is determined (line 24).

a. Determine the number of SIs per SU

Since a SU can group components serving one or multiple SIs, we can group components in multiple ways to form a SU. There are two extremities: 1) the SU serving a single SI; 2) the SU serving the maximum number of SIs.

i. The SU serving a single SI

A SU may contain components belonging to different types and each of them may recover based on their configured recovery action. The actual recovery of components in the context of configuration is determined using the actual recovery algorithm defined in [11]. When the actual recovery of any one of the components composing the SU is not component restart or component failover, then single SI per SU solution is preferred. For example, if a SU has two types of

components that recover with component restart and SU restart respectively. If many components that recover with component restart fail frequently, the impact is only at the component level. That is, component restart recovery action will not affect the availability of other SIs provided by that SU. On the other hand, if one component that recover with SU restart fails, then the entire SU will be restarted and all SIs served by the SU would be impacted as shown in Figure 4-8. The number of impacted SIs can be reduced by reducing the number of SIs the SU serves. That is grouping into the SU only component(s) required for one SI. This is a cost-effective approach to minimize the interference between collocated components because SUs serve as a fault-isolation unit, but they are only logical groupings. They do not imply any overhead as opposed to VMs.

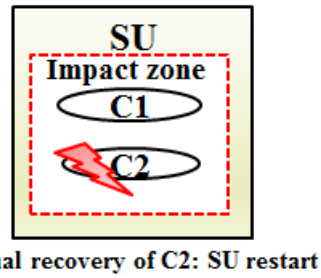

Figure 4-8 *Impact zone when actual recovery is SU restart*

If a SG type is following No-redundancy redundancy model, then SUs in that SG can take at most one SI assignment [10]. In this case also, the *No of SIs per SU* is one.

ii. ***The SU serving the maximum number of SIs***

This solution is preferred if the actual recovery of all the component types in a SU is at the component level. If the actual recovery of components is component failover, then the impact of the component failure will not affect the other SIs provided by a SU. If the actual recovery of the components in the SU is component restart, then there is only a small probability that the recovery action will escalate – due to repeated failures – from component restart to the SU and VM levels. As a result, in this case the maximum number of components that can be grouped together is

preferred. *No of SIs per SU* may be limited by the VM flavor, or by the SU type provided by the vendor or the *No of SIs* as shown in Equation (5).

$$No\ of\ SIs\ per\ SU = min\left(\begin{matrix} min_{1\le j\le N}\, floor\left(\dfrac{Max\ no\ of\ comps\ per\ CT_j}{Min\ no\ of\ comps\ per\ CT_j}\right), \\ No\ of\ SIs\ per\ VM, No\ of\ SIs \end{matrix}\right) \quad (5)$$

Next, depending on the *No of SIs per SU*, the number of components in a SU is calculated using Equation (6) and (7). Note that, in Equation (5) and (6) *j* iterates through the *N* component types in a SU.

$$No\ of\ comps\ per\ SU = \sum_{j=1}^{j\le N} No\ of\ comps\ per\ CT_j \quad (6)$$

$$No\ of\ comps\ per\ CT = No\ of\ SIs\ per\ SU * Min\ no\ of\ comps\ per\ CT \quad (7)$$

b. Determine the number of SUs per SG and the number of SGs

The *No of SUs per SG* and the *No of SGs* are determined based on the redundancy model of the SG type. Except for the No-redundancy redundancy model, in each redundancy model the redundancy is considered on the service side as well as on the service provider side [10]. The number of active and standby assignments per SI defines the redundancy on the service side and the number of SUs per SG defines the redundancy on the service provider side. The redundancy requirement from the service side dictates the number of redundant service providers. Note that Equations (11-18) and (20) are obtained from [9].

i. For the No-redundancy redundancy model, each SI has at most one active assignment and no standby assignments. This redundancy model does not have redundancy on the service side. The redundancy on the service provider side is ensured by having spare SUs in the SG. As

long as there are enough spare SUs to protect the SIs against failure, this redundancy model does not require all the SUs of a SG to be hosted on different VMs and physical hosts. The *No of SUs per SG* required is determined using Equation (8). To provide the entire SIs specified in the CR, the *No of SGs* required is calculated using Equation (9).

$$No\ of\ SUs\ per\ SG = \min(No\ of\ SIs\ per\ VM, No\ of\ SIs) * 2 \quad (8)$$

$$No\ of\ SGs = ceil\left(\frac{No\ of\ SIs}{No\ of\ SIs\ per\ VM}\right) \quad (9)$$

ii. In the 2N redundancy model, each SI has at most one active assignment and one standby assignment [10]. At run-time, AMF assigns all the active assignments of all the SIs to one SU in the SG – which becomes the active SU – and all the standby assignment to another – the standby SU. In the event of a failure of the active SU, the standby SU takes over the active role and starts providing all the SIs [10]. This implies that, a minimum of two SUs is required in a SG to provide and protect a SI (Equation (10)). The next step is to analyze the maximum number of active and standby assignments a SU can handle (Equation (11) and (12)). The *No of comps per CT* used in Equations (11 and 12) represents the number of components per component type in a SU, calculated using Equation (7). Also, *act cap per CST* represents the active capability of a component type to provide a service and *std cap per CST* denotes the standby capability of a component type. The *No of CSIs per CST* represents the number of CSIs in a SI. The *Max no of SIs per SG* is calculated using Equation (13). It is the minimum of *Max no of SIs per act SU* and *Max no of SIs per std SU*. Finally, using Equation (14), the *No of SGs* is calculated based on the *No of SIs* specified in the CR and *Max no of SIs per SG*.

$$No\ of\ SUs\ per\ SG = 2 \quad (10)$$

$$Max\ no\ of\ SIs\ per\ act\ SU = floor\left(\frac{No\ of\ comps\ per\ CT * act\ cap\ per\ CST}{No\ of\ CSIs\ per\ CST}\right) \quad (11)$$

$$Max\ no\ of\ SIs\ per\ std\ SU = floor\left(\frac{No\ of\ comps\ per\ CT * std\ cap\ per\ CST}{No\ of\ CSIs\ per\ CST}\right) \quad (12)$$

$$Max\ no\ of\ SIs\ per\ SG = Min(Max\ no\ of\ SIs\ per\ act\ SU, Max\ no\ of\ SIs\ per\ std\ SU) \quad (13)$$

$$No\ of\ SGs = ceil\left(\frac{No\ of\ SIs}{Max\ no\ of\ SIs\ per\ SG}\right) \quad (14)$$

iii. In the N+M redundancy model, each SI has an active assignment and a standby assignment [10]. As opposed to the 2N redundancy model, this model allows for N active SUs and M standby SUs in the SG [10]. Therefore, to determine the N and M numbers of active and standby SUs of a SG, initially the total number of active SUs and standby SUs are calculated using Equations (15) and (16). The *Max no of SIs per act SU* and *Max no of SIs per std SU* in Equation (15) and (16) are calculated using Equation (11) and (12) respectively. Equation (17) and (18) represents the active and standby SUs proportion that can be used to construct a SG. In [9], the *No of SUs per SG* is constructed in such a way that the *Act proportion* and *Std proportion* does not exceed the number of nodes (number of nodes is given as input [9]). In contrast, in our approach, since the number of nodes is not given as input, the *No of SUs per SG* is the sum of active and standby SUs proportion as shown in Equation (19). This represents the minimum number of redundant entities required i.e. the number of VMs in a VM group or the number of redundant physical hosts. Finally the *No of SGs* is calculated using Equation (20).

$$No\ of\ act\ SUs = ceil\left(\frac{No\ of\ SIs}{Max\ no\ of\ SIs\ per\ act\ SU}\right) \quad (15)$$

$$No\ of\ std\ SUs = ceil\left(\frac{No\ of\ SIs}{Max\ no\ of\ SIs\ per\ std\ SU}\right) \quad (16)$$

$$Act\ proportion = \left(\frac{No\ of\ act\ SUs}{min(No\ of\ act\ SUs, No\ of\ std\ SUs\ )}\right) \quad (17)$$

$$Std\ proportion = \left(\frac{No\ of\ std\ SUs}{min(No\ of\ act\ SUs, No\ of\ std\ SUs\ )}\right) \quad (18)$$

$$No\ of\ SUs\ per\ SG\ = Act\ proportion + Std\ proportion \quad (19)$$

$$No\ of\ SGs = ceil\left(\frac{No\ of\ act\ SUs + No\ of\ std\ SUs}{No\ of\ SUs\ per\ SG}\right) \quad (20)$$

iv. In the N-way-active redundancy model, each SI has two or more active assignments and no standby assignments [10]. The required number of active assignments (*No of active assignments*) is specified in the CR and it is assumed that all the SIs have the same number of active assignments. At run-time, AMF assigns each active assignment of a SI to a different SU in the SG [10]. In the event of a failure of any one of the SUs, the service is not interrupted as the service – the SI – is still provided by the other SUs active for the SI in the SG [10]. Therefore, *No of SUs per SG* is equal to the *No of active assignments* as shown in Equation (21). The *No of SGs* is calculated using Equation (14). However, *Max no of SIs per SG* used in Equation (14) is calculated using Equation (22).

$$No\ of\ SUs\ per\ SG = No\ of\ active\ assignments \quad (21)$$

$$Max\ no\ of\ SIs\ per\ SG = min\ (Max\ no\ of\ SIs\ per\ act\ SU, No\ of\ SIs) \quad (22)$$

v. For the N-way redundancy model, each SI has one active assignment and one or more standby assignments. The required number of standby assignments (*No of std assignments*) is specified

in the CR. It is assumed that each SI has the same number of standby assignments. The *No of SUs per SG* is equal to sum of the *No of std assignments* and one for the active assignment and one for the spare SU as shown in Equation (23). The *No of SGs* is calculated using Equation (14). However, the *Max No of SIs per SG* used in Equation (14) is calculated using Equation (24).

$$No\ of\ SUs\ per\ SG = One\ active\ assignment + No\ of\ std\ assignments + 1 \quad (23)$$

$$Max\ no\ SIs\ per\ SG = min\,(Max\ no\ of\ SIs\ per\ act\ SU * (No\ of\ SUs\ per\ SG - 1),\ Max\ no\ of\ SIs\ per\ std\ SU, No\ of\ SIs) \quad (24)$$

c. *Determine the number of SUs per VM and the number of SIs per VM*

The number of SUs per VM (*No of SUs per VM*) is the minimum of maximum number of SUs a VM can host based on its capacity and the *No of SGs* as shown in the Equation (25). For No-redundancy model, Equation (26) is used to calculate the *No of SUs per VM.* Based on the *No of SUs per VM* and the *No of SIs per SU,* the *No of SIs per VM* is calculated as shown in the Equation (27). It is important to recalculate the *No of SIs per VM* due to the distribution of SIs to the SUs in a VM vertically and also to the redundant SUs in SGs horizontally.

$$No\ of\ SUs\ per\ VM = min\left(floor\left(\frac{No\ of\ SIs\ per\ VM}{No\ of\ SIs\ per\ SU} \right), No\ of\ SGs \right) \quad (25)$$

For No-redundancy model,

$$No\ of\ SUs\ per\ VM = \min(No\ of\ SIs\ per\ VM, No\ of\ SIs) \quad (26)$$

$$No\ of\ SIs\ per\ VM = No\ of\ SUs\ per\ VM * No\ of\ SIs\ per\ SU \quad (27)$$

d. Determine the number of VM groups, the number of VMs and the number of physical hosts

The total physical memory (*TPM*) of a host is used by the hypervisor, host OS and by the VMs residing on that host. Therefore, next to calculate the physical memory that is available (*APM*) to host the VMs, the memory required for the hypervisor and the host OS are excluded from the *TPM* as shown in the Equation (28). The number of VMs a physical host can host (*No of VMs per PH*) is calculated based on the *APM*, the *TGM* of the VM flavor and an overbooking factor using Equation (29). The overbooking factor indicates to what extent the number of VMs per physical host can be increased by serializing their execution [33]. The *No of VM groups* i.e. AMF node groups is calculated based on the *No of SGs* and the *No of SUs per VM* as shown in Equation (30). For No-redundancy model, Equation (31) is used to calculate the *No of VM groups*.

$$APM = TPM - (HOSM + HM) \tag{28}$$

$$No\ of\ VMs\ per\ PH = floor\left(\frac{APM * OBF}{TGM}\right) \tag{29}$$

$$No\ of\ VM\ groups = ceil\left(\frac{No\ of\ SGs}{No\ of\ SUs\ per\ VM}\right) \tag{30}$$

For No-redundancy model,

$$No\ of\ VM\ groups = ceil\left(\frac{No\ of\ SIs}{No\ of\ SIs\ per\ VM}\right) \tag{31}$$

The maximum number of VMs per physical host (*Max no of VMs per PH*) is the minimum of the *No of VMs per PH* required and the *No of VM groups* (Equation (32)). The *No of SUs per*

PH is the product of *Max no of VMs per PH* and the *No of SUs per VM* as shown in the Equation (33).

$$Max\ no\ of\ VMs\ per\ PH = min\{No\ of\ VMs\ per\ PH, No\ of\ VM\ groups\} \quad (32)$$

$$No\ of\ SUs\ per\ PH = Max\ no\ of\ VMs\ per\ PH * No\ of\ SUs\ per\ VM \quad (33)$$

By multiplying the *Max no of VMs per PH* and the *No of SIs per VM*, the *No of SIs per PH* determined (Equation (34)). The total *No of VMs* and the total *No of PHs* required to deploy the SGs are calculated using Equations (35-37) respectively. *K* denotes the number of redundant entities i.e. redundant VMs per VM group and the number of redundant physical hosts. For No-redundancy redundancy model, the number of redundant entities required is two (*K*=2). However, for the other redundancy models, the *No of SUs per SG* determines the number of redundant entities.

$$No\ of\ SIs\ per\ PH = Max\ no\ of\ VMs\ per\ PH * No\ of\ SIs\ per\ VM \quad (34)$$

$$No\ of\ VMs = No\ of\ VM\ groups * K \quad (35)$$

$$No\ of\ PHs = ceil\left(\frac{No\ of\ SGs}{No\ of\ SUs\ per\ PH}\right) * K \quad (36)$$

For No-redundancy redundancy model,

$$No\ of\ PHs = ceil\left(\frac{No\ of\ SIs}{No\ of\ SIs\ per\ PH}\right) * K \quad (37)$$

e. Determine the number of collocated entities

A SU may host components serving one or multiple SIs. If availability is estimated for a SI in a SU, then components that provide other SIs but hosted in the same SU are called collocated

components. SUs that are serving other SIs but hosted in the same VM are called collocated SUs and VMs that are providing other SIs but collocated in the same physical host are called collocated VMs. The *No of coll comps per SU* is calculated using Equation (38). The number of collocated components for each component type in a SU is calculated by excluding the components that provides one SI from the components in a SU (Equation (39)). Using Equation (40), the number of collocated SUs in a VM (*No of coll SUs*) is calculated by excluding one SU whose SI's availability is being estimated. Similarly, the number of collocated VMs (*No of coll VMs*) is determined by excluding one VM from the *Max No of VMs per PH* as shown in Equation (41).

$$No\ of\ coll\ comps\ per\ SU = \sum_{j=1}^{j \le N} No\ of\ coll\ comps\ per\ CT_j \quad (38)$$

j iterates through the N component types in a SU

$$No\ of\ coll\ comps\ per\ CT = No\ of\ comps\ per\ CT - Min\ no\ of\ comps\ per\ CT \quad (39)$$

$$No\ of\ coll\ SUs = No\ of\ SUs\ per\ VM - 1 \quad (40)$$

$$No\ of\ coll\ VMs = Max\ no\ of\ VMs\ per\ PH - 1 \quad (41)$$

C. Availability estimate method

The service availability is calculated per service instance (SI). The calculations we show are for estimating the availability for a SI of a given service type.

$$Availability\ of\ SI = A_{components} * A_{infrastructure} * A_{collocated\ entities} \quad (42)$$

As shown in Equation (42), availability of the service broadly depends on the following factors: A) availability of the components providing the service; B) availability of virtual and physical infrastructure; C) interference caused by the collocated entities.

To calculate the availability of each entities in the system, two factors are required: MTTF and MTTR (Mean Time To Repair). From a service perspective, MTTF is the mean time that an element takes to fail while the MTTR is the mean time required to recover the service provided by the failed element [1]. Once the MTTF and MTTR are known, the availability of the service can be determined using Equation (43).

$$Availability = \frac{MTTF}{MTTF + MTTR} \tag{43}$$

a. Availability of the components providing the service

For availability due to the failure of components, it is assumed that software vendors provide the MTTF for each component type, which may be a result of benchmark analysis. To calculate the time required to recover the service due to the failure of the components ($MTTR_{component}$), actual recovery actions of the components in the context of the configuration is analyzed [11]. Based on this actual recovery action, estimated time to recover the service is calculated [9]. Equation (44) is used to calculate the availability of components providing a SI.

$$A_{components} = \prod_{j=1}^{j \le N} \left(\frac{MTTF_{component_j}}{MTTF_{component_j} + MTTR_{component_j}} \right)^{p_j} \tag{44}$$

j iterates through N component types in a SU type. $MTTF_{component}$ and $MTTR_{component}$ represents the mean time for a component type to fail and time required to recover the SI respectively. p_j is the required number of components of a component type to provide one SI.

b. ***Availability of the infrastructure***

As shown in Equation (45), the availability of the infrastructure is calculated as the product of availability of the virtual infrastructure (A_{vi}) and availability of the physical infrastructure (A_{pi}).

$$A_{infrastructure} = A_{vi} * A_{pi} \quad (45)$$

A_{vi} is calculated as the product of availability of the VM (A_{vm}) and availability of the guest OS ($A_{guestOS}$) as shown in Equation (46). As mentioned before, *MTTF* of the VM and guest OS are obtained from the infrastructure file. While calculating $MTTR_{infra}$, it is assumed that the SIs are failed over to another healthy VM hosted on a redundant host. As shown in Equation (48), $MTTR_{infra}$ is the time required to detect a VM failure (*Detection time*) and also to recover the service from the failed VM (*Failover time*). It should be noted that VM failure is detected by the Cluster Membership Service (CLM) [31] or by the failure detection mechanism in the infrastructure.

$$A_{vi} = A_{vm} * A_{guestOS} \quad (46)$$

$$A_{vi} = \left(\left(\frac{MTTF_{vm}}{MTTF_{vm} + MTTR_{infra}} \right) * \left(\frac{MTTF_{guestOS}}{MTTF_{guestOS} + MTTR_{infra}} \right) \right) \quad (47)$$

$$MTTR_{infra} = Detection\ \ time + Failover\ \ time \quad (48)$$

$$Failover time = Max_{1 \le j \le N}(CSS_j) \quad (49)$$

Here, failover is assumed to occur in parallel, therefore the failover time is the maximum time required to set HA state assignment to components (*CSS*). *j* iterates through *N* component types hosted per VM.

A_{pi} (Equation (50)) is calculated as the product of availability of the physical hardware (A_{ph}), availability of the hypervisor ($A_{hypervisor}$) and availability of the host OS (A_{hostOS}).When any of these (physical hardware, hypervisor or host OS) fail, it is assumed that each VM is failed over independently to another healthy VM hosted on a redundant host. $MTTR_{infra}$ is the time required to detect the VM failure and the time required to failover the SIs from that failed VM as shown in the Equation (48). The MTTF values ($MTTF_{ph}$, $MTTF_{hostOS}$ *and* $MTTF_{hypervisor}$) are obtained from infrastructure file.

$$A_{pi} = A_{ph} * A_{hypervisor} * A_{hostOS} \tag{50}$$

$$A_{pi} = \left(\left(\frac{MTTF_{ph}}{MTTF_{ph} + MTTR_{infra}} \right) * \left(\frac{MTTF_{hostOS}}{MTTF_{hostOS} + MTTR_{infra}} \right) * \left(\frac{MTTF_{hypervisor}}{MTTF_{hypervisor} + MTTR_{infra}} \right) \right) \tag{51}$$

c. Availability of the service due to interferences of collocated entities

The failure of collocated components may affect the availability of other SIs provided in that environment. Considering the example illustrated in Figure 4-9, the availability of SI 1 is affected by the failure of collocated component C2 or SU2 or VM 2. The following sub-sections show the calculations of the availability for: the components collocated in a SU, the SUs collocated in a VM and for the VMs collocated in a physical host.

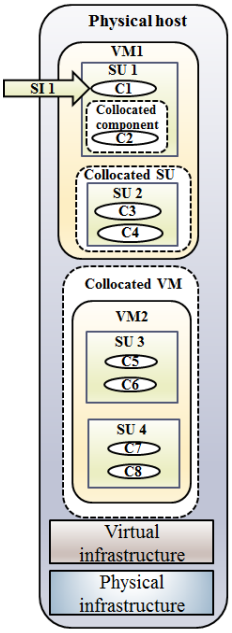

Figure 4-9 *Availability of a SI*

i. Availability due to collocated components interferences

When any one of the collocated components fails, there is a probability that the recovery action is escalated to SU restart or SU failover or VM failover or VM reboot. To calculate the availability of collocated components due to the interferences, the following probabilities are calculated.

a) Probability of escalating the recovery action to SU restart

b) Probability of escalating the recovery action to SU failover

c) Probability of escalating recovery action to VM failover

d) Probability of escalating recovery action to VM reboot

a) Probability of escalating recovery action to SU restart

For the first time when a component fails, AMF performs component restart recovery action. When too many components of the SU need to be restarted it is unlikely that the components carry the fault. In order to capture the fault, along with the failed components, its

siblings are also restarted (Level 1 escalation) [10]. Note that the level 1 escalation is applicable only if all the components in a SU are restartable.

In particular, level 1 escalation is activated when the maximum number of allowed components restarts is reached in a time period. To calculate the probability of maximum number of component failures (x) occurring in a time period (t), Poisson distribution [32] can be used as shown in Equation (52). Since component restarts occur only after the components have failed, the probability of maximum number of component restarts occurring in a probation time can be calculated using Equation (53).

$$P(x) = \left(\frac{e^{-\lambda t} \lambda t^{x}}{x!} \right) \tag{52}$$

For a SU with restartable components,

$$P(escalation\ to\ SU\ restart) = P(x)_{level\ 1} = \left(\frac{e^{-\lambda t_{level\ 1}} \lambda t_{level\ 1}{}^{x_{level\ 1}}}{x_{level\ 1}!} \right) \tag{53}$$

In Equation (53), λ represents the failure rate of the components and $x_{level\ 1}$ denotes the maximum number of allowed component restarts in a time period $t_{level\ 1}$. If there are N component types in a SU type and p_i represents the number of components per component type then the failure rates of the components are added up as shown in Equation (54).

$$\lambda = \sum_{i=1}^{i \le N} p_i \lambda_i = \sum_{i=1}^{i \le N} p_i * \left(\frac{1}{MTTF_{component_i}} \right) \tag{54}$$

b) Probability of escalating recovery action to SU failover

Since level 1 escalation does not guarantee the resolution of the fault, further escalation levels are considered: Once level 1 escalation has been activated for a SU, whenever one of its

components fails, the component is restarted along with its sibling components in that SU. If the components of a SU continue to fail and reach a second threshold within a second probation time period, then the SU restarts deemed to be futile and the recovery action is escalated to SU failover, i.e. level 2 escalation is reached [10]. Note that the level 2 escalation is applicable only if all the components in a SU are restartable.

The probability of maximum number of allowed SU restarts occurring in a probation time $P(x)_{level2}$ is calculated using Equation (55) where x_{level2} and t_{level2} parameters are used as defined for level 2 escalation. *P (escalation to SU failover)* is calculated by multiplying the $P(x)_{level2}$ and the probability that the SU was already in level 1, *P (escalation to SU restart)* as shown in Equation (56).

For a SU with only restartable components,

$$P(x)_{level\,2} = \left(\frac{e^{-\lambda t_{level\ 2}} \lambda t_{level\,2}{}^{x_{level\ 2}}}{x_{level\,2}!} \right) \qquad (55)$$

$$P(escalation\ to\ SU\ failover) = P(x)_{level2} * P(escalation\ to\ SU\ restart) \qquad (56)$$

c) Probability of escalating recovery action to VM failover

When a component of a SU on which level 2 escalation is active fails, then the SU is failed over. When the maximum number of permitted SU failovers (failover of SUs residing on the same VM) is reached within a time period $t_{level\ 3}$, then AMF assumes that the VM is faulty and it will failover the VM [10]. The probability of maximum number of allowed SU failover $P(x)_{level3}$ occurring in a time period $t_{level\ 3}$ is calculated using Equation (57). $x_{level\ 3}$ and $t_{level\ 3}$ parameters are used as defined for level 3 escalation.

$$P(x)_{level\,3} = \left(\frac{e^{-\lambda t_{level\,3}} \lambda t_{level\,3}^{\;x_{level\,3}}}{x_{level\,3}!} \right) \quad (57)$$

If a SU has only restartable components, *P (escalation to VM failover)* is calculated by multiplying $P(x)_{level3}$ and *P (escalation to SU failover)*. *P (escalation to SU failover)* is obtained from Equation (58).

$$P(escalation\ to\ VM\ failover) = P(x)_{level\,3} * P(escalation\ to\ SU\ failover) \quad (58)$$

If a SU has at least one non-restartable component, the failure of a component itself, triggers the SU-failover. In that case, *P (escalation to VM failover)* is calculated using Equation (59).

$$P(escalation\ to VM\ failover) = P(x)_{level\,3} \quad (59)$$

d) Probability of escalating recovery action to VM reboot

During component restart recovery action, AMF cleans up the faulty component and then it tries to re-instantiate the component. However, if the cleanup action is unsuccessful or if the all the allowed attempts of instantiation fail, then AMF assumes the fault is in the VM and escalates the recovery action to VM reboot [10] and the services provided by the VM are impacted. In this case, the probability of escalating recovery action to VM reboot is calculated.

The restart recovery action may fail either during cleanup or while instantiating a component with delay or while instantiating a component without delay [10]. In Equation (60), *P (CF)* represents the probability of cleanup failures occurring during the component restart recovery action. When the cleanup action fails, the recovery action is escalated to VM reboot. If the cleanup action is successful for the first time, but if the instantiation attempt fails, again the cleanup action

is performed before attempting to instantiate a component. Therefore, there is a probability that the cleanup actions may fail while attempting to instantiate components with delay i.e. *P (CFIWOD),* calculated using Equation (62) or without delay i.e. *P (CFIWD),* calculated using Equation (63). As shown in Equation (61), the total number of instantiation attempts *nia* is given by the sum of *NIWOD* (Number of instantiation attempts without delay) and *NIWD* (Number of instantiation attempts with delay). Note that part of the Equations (61-64) are taken from [9].

$$P(CF) = P(CFIWOD) + P(CFIWD) \tag{60}$$

$$nia = NIWOD + NIWD \tag{61}$$

$$P(CFIWOD) = \sum_{i=1}^{NIWOD} PCS^{i-1} * PINS^{i-1} * PCNS \tag{62}$$

$$P(CFIWD) = PINS^{NIWOD} * \left[\sum_{i=NIWOD+1}^{nia} PCS^{i-1} * PINSD^{(i-NIWOD-1)} * PCNS \right] \tag{63}$$

PCS and *PCNS* represents the probability of cleanup successful and failure respectively. *PINS* and *PINSD* denotes the probability of instantiation not successful without and with delay respectively.

Even though the cleanup is successful, there is a probability that all the instantiation attempts may fail and this is calculated using Equation (64). The probability of escalating the recovery action to VM reboot due to instantiation or termination failure is calculated using Equation (65).

$$P(IF) = PCS^{nia} * PINS^{NIWOD} * PINSD^{NIWD} \tag{64}$$

$$P(escalation\,to\,VM\ reboot) = P(CF) + P(IF) \tag{65}$$

Equation (66) calculates the $MTTF_{int\ of\ comps}$ for a component type based on the probability of escalating the recovery actions to SU level or VM level.

$$MTTF_{int\ of\ comps} = \frac{MTTF_{component}}{\begin{pmatrix} P(escalation\ to\ SU\ restart) + P(escalation\ to\ SU\ failover) + \\ P(escalation\ to\ VM\ failover) + P(escalation\ to\ VM\ reboot) \end{pmatrix}} \quad (66)$$

Once $MTTF_{int\ of\ comps}$ is calculated, then the next step is to calculate the time required to perform VM failover and VM reboot recovery actions. To failover the SIs from a VM, primarily the components are cleaned up and if the cleanup action is successful, then the failed component's CSI is failed over and the healthy component's CSI are switched over [10]. Switch over is a smooth transition of CSI. In Equation (67), the time required to perform VM failover is sum of the maximum time required to perform cleanup action and the maximum time required to perform failover action. Since the cleanup action for all components are executed in parallel, the maximum time required is considered as the cleanup time. Similarly, the maximum time required to perform failover action is considered as the failover time [9]. Note that in the below Equation (67), *j* iterates through the *N* component types.

$$T_{VM\ failover} = Max_{1 \le j \le N}[Cleanup\ time_j] + Max_{1 \le j \le N}[Failover\ time] \quad (67)$$

In Equation (68), *clt* represents the time required for a component type to perform cleanup action. The cleanup probabilities and the cleanup time are described in the extended ETF [9]. Note that, if the cleanup action fails then the VM is rebooted. $T_{VM\ reboot}$ denotes the VM reboot time. It is calculated using Equation (69) [9]. *NST* represents the time required by a VM to shut down and it is described in the infrastructure file. *CSS* represents the time required to set the HA assignment state for a component belonging to a component type. Equation (70) gives the maximum failover time required by all the assignments provided by a VM. *SOT* represents the time required by

components to switch over the active assignments to healthy components hosted on another VM. It is calculated using Equation (71).

$$Cleanup\ time = [PCS * clt] + [PCNS * (clt + T_{VM\ reboot})] \quad (68)$$

$$T_{VM\ reboot} = NST + Max_{1 \le j \le N}(CSS_j) \quad (69)$$

$$Failover\ time = Max_{l \le j \le N}[PCS_j * Max_{1 \le j \le N}(CSS_j, SOT_j)] \quad (70)$$

$$SOT = 2 * CSS \quad (71)$$

For each component belonging to a component type, there is a probability that the recovery action is SU restart or SU failover or VM failover or VM reboot. Equation (72) is used to calculate the mean time to recover the service due to the interference of the collocated components $MTTR_{int\ of\ comps}$. Time required to perform SU restart $T_{SU\ restart}$, time required to perform SU failover $T_{SU\ failover}$ are calculated using [9]. Time required to perform VM failover $T_{VM\ failover}$ and the time required to perform VM reboot $T_{VM\ reboot}$ are calculated using Equations (67) and (69) respectively.

$$MTTR_{int\ of\ comps} = \begin{pmatrix} (P(escalation\ to\ SU\ restart) * T_{SU\ restart}) + \\ (P(escalation\ to\ SU\ failover) * T_{SU\ failover}) + \\ (P(escalation\ to\ VM\ failover) * T_{VM\ failover}) + \\ (P(escalation\ to\ VM\ reboot) * T_{VM\ reboot}) \end{pmatrix} \quad (72)$$

$$A_{collocated\ components\ in\ a\ SU's\ interference} = \prod_{j=1}^{j \le N} \left(\frac{MTTF_{int\ of\ comps\ j}}{MTTF_{int\ of\ comps\ j} + MTTR_{int\ of\ comps\ j}} \right)^{n_j} \quad (73)$$

Equation (73) is used to calculate the availability of collocated components in a SU. $MTTF_{int\ of\ comps}$ is calculated using Equation (66) and $MTTR_{int\ of\ comps}$ is calculated using Equation (72). j iterates through the N component types in a SU and n_j represents the number of collocated

components per component type in a SU. If there are no collocated components in a SU, then the availability due to collocated components is one.

ii. Availability due to collocated SUs interferences

It is possible that availability of SI may be affected when recovery action is performed at the VM level due to collocated SUs in a VM. Here, $MTTF_{int\ of\ SUs}$ is calculated using Equation (74) and the mean time to recover the service due to the interference of the collocated SUs ($MTTR_{int\ of\ SUs}$) is calculated using Equation (75). Equation (76) is used to calculate the availability of collocated SUs in a VM. N represents the number of component types in a SU type, p_j denotes the required number of components per component type and r is the number of collocated SUs in a VM. Note that if there are no collocated SUs in a VM, then the availability due to collocated SUs interference is one.

$$MTTF_{int\ of\ SUs} = \frac{MTTF_{component}}{P(escalation\ to\ VM\ \ failover) + P(escalation\ to\ VM\ \ reboot)} \tag{74}$$

$$MTTR_{int\ of\ SUs} = \begin{pmatrix} \left(P(escalation\ to\ VM\ \ failover) * T_{VM\ \ failover}\right) + \\ \left(P(escalation\ to\ VM\ \ reboot) * T_{VM\ reboot}\right) \end{pmatrix} \tag{75}$$

$$A_{collocated\ SUs\ in\ a\ vm's\ interference} = \left(\prod_{j=1}^{j \le N} \left(\frac{MTTF_{int\ of\ SUs\ j}}{MTTF_{int\ of\ SUs\ j} + MTTR_{int\ of\ SUs\ j}} \right)^{p_j} \right)^{r} \tag{76}$$

iii. Availability due to collocated VMs interference

A fault in the physical hardware, or in the host operating system, or in the hypervisor will affect all the services provided by the application components running on that host. Such faults may or may not cause the failure of the faulty entity itself. They may propagate to one of the hosted entities and cause it to fail. As a result when components fail due to one of the above mentioned faults,

AMF cannot identify the source of the failure, the faulty entity, it is not even aware of the fact that the node on which it manages the components are VMs deployed on physical hosts, but potentially collocated. AMF will failover services provided by components of the other VMs residing on that physical host. AMF considers these failures to be independent and therefore the recovery action is taken per VM. However these failures are dependent (e.g. due to physical hardware fault) and physical hardware reboot could solve this issue, but AMF performs VM failovers independently.

To handle this issue, it is assumed that the same escalation is applied for the VMs as for the components. When the maximum number of permitted VM failovers or VM reboot is reached within a time period, physical hardware reboot is performed. As a result, a service is affected when the collocated VMs trigger reboot of the physical host it is hosted on. If N represents the number of component types in a SU, p_j denotes the number of components of a component type, $r+1$ is the total number of SUs in a VM and s is the number of collocated VMs in a physical host then, Equation (77) is used to calculate $A_{collocatedVM'sinterference.}$ Note that $MTTF_{int\ of\ SUs}$ is calculated using Equation (74) and $MTTR_{int\ of\ SUs}$ is calculated using Equation (75). Note that if there are no collocated VMs in a physical host, then the availability due to collocated VMs interference is one.

$$A_{collocated\ VMs\ in\ a\ ph's\ interference} = \left(\left(\prod_{j=1}^{j \le N} \left(\frac{MTTF_{int\ of\ SUs\ j}}{MTTF_{int\ of\ SUs\ j} + MTTR_{int\ of\ SUs\ \ j}} \right)^{p_j} \right)^{r+1} \right)^{s} \quad (77)$$

From (73), (76) and (77)

$$A_{collocated\ entities} = \begin{pmatrix} A_{collocated\ components\ in\ a\ SU's\ interference} \\ * A_{collocated\ SUs\ in\ a\ vm's\ interference} \\ * A_{collocated\ VMs\ in\ a\ ph's\ interference} \end{pmatrix} \quad (78)$$

Finally substituting Equations (44), (45) and (78) in Equation (42) gives the estimated availability of a SI.

D. Example illustrating the calculation of number of SIs per VM flavor from the perspectives of availability and resource utilization for a type stack

Let us assume a service type ST A is composed of two component service types CST 1 and CST 2. The information related to the number of SIs of ST A, the number of CSIs per CST1 and CST 2 in each SI and the requested availability is obtained from the configuration requirements. Based on the ETF model and the configuration requirements the type stacks are formed using [9]. The following example, illustrates the first phase in the AMF entities creation step for a type stack.

From configuration requirements

- Number of SIs of ST A = 40
    - Number of CSIs of CST 1 = 2
    - Number of CSIs of CST 2 = 3
- Requested availability = 0.999

From ETF

- Capability of components
    - Active capability of CT1 for CST1 = 2
    - Standby capability of CT1 for CST1 = 5
    - Active capability of CT2 for CST2 = 2
    - Standby capability of CT2 for CST2 = 3

- Maximum number of components per SU
    - Max no of comps per CT1 = 80
    - Max no of comps per CT2 = 64

- Memory requirement
    - Memory required per CST1 = 3 MB
    - Memory required per CST2 = 1 MB

From infrastructure file

- Set of VM flavors = { small = 512 MB, medium = 2048 MB}
- Memory used by the hypervisor = 256 MB
- Memory used by the host OS = 72 MB
- Memory used by the guest OS = 72 MB
- Memory of the physical host = 4608 MB
- Over booking factor = 1

The following calculations are illustrated for the small VM flavor. Initially, the memory required for a SI is determined using Equations (1-2).

*Memory required per CST1= Memory required per CSI1 * No of CSIs per CST1 = 3*2 = 6*

*Memory required per CST2= Memory required per CSI2 * No of CSIs per CST2 =1*3 = 3*

Memory required per SI= Memory required per CST1 + Memory required per CST2 = 9 MB

Using Equation (3), the AGM is calculated for the small VM flavor.

AGM =TGM – GOSM =512-72 =440MB

Next, using Equation (4), the *No of SIs per VM* is calculated

$$No\ of\ SIs\ per\ VM = floor\left(\frac{AGM}{Memory\ required\ per\ SI}\right) = floor\left(\frac{440}{9}\right) = 48\ \text{SIs}$$

The small VM flavor can at most host 48 SIs. Next, the availability is estimated for the best and the worst case scenarios using Algorithm-1. From the perspective of availability estimation, the best case scenario is when the *No of SIs per VM* is 1. Using the Algorithm-2 the number of entities and the number of collocated entities are determined.

i. *Determine number of SIs per SU:* Initially, the actual recovery of the components and the minimum number of components per CT is determined using [11] and [9] respectively.

— Actual recovery for both the component types (CT1 and CT2) is component restart.

— Minimum number of components per CT1 required to provide one SI = 1

— Minimum number of components per CT2 required to provide one SI = 2

Since the actual recovery of both the component types is at the component level, the SU serving maximum number of SIs is selected.

$$No\ of\ SIs\ per\ SU = \min\left(\begin{array}{l}\min_{1\le j\le N}\left(\begin{array}{l} floor\left(\dfrac{Max\ no\ of\ comps\ per\ CT1}{Min\ no\ of\ comps\ per\ CT1}\right), \\ floor\left(\dfrac{Max\ no\ of\ comps\ per\ CT2}{Min\ no\ of\ comps\ per\ CT2}\right)\end{array}\right), \\ No\ of\ SIs\ per\ VM, No\ of\ SIs\end{array}\right)$$

$$No\ of\ SIs\ per\ SU = \min\left(\min\left(\frac{80}{1},\frac{64}{2}\right),1,40\right) = 1$$

*No of comps per CT1 =No of SIs per SU*Min no of comps per CT1 =1*1 =1*

*No of comps per CT2 =No of SIs per SU*Min no of comps per CT 2=1*2 =2*

ii. *Determine number of SUs per SG and number of SGs*

From the SG type, the redundancy model is inferred to be N+M. For each component type the *No of SGs* is calculated using Equations (11-12 and 15-20).

	CT1	CT2
Max no of act SIs per SU	1	1
Max no of std SIs per SU	2	2
No of act SUs	40	40
No of std SUs	20	20
Act proportion	2	2
Std proportion	1	1
No of SUs per SG	3	3
No of SGs	20	20

Table 4-1 *Calculation of No of SGs and No of SUs per SG*

iii. *Determine the number of SUs per VM and the number of SIs per VM*

$$No\ of\ SUs\ per\ VM = min\left(floor\left(\frac{No\ of\ SIs\ per\ VM}{No\ of\ SIs\ per\ SU} \right), No\ of\ SGs \right) = min\ (1,\ 20) = 1$$

$$No\ of\ SIs\ per\ VM = No\ of\ SUs\ per\ VM * No\ of\ SIs\ per\ SU = 1$$

iv. *Determine the number of VM groups, the number of VMs and the number of physical hosts*

$$APM = TPM - (HOSM + HM) = 4608 - 328 = 4280\ \text{MB}$$

$$No\ of\ VMs\ per\ PH = floor\left(\frac{APM * OBF}{TGM} \right) = floor\left(\frac{4280 * 1}{512} \right) = 8$$

$$No\ of\ VM\ groups = ceil\left(\frac{No\ of\ SGs}{No\ of\ SUs\ per\ VM} \right) = ceil\left(\frac{20}{1} \right) = 20$$

$$Max\ no\ of\ VMs\ per\ PH = min\{No\ of\ VMs\ per\ PH, No\ of\ VM\ groups\} = min\ (8,\ 20) = 8$$

$$No\ of\ SUs\ per\ PH = Max\ no\ of\ VMs\ per\ PH * No\ of\ SUs\ per\ VM = 8 * 1 = 8$$

$$No\ of\ VMs = No\ of\ VM\ groups * K = 20 * 3 = 60$$

$$No\ of\ PHs = ceil\left(\frac{No\ of\ SGs}{No\ of\ SUs\ per\ PH} \right) * K = ceil\left(\frac{20}{8} \right) * 3 = 9$$

v. *Determine the number of collocated entities*

When the *No of SIs per SU* is one, the *No of coll comps* and the *No of coll SUs* is zero.

No of coll comps per CT1 = No of comps per CT1- Min no of comps per CT1= 0
No of coll comps per CT2 = No of comps per CT2- Min no of comps per CT2= 0

No of coll SUs = No of SUs per VM -1 = 0
No of coll VMs =8-1=7

Using the availability estimate method, the availability of a SI for best case scenario is estimated. The information about the component types and the infrastructure elements are obtained from Table 4-2 and 4-3 respectively.

	CT 1	CT 2
Clean up time (clt) in sec	3	2
CSS time in sec	2	1
Instantiation time (IT) in sec	1	2
PCS	0.4	0.6
PCNS	0.6	0.4
PINS	0.1	0.1
PINSD	0.1	0.1
MTTF	530000	780000
NIWOD	2	2
NIWD	1	1

Table 4-2 *Information about the component types*

Infrastructure elements	MTTF (sec)
Physical host	8300000
Hypervisor	6200000
Host OS	5400000
VM	7100000
Guest OS	5400000

Table 4-3 *Information about the infrastructure elements*

a) Availability due to failure of components

The MTTR calculated using [9] is 3.3 and 2.1 respectively. Using MTTF values from Table 4-3, $A_{components}$ is calculated using Equation (44).

$$A_{components} = \prod_{j=1}^{j \le N} \left(\frac{MTTF_{Componentj}}{MTTF_{Componentj} + MTTR_{componentj}} \right)^{p_j} = \left(\frac{530000}{530000 + 3.3} \right)^{1} * \left(\frac{780000}{780000 + 2.1} \right)^{2} = 0.999990$$

b) Availability of the infrastructure

The MTTF values for the virtual infrastructure elements and the physical infrastructure elements are obtained from Table 4-3.

Using Equation (48), the $MTTR_{infra}$ is calculated.

$MTTR_{infra}$ = *Detection time* +*Failover time* = 3.2 + 2 = 5.2

$$Failovertime = Max_{1 \le j \le N}(CSS_j) = Max(2,1) = 2$$

Using Equation (47), the availability of the virtual infrastructure (A_{vi}) is determined.

$$A_{vi} = \left(\frac{7100000}{7100000 + 5.2} \right) * \left(\frac{5400000}{5400000 + 5.2} \right) = 0.9999982$$

Using Equation (51), the availability of the physical infrastructure (A_{pi}) is determined.

$$A_{pi} = \left(\frac{8300000}{8300000 + 5.2} \right) * \left(\frac{5400000}{5400000 + 5.2} \right) * \left(\frac{6200000}{6200000 + 5.2} \right) = 0.9999974$$

$$A_{infrastructure} = A_{vi} * A_{pi} = 0.9999982 * 0.9999974 = 0.9999957$$

c) Availability due to the collocated components interferences

i. Probability of escalation due to SU restart

When the $t_{level\ 1}$ = 10000 s and $x_{level\ 1}$ = 1

Using Equation (54), the failure rate of the components is determined

$$\lambda = p_{component1}\lambda_{component1} + p_{component2}\lambda_{component2} = (1/530000) + (2/780000) = 4.45*10^{-6}$$

Using Equation (53), the probability of escalating the recovery action to SU restart is determined.

$$P(escalation\ to\ SU\ restart) = \frac{e^{-(4.45*10^{-6})(10000)}[(4.45*10^{-6})(10000)]^1}{1!} = 0.04255$$

ii. Probability of escalation due to SU failover

When $t_{level\ 2}$ = 10000 s and $x_{level\ 2}$ = 1

Using Equation (55), the $P(x)_{level2}$ is determined.

$$P(x)_{level\,2} = \frac{e^{-0.0445}0.0445^1}{1!} = 0.04255$$

Using Equation (56), the probability of escalating the recovery action to SU failover is determined.

$$P(escalation\ to\ SU\ failover) = (0.04255)^2 = 1.81*10^{-3}$$

iii. Probability of escalation due to VM failover

When $t_{level\ 3}$ = 10000 s and $x_{level\ 3}$ = 1

Using Equation (58), probability of escalating the recovery action to VM failover is determined.

For restartable components,

P (escalation to VM failover) = $7.70*10^{-5}$

iv. Probability of escalation to VM reboot

For CT 1,

$nia = NIWOD + NIWD = 2+1=3$

$P(CF) = P(CFIWOD) + P(CFIWD)$

$$P(CFIWOD) = \sum_{i=1}^{NIWOD} PCS^{i-1} * PINS^{i-1} * PCNS = \sum_{i=1}^{2} (0.4)^{i-1} * (0.1)^{i-1} * (0.6) = 0.624$$

$$P(CFIWD) = 0.1^2 * \sum_{i=2+1}^{3} (0.4)^{i-1} * (0.1)^{i-2-1} * (0.6) = 0.00096$$

$$P(CF) = 0.624 + 0.00096 = 0.62496$$

$$P(IF) = PCS^{nia} * PINS^{NIWOD} * PINSD^{NIWD} = (0.4)^3 * (0.1)^2 * (0.1)^1 = 6.4 * 10^{-5}$$

Using Equation (65), the probability of escalation to VM reboot for CT 1 is determined.

$$P(escalation\ to\ VM\ reboot)_{CT1} = 0.62496 + 6.4 * 10^{-5} = 0.625024$$

For CT 2,

$nia = 2+1=3$

$$P(CFIWOD) = \sum_{i=1}^{2} (0.6)^{i-1} * (0.1)^{i-1} * (0.4) = 0.424$$

$$P(CFIWD) = 0.1^2 * \sum_{i=2+1}^{3} (0.6)^{i-1} * (0.1)^{i-1} * (0.4) = 0.1^2 * 0.0144 = 1.44 * 10^{-4}$$

$$P(CF) = 0.424 + 1.44 * 10^{-4} = 0.4241$$

$$P(IF) = (0.6)^3 * (0.1)^2 * (0.1) = 2.16 * 10^{-4}$$

$$P(escalation\ to\ VM\ reboot)_{CT2} = 0.4241 + 2.16 * 10^{-4} = 0.424316$$

Using Equation (66), the MTTF $_{int\ of\ comps}$ for CT 1 and CT 2 is determined.

For CT 1, $MTTF_{int\,of\,comps} = \frac{530000}{0.66946} = 791681.66$

For CT 2, $MTTF_{int\,of\,comps} = \frac{780000}{0.46875} = 1663989.35$

Using Equation (72), the MTTR $_{int\,of\,comps}$ for CT 1 and CT 2 is determined.

$$T_{VM\ failover} = Max_{1\le j\le N}[Cleanup\ time_j] + Max_{1\le j\le N}[Failover\ time_j]$$

$$T_{VMreboot} = NST + Max_{1\le j\le N}[CSS_j] = 5 + [2,1] = 7$$

$$Cleanup\ time_{CT1} = [0.4 * 3] + [0.6 * (3+7)] = 7.2$$

$$Failover\ time_{CT1} = [0.4 * Max(2,4)] = 1.6$$

$$Cleanup\ \ time_{CT2} = [0.6 * 2] + [0.4 * (2+7)] = 4.8$$

$$Failover\ time_{CT2} = [0.6 * Max(1,2)] = 1.2$$

$$T_{VM\ failover} = Max[7.2,4.8] + Max[1.6,1.2] = 8.8$$

$T_{SU\,failover}$ and $T_{SU\,restart}$ calculated using [9] are 8.78 and 4.8 respectively.

$$MTTR_{int\,of\,comps\ CT1} = \begin{pmatrix} (0.04255) * 4.8 + (1.81 * 10^{-3}) * 8.7 \\ + (7.70 * 10^{-5}) * 8.8 + (0.625024) * 7 \end{pmatrix} = 4.59$$

$$MTTR_{int\,of\,comps\ CT2} = \begin{pmatrix} (0.04255) * 4.8 + (1.81 * 10^{-3}) * 8.7 + \\ (7.70 * 10^{-5}) * 8.8 + (0.42431) * 7 \end{pmatrix} = 3.1$$

If the number of collocated components of both the component types is zero, therefore the availability of collocated components in a SU's interference is 1.

$$A_{collocated\,components\,in\,a\,SU's\,interference} = \left(\frac{791681.66}{791681.66 + 4.59}\right)^{0} * \left(\frac{1663989.35}{1663989.35 + 3.1}\right)^{0} = 1$$

Using Equation (74),

For CT 1,

$$MTTF_{int\,of\,SUs} = \frac{MTTF_{component}}{P(escalation\ to\ VM\ \ failover) + P(escalation\ to\ VM\ \ reboot)} = \frac{530000}{0.625101} = 847862.98$$

$$MTTR_{int\,of\,SUs} = \left(7.70 * 10^{-5} * 8.8\right) + \left(0.625024 * 7\right) = 4.37$$

For CT 2,

$$MTTF_{int\,of\,SUs} = \frac{780000}{0.42439} = 1837932.091$$

$$MTTR_{int\,of\,SUs} = \left(7.70 * 10^{-5} * 8.8\right) + \left(0.424316 * 7\right) = 2.97$$

For the best case scenario, the number of collocated SUs in a VM is zero, therefore the availability of collocated SU's interference is 1.

$$A_{collocated\,SU's\,interference} = \left(\left(\frac{847862.98}{847862.98 + 4.37}\right)^{1} * \left(\frac{1837932.091}{1837932.091 + 2.97}\right)^{2}\right)^{0} = 1$$

The number of collocated VMs in a physical host is 7 and there is one SU in a collocated VM and each SU has one and two components in each component type respectively, then the availability of collocated VM's interference is 0.9999.

$$A_{collocated\,VM's\,interference} = \left(\left(\left(\frac{847862.98}{847862.98 + 4.37}\right)^{1} * \left(\frac{1837932.091}{1837932.091 + 2.97}\right)^{2}\right)^{1}\right)^{7} = 0.999940$$

$$A_{collocated\,entities} = 1 * 1 * 0.999940 = 0.999940$$

$$Availability\ of\ SI = 0.999990 * 0.9999957 * 0.999940 = 0.999925$$

For the best case scenario, the estimated availability is greater than the requested availability. Next step is to estimate availability for the worst case scenario. Even though the maximum number of SIs provided by a VM is 48, due to the limitation on the number of SIs a SU can handle and the number of SGs, the *No of SIs per VM* is limited to 32.

$$No\ of\ SIs\ per\ SU = \min\left(\min\left(\frac{80}{1}, \frac{64}{2}\right), 48, 40\right) = 32$$

$$No\ of\ SUs\ per\ VM = min\left(floor\left(\frac{No\ of\ SIs\ per\ VM}{No\ of\ SIs\ per\ SU}\right), No\ of\ SGs\right) = \min\left(floor\left(\frac{48}{32}\right), 1\right) = 1$$

$$No\ of\ SIs\ per\ VM = No\ of\ SUs\ per\ VM * No\ of\ SIs\ per\ SU = 32 * 1 = 32$$

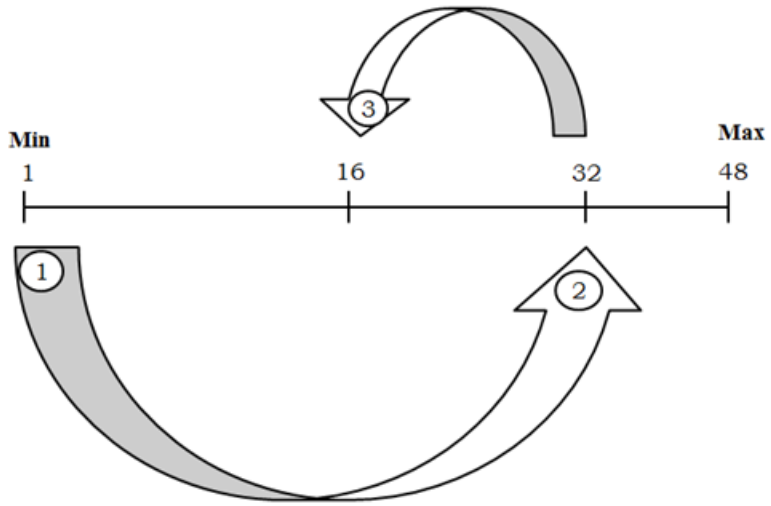

Figure 4-10 *Calculating number of SIs per VM*

No of SIs per VM	Number of entities							No of collocated entities				Estimated availability
	No of SIs per SU	*No of SGs*	*No of SUs per VM*	*No of SUs per SG*	*No of SIs per VM*	*No of VM groups*	*No of PHs*	*No of coll comps in SU*		*No of coll SUs*	*No of coll VMs*	
								CT 1	*CT2*			
1	1	20	1	3	1	20	9	0	0	0	7	0.9999
48	32	1	1	3	32	1	3	31	62	0	0	0.99
16	16	1	1	4	16	1	4	15	30	0	0	0.999

Table 4-4 *Number of SIs per VM with corresponding number of entities and the estimated availability*

The number of entities and the number of collocated entities are calculated for 32 SIs per VM as shown in Table 4-4. When availability is estimated at this point, it is less than the requested availability. Again, when the availability is estimated at the middle point, the estimated availability is equal to the requested availability. Therefore, 16 represents the number of SIs per VM from the perspectives of availability and resource utilization. Figure 4-10 illustrates the various scenarios at which the availability is estimated and the number in the circle denotes the order in which they are estimated.

2) VM flavor selection for a type stack

A physical host has a finite capacity and the number of identical VMs it can host depends on the VM flavor, which among others specifies the total guest memory associated with the flavor. From the set of available VM flavors specified in the infrastructure file, the above mentioned procedure to determine the *No of SIs per VM* from the availability and resource perspective is repeated. During this process, the number of entities and the *No of SIs per PH* are determined for each VM flavor using Algorithm-1 and 2. To select the VM flavor, the VM flavor with the highest *No of SIs per PH* is selected. This selection provides the highest utilization of the host as its

resources are mostly used by the SIs and it infers minimum number of physical hosts. However, if two or more VM flavors support maximum *No of SIs per PH*, then the VM flavor with the smallest TGM is selected based on the availability considerations. Namely, the smallest VM flavor provide better fault isolation compared to other VM flavors.

4.2.3 Distribute AMF entities for deployment

While distributing the AMF entities for deployment, the affinity/anti-affinity relation between them is defined and they are configured. The relation between the SUs is defined by the SGs: SUs that are providing and protecting the same SIs, i.e. they are part of the same SG should not be hosted on the same VM or physical host. The relation between the VMs is defined by the VM group. While determining the number of entities, along with the *No of VMs*, the *No of VM groups*, the number of redundant VMs in a VM group is also defined. In addition to that, the number of SUs that can be hosted per VM is also determined, based on this the configuration attributes related to the distribution of SGs on the VM groups is set. Finally, an AMF configuration is generated for a type stack.

4.2.4 Repeating the process for all the type stacks

As described in the flowchart Figure 4-11 (a) and 4-11 (b), for all the type stacks created for each prototype in the step one of the configuration generation process [9], the AMF type creation [9], AMF entities creation step and the distribution of AMF entities for deployment step are repeated. In the third step, if the best case scenario for all the VM flavors do not meet the requested availability, then the type stack is discarded. The other case where a type stack is discarded is when it is not able to provide a SI. For those prototypes that met the requested availability, AMF configurations are generated. Out of these, only the configuration that uses minimum number of physical hosts is selected for deployment.

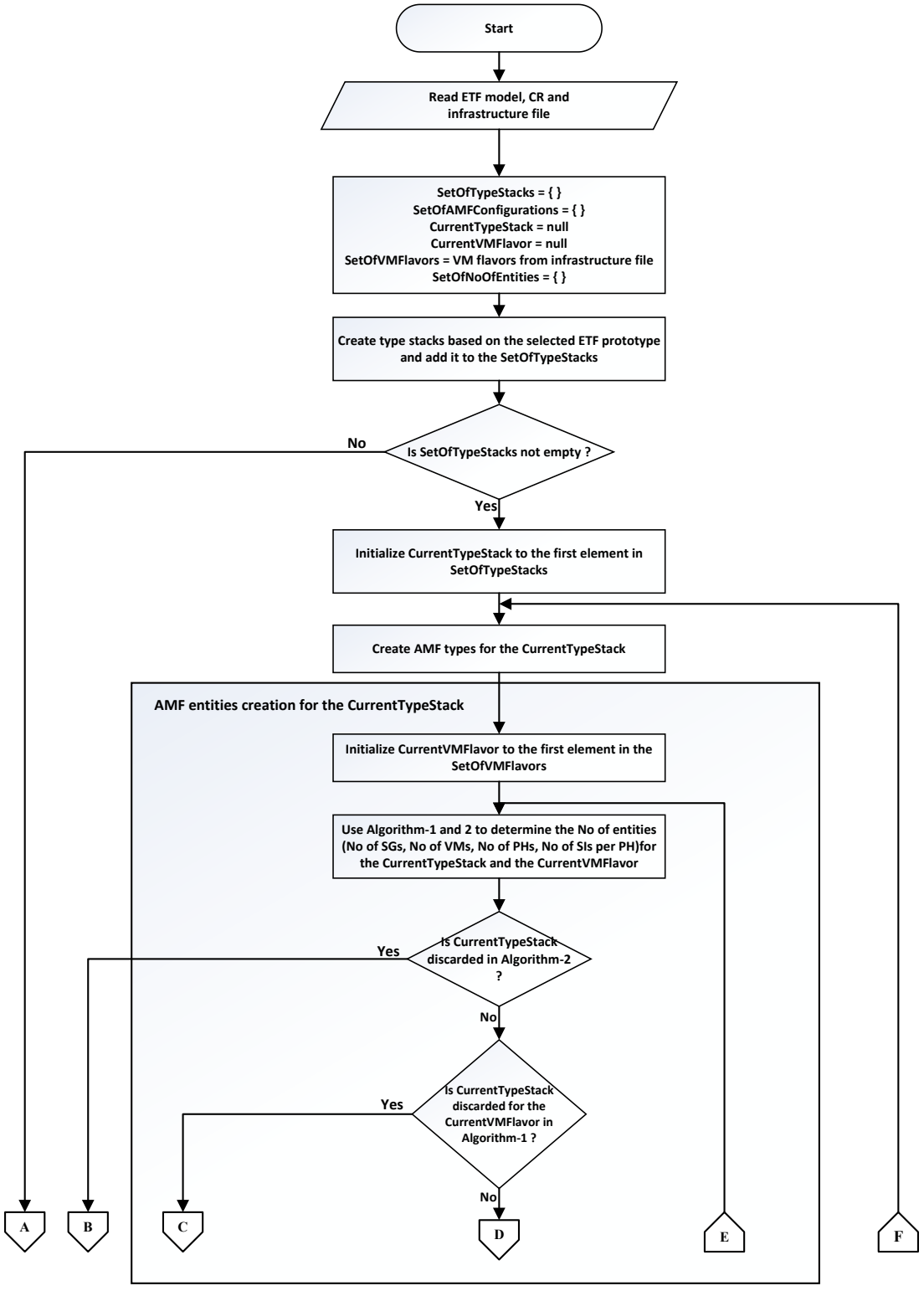

Figure 4-11 (a) *Repeating the process for all the type stacks*

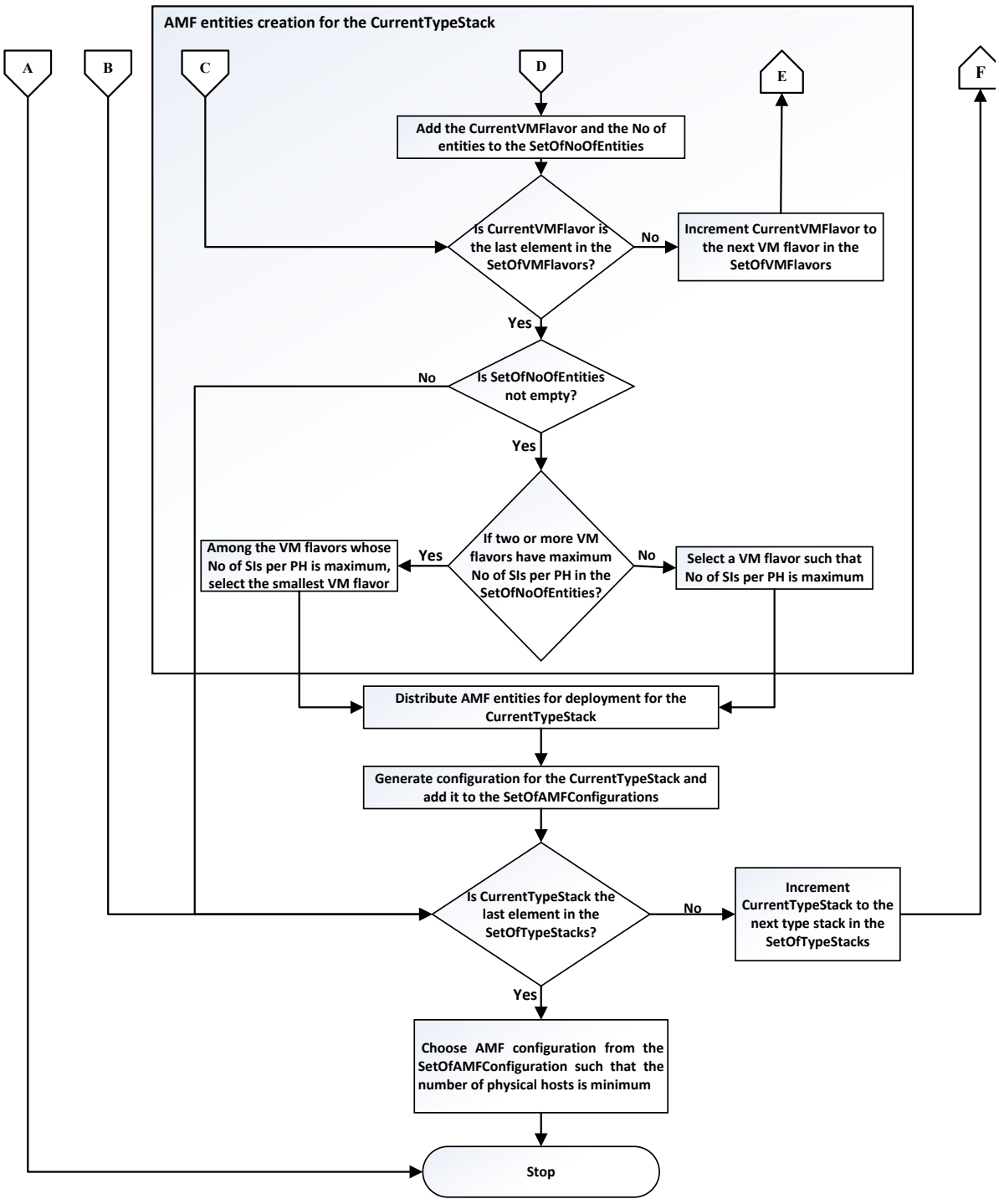

Figure 4-11(b) *Repeating the process for all the type stacks*

4.3 Deployment in the cloud

Deployment is the process involved in installing and running AMF applications in the cloud using the generated configuration. This process consists of three steps: a) deployment information file generation; b) VM image creation; c) initial deployment. The first two steps is about generating pre-requisite files required for the initial deployment. They can be executed in parallel as shown in Figure 4-12. The third step describes the automatic deployment of AMF applications. Note that the following deployment process is discussed with respect to the OpenStack [18] cloud and OpenSAF [34] is used as an open source AMF implementation.

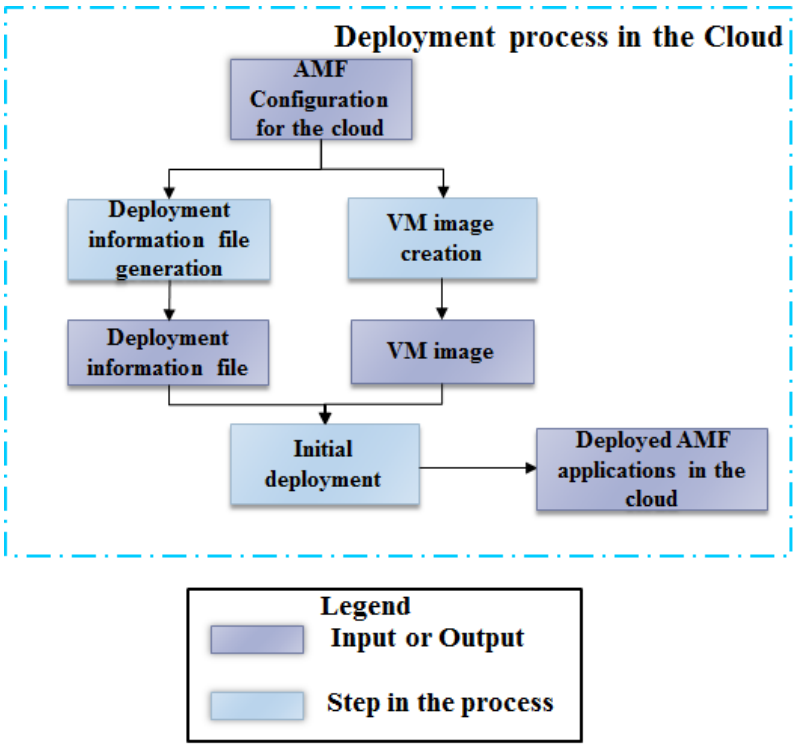

Figure 4-12 Deployment process in the cloud

4.3.1 Deployment information file generation

From the deployment perspective, AMF configuration defines the relation between SGs and VM groups (i.e. SGs are configured to VM groups). This ensures that each SU of a SG will

be hosted on a VM in a VM group. In order to avoid single point of failure, each VM in a VM group should be hosted on a different physical host. This is because, VMs in a VM group provide and protect the same SIs. When these VMs are hosted on the same physical host the failure of the physical host will jeopardize the availability. However, this relation between VM groups and physical hosts (i.e. configuring VMs to physical host) is not defined in the AMF configuration. Anti-affinity VM groups defined in OpenStack cloud can be useful in this context. Therefore, VM groups (i.e. AMF node groups) defined in the configuration are mapped to anti-affinity groups in the OpenStack as shown in Figure 4-13.

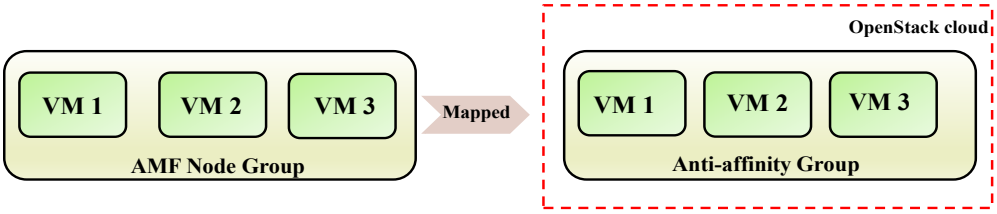

Figure 4-13 Mapping of AMF node group to anti-affinity group in the OpenStack

To achieve this mapping, it is necessary to extract deployment information like the number of VM groups, the number of VMs in each group, name of the VM group and selected VM flavor from the generated configuration. These details are added to a deployment information file using a parser.

4.3.2 VM image creation

In this thesis, AMF applications are designed to be deployed over a cluster of identical VMs. For this purpose, a VM image that contains OS, OpenSAF [34], Monitoring [21] and Elasticity Engine stack [22] and the executable code of the components is created in this step as shown in Figure 4-14. Monitoring [21] and Elasticity Engine [22] stack contains monitoring server, monitoring client and elasticity engine which is used to manage the workload of applications at run-time. The main advantage of creating this VM image is that it reduces the effort of installing

the above mentioned entities into each VM separately and as a result, this VM image can be used as a template to boot VMs.

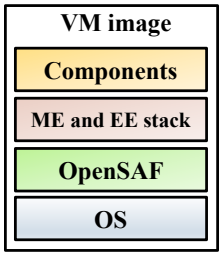

Figure 4-14 VM image

4.3.3 Initial deployment

Using the deployment information file and the created VM image, this step aims to deploy AMF applications in the cloud. Before creating VMs, it is important to create the required number of anti-affinity groups as specified in the deployment information file. Also, based on the number of VMs in each VM group, OpenStack Nova's scheduler [20] is used to provision the required number of VM instances in each anti-affinity group.

OpenStack is responsible for managing VMs and physical hosts. However, it is agnostic to applications running inside the VMs. To have the cluster up and running, OpenSAF should be configured and all the AMF entities should be instantiated and workloads should be assigned to them. To achieve this, VMs are accessed remotely and deployment specific attributes in OpenSAF are configured also the generated configuration is loaded and finally the VMs are restarted to make the changes come into effect. Each VM joins the cluster and successful instantiation of AMF entities indicates that the AMF application is deployed in the cloud and AMF can manage the lifecycle of the entities.

4.4 Managing AMF applications in the cloud

Applications deployed in the cloud can be scaled out/in based on the workload demand. For this purpose, the existing Monitoring architecture [21] and the Elasticity Engine [22] can be used. The reason for choosing these existing engines is that, the elasticity actions are performed based on the service level workload changes (i.e. SI level). Managing AMF applications based on the SI level workload provides finer granularity as opposed to managing applications based on VM level workload changes [21][22]. Many existing solutions in the cloud map an application to a VM and whenever there is an increase in workload of application, a new VM is spawned [35]. In contrast, if a SI workload increase is detected by Monitoring server, then the Elasticity Engine resolves to add a new VM only after attempting to adjust the SG or cluster to provide room for the increased workload [22].

4.4.1 Integration with Monitoring architecture and Elasticity Engine

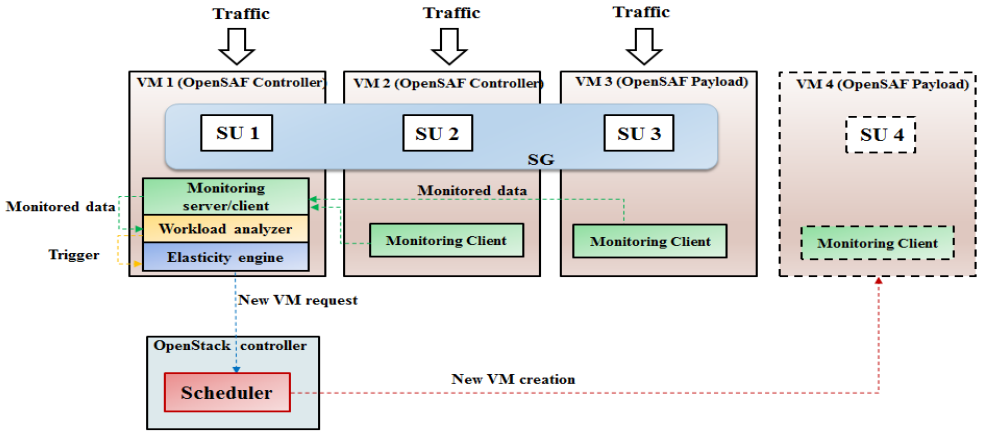

Figure 4-15 Integration of AMF applications with ME and EE

Figure 4-15 illustrates the integration of AMF application with Monitoring architecture and Elasticity Engine and the data flow between the entities. Each VM booted from the above created VM image contains monitoring server, monitoring client, workload analyzer and elasticity engine

stack. However, only one of the VM (OpenSAF controller) is configured with monitoring server, workload analyzer and elasticity engine. Monitoring clients are configured on each VM to monitor the service level workload for all the components hosted on that VM and this monitoring data is sent to the monitoring server [21].

Monitoring data corresponding to each component that participates in providing a SI are aggregated by the monitoring server and sent to the workload analyzer [21]. Further, the monitored data is compared with pre-defined threshold values. In this case, two scenarios are possible:

a. If the monitored data is less than the pre-defined threshold values then over provisioned trigger is generated and sent to the Elasticity Engine [21]. Based on the policies defined in the Elasticity Engine, it modifies the configuration by moving assignments of a SI or removing the SG etc. However, it also ensures that the cluster will not be contracted beyond the minimum configuration [22].
b. If the monitored data is greater than the pre-defined threshold values, then under provisioned trigger is generated and sent to the Elasticity Engine [21]. It is the responsibility of the Elasticity Engine to take necessary elasticity actions including swapping SIs to make room for the increased SI workload or adding assignments to the SI or adding SGs or to add new VM(s) [22]. To add a new VM, Elasticity Engine communicates with the OpenStack scheduler and boots up a new VM using the created VM image. Further, the newly added VM is remotely accessed to configure OpenSAF. Finally, this VM joins the cluster and starts providing the service. Figure 4-15 illustrates an example that Elasticity Engine requests for a new VM to OpenStack scheduler and VM 4 is created and this VM is configured to join the cluster.

4.5 Summary

This chapter discussed the three main contributions of the thesis, namely the availability estimate method, the AMF entities creation method and a method for deploying AMF applications in the cloud. Furthermore, in order to scale AMF applications in the cloud, their integration with the existing Monitoring architecture [21] and Elasticity Engine [22] is also described.

a. The availability estimate method is used to evaluate the availability of a service (i.e. availability of a SI) considering the potential factors that could affect the service. This includes availability of components providing the service, availability of virtual and physical infrastructure and availability of service due to the interference of the collocated entities. This method is used in the AMF entities creation step to estimate the availability and to eliminate the type stacks that do not meet the requested level of service availability.

b. The AMF entities creation method is responsible for calculating the number of AMF entities (i.e. components, SUs, SGs, VMs) that meets the requested level of service availability and they can be deployed using a minimum number of physical hosts. More specifically, this method calculates the number of entities by taking into account the minimum number of redundant entities, potential interference that may occur due to the effect of collocation and also the physical host and VM's capacity limitation.

c. The main goal of the deployment approach is to run AMF applications in the cloud using the generated configuration without jeopardizing the availability. The deployment takes place at two levels namely: the physical host level and the VM level. During the physical host level integration, the required number of VMs are created in

the appropriate VM groups. At VM level integration, the applications are installed and configured automatically so that the AMF application can provide the service functionality. Finally, the deployed application is integrated with the existing Monitoring architecture [21] and Elasticity Engine [22]. Thereby like any other application, AMF applications can also be scaled out/in accordingly in the cloud.

Chapter 5

Prototype Tool

5.1 Introduction

The main goal of this chapter is to describe a proof of concepts prototype tool for the generation and the deployment of AMF configurations in the cloud. This tool includes all the solutions proposed in the Chapter 4. Existing AMF configuration generation tool [9] is extended to generate configurations for the cloud. Further, the deployment modules (parser module, VM image creation module and initial deployment module) are developed to deploy the AMF applications in the OpenStack cloud. Finally, with the help of the Monitoring architecture [21] and the Elasticity Engine [22], applications can be managed (scaled in or out) accordingly.

5.2 AMF configuration generation module

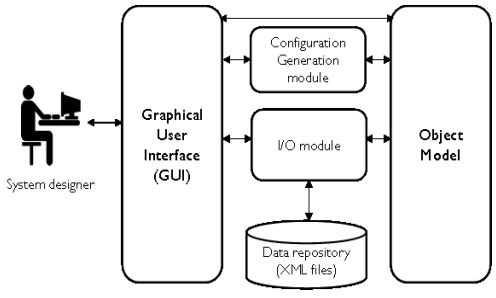

Figure 5-1 *Data flow in the configuration generation module [9]*

As shown in Figure 5-1, the configuration generation module [9] consists of four parts namely; 1) The Graphical User Interface (GUI); 2) Object model; 3) The I/O module; 4) The configuration generation module.

1) *Graphical User Interface (GUI)*

The system designer is responsible for providing the ETF XML files and specifying the configuration requirements like the number of SIs in a SI template, the number of CSIs per SI, minimum requested service availability and redundancy models [9]. Note that, templates are introduced to generically create entities that share common characteristics [9]. The GUI uses Eclipse Modelling Tools [36] and Java Swing [37] to input ETF XML file and other requirements [9].

2) The Object Model

This object model is a repository for models that includes the AMF, ETF, CR and infrastructure models. These UML models are described as Ecore models [38] using Eclipse Modelling Framework (EMF) [39]. The AMF model is created according to the information model described in [10] and the infrastructure model is defined as explained in the Chapter 4 (Section 4.2.1). Also, ETF model is defined based on ETF schema described in [40] and the CR model is designed according to [9]. Note that the ETF and CR models also includes the modifications mentioned in Section 4.2.

3) The I/O Module

The input/output files are instances of the object model. Using this module, an input file is parsed against a model in the repository. Also, using this module, an output file (i.e. generated AMF configuration) is saved as a XML file [9] which conforms to the AMF model in the data repository.

4) The Configuration Generation Module

This module is responsible for generating AMF configurations. As shown in Figure 5-2, the input wizard GUI takes ETF XML file and validates it against the ETF model [9].

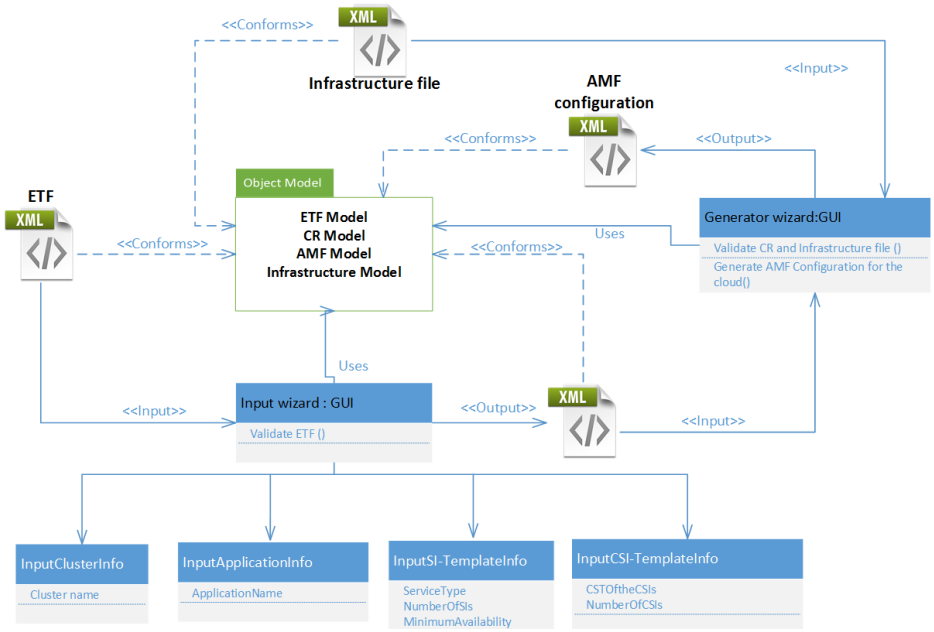

***Figure 5-2** Extended AMF configuration generation prototype*

If the validation is successful, GUI expects configuration requirements. It then outputs a CR object model. The CR object model, infrastructure object model are given as input to the Generator wizard GUI. It validates both the models and if the validation is successful, then the configuration generation process starts by selecting the prototypes and creates AMF types [9]. Furthermore, using the AMF entities creation method and the distribute AMF entities for distribution methods explained in the Chapter 4, the number of AMF entities are created and they are configured for deployment. Finally, AMF configuration is generated and saved in an XML format.

5.3 Deployment modules

The deployment process consists of three modules namely parser, VM image creation and initial deployment module.

1) *Parser module*

The purpose of the parser module is to generate a deployment information file from an AMF configuration. For this purpose, DOM4J parser [41] is used to extract the number of VMs, the number of VM groups and the number of VMs in each group, name of the VM group from the configuration and stores these data in a JSON file. In addition to that, OpenSAF specific parameters like node type (controller or payload) required to configure each VM are also added to the deployment information file.

2) *VM image creation module*

A VM image that contains OS, OpenSAF middleware, Monitoring and Elasticity Engine stack and the executable code of components is created. The created VM image captured the state and data of a VM at one point in time [42] that is used to create identical VMs with the above mentioned entities.

3) Initial deployment module

The initial deployment module requires the created VM image and the deployment file to boot VMs in the OpenStack cloud. It first requests the nova service to create the required number of anti-affinity groups. It then communicates with the OpenStack scheduler and specifies the VM image, flavor of the VMs, the number of VMs in an anti-affinity group, name of the anti-affinity group and name of the VMs. When scheduler receives this command, it automatically schedules and boot VMs from the VM image. Since the VMs are created from the same VM image, it is possible that they possess the same host name (host name is different from name of the VM during creation and host name is more specifically used by OpenSAF). The deployment module then remotely logs into each of the VM in the cluster, configures and

starts OpenSAF. Successful instantiation of application entities indicates that the initial deployment is complete and AMF applications are deployed in the cloud.

5.4 Illustration with an application

Let us consider a deployment of HTTP service in an OpenStack cloud with 4 SIs. In the first step, the system designer inputs the ETF file using the GUI module, which is validated against the ETF schema by the object model. The Figure 5-3 illustrates the conformance message that the ETF is parsed successfully. Using the input wizard GUI module, information about the SG types (redundancy model, number of active/standby assignments), SI types and CSI types is given in the form of templates [9].

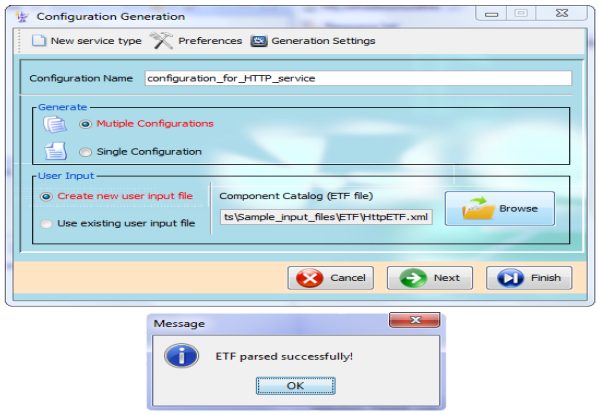

Figure 5-3 *Providing ETF input*

As shown in Figure 5-4 (a), the system designer specifies N-way active redundancy model using *SG template-Pattern-Based* dialog box. Further, the system designer inputs the name of the SI template as HTTP_SI_template, type of service as HTTP service, the number of SIs to be 4, number of active assignments to be 3 and the requested level of availability to be 0.999 using *Regular SI template-Pattern-Based* dialog box (Figure 5-4 (b)). The system designer then inputs the HTTP_CSI template associated with the HTTP_SI_template using the *CSI Template* dialog

box (Figure 5-4 (c)). This process outputs the CR file. Finally, the infrastructure file and the outputted CR file are given as input to the Generator wizard GUI and they are validated against the infrastructure model and CR model respectively. Figure 5-4 (d) illustrates the conformance message that the infrastructure file has been parsed successfully.

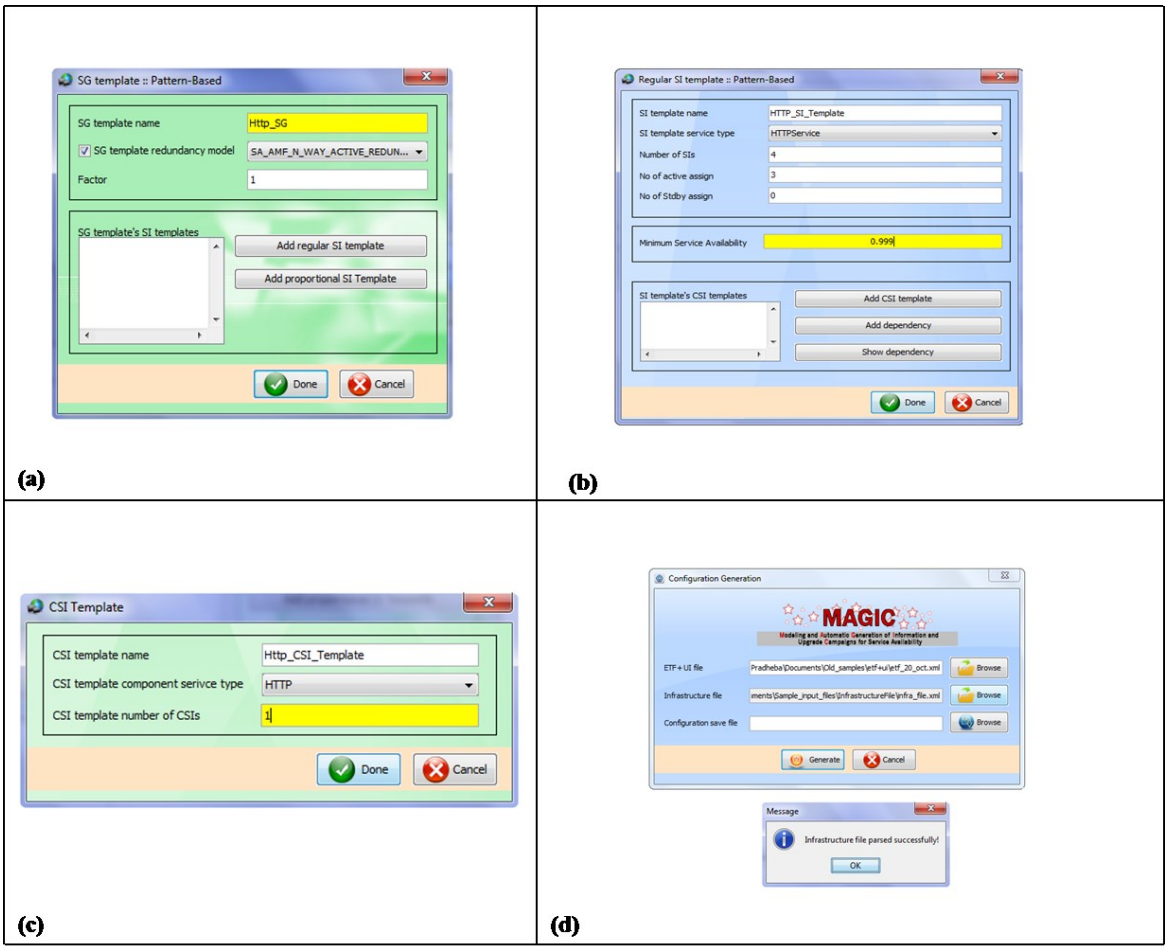

Figure 5-4 *(a) SG template-Pattern based dialog box; (b) SI template pattern based dialog box; (c) CSI template; (d) Providing infrastructure file and CR file*

Using the first two steps of the configuration generation process, two type stack (SR-0 and SR-1) that can provide the HTTP service type with the requested level of service availability is created [9]. Figure 5-5, illustrates the number of SIs per VM calculation and the VM flavor selection for SR-0 type stack. The small VM flavor is not considered because the memory required

to provide one SI is greater than the available guest memory. For the large VM flavor, when the *No of SIs per VM* is one, to deploy 4 SIs the *No of SGs* required is 4 and the *No of SIs per SU* is 1. The number of active assignments specified in the configuration requirements is three therefore, the *No of SUs per SG* is three. To deploy 4 SGs, *No of VM groups* required is also 4. The *No of VMs per VM group* based on the *No of SUs per SG* is also 3. Based on the capacity of the physical host, the number of large VMs it can accommodate is 1. Therefore, the total *No of PHs* required is 12. Based on this, the number of collocated entities are calculated and the availability is estimated. For the best case scenario, the estimated availability is greater than the requested availability. Therefore, the availability is estimated for the worst case scenario (i.e. 4 SIs per VM). Here, the estimated availability is less than the requested availability. Therefore, the availability is estimated for the midpoint interval i.e. at 2 SIs per VM, which satisfies the availability requirement. The large VM flavor is selected for this case and the *No of SIs per PH* is 2 and the *No of PHs* required is 6.

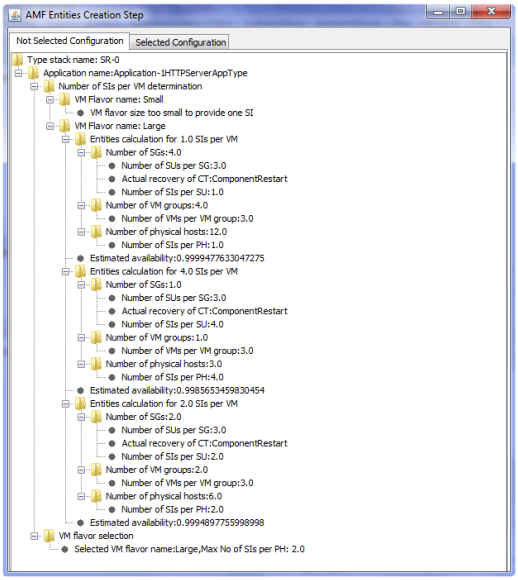

Figure 5-5 *No of SIs per VM determination and VM flavor selection for SR-0 type stack*

For SR-1 type stack the same procedure to determine the number of SIs per VM is performed for small VM flavor (Figure 5-6) and large VM flavor (Figure 5-7). The *No of SIs per PH* for small VM flavor is 4 and the large flavor is 2. Therefore, the small VM flavor is selected for this type stack. Finally, the type stack SR-1 is selected because, the *No of PHs* required (i.e. 3) is minimum compared to the *No of PHs* (i.e. 6) required for SR-0.

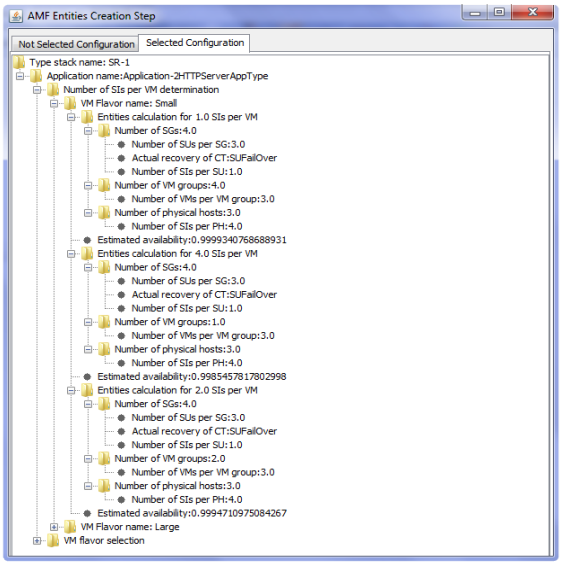

Figure 5-6 *No of SIs per VM determination for small VM flavor and SR-1 type stack*

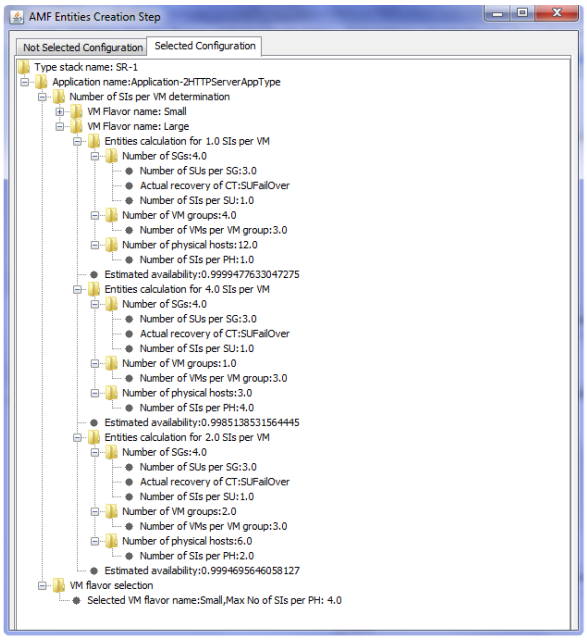

Figure 5-7 *No of SIs per VM determination for large VM flavor and VM flavor selection for SR-1 type stack*

As explained in the Section 5.3, based on the generated AMF configuration the deployment details are extracted and the VM image is created. Using the initial deployment module, two anti-affinity groups (*AntiAffinityServerGrp_0* and *AntiAffinityServerGrp_1*) and 6 VMs (*AF_VM_1 to AF_VM_6*) are created in an OpenStack cloud. Figure 5-8 shows the created VMs and grouping of VMs in each anti-affinity group.

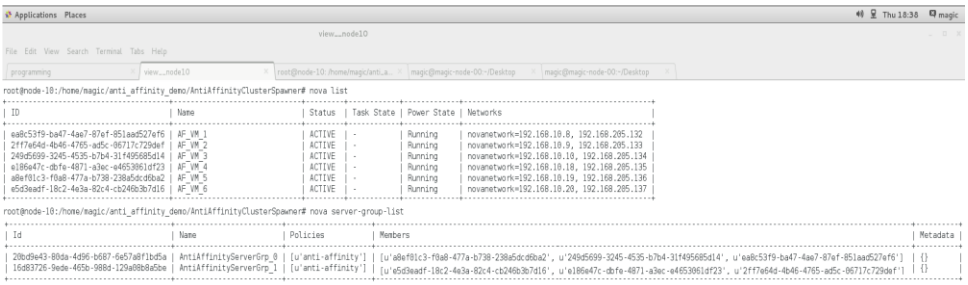

Figure 5-8 *Created VMs and anti-affinity groups in OpenStack cloud*

The overall view of the deployed AMF application is shown using Monitoring GUI [21] (Figure 5-9). The first two SGs (*Service_Group-00HTTP_server* and *Service_Group-01HTTP_server)* are deployed over the *AntiAffinityServerGrp_0 (node 1, node 3 and node 5). The next two SGs (Service_Group-02HTTP_server and Service_Group-03HTTP_server) are deployed over AntiAffinityServerGroup_1 (node 2, node 4 and node 6).* Note that node 1 to node 6 refer to the AMF nodes which are mapped to *AF_VM_1 to AF_VM_6*. Each SG provides one SI and the workload of each SI is monitored by the Monitoring Engine.

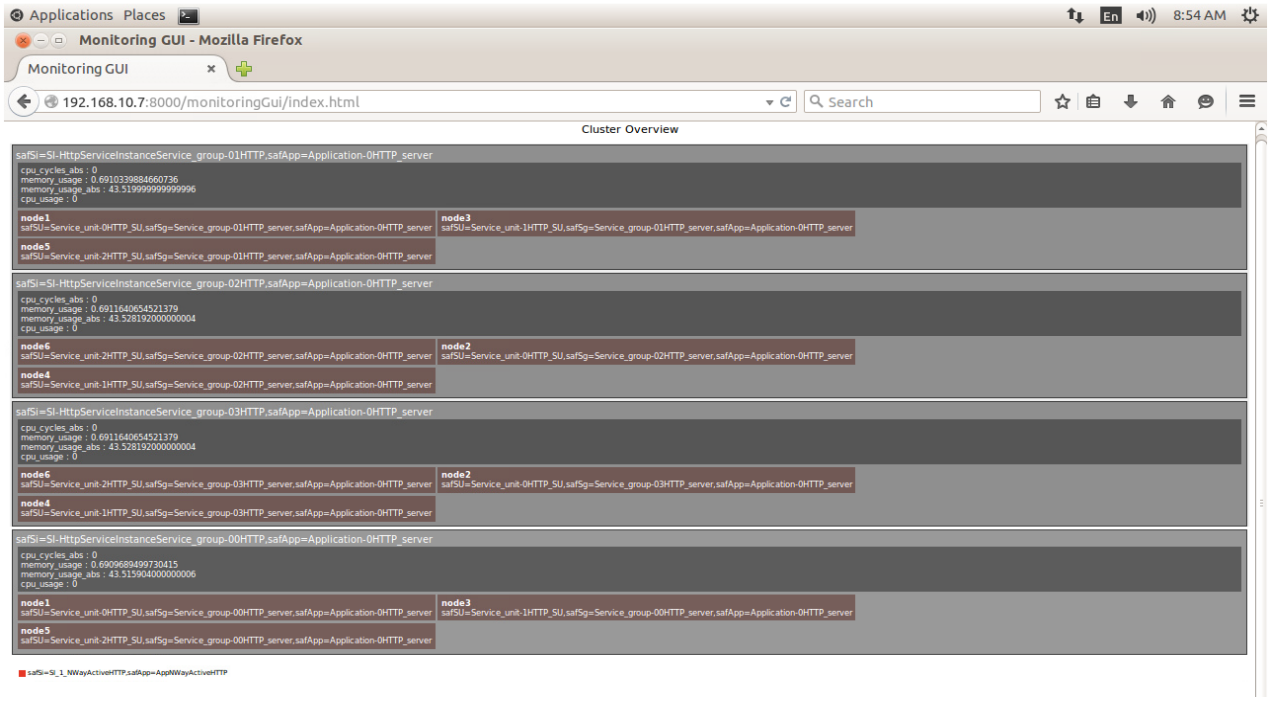

Figure 5-9 *Monitoring GUI showing the deployed AMF applications in the cloud before scaling*

Consider a scenario where the workload of SI 1 provided by *Service_Group-01HTTP_server* is increased using Apache JMeter [43]. The Monitoring Engine notices the increase in workload and sends the workload to workload analyzer. Further, the workload analyzer compares the increased workload with the pre-defined threshold and triggers the Elasticity Engine to take necessary elasticity actions [21]. The Elasticity Engine modifies the configuration by increasing the number of active assignment of SI 1 from 3 to 4 and requests OpenStack to create a new VM. Figure 5-10 depicts this scenario where a new VM (i.e. node 7) has joined the cluster and the SU hosted on it is assigned a new assignment.

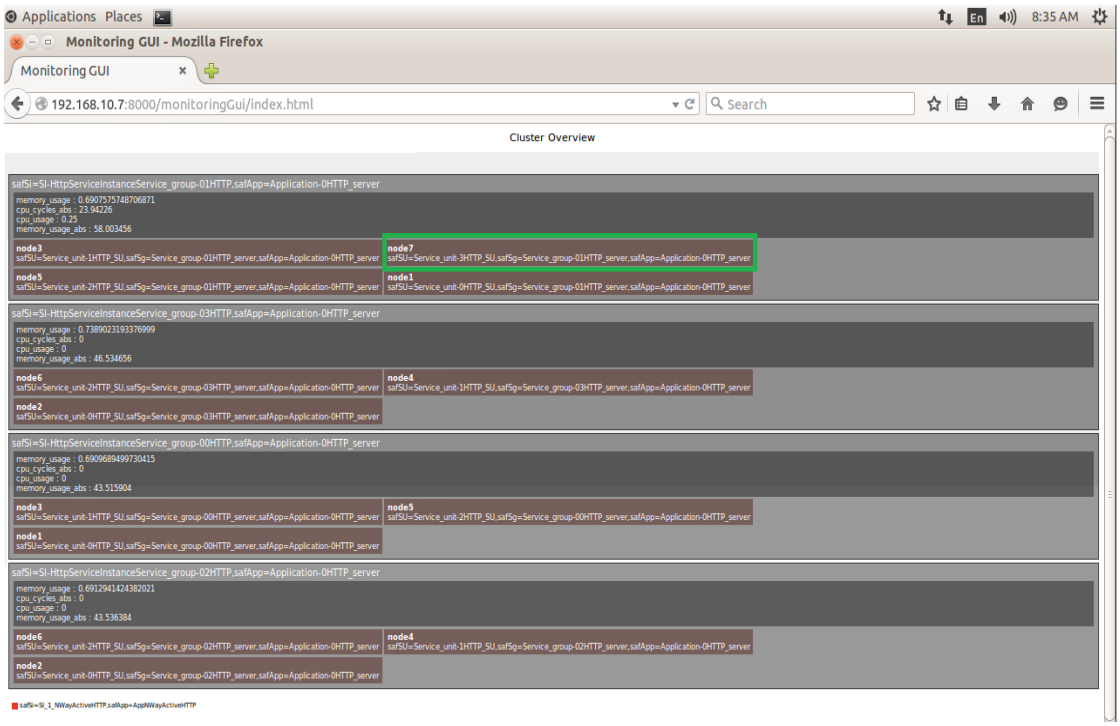

Figure 5-10 *Monitoring GUI showing the deployed application in the cloud after scaling*

5.5 Summary

This chapter presented the prototype tool for the generation and the deployment of AMF configurations in the cloud. This tool implemented the proposed AMF entities creation method, availability estimate method and the method to deploy AMF applications in the OpenStack cloud. In addition, this chapter also presented an example to illustrate the prototype tool and also a case highlighting the usage of existing Monitoring [21] and Elasticity Engine [22] to scale the deployed application.

Chapter 6

Application – Configuration Generation for AMF Managed VNFs

6.1 Introduction

The main objective of this chapter is to illustrate an application of the proposed AMF configuration generation process in the domain of NFV [45]. This chapter introduces the necessary background on NFV and then discuss about the proposed mapping between NFV and AMF domain. It is followed by the configuration generation process for AMF managed VNFs.

6.2 Background on NFV

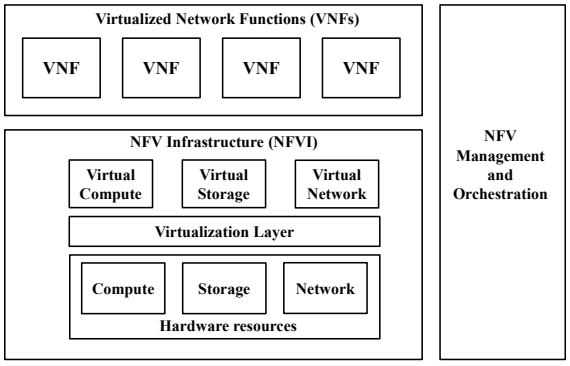

Figure 6-1 *High-level NFV framework [45]*

NFV is changing the way Network Services (NS) are designed, deployed and managed [45]. It leverages virtualization technology and cloud technologies to roll out NS faster as opposed to traditional networks. For this purpose, NFV unveils a new set of concepts called VNFs, NFV Infrastructure (NFVI) and NFV Management and Orchestration (NFV-MANO) [46] as shown in

Figure 6-1. VNFs are network functions that are virtualized and can run over shared compute, storage and networks in NFVI. NFVI encompasses heterogeneous physical hardware, software and networking elements necessary to run VNFs. NFV-MANO is responsible for managing the life cycle of the NS and its constituent VNFs [46].

A VNF may consist of a single software component capable of providing the network function or several software components that collaborate to provide the network function. These software components are referred to as VNF Components (VNFCs) [44]. VNF Component Instance (VNFCI) represent the run-time instantiation of a VNFC [44].

6.3 Mapping between NFV and AMF domain

A VNF is deployed as a cluster of VMs in the NFVI [44]. These VNFs are expected to be highly available and provide the required functions with minimal downtime. AMF can manage the availability of any application through an AMF configuration. Therefore, managing VNF as an AMF application is not an exception to this. To be able to do this, we need to map the concepts in the NFV domain to the concepts in the AMF domain.

Each VNF exposing a specific network functionality is mapped to an AMF App type that provides a service type. A VNF may consist of one or more VNFCs and each VNFC provides a specific service type (i.e. sub-functionality of the network function) and it is packaged as a software image [50]. To be able to map a VNFC to the concepts in the AMF domain, a new concept called AMF node type is proposed. This AMF node type represents a collection of the software images necessary for the service unit type that provides the service type of a VNFC. Each VNFCI runs in a dedicated virtualization container (e.g. VM) [44]. In the AMF domain, each instance of AMF

node can be mapped against a VM. Thus the VNFCI can be mapped to the AMF node, and the VNFCIs of a given VNFC to an AMF node group.

6.4 AMF configuration generation process for VNFs

The AMF node groups for VNFCs are disjoint, we generate an AMF configuration for a service type i.e. the service type to be provided by a VNFC. We design a VNFC by grouping one or more AMF component types into a service unit type so that it can provide the service type. Based on the number of workload units (i.e. SIs) to be provided for the given service type (i.e. service capacity) and the requested level of service availability, we determine the number of AMF entities (i.e. SUs, SGs, VMs) so that the VNFCIs providing the service can be deployed using a minimum number of physical hosts. The affinity and anti-affinity relations between the AMF entities are defined by AMF and therefore are reflected in the AMF configuration. In particular, nodes of an AMF node group are redundant entities therefore cannot be collocated. During deployment, the required number of VNFCIs of a VNFC are deployed using the AMF configuration. The proposed AMF configuration generation approach in Chapter 4 is applied in the same manner for each of the service types the VNF to configure the VNFCIs of each VNFC.

The information about the number of entities calculated can be used to deploy and configure a VNF in the NFVI. For example, the *No of VMs* represents the required number of VNFCIs of a VNFC. The number of VMs in a VM group represents the number of VNFCIs of a VNFC in anti-affinity relation. The selected VM flavor can be used to deploy a VNFCI and the *Max no of VMs per PH* represents the number of VM groups per physical host. The calculated number of physical hosts can be used to deploy the VNFCIs of a VNFC.

6.5 Conclusion

This chapter presented an application for AMF configuration generation process in the field of NFV. Here, we proposed to use AMF as middleware to manage the availability of the services provided by the VNFs. To achieve this, we proposed an approach for generating AMF configurations for VNFs. In this approach, we mapped the concepts in the NFV domain to the concepts in the AMF domain and we designed a VNFC by grouping one or more AMF component types to provide the service type. Next, we determined the number of AMF entities with a goal that the requested availability should be met and the VNFCIs of a VNFC should be deployed using a minimum number of physical hosts.

For the configuration and deployment of VNFs in the NFVI, the information about concrete the number of VNFCIs, their colocation/anti-colocation relationship and the VM flavor is required. These information are reflected in the AMF configuration and they can be extracted to design a VNF configuration.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, an approach for generating AMF configurations for applications and deploying them in the cloud is proposed. The generation process is devised with an intention to generate configurations that can provide and protect the services with requested level of service availability and also to deploy them using a minimum number of physical hosts.

An availability estimate method is proposed to evaluate the availability of service by considering the availability of all the entities that participate in providing the service. In particular, this method takes into account the impact due to collocation. This method is used to eliminate the configurations that do not meet the requested level of service availability.

An AMF entities creation method is proposed to determine the number of AMF entities that satisfies availability and resource utilization requirements. In particular, this method determines the number of AMF entities required to build an AMF application by taking into account the minimum number of redundant entities, the potential interference that may occur due to collocation and also the physical host and VM's capacity limitation.

A method to deploy AMF applications in the cloud is proposed. This deployment process is defined to install and run AMF applications in the cloud without jeopardizing the availability. Further, the deployed applications are integrated with the existing Monitoring architecture [21]

and Elasticity Engine [22], so that the workload of the applications can be managed effectively. Finally, as a proof of concepts of the above proposed solutions has been developed as a prototype tool.

An application for the AMF configuration generation process in the domain of NFV [45] has been developed. Using this process, configurations can be generated for AMF managed VNFs [44]. For this purpose, the appropriate mapping between both fields has been proposed. The information about the number of entities calculated during this process can be used to configure and deploy VNFs in the NFVI.

7.2 Potential future research direction

There are few aspects that can be investigated further in this research. The AMF configuration generation and deployment process is designed for a service type. In future, one may consider to generate and deploy configurations for many service types. Also, while determining the number of entities, one may consider to deploy the applications over heterogeneous physical hosts. Furthermore, configuration management tools like Chef [47] and Puppet [48] can be considered to deploy AMF applications in the cloud.

References

[1] M.Toeroe and F.Tam, *Service Availability: Principles and Practice*, 1st edition, Wiley 2012

[2] L.M.Vaquero, L.Rodero-Merino, J.Caceres, M.Linder, "A Break in the Clouds: Towards a Cloud Definition", Volume 39, Number 1, January 2009

[3] M.Armbust, A.Fox, R.Griffith, A.D.Joseph, R.Katz, A.Konwinski, G.Lee, D.Patterson, A.Rabkin, I.Stoica and M.Zaharia, "A View of Cloud Computing", DOI: 10.1145/ 1721654 .1721672

[4] K.Clay,"Amazon.com Goes Down, Loses $66,240 Per Minute",[Online] Available: http://www.forbes.com/sites/kellyclay/2013/08/19/amazon-com-goes-down-loses-66240-per-minute/#693b4d7a3c2a, [Accessed 19th September 2016]

[5] Q.Zhang, L.Cheng,R.Boutaba "Cloud computing: state-of-the-art and research challenges " J Internet Serv Appl (2010) 1:7-18

[6] M.Pearce, S.Zeadally, R.Hunt, "Virtualization: Issues, Security Threats and Solutions", ACM Computing Society, Vol. 45, No.2, Article 17.

[7] P.T.Endo,A.V.A.Palhares,N.N.Pereira,G.E.Goncalves,D.Sadok,J.Kelner,B.Melander and J.Mangs "Resource Allocation for Distributed Cloud: Concepts and Research Challenges", IEEE Network, July/August 2011

[8] "Service Availability Forum" [Online] Available: www.saforum.org, [Accessed 19th September 2016]

[9] P.Pourali, "Pattern-based Generation of AMF Configurations", Master Thesis, Concordia University, 2014

[10] Service Availability Forum, Application Interface Specification. Availability Management Framework SAI-AIS-AMF-B.04.01.," 2011.

[11] A. Kanso, "Automated Configuration Design and Analysis for Service High-Availability," PhD Thesis, Concordia University, 2012.

[12] P. Salehi, "A Model Based Framework for Service Availability Management," PhD Thesis, Concordia University, 2012.

[13] SAForum, "Service Availability Forum, Service Availability Interface Overview," 2011.

[14] National Institute of Standards and Technology, "The NIST definition of Cloud computing", Special publication 800-145, 2011 pg-2-3

[15] SA Forum, "Service Availability Forum, Application Interface Specification. Software Management Framework SAI-AIS-SMF-A.01.02.AL.," 2011

[16] Google App Engine, [Online] Available: https://cloud.google.com/appengine/ [Accessed: 19th September 2016]

[17] A. Kumawat, "Cloud Service Models (IaaS, SaaS, PaaS)+How Microsoft Office 365 Azure Fit In"[Online] Available: http://www.cmswire.com/cms/information-management/cloud-service-models-iaas-saas-paas-how-microsoft-office-365-azure-fit-in-021672.php [Accessed: 19th September 2016]

[18] OpenStack , [Online] Available: https://www.openstack.org/ [Accessed: 19th September 2016]

[19] Nova System Architecture, [Online] Available: http://docs.openstack.org/developer/nova/architecture.html [Accessed: 19th September 2016]

[20] OpenStack compute Scheduler, [Online] Available: http://docs.openstack.org/kilo/config-reference/content/section_compute-scheduler.html [Accessed: 19th September 2016]

[21] M.N.A.Khan, "Monitoring Service Level Workload of Highly Available Applications," Master Thesis, Concordia University, 2015.

[22] N.Pawar, "Managing High-Availability and Elasticity in a Cluster Environment", Master Thesis, Concordia University, 2014

[23] SAForum, "Service Availability Forum, Information Model Management Service"

[24] D.Jayasinghe, C.Pu, T.Eilam, M.Steinder, I.Whalley,E.Snible,"Improving Performance and Availablity of Services Hosted on IaaS Clouds with Structural Constraint-aware Virtual Machine Placement" 2011 International Conference on Services Computing.

[25] E.Bin,O.Biran,O.Boni,E.Hadad,E.K.Kolodner,Y.Moatti,D.H.Lorenz,"Guaranteeing High Availability Goals for Virtual Machine Placement" 2013 31st International Conference on Distributed Computing Systems.

[26] M.Jammal,A.Kanso,A.Shami,"High Availability-Aware Optimization Digest for Applications Deployment in Cloud" IEEE ICC 2015-Communications Software, Services and Multimedia Applications Symposium

[27] F.Machida, M.Kawato and Y.Maeno,"Redundant Virtual machine Placement for Fault-tolerant Consolidated Server Clusters" 2010 IEEE/IFIP Network Operations and Management Sysmposium- NOMs 2010:Mini-Conference.

[28] J.Li,Q.Lu,L.Zhu,L.Bass,X.Xu,S.Sakr,P.L.Bannerman,A.Liu,"Improving Availability of Cloud-Based Applications through deployment Choices"2013 IEEE Sixth International Conference on Cloud Computing.

[29] A.Abouzamazem, P.Ezhilchelvan,"Efficient Inter-Cloud Replication for High Availability Services" 2013 IEEE International Conference on Cloud Engineering.

[30] M.E.Frincu,C.Cracium,"Multi-objective Meta-heuristics for Scheduling Applications with High-Availability Requirements and Cost Constraints in Multi-Cloud Environments" 2011 Fourth IEEE International Conference on Utility and Cloud Computing.

[31] SA Forum, "Service Availability Forum, Application Interface Specification. Cluster Membership Service, SAI-AIS-CLM-B.04.01" 2011

[32] M.J de Smith, Statistical Analysis Handbook [Online].Available: http://www.statsref.com/HTML/index.html?poisson.html (accessed September 11, 2016).

[33] J.Heo,X.Zhu,P.Padala,Z.Wang, "Memory Overbooking and Dynamic Control of Xen Virtual Machines in Consolidated Environments", IFIP/IEEE International Symposium on Integrated Network Management, 2009.

[34] OpenSAF [Online]. Available: http://devel.opensaf.org/

[35] Amazon auto scaling [Online].Available: https://aws.amazon.com/autoscaling/

[36] Eclipse Modelling tools [Online]. Available: https://eclipse.org/downloads/ packages/release/Luna/SR2

[37] Java Swing [Online]. Available: http://docs.oracle.com/javase/6/docs/technotes/guides/swing/

[38] Eclipse Modeling Framework Core (Ecore) [Online]. Available: http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html

[39] Eclipse Modeling Framework (EMF) [Online]. Available: http://www.eclipse.org/modeling/emf/

[40] Service Availability Forum, Application Interface Specification (ETF schema describing the software bundle and the entity types' relations and features) SAI-AIS-SMF-ETF-A.01.02.xsd," 2011.

[41] DOM4J parser [Online]. Available: http://dom4j.sourceforge.net/dom4j-1.6.1/guide.html

[42] VMware – Using the snapshot [Online]. Available: https://www.vmware.com/support/ws4/doc/preserve_snapshot_ws.html

[43] Apache JMeter [Online]. Available: http://jmeter.apache.org/usermanual/

[44] Network Functions Virtualization (NFV); Virtual Network Functions Architecture ETSI GS NFV-SWA 001 V1.1.1 (2014-12).

[45] Network Functions Virtualization (NFV); Architectural Framework, ETSI GS NFV 002 V1.2.1 (2014-12)

[46] Network Functions Virtualization (NFV); Management and Orchestration ETSI GS NFV-MAN 001 V1.1.1 (2014-12)

[47] Chef configuration management [Online]. Available: https://www.chef.io/solutions/infrastructure-automation/

[48] Puppet configuration management [Online].Available: https://puppet.com/solutions/cloud-management

[49] Microsoft Office 365 [Online]. Available: https://technet.microsoft.com/en-us/cloud/gg697163.aspx

[50] Network Functions Virtualization (NFV); Management and Orchestration; VNF Packaging Specification, ETSI GS NFV-IFA 011 V2.1.1 (2016-10)