HIGH-AVAILABILITY RESOURCE MONITORING FRAMEWORK ON DYNAMIC DISTRIBUTED ENVIRONMENT

By

WONG SIAW LING

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

In partial fulfilment of the requirements

For the degree of

BACHELOR OF COMPUTER SCIENCE (HONS)

Faculty of Information and Communication Technology

(Perak Campus)

Jan 2014

DECLARATION OF ORIGINALITY

I declare that this report entitled "**High-Availability Resource Monitoring Framework on Dynamic Distributed Environment**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature	:	____________________
Name	:	WONG SIAW LING
Date	:	11th April 2014

ACKNOWLEDGEMENTS

First and foremost, I would like to express my greatest gratitude to my supervisor Dr. Alex Ooi Boon Yaik for his excellent guidance, advice and support throughout this project. Without his consistent encouragement, motivation and inspiration, this work will never come into existence, and I will never develop my interest on distributed system. Sincerely thanks to him for being a great mentor of mine for both academically and personally, throughout this project and my university life.

I would also like to thank Mr. Ng Eng Siong, Mr. Muhamad Shukri Bin Muhamad Shukur, Mr. Kurt Strebel and other colleagues as well who guided and shared their knowledge with me during my internship with Hilti Asia IT Services Sdn. Bhd. Thanks to them for giving me opportunity to work and gain exposure on enterprise level IT setup, particularly on resource monitoring area. Those experiences are extremely useful to this work. Also, thank you for their willingness and time to discuss and feedback on my project.

Also, sincerely thank you to Mr. Wong Chee Siang, Dr. Liew Soung Yue, Dr. Tan Hung Khoon and Mr. Gan Chee Tak for their constructive advices and recommendations on my project.

Last but not least, my heartily thank dedicated to my parents, my love, family and friends who being supportive and considerate for all the time.

ABSTRACT

Resource Monitoring System (RMS) is one of the most crucial components under an IT landscape. It oversees all the sub-systems and services' performance to maintain their availability, and responsible to raise alert when any component is under critical situation. Thus, the RMS must achieve the highest availability among all the computing resources in the monitored system which otherwise its monitoring would not be complete as it may fail to monitor some events. However, due to the increasing complexity and heterogeneous of an IT over the years, the mission of maintaining a High-Availability RMS has become very challenging. To counteract this, the RMS should have self-managed properties within the system itself. This work proposed a novel approach to improve RMS availability via automated failover and fallback mechanism through automated service placement. As compared to conventional approach that uses dedicated and redundant servers, together with much human intervention to achieve high availability, this work simplifies and automates most of the processes to realizes the HA operational goals. A prototype of proposed solution is being implemented on both controlled and dynamic environments, and experiments were carried out to investigate the feasibility and limitation of the proposed approach. Based on the experimental result, the proposed solution is promising under controlled environment with human-injected failure. However, it does not work as expected and we face some reliability issues when it is deployed under dynamic environment. These problems are identified and will be improved in the future work.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

RMS	Resource Monitoring System
IT	Information Technology
CPU	Central Processing Unit
IP	Internet Protocol
LAN	Local Area Network
WAN	Wide Area Network
OS	Operating System
API	Application Programming interface
HPC	High Performance Computing
REST	Representational state transfer
DBMS	Database Management System
GUI	Graphical User Interface
TCP/IP	Transmission Control Protocol - Internet Protocol
SSH	Secure Shell
SFTP	SSH File Transfer Protocol
PHP	Hypertext Preprocessor
I/O	Input-Output
ICT	Information and Communication Technology

CHAPTER 1 INTRODUCTION

1.1 Introduction

Resource Monitoring is one of the most crucial components under an IT landscape as it oversees the system performance, availability, and responsible to raise alerts when any component is undergoing critical situations such as server failure, network failure, critical service is unavailable and etc. However, due to the rapid growth of the complexity and heterogeneity of computer systems and software solutions, resource monitoring is becoming more challenging. In order to provide an effective RMS, most of the solution providers have continuously improved their RMS in terms of functionality and reliability. Unfortunately, as the RMS advances, its own complexity increases proportionally. Consequently, the task of managing RMS is getting more challenging and requires trained and experience administrators to maintain and manage the system.

Most of the present RMS use centralized and client-server approach to perform monitoring meaning that the RMS must be installed in one centralized and dedicated server. Centralized and dedicated hardware approach is often the cause of single point of failure problem. Thus, in order to maintain high-availability of the RMS, additional hardware is often required to achieve high-availability which increases operating cost and management complexity. Apart from that, most of the current RMSs are having difficulty to maintain high-availability. For conventional practice, there will be only one redundant server prepared for hot swap in case of failure and such configuration can only withstand at most one failure. More redundant servers are required for RMS to achieve better availability. Unfortunately, more redundant servers will require more complex configuration and more time to perform the configuration which eventually increases the overall operating cost to IT management.

In view of the importance of HA-RMS and additional servers are expensive, cloud-based monitoring is introduced and has started to gain popularity for recent years. Cloud-based monitoring solved the hardware cost problem by renting services (Resource monitoring-as-a-Service) from cloud vendors. Users are freed from configuration and maintenance problems and they use the monitoring services directly. Majority of the cloud-based monitoring service vendors offer 99% of availability of service level agreement. Despite the HA advantages, cloud monitoring services are very network dependent and do not guarantee lower cost compare to in-house monitoring in longer term.

This work proposed a high-availability resource monitoring framework called Navigator, which aims to improve the management complexity and scalability of the RMS, and ultimately achieve automated high-availability with minimal human intervention via automated failover and fallback mechanism through automated service placement of virtualized RM.

This work is inspired by the vision of autonomic computing: *System manage themselves according to an administrator's goals*. (Kephart and Chess 2003, p. 41–50) The essence of autonomic computing systems is self-management, which intended to free system administrator from the details of system operation and maintenance, and to provide users with a machine that run at peak performance 24/7. With this inspiration, the work emphasizes on the automation of various administration process to achieve high-availability resource monitoring goal with minimal human intervention.

1.2 Project Motivations

The key motivation of this project is to enhance current RMS' high-availability architecture and its scalability via automation. Constructing a high-availability RMS isn't new. However, most of the existing solutions require intensive human intervention to configure, maintain and manage the high-availability architecture. Consequently, the reliability of the RMS and the high-availability setup is greatly depend on how the system administrator setups the whole system. In this project, we aim to improve this situation via pre-configuration of the high-availability system, together with automation of most the operating processes, so that only minimal human intervention is required to maintain the RMS.

Apart of that, although hardware redundancy is one of the most common method to secure high-availability and it has proven successful, but this method is poor in scalability and the number of consecutive failure it can withstand is subjected to the number of extra hardware is deployed. Thus, in this project, we will focus on how to lessen the hardware dependency, in order to improve the overall scalability of the high-availability architecture, and eventually allow the system to resist to more consecutive failure with minimal operational process and cost.

1.3 Problem Statements

Contemporary monitoring systems provide comprehensive monitoring features which sufficient to cover different type of complex heterogeneous or homogeneous computer system landscape. Also, high availability can be achieve via hardware-redundancy with a series of complicated configuration and maintenance procedure.

However, such approach required intensive human intervention and high operation cost.

In order to allow a RMS running under high-availability, it involved activities such as setting up the extra monitoring service in redundant server, manage the data synchronization and etc. Those processes usually **involve a series of detailed and advanced configuration steps**. System administrators have to proceed all the steps meticulously in order to ensure the system is working properly. Consequently, it is difficult to maintain and manage the RMS. This situation becomes worse when come to the substituting experience staff with new staff who without any experience. The learning curve of new staff to handle such system is steep due to its complexity.

Most of the existing RMSs are hardware dependent and so with the high-availability setup. For common high-availability setup, the system administrator will required to deploy two dedicated hardware in the IT landscape. One of the hardware will actively running the RMS while another redundant hardware will configured to stay idle for immediate failover. However, such architecture is hard to scale as in it could be difficult and complex to add the third hardware to achieve higher availability.

Inherited from scalability issue, **high availability via redundancy only can withstand a limited number of consecutive failover** and it is subjected to the number of redundant hardware is deployed. Besides, in most of the case, fallback process need to be done manually due to lack of automation mechanism. As a result, the RMS might not have enough room to perform failover if there are multiple failure happens consecutively

Therefore, to mitigate the issues listed above, a high availability Resource Monitoring Framework on dynamic distributed environment is proposed in this project.

1.4 Project Objectives

This work is presenting a self-managed high-availability resource monitoring framework by using automation mechanism. Thus, in order to achieve the goal, the project objectives are listed as below.

1. To design a dynamic service placement mechanism which enable dynamic service migration and shifting among heterogeneous hardware with the integration of service encapsulation. This mechanism will self-managed and performs all the configuration and deployment work involved in service migration and placement without human intervention.

2. To define and automate processes on failover and fallback to achieve high-availability without human intervention as well as additional hardware. At the same time the system able to resist multiple consecutive failovers.

3. To investigate and validate the usability of the defined framework, a prototype of the system will be built deployed under a small-scale system landscape. Various testing include examine the automated failover processes, service performance measure, overhead and etc. will be carried out.

1.5 Project Scopes

In general, this project mainly involved in constructing the high-availability resource monitoring framework and architecture namely Navigator, followed by the development of prototype the exam the usability and the performance of the framework. Hence the project scope is defined as below.

1. The prototype is developed to run under Windows .NET environment. Windows .NET is chosen because most of the machines in laboratory labs are installed with Windows platform. Develop under Windows .NET will ease the deployment of testing.
2. The proposed High-Availability Resource Monitoring Framework only operates within a Local Area Network (LAN). Although the resource monitoring scope might cover up to Wide Area Network (WAN), but Navigator framework will only enforced within the LAN where the RMS is deployed.

3. The RMS including the core monitoring application, web server and database are encapsulated with virtualization technology. These components have to be install in one single instance and it is not allowed to reside in different physical hardware under Navigator framework.

4. This work will only focus on infrastructure monitoring which cover metrics such as system availability, CPU and memory usage, and network utilization of the monitored host.

1.6 Project Contributions

As RMS is crucial to an IT landscape, failure of resource monitoring is almost intolerable and it should not require too much downtime for maintenance purpose. In this project, a high availability resource monitoring framework is designed to reduce the RMS management complexity and the same time the RMS's availability is maintained via automation process. As a result, the operation processes will be much simplified and system administrator will be freed from the detail of system operations and maintenance, which is corresponded to the motivation of autonomic computing. .

In the future, similar framework can be used to implement any generic services for instance web services and database services with minimal management complexity to achieve high-availability. As the complexity of management is reduced, system administrators can now devote their time in higher level system management such as optimization of the resource utilization of an IT landscape.

High-availability via redundancy is well-known to be success but it exposed to scalability issues. In this project, we improved the scalability issue with automated dynamic service placement method. Without the need of introducing a complete set of dedicated redundant hardware to the landscape, the high-availability is scaled by harnessing the redundant resources available under the IT landscape.

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

Plenty of RMSs and high-availability architecture have been developed and studied over years. They differentiate each of other in term of architecture, features and type of computer systems they specialized. In general, this work classifies all the related work into 3 major domain: high performance computing monitoring, enterprise or industry computing monitoring and cloud-based monitoring. Each of this domain will be review in section 2.2.

Literature review on the existing services encapsulation technologies will be done in section 2.3 and lastly a discussion and conclusion will be made in section 2.4 and section 2.5

2.2 Review on Existing RMSs

2.2.1 High Performance Computing Monitoring

High performance computing system such as Grid and cluster usually consist large amount of interconnected node. Therefore, scalability is very important in order to monitor this type of distributed system. Ganglia(Massie, Chun and Culler 2004, p. 817–840), is a RMS which leverage hierarchical architecture to perform monitoring for large-scale distributed system. Even though Ganglia still using centralized management, but Ganglia solved the problem of single point failure via hierarchical monitors with the cost of redundancy. In order to ensure data availability for a set of cluster, data of each node will publish to other nodes within the same cluster via multicast protocol. Similarly, monitoring service availability is maintained via introduce multiple redundant gmetad (a Ganglia daemon which allows monitoring information for multiple cluster to be aggregated). Although high availability can be easily achieved in Ganglia via introducing redundancy in various components, but it might create a lot of overhead to the system. The multicast protocol itself creates high bandwidth consumption to the landscape and furthermore the number of failure is limited to the number of redundancy deployed. Also, there is no clear automated fallback mechanism if defined in the system if failover occurred.

Instead of a pure RMS, Astrolabe(Van Renesse, Birman and Vogels 2003, p. 164–206) is an integrated system for distributed system monitoring, management and data mining. The design of the system is focus on its scalability and it achieves scalability through its zone hierarchy. Astrolabe uses Peer-to-Peer protocol for communication to eliminate single-point of failure and increase the system robustness. By utilizing the design of zone hierarchy and peer-to-peer communication, Astrolabe is highly scalable and basically the failure of any single node in the system will not affect the service, however the performance of Astrolabe is very much depends on the zone definition and unfortunately system administrators responsible for configure the system and assign zone. System administrator is required to have in-depth knowledge of their IT landscape's network in order to define all the zones in the consideration of different topology might result in different overhead resulted from communication of Astrolabe system. In this case, management of Astrolabe becomes difficult and complex. Furthermore, Astrolabe uses IP multicast to communicate between hosts. If monitored machines does not support IP multicast, Astrolabe will do broadcast communication which might result vast overhead to the network.

Both Ganglia and Astrolabe are highly-scalable and resilient to failure based on their architecture. However, approach such as introduce multiple redundancy and vast communication between nodes to control failure can generate a lot of overhead. Such amount of overhead can only be justified if we want to monitor a very large scale distributed system else apart of wasting resource, it might overload the landscape for instance a distributed system which only contain less than hundred nodes. Furthermore, hierarchical monitoring is not easy to be configured and maintained if the system administrators do not have in-depth knowledge of the network. Misconfiguration will affect the performance of the system. Furthermore, the management of the system will getting more complex while the landscape is scale.

2.2.2 Enterprise/Industry Computing Monitoring

Depends on the size of the organization, the IT infrastructure can scale from few ten nodes to few hundred or even thousand nodes. Generally, Enterprise/Industry computing are more heterogeneous in the sense of different resources are integrated under one landscape. For example under an enterprise IT landscape, it might consist of workstation, server farm, cluster, virtual instances, cloud and numbers of network device. RMS like Nagios(Nagios 2002), Icinga(Icinga 2009), Zabbix (Zabbix 2001), Zenoss (Zenoss, Inc. 2005), openNMS (The OpenNMS Group Inc. 2002) and Solarwinds (SolarWinds 2003) are dedicated to monitor this kind of IT infrastructure.

These RMS differentiate themselves in term of features provided for instance reporting, system performance analyzer, open source and etc. However most of them are design under similar architecture which is centralized management of the monitoring server. RMS are required to installed is dedicated hardware and the monitoring process is carried in a client-server fashion. Such approach is exposed to single point failure. Some RMS like Icinga has documented how to set up redundant monitoring to handle failover. Redundant monitoring means user will required to set up another identical monitoring server. Furthermore, users will need to write some automation script to control the process. Although the guidelines is given, but the process is complex and any wrong configuration might fail the whole high availability setup. Apart of this, there is no explicit guideline in term of fallback after failover and the number of consecutive failure the system can withstand is subjected to the number of redundant servers present.

2.2.3 Cloud-based Monitoring

With the rise of cloud computing, cloud-based monitoring is introduced since past few years. Basically, cloud-based monitoring provides monitoring as a services to customers. In this case, user will not need to install the monitoring system on premise but just required to subscribe to the monitoring services. There are numbers of cloud-based RMS namely: Rackspace Cloud Monitoring(Rackspace, US Inc 2014), Monitis(Monitis.com 2006), Logic Monitor(LogicMonitor 2008) and Opsview(Opsview Ltd. 2014). Similar to enterprise computing monitoring, cloud-based monitoring perform centralized infrastructure and application monitoring. However, scalability of the monitoring is based on how much subscription fees that the user willing to pay.

Since the RMS is hosted on the cloud, the user will not require to configure and maintain the RMS. Generally, he or she only required to add or remove monitoring host and metrics through the administrator panel or API provided. For instance Rackspace Cloud Monitoring allowed user to monitor anything by creating entity via their API provided. Compare to enterprise or HPC monitoring, the hassle of installation and maintenance is greatly reduced by using cloud-based monitoring as those tasks have become the responsibility of the service provider. Apart of that, for instance Rackspace Cloud Monitoring maintain the service availability by having multiple monitoring zone globally so services and failover to different if any problem occur.

Compare to conventional enterprise and HPC monitoring, cloud-based monitoring indeed eliminates most of the service management problem and we have choice to acquire monitoring on-demand. However, there are several potentials pitfall with cloud-based monitoring. First of all, cloud-based monitoring required user to have consistent communication to the cloud. For example Rackspace Cloud Monitoring uses REST to perform the pulling and pushing of data continuously. This might impose high bandwidth consumption when the monitoring scope is scaled. The situation becomes worst if user would like to have high frequency checking onto the IT landscape. Also, cloud-based monitoring charge user per usage. In long term, it might be more expensive than hosting a monitoring services in-house.

2.3 Service Encapsulation

In order to achieve dynamic service placement, the RMS need to have portability in order to move within different environment or even physical hardware. However, most of the existing RMS required numbers of dependency package to be installed. For instance, a database management system (DBMS) to record data and web server to host the graphical user interface (GUI) is required to run a monitoring services. Such dependencies make the service lack of portability. Thus, in order to solve this issue, we need to encapsulate the RMS into a single component so that RMS is portable and able to perform dynamic service placement between hosts.

2.3.1 Virtualization

Computer resources have become more powerful and inexpensive compare to the past. There are always excessive computing resources in one physical machine which sufficient to run extra set of operating system application. Virtualization realizes the concept of running multiple virtual machines in one single hardware platform. By using virtualization technology, a single machine able to aggregate all kinds of data resources, software resources and hardware resources for different tasks. Moreover, virtualization separates hardware and software management, and provide useful features such as performance isolation, server consolidation and live migration(Clark et al. 2005, p. 273–286). In addition virtual technology can also provide portable environments for the modern computing systems.(Li, Li and Jiang 2010, p. 332–336) Thus, we can encapsulate the whole monitoring services into a virtual machine and make it portable across different hardware platform.

Apart of that, although most of the existing resources monitoring solution did not officially support hosting in a virtual machine, but for demonstration or simplicity purpose, resources monitoring application such as Icinga, Zabbix, Hyperic and Nagios provide pre-configured virtual machine which allows user to deploy and experience the RMS without the need of carry out any configuration and installation. This proved that, it is practical to encapsulate the resources monitoring application together with necessary dependency packages into a virtual machine. It will eliminate the effort of configuration and installation and with the current technology, virtual machine can be easily port within different physical hardware by using different virtual machine migration technique. In this work, virtualization will be leverage in order to achieve dynamic services placement.

2.3.2 Operating-system level virtualization

A subset from virtualization, OS level virtualization or containers virtualization in general, can be define as a technique of virtualization which provide the required isolation and security to run multiple applications or copies under a single OS.(OpvenVZ.org 2005) Compare to full-system or hardware virtualization, OS level virtualization usually generate lesser footprint since they only required to manage a single OS kernel compare to multiple OS kernel, OS-level virtualization lack of portability as it only as the guest container only can hosted in one OS kernel but not different OS, and thus, migration of services is somehow challenging in OS-level virtualization.

Currently, there are numbers of OS-level virtualization implementation is available namely openVZ(OpvenVZ.org 2005), Linux Containers (LXC)(linuxcontainers.org 2014), Jails(DVL Software Ltd. n.d.), Zones(Tucker and Comay 2004) and Docker(Docker Inc. n.d.). In particular, Docker utilizes LXC to provide an open-source engine that automates the deployment of any application as a lightweight, portable, self-sufficient container that will run virtually anywhere regardless different hardware platforms. The vision of Docker is *built once run..run anywhere* and *configure once..run anything*. Docker container is highly portable as almost everything can be encapsulate into Docker and the containers able to deploy in almost any platform without dependency issues at the same time the consistency of the container is maintained. Docker is first introduced in year 2013 and it considers a relatively new technology. Thus it is still not advisable to use in production system since it still under heavy development.

2.4 Discussion

	Feature / System	High Availability features	Automated HA setup	Automated fallback process	Number of consecutive failover can withstand	Dynamically service placement
Enterprise/Industry Monitoring	Nagios	✓	X	X	Subjected to number of redundancy	X
	Icinga	✓	X	X	Subjected to number of redundancy	X
	Zabbix	✓	X	X	Subjected to number of redundancy	X
	Zenoss	✓	X	X	Subjected to number of redundancy	X
	openNMS	✓	X	X	Subjected to number of redundancy	X
	SolarWinds	✓	X	X	Subjected to number of redundancy	X
HPC Monitoring	Ganglia	✓	X	X	Subjected to number of redundancy	X
	Astrolable	✓	✓	X	Subjected to number of nodes	X
Cloud-based Monitoring	Rackspace Cloud Monitoring	Subject to SLA agreement				X
	Monitis	Subject to SLA agreement				X
	Logic Monitor	Subject to SLA agreement				X
	OpsView	Subject to SLA agreement				X

(✓ indicates the feature is provided; X indicates the feature is not provided)

Table 2-1: Comparison of existing RMS

Based on the tabulated data shown in table 2-1, conclusion can be made as most of the RMSs do provide high-availability via hardware redundancy. However, in most the case especially for enterprise/Industry monitoring, the high-availability did not come off-the-shelf with application. System administrator need to setup the high-availability function by referring to the manual provided and very often the steps involved is complex.

For instance, according to Zabbix high Availability setup manual(Zabbix.org 2014), in order to setup the high-availability features, system administrator need to do perform configuration on database, Zabbix web frontend, Zabbix server, Zabbix Agent and firewall. Additionally, OpenAIS/Corosync is required for low availability checking and pacemaker to ensure all services are properly switched from on node to another. These instructions are inter-related. Any mistake in the process might cause the high-availability to be fail.

Compare to enterprise monitoring, HPC monitoring have more sophisticated method to ensure high-availability. Ganglia replicates the monitoring data in many instances so that when a particular node failed, the monitoring data still available to user whilst Astrolabe aggregate the monitoring data through Peer-to-Peer protocol and zone definition. However, too much redundancy might intrusive to the system and it is inefficient, thus such high-availability concept only justified in large-scale distributed system like clustering and Grid.

Cloud-based monitoring hides all the configuration and architecture of the monitoring application from the user. They provide monitoring as a service and what user's is needed to is define all the monitoring metrics provided at the service level and hence, the high-availability setup is not visible to user. In most of the time, if users would like to acquire certain level of high-availability, they will need to define such requirement in the service-level agreement with their vendor.

Although, cloud-based monitoring is good in such a way that it eliminates all the configuration and maintenance process compare to conventional monitoring, but user have no control on the monitoring system and eventually might fall to the risk of vendor lock-in. For instance, software vendor might only allow specific high availability module to be integrated with the cloud-based monitoring services and leave user no choice

Despite of cloud-based monitoring, almost all the existing RMSs do not offer automated fallback. Imagine the scenarios of one active monitoring server and one redundancy server is up. At the moment, the availability of RMS is secured, once the active monitoring server is failed it will automatically failover to redundancy server, given all the high-availability configuration is setup properly. At this point of time, the high-availability is broken down because there is only single monitoring server is running. If any failure is happened again at this critical period, the monitoring service will no longer be available to user.

If the cost of introducing more redundancy hardware is acceptable, the problem mentioned above can be relieved by introduce more redundant monitoring servers so that the architecture can withstand more than one failure event. However, in most of the time it is not cost-effective and the effort and complexity of maintenance increase proportionally with the number of redundancy implemented in the landscape.

Virtualization technology promotes portability and simplicity in computing system. In general, virtualization is platform or framework which allows multiple independent operating system or even services to work on top of a single hardware instances. It makes sense to encapsulate a RMS into a virtual container to grant the monitoring services better portability and eventually it be used to design a dynamic service placement architecture.

As conclusion, this project will focus on designing an automated high-availability resource monitoring framework which capable to perform self-configuration and self-management and ultimately, increase the efficiency and effectiveness of the whole resource monitoring management with minimal human intervention. To achieve that, a dynamic placement architecture will be designed and together with monitoring services encapsulation, a high-availability framework will be defined.

CHAPTER 3 METHODOLOGY AND TOOLS

3.1 Methodology

In this project, spiral model will be adapted as the software development methodology. Each of the alphabet inside the spiral represents the milestone of the project.

The process of the software development can be depicted visually as figure 3-1.

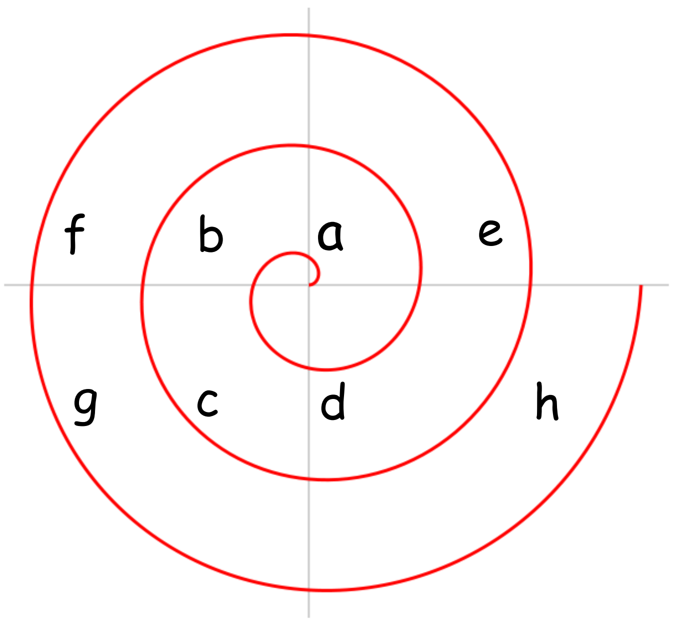


Figure 3-1: Spiral Model of software development

(Please refer to the next section for the detail description of each phase from a-h)

3.1.1 Prototype 1

The first spiral will be started with requirement gathering and feasibility study of the project. Activities such as literature review, objective and problem statement setting will be done subsequently. After the all the objectives is set. The process will be continue by developing the 1st prototype. The requirement of 1st prototype will be listed as below.

a. RMS is encapsulated in a virtual machine and capable to perform monitoring in a controlled computer systems.

b. The virtualized RMS is capable to migrate to any hosts without the monitoring process being interrupt.

After the 1st prototype is completed, benchmark on the resource consumption of the virtualized RMS to the physical host will be carried out.

3.1.2 Prototype 2

Spiral 2 will begin with the evaluation of the 1st prototype. After the evaluation completed, the development of 2nd prototype will begin with the requirement listed as below.

c. Migration process will be automated by a coordinator program

d. Coordinator program are required to handle the migration process for both failover and fallback process

e. Coordinator program is required to have the capability of making decision on who is the optimal host to migrate.

Testing on the coordinator program’s functionality will be carried out after the 2nd prototype is completed.

3.1.3 Prototype 3

Similarly, evaluation of 2nd prototype will be done before the development of 3rd prototype begin. The 3rd prototype’s requirements are listed as below.

f. A resource manager program is developed to manage the RMS and a proxy program is developed to allow communication between resource manager program and coordinator program

g. Full system implementation

Benchmarking on the overall performance of the resource monitoring in aspect of native environment, virtualized environment and cloud environment will be done.

3.2 Development Platform and Technology Used

The detail of development platform and technology used in this project implementation is elaborate at each of the section below.

3.2.1 Operating System

1. The high-availability resource monitoring framework is developed an operate under Microsoft Windows .NET environment

2. The RMS will be installed in a Linux CentOS operating system.

3.2.2 Programming Platform

1. C# programming language will be used to develop all the programs which define the high availability resource monitoring framework.

2. Bash shell script is used to achieve some automation in the Linux OS where the RMS resides.

3. PHP scripting is used to developed web-frontend of the high-availability resource monitoring framework.

3.2.3 Resources Monitoring Application

1. Zabbix is used as the RMS.

3.2.4 Virtualization

1. Virtualization technology from VMware is used to achieve service encapsulation.

3.3 System Architecture

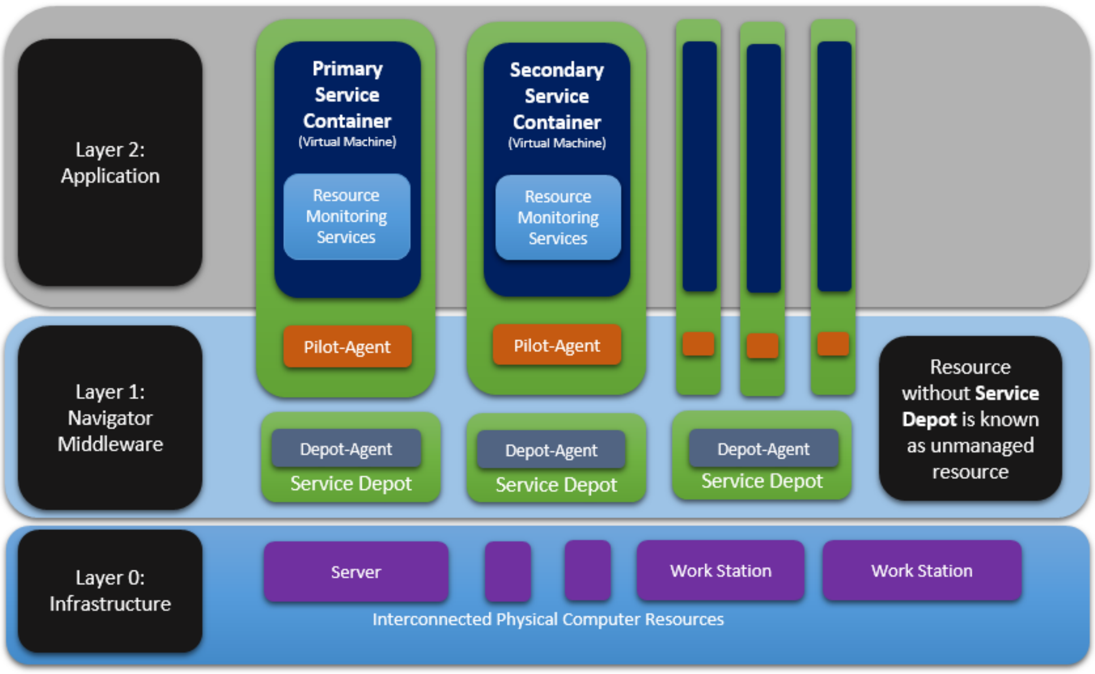

Layer 2: Application
Primary Service Container (Virtual Machine)
Resource Monitoring Services
Secondary Service Container (Virtual Machine)
Resource Monitoring Services
Layer 1: Navigator Middleware
Pilot-Agent
Pilot-Agent
Depot-Agent
Service Depot
Depot-Agent
Service Depot
Depot-Agent
Service Depot
Resource without Service Depot is known as unmanaged resource
Layer 0: Infrastructure
Server
Work Station
Work Station
Interconnected Physical Computer Resources

Figure 3-2: The Architecture Design of Navigator

Inspired from sea port management, we adopted some naming convention from there to describe our architecture design. Under Navigator System, the RMS is encapsulated inside a virtual machine to form a service container. In this case, virtualization helped us to solve the issues of portability. By using dynamic service placement technique, we given the freedom to move the RMS between different resources and ultimately achieve automated high availability without affect the normal monitoring process. In specific, the architecture can be separated in three layer which is:

1. Layer 0: This layer consist a set of interconnected physical resources such as workstation, cluster, workstation or server farm. Navigator did not operate at the hardware layer, thus, only resources which managed by operating system will be consider usable infrastructure under layer 0.

2. Layer 1: The Navigator middleware layer consist three agents which responsible to perform resources management, maintain high availability via automated failover and fallback mechanism. As show in figure 1, Service Depot and Depot Agent will reside under each of the computer resources. They execute on Operating System level. Service Depot manages the local computer resources in the Navigator system. Thus any resource without Services Depot is known as unmanaged resources and will not fall under the scope of Navigator System. As mentioned previously, the RMS will encapsulated into a virtual services container. Pilot agent will take care of this service container and does all the automation process to ensure the availability of services container and eventually the resources monitoring services too. Meanwhile, Pilot Agent will act as a proxy in order to allow communication between Service Depot and Pilot Agent in matter of resources management and coordination. All the communication here will be done by using TCP and UDP protocol.

3. Layer 3: In application layer, Resource Monitoring is encapsulated in a virtual machine together with all the required dependencies package such as web server and database management systems and form a service container. Created an abstraction layer between the RMS and the hardware resources by using three of the agents in middleware layer. The service container will have no knowledge on what resources they are hosted on but the middleware layer

will ensure the availability of services container 24/7 and so with the RMS. Also, at any point of time, a part of primary monitoring services container, there will always be another secondary services container to serve the purpose of redundancy for failover.

3.4 Requirement Specification

3.4.1 Class Diagram

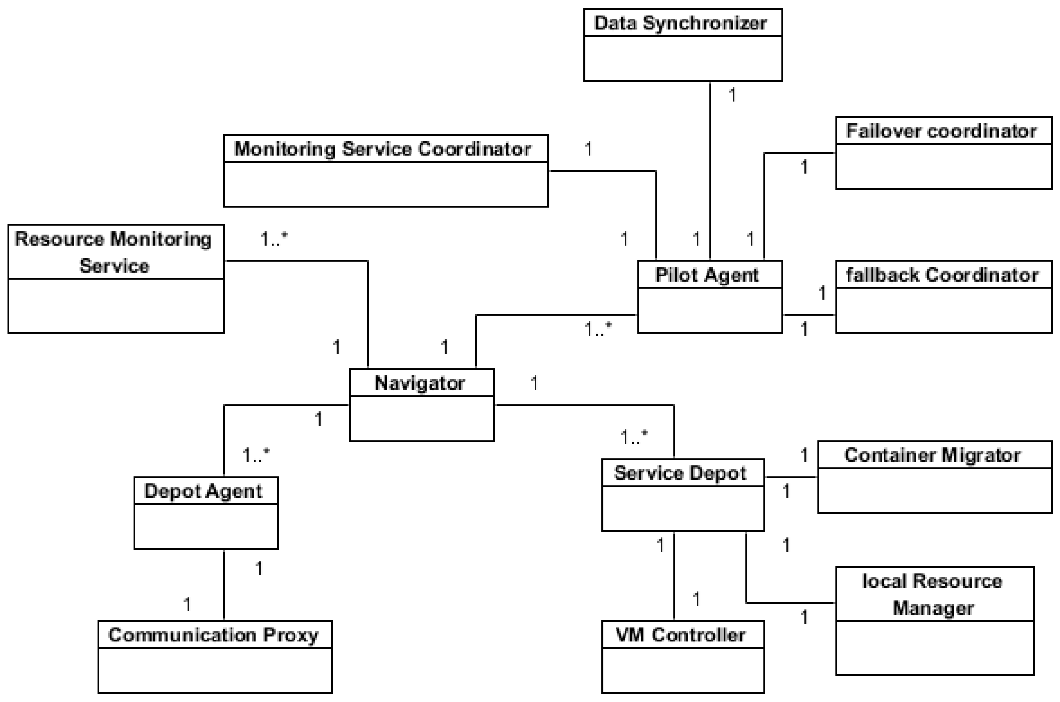


Figure 3-3: Class Diagram of Navigator

Navigator framework composed by three main components which are the RMS, Depot Agent, Service Depot and Pilot Agent. Service Depot and Depot Agent will be installed in every single computing resource under Navigator framework, where else at one point of time, there will be primary and secondary RMS is running under the framework, which manage by two different instance of pilot agent respectively.

Service Depot can be seen as a local coordinator of a computing resources under the scope of Navigator. When there is any service container or in another word virtual machine is hosted under a particular resources, Service Depot will be responsible to perform the instruction to control the virtual machine. Also Service Depot executes the actual operation that needed to be done for service container migration. Apart of that, Service Depot is responsible to provide local resource information to Navigator as per request too.

Unlike from Service Depot, Pilot Agent acts as the coordinator of the resource monitoring service container. At one point of time, there will be always two Pilot Agent working hand in hand to take care both the primary and secondary resource monitoring service container. Pilot Agent makes the decision of which computing resources should host the services by sending appropriate instruction to Service Depot. Pilot Agent is also the component which ensures the availability of the RMS. Pilot Agent responsible for all the failover and fallback process under Navigator framework. Apart of that, Pilot Agent also required to perform data synchronization between primary and secondary resources monitoring services container.

Pilot Agent is the decision maker whilst Service Depot is the daemon who executes all the actual operation upon request. Both of them need a communication channel to talk to each other. In this case, Depot Agent is the proxy between Pilot Agent and Service Depot

3.4.2 Use Case Diagram

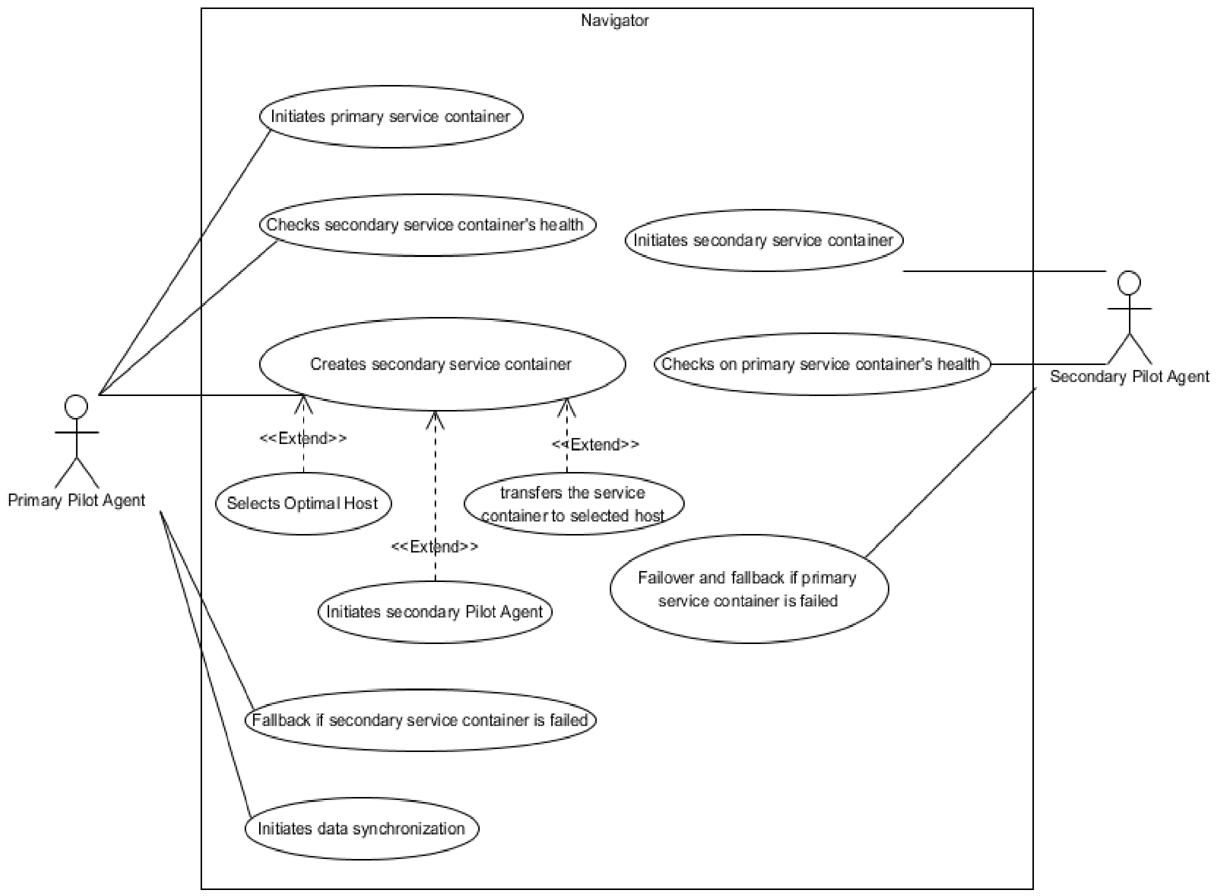

Navigator
Initiates primary service container
Checks secondary service container's health
Initiates secondary service container
Creates secondary service container
Checks on primary service container's health
Secondary Pilot Agent
<<Extend>>
<<Extend>>
Primary Pilot Agent
Selects Optimal Host
transfers the service container to selected host
<<Extend>>
Failover and fallback if primary service container is failed
Initiates secondary Pilot Agent
Fallback if secondary service container is failed
Initiates data synchronization

Figure 3-4: Use Case Diagram of Navigator

Both primary and secondary Pilot Agents are the decision makers under Navigator framework. By default, primary and secondary Pilot Agent is identical in term of their capability, but they will perform appropriate operation according to the role they are opposed, which is either primary or secondary.

Primary Pilot Agent responsible for the setup of primary resource monitoring service container. After the setup completed, primary Pilot Agent is required to initiates the setup of secondary service container under Navigator landscape to ensure high-availability. Under this event primary Pilot Agent need to select the optimal host and transfer the service container disk image to the selected host. The responsibility of Pilot Agent is consider done once it initiates the execution of secondary Pilot Agent on the selected host.

After the secondary Pilot Agent is started to operate, he will perform all the necessary work to setup the secondary service container. Once the secondary service container is ready, the high-availability is secured. Both Pilot Agent will check on each other service container to ensure their availability. If any service container failed, both Pilot Agent will execute failover and fallback process to ensure the availability of the RMS. Apart of that, primary Pilot Agent will perform data synchronization between primary and secondary service container to achieve data consistency between both service containers.

3.4.3 Activity Diagram

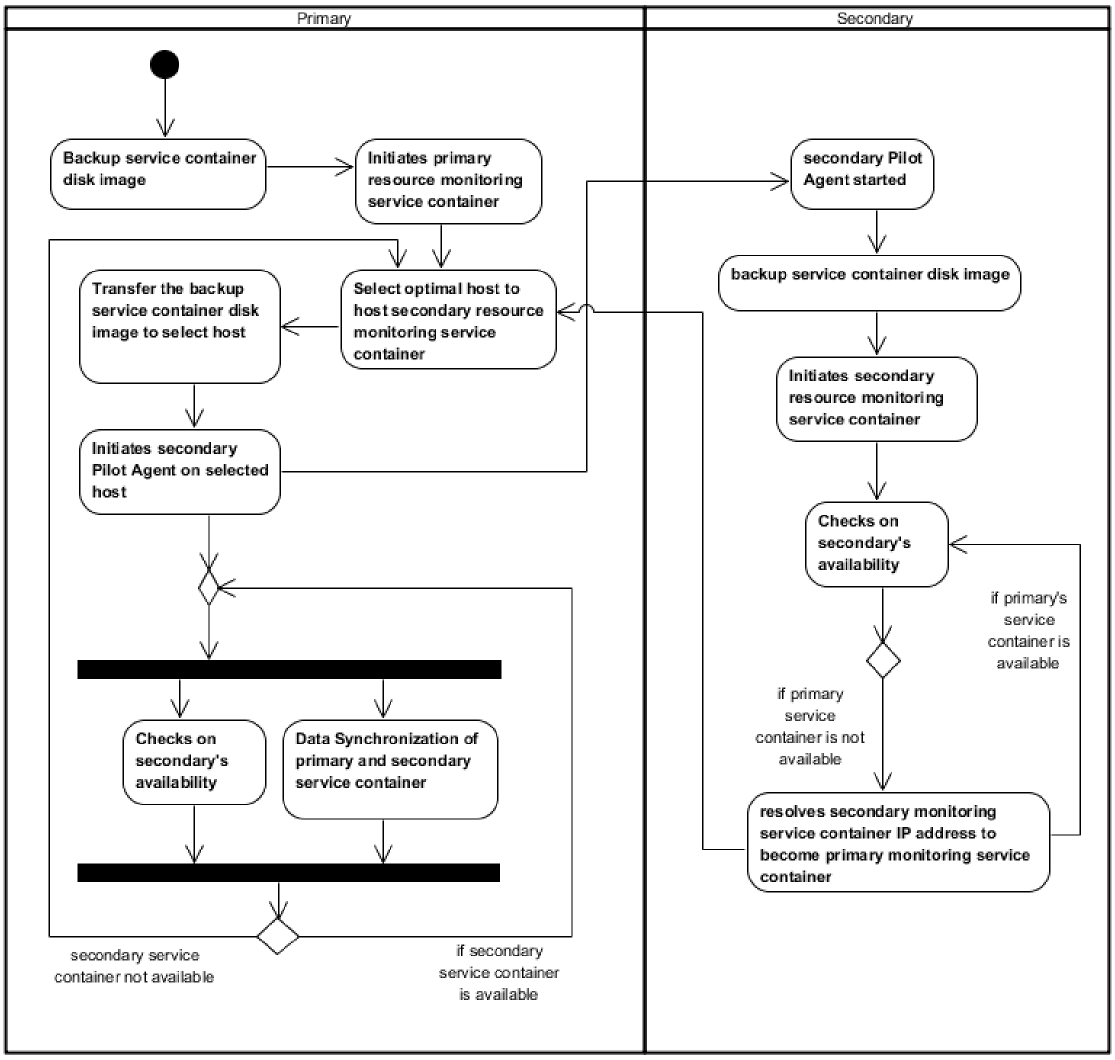

Primary
Secondary
Backup service container disk image
Initiates primary resource monitoring service container
Select optimal host to host secondary resource monitoring service container
Transfer the backup service container disk image to select host
Initiates secondary Pilot Agent on selected host
Checks on secondary's availability
Data Synchronization of primary and secondary service container
secondary service container not available
if secondary service container is available
secondary Pilot Agent started
backup service container disk image
Initiates secondary resource monitoring service container
Checks on secondary's availability
if primary's service container is available
if primary service container is not available
resolves secondary monitoring service container IP address to become primary monitoring service container

Figure 3-5: Activity Diagram of Navigator

Once the primary Pilot Agent is started to execute, it will first backup the service container disk image it opposed. Then, primary Pilot Agent will initiates the primary recourse monitoring service container and kick start the RMS in the landscape. Once the RMS is up and running, primary Pilot Agent will started to find a suitable computing resources under Navigator landscape to host the secondary resource monitoring container. When a suitable host is found, primary Pilot Agent will send the backup disk image to the particular host. Once the transfer is done, Primary Pilot Agent will then kicks start secondary Pilot Agent at the selected host.

On the selected host, the secondary Pilot Agent will kicks start the service container but the RMS will remain idle. When the secondary service container up and running, primary Pilot Agent will periodically check on the secondary service container's availability status and data synchronization will be performed by primary Pilot Agent to ensure the data consistency between primary and secondary service container. Also, at the same, secondary Pilot Agent will periodically check on primary service container's availability to ensure the RMS is running fine.

If the secondary service container is not available in the landscape, this even will be detected by primary Pilot Agent and it will perform failover process of the failure of secondary service container. Primary Pilot Agent will find another suitable computing resources to hose the secondary service container. The process is identical to previous secondary service container setup.

In contrary, if the primary service container is failed, secondary Pilot Agent will resolve the IP address of the secondary service container to be identical to primary service container. At the same time the RMS will be started. Once the failover is completed, the RMS will be again available and the process is hidden from user. Once the secondary resource monitoring container has completely took up the role of primary service container, secondary Pilot Agent will then take up the role of primary Pilot Agent and execute all the responsibility fall under primary Pilot Agent.

3.4.4 Sequence Diagram

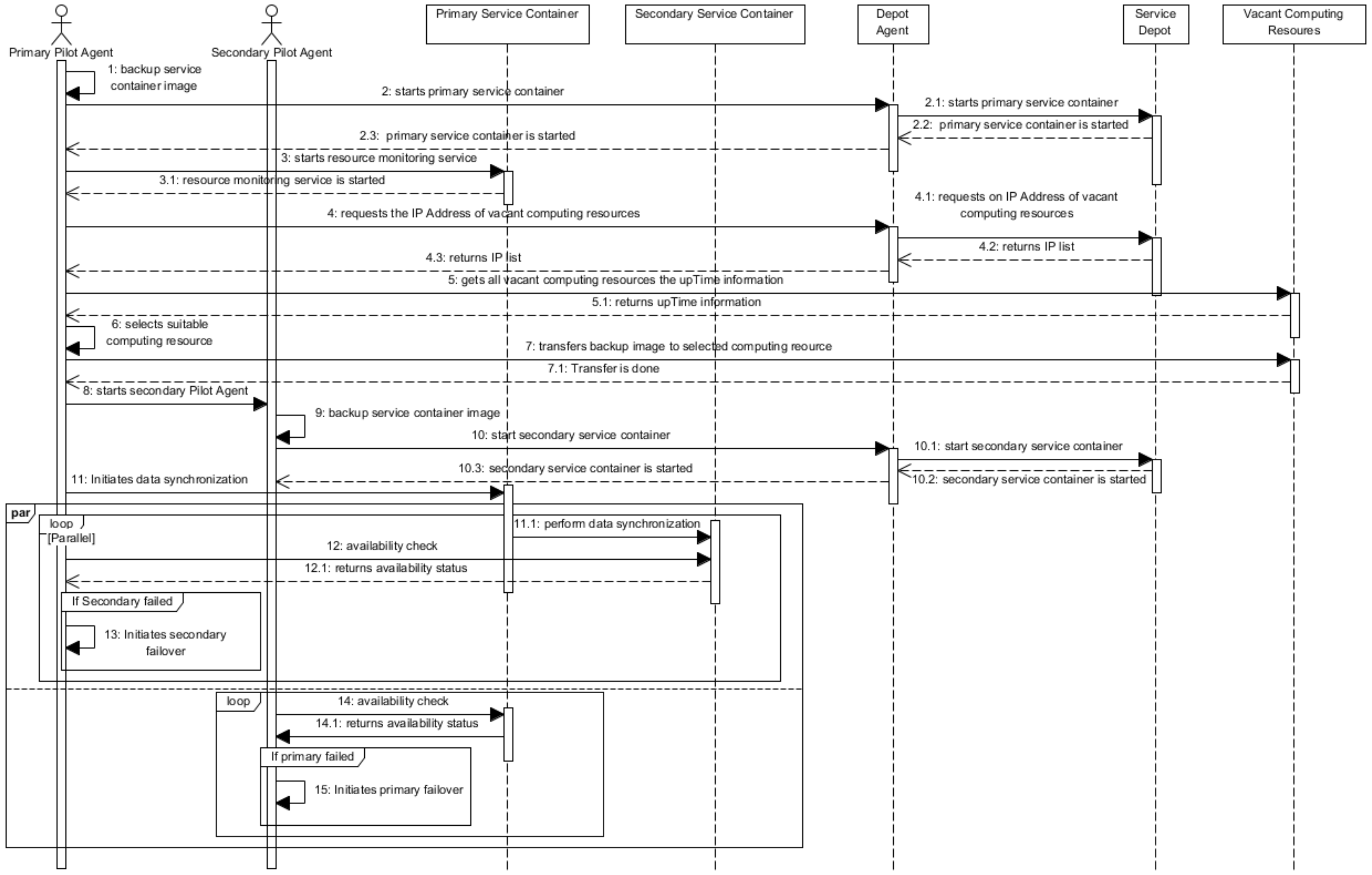

Primary Pilot Agent
Secondary Pilot Agent
Primary Service Container
Secondary Service Container
Depot Agent
Service Depot
Vacant Computing Resoures
1: backup service container image
2: starts primary service container
2.1: starts primary service container
2.2: primary service container is started
2.3: primary service container is started
3: starts resource monitoring service
3.1: resource monitoring service is started
4: requests the IP Address of vacant computing resources
4.1: requests on IP Address of vacant computing resources
4.2: returns IP list
4.3: returns IP list
5: gets all vacant computing resources the upTime information
5.1: returns upTime information
6: selects suitable computing resource
7: transfers backup image to selected computing reource
7.1: Transfer is done
8: starts secondary Pilot Agent
9: backup service container image
10: start secondary service container
10.1: start secondary service container
10.2: secondary service container is started
10.3: secondary service container is started
11: Initiates data synchronization
11.1: perform data synchronization
par
loop [Parallel]
12: availability check
12.1: returns availability status
If Secondary failed
13: Initiates secondary failover
loop
14: availability check
14.1: returns availability status
If primary failed
15: Initiates primary failover

Figure 3-6: Sequence Diagram of Navigator

Figure 3-6 depicted the interaction between each of the component inside Navigator framework. When the primary Pilot Agent is started, it will first backup the service container image. Then, via Depot Agent, it will request for starting the primary resource monitoring service container on the computing resource where it reside on. Once the resource monitoring is up and running, primary Pilot Agent will start to setup the secondary service container. Through Depot Agent, primary Pilot Agent will able to retrieve a list of available computing resource's IP address. From the list, primary Pilot Agent will one by one request the upTime information directly from the particular computing resource and select the most suitable candidate to host the secondary resource monitoring service container.

Once suitable candidate is found, primary Pilot Agent will then transfer the disk image to the selected candidate and kicks start the secondary Pilot Agent. Again, the secondary Pilot Agent will back up the disk image before it starts the secondary service container via Service Depot. The high-availability setup is considers done at the moment of both primary and secondary service container is up and running at the same time.

Primary and secondary Pilot Agent continues to secure availability of the RMS by cross-check on each other service container's availability status. If any service container is failed, corresponding Pilot Agent will initiate the failover process to ensure the availability of recourse monitoring services.

3.4.5 State Machine Diagram

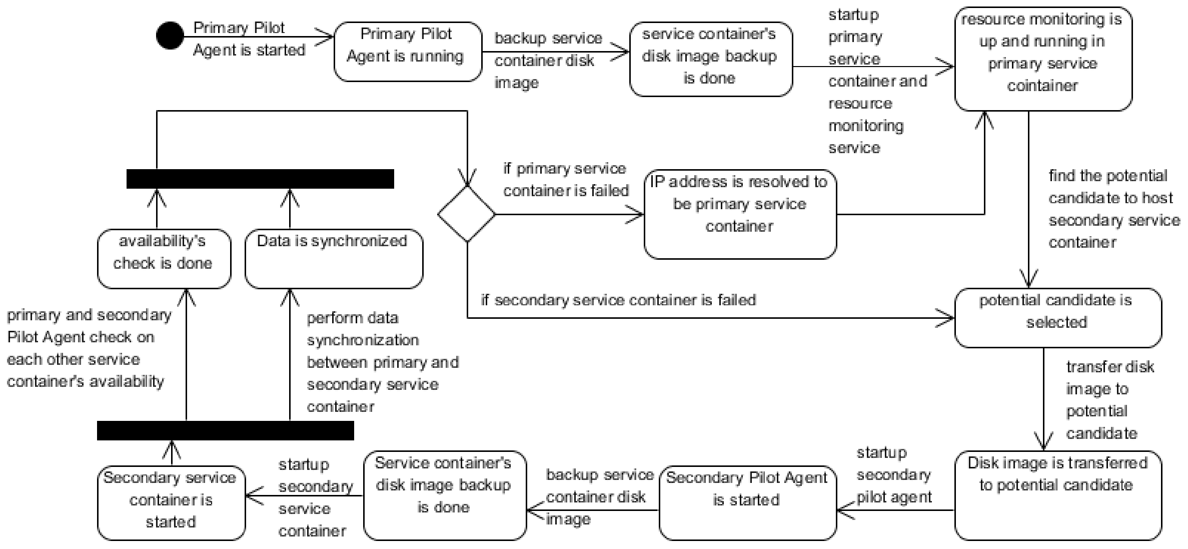
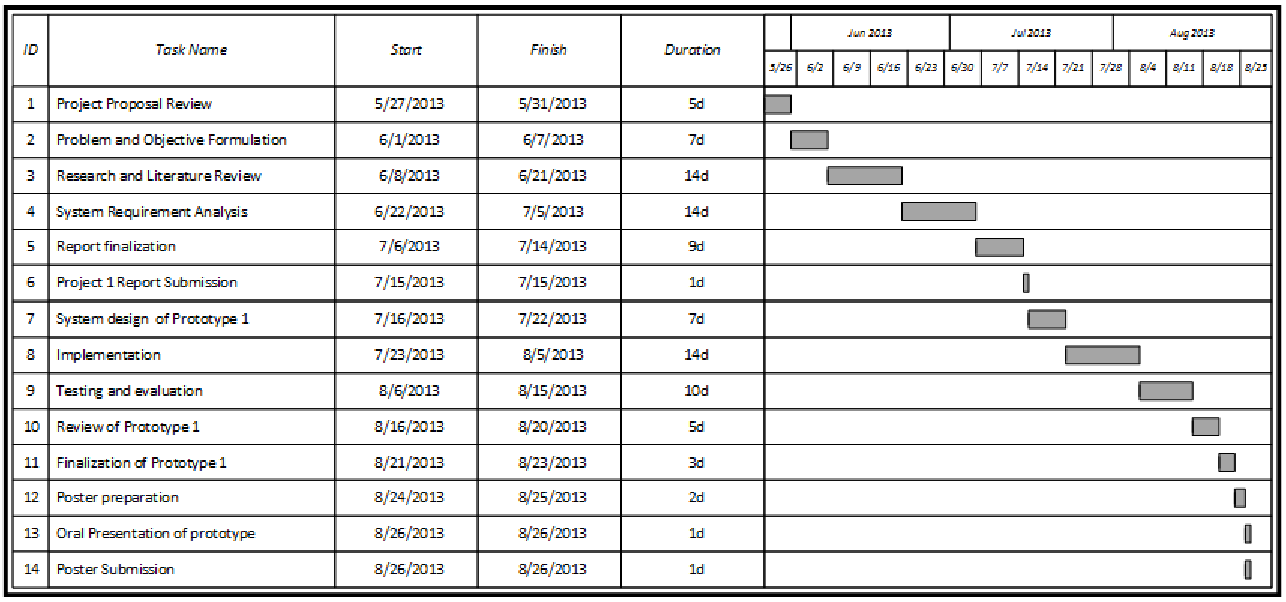

Primary Pilot Agent is started
Primary Pilot Agent is running
backup service container disk image
service container's disk image backup is done
startup primary service container and resource monitoring service
resource monitoring is up and running in primary service cointainer
if primary service container is failed
IP address is resolved to be primary service container
find the potential candidate to host secondary service container
potential candidate is selected
if secondary service container is failed
availability's check is done
Data is synchronized
primary and secondary Pilot Agent check on each other service container's availability
perform data synchronization between primary and secondary service container
transfer disk image to potential candidate
Disk image is transferred to potential candidate
startup secondary pilot agent
Secondary Pilot Agent is started
backup service container disk image
Service container's disk image backup is done
startup secondary service container
Secondary service container is started

Figure 3-7: State Machine Diagram of Navigator

The state machine diagram on figure 3-7 shows the behavior of the whole Navigator framework. The execution begins with starting up the primary Pilot Agent. Once the primary Pilot Agent is running, it will starts to setup the primary service container together with starts up the RMS. Once the primary service container is been setup successfully, primary Pilot Agent will find a suitable candidate to host the secondary service container. When the suitable candidate is found, primary Pilot Agent will then send the backup disk image to the candidate and start up the secondary Pilot Agent over there.

When the secondary Pilot Agent is started, it will setup the secondary service container to achieve high availability through redundancy. Once the secondary service container is up and running, both primary and secondary Pilot Agent will check on each other service container's availability at the same time, primary Pilot Agent will perform data synchronization between both of the host. Both availability checking and data synchronization will be carried out periodically. If any service container is failed, the corresponded Pilot Agent will initiate failover and fallback process to recover the service.

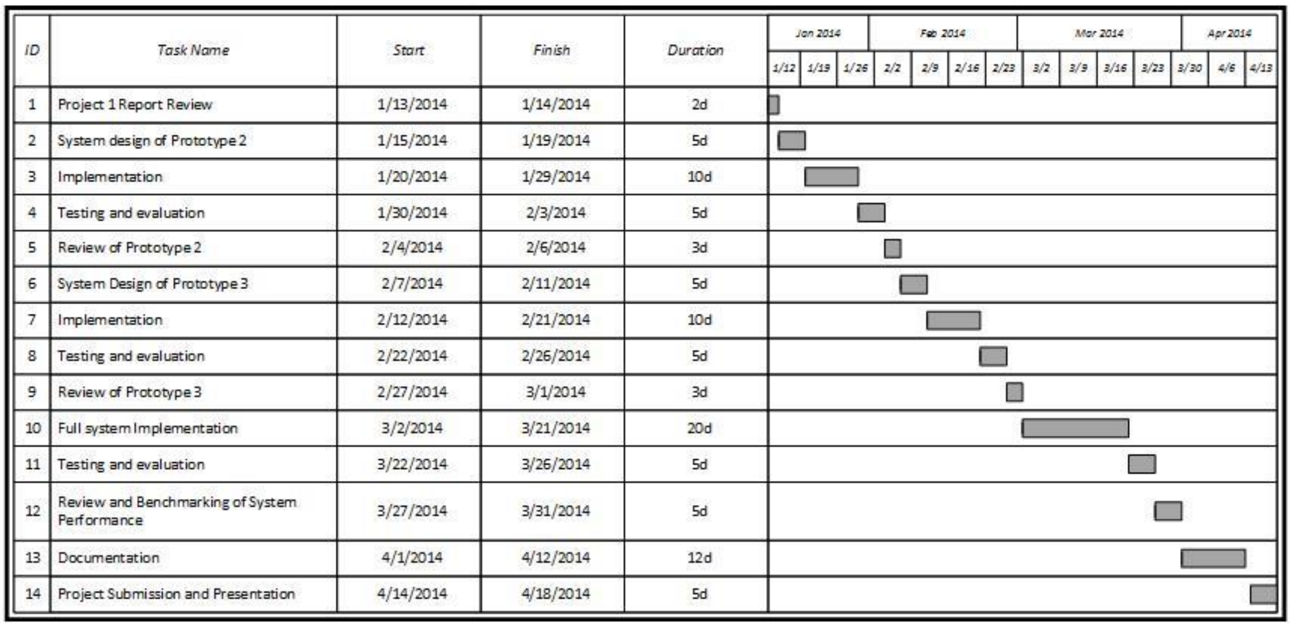
3.5 Project I Timeline



ID	Task Name	Start	Finish	Duration
1	Project Proposal Review	5/27/2013	5/31/2013	5d
2	Problem and Objective Formulation	6/1/2013	6/7/2013	7d
3	Research and Literature Review	6/8/2013	6/21/2013	14d
4	System Requirement Analysis	6/22/2013	7/5/2013	14d
5	Report finalization	7/6/2013	7/14/2013	9d
6	Project 1 Report Submission	7/15/2013	7/15/2013	1d
7	System design of Prototype 1	7/16/2013	7/22/2013	7d
8	Implementation	7/23/2013	8/5/2013	14d
9	Testing and evaluation	8/6/2013	8/15/2013	10d
10	Review of Prototype 1	8/16/2013	8/20/2013	5d
11	Finalization of Prototype 1	8/21/2013	8/23/2013	3d
12	Poster preparation	8/24/2013	8/25/2013	2d
13	Oral Presentation of prototype	8/26/2013	8/26/2013	1d
14	Poster Submission	8/26/2013	8/26/2013	1d

Figure 3-8: Grantt Chart & Milestone of Project 1

3.6 Project II Timeline



ID	Task Name	Start	Finish	Duration
1	Project 1 Report Review	1/13/2014	1/14/2014	2d
2	System design of Prototype 2	1/15/2014	1/19/2014	5d
3	Implementation	1/20/2014	1/29/2014	10d
4	Testing and evaluation	1/30/2014	2/3/2014	5d
5	Review of Prototype 2	2/4/2014	2/6/2014	3d
6	System Design of Prototype 3	2/7/2014	2/11/2014	5d
7	Implementation	2/12/2014	2/21/2014	10d
8	Testing and evaluation	2/22/2014	2/26/2014	5d
9	Review of Prototype 3	2/27/2014	3/1/2014	3d
10	Full system Implementation	3/2/2014	3/21/2014	20d
11	Testing and evaluation	3/22/2014	3/26/2014	5d
12	Review and Benchmarking of System Performance	3/27/2014	3/31/2014	5d
13	Documentation	4/1/2014	4/12/2014	12d
14	Project Submission and Presentation	4/14/2014	4/18/2014	5d

Figure 3-9: Grantt Chart & Milestone of Project II

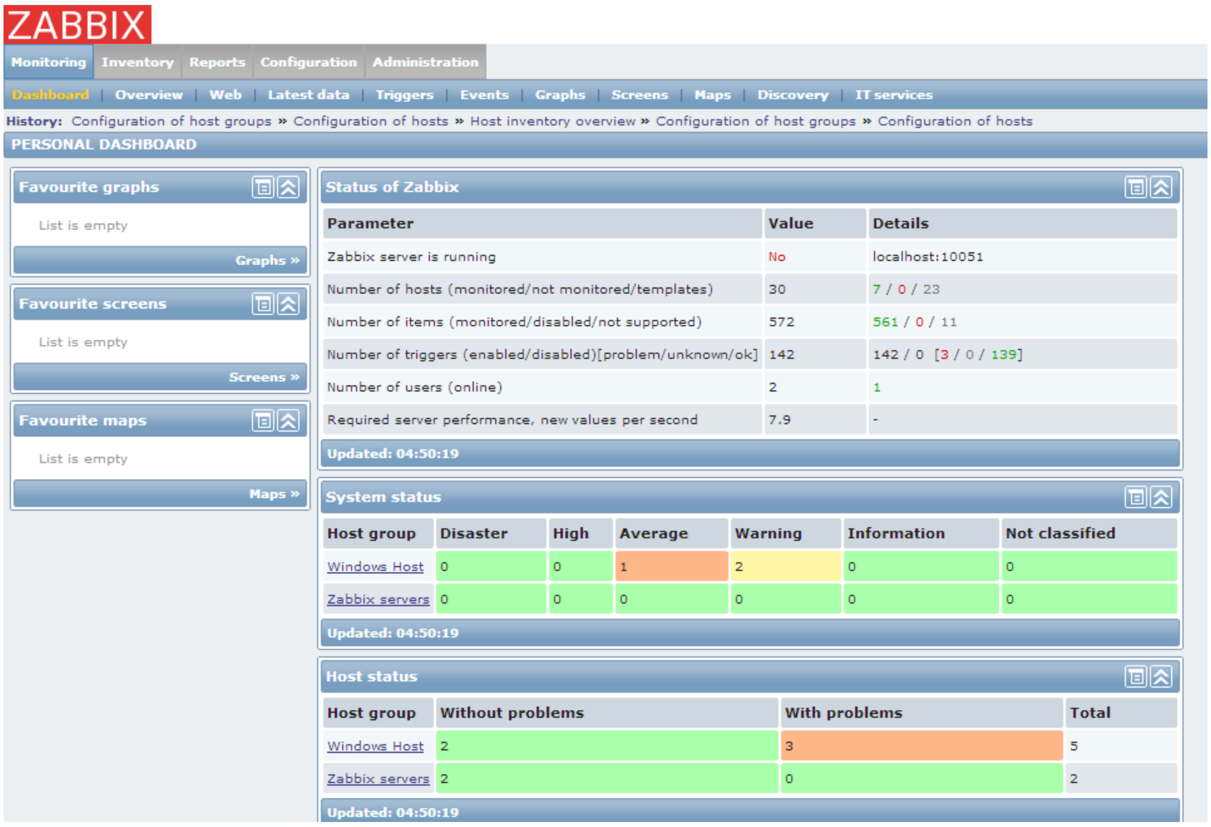
CHAPTER 4 IMPLEMENTATION

The implementation consists of two part. The first part is deploying the monitoring service and encapsulates it into virtual machine whilst the second part involves the deployment of Navigator.

4.1 Zabbix

We have choosen Zabbix as our RMS. Zabbix opposed few features which favor to us. First of all, Zabbix is one of the top-notch open source RMS in the current market. Zabbix can be fully configure via their web GUI and thus it is easy to use and maintains Zabbix without complex configuration on OS level. Monitoring process of Zabbix is straightforward. User only required to install an agent in monitored host, include the monitored host via Zabbix GUI and the monitoring process will begin. One of the crucial design with Zabbix is Zabbix stores all the historical data and configuration in database, this brought great advantages to us in term of setting up primary and secondary monitoring services synchronization. Unlike Zabbix, other RMS for instance Icinga and Nagios, the configuration involve modify config file at OS level and all the config file is store text file separately. It is hard to perform the synchronization and consistency is harder to maintain since the config file might located in different place. In this case, Zabbix solved this issues by store all the configuration together with historical data in a centralized database.

Back to Navigator, we installed Zabbix with all the required dependencies inside a virtual machine. The OS chosen is CentOS 6.5. Also, we installed Zabbix agent in the host we which to monitor and added all the host into the monitoring scope via Zabbix Web interface. The services container is considered done at this moment. Since the service container is a virtual machine, it will be given a static IP address and hence user can login via the IP address. A note here is, this work presents a high-availability framework for RMS, and service container can consider as black box which able to perform monitoring. What application is installed in the service container will not affect the performance of high-availability framework.


ZABBIX
Monitoring | Inventory | Reports | Configuration | Administration
Dashboard | Overview | Web | Latest data | Triggers | Events | Graphs | Screens | Maps | Discovery | IT services
Graphs »
Screens »
Maps »

History: Configuration of host groups » Configuration of hosts » Host inventory overview » Configuration of host groups » Configuration of hosts

PERSONAL DASHBOARD

Favourite graphs

List is empty

Favourite screens

List is empty

Favourite maps

List is empty

Status of Zabbix

Parameter	Value	Details
Zabbix server is running	No	localhost:10051
Number of hosts (monitored/not monitored/templates)	30	7 / 0 / 23
Number of items (monitored/disabled/not supported)	572	561 / 0 / 11
Number of triggers (enabled/disabled)[problem/unknown/ok]	142	142 / 0 [3 / 0 / 139]
Number of users (online)	2	1
Required server performance, new values per second	7.9	-

Updated: 04:50:19

System status

Host group	Disaster	High	Average	Warning	Information	Not classified
Windows Host	0	0	1	2	0	0
Zabbix servers	0	0	0	0	0	0

Updated: 04:50:19

Host status

Host group	Without problems	With problems	Total
Windows Host	2	3	5
Zabbix servers	2	0	2

Updated: 04:50:19

Figure 4-1: Screenshot of Zabbix Monitoring Page

4.2 Navigator Implementation

In navigator, every single hardware resources represent a potential candidate to host the dynamic services container. To bring hardware resources into Navigator, we need to install Depot Agent and Service Depot to manage and coordinate the local hardware resource allocation on each of the instances we which to include in Navigator scope. To start the monitoring services for the very first time, user required to choose a host randomly and place the service container which configure previously into that particular host together with Pilot Agent, and finally, a high availability RMS will be running under Navigator framework.

4.3 High-Availability

Once the RMS is initiated, the Pilot Agent will start to look for a potential candidate to host the secondary monitoring service container. Pilot Agent asks the local Service Depot through Depot Agent to find out who is the best candidate of hosting the secondary services across the network. After the Service Depot received the message, it will broadcast a message to other potential candidates in the network to retrieve all the resources up time information. From there, the local service depot will find out the best candidate based on the longest uptime and respond back to Pilot Agent.

While Pilot Agent knows who the best candidate in the network is, it will initiate the process of creating a secondary monitoring services container. Pilot Agent will send a pre-backup service container to the secondary monitoring services container together with a Pilot Agent to the candidate via Secure Copy. After all the transfer is done, the primary Pilot Agent will start up the secondary Pilot Agent at the candidate itself. Once the Secondary Pilot Agent is started up, it will start to manage the secondary monitoring services container and eventually automate all the failover and fallback process if there is any failure happens on the primary monitoring services. In order to ensure the consistency between both primary and secondary monitoring service container, primary Pilot Agent will periodically backup the database to secondary service container. As mentioned before, all the configuration and historical data is stored under a database, backing up database is sufficient to ensure the consistency between both service containers. At this moment, monitoring services is conducted from primary services container and secondary services container will stay idle. In this case, an automated high availability setup is done. Primary and secondary Pilot Agent will periodically check on each other availability to ensure both primary and secondary services container is up and running fine.

4.4 Failover and Fallback

If the primary services container failed, for instance a hardware shut down on where the primary service container hosted, the secondary pilot agent will initiate the failover which involve the step listed as below:

1. Secondary Pilot Agent will take up the role to become Primary Pilot Agent. It will change the IP address of the secondary service container on-the-fly to become the primary services container.

2. Pilot Agent will start up the Monitoring Services in the services container and ensure the service is started to perform the monitoring on the network.

3. Pilot Agent started to find a suitable candidate to host the secondary services container and the process will be same as what mentioned is section 3.1

In contrary, if the secondary services container fails, the primary Pilot Agent will again find another suitable candidate to host the secondary services container.

We designed all the agents in a way there are identical but have the ability to taking up different role. For instance Pilot Agent has two different role which is primary and secondary according to what services container they are governing on. By doing so, we improved the scalability of the high availability framework. Since a single Pilot Agent can take up any role while necessary, for instance once failure detected, the secondary Pilot Agent will take up primary role and continue to maintain the high availability, we can scale the high availability to withstand more failure by just utilize the existing resources own in the network. Let said we have 5 active hosts under Navigator system, the monitoring service can at least withstand 4 times of failure until the 5th workstation is failed. All the failover and fallback process is automated via the coordination of all agents. Complex configuration and fallback process is omitted.

4.5 Communication

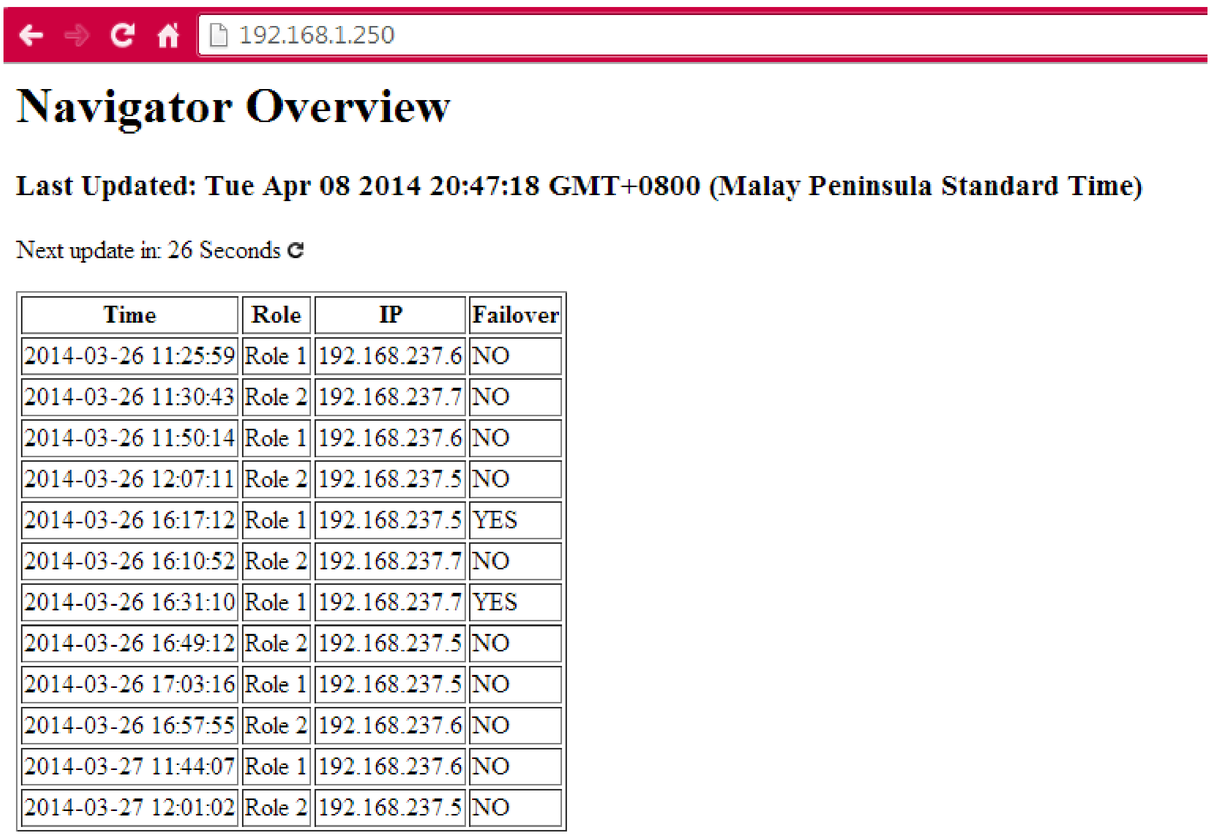
In order to coordinate between agents under the Navigator framework, agents need to talk to each other constantly and so with service containers. There are two type of communication technique is being used by Navigator. As mentioned previously Pilot Agent is the decision maker at the same time Service Depot a daemon to execute the actual operation based on the instruction it receive. In this case, Depot Agent is the one whole act as the proxy between this two agents. In the actual design, Service Depot will only listened to its own Depot Agent, while Depot Agent will listen to any Pilot Agent request. By doing so, all the message posting and respond between Pilot Agent and Service Depot will be centralized with Depot Agent. During any point of time, Depot Agent can handle multiple connection from multiple Pilot Agent and pass the message to the Service Depot. The communication channel between Pilot Agent and Depot Agent is through process message passing whereas Pilot Agent and Service Depot communicate via TCP/IP protocol.

In order to establish connection between Pilot Agent and service container, Secure Shell (SSH) protocol is being used. SSH allowed secure remote login and other secure network service over an insecure network(T. Ylonen 2006). By using SSH, Pilot Agent can remotely login into its service container and perform necessary operation. Besides, SSH File Transfer Protocol (SFTP) is used and programmed to transfer disk image between hosts within the network.

4.6 Service Container movement Tracking

Due to the nature of dynamic service placement of the resource monitoring service containers, there do not have a fix location of where both of the service containers located. Thus in order to track the physical locations where the service containers are located, a tracking mechanism is needed. Under Navigator framework, the one who have the highest availability is the resource monitoring service container itself. So we hosted the tracking mechanism inside the resource monitoring service container so that user able to know where is the primary and secondary service container's physical location at any moment.

The tracking mechanism comprised of two element, database and a PHP web page. Every time when a service container is started in a new location, Pilot Agent will update the location information into tracking database and then, the information will be retrieved and displayed on the PHP web page. User can log into the web page by simply access the IP address of the primary resource monitoring container to see all the movement for both primary and secondary service container in the landscape. Figure below showed the screenshot of the tracking page.



192.168.1.250

Navigator Overview

Last Updated: Tue Apr 08 2014 20:47:18 GMT+0800 (Malay Peninsula Standard Time)

Next update in: 26 Seconds

Time	Role	IP	Failover
2014-03-26 11:25:59	Role 1	192.168.237.6	NO
2014-03-26 11:30:43	Role 2	192.168.237.7	NO
2014-03-26 11:50:14	Role 1	192.168.237.6	NO
2014-03-26 12:07:11	Role 2	192.168.237.5	NO
2014-03-26 16:17:12	Role 1	192.168.237.5	YES
2014-03-26 16:10:52	Role 2	192.168.237.7	NO
2014-03-26 16:31:10	Role 1	192.168.237.7	YES
2014-03-26 16:49:12	Role 2	192.168.237.5	NO
2014-03-26 17:03:16	Role 1	192.168.237.5	NO
2014-03-26 16:57:55	Role 2	192.168.237.6	NO
2014-03-27 11:44:07	Role 1	192.168.237.6	NO
2014-03-27 12:01:02	Role 2	192.168.237.5	NO

Figure 4-2: Navigator Service Container Tracking Page

4.7 Data Synchronization

Data synchronization is critical to ensure all the configuration and history monitoring data is conserved. In order to achieve, when both primary and secondary container are up and running, the primary Pilot Agent will initiate the data synchronization on the primary service container. For every 1 minute, the service container will execute mysqldump(Oracle Corporation 2008)backup to the secondary service container. The process is identical to movement tracking database. The only different here is instead of every 1 minute, the backup is only done once when a new location record is inserted to the particular database.

CHAPTER 5 TESTING

In order to evaluate and benchmark the usability and reliability of Navigator, Navigator is being tested in various perspective under different environment. The testing phase can be divided in to 2 main parts, which is under controlled environment and a dynamic environment. Under controlled environment, we focused on investigating the footprint of Navigator imposed to the network whereas under a dynamic environment, we aim to study the effectiveness of the high-availability mechanism and the movement of service container under Navigator framework.

All the absolute value documented in this section is obtain by calculating the average value of 30 sampling, each sampling is taken at 5 seconds interval.

5.1 Navigator Footprint

Four workstation is used to deploy Navigator. Four of them are interconnected under a local area network and each of them is given a static IP. The specification of the workstation is shown in the table below.

Operating System	32-bit Windows 7
Process	Intel Core 2 Quad Processor Q8400
Memory	3072MB

Table 5-1: Specification of testing workstation

Each of the workstations is installed with Depot Agent and Service Depot. Once the installation is done, the service container is placed inside one of the workstation and Pilot Agent is started on the particular machine. The testing and data collection is begun once the Pilot Agent. Along the testing, multiple failover injection is done to exam behavior the failover and fallback process and the footprint imposed.

Apart from workstation specification, the specification of the resource monitoring service container is tabulated in table 5.2.

Operating System	CentOS 6.5 64-bit
RMS used	Zabbix
Number of process being assigned	1
Amount of memory being assigned	1G
Size of the disk image	1.25G
Hypervisor	VMware Player
Type of network connection	Bridged Connection

Table 5-2: specification of Service Container

5.1.1 Agents Resource Consumption

The Resource Consumption in term of CPU, Memory, Disk and Network I/O is recorded and tabulated in table 5.3.

Name	CPU Usage (in %)	Memory Usage (in MB)			Disk I/O (Kb/sec)	Network I/O (Kb/sec)
		Private Bytes	Sharable Bytes	Working Set		
Pilot Agent	< 0.1	23.8	20.9	44.54	0.4	0.9
Service Depot	< 0.1	9.46	3.86	13.32	< 0.1	<0.1

Table 5-3: Agents Resource Consumption

Depot Agent is not a long run process. It will only being executed when it being called by Pilot Agent and it will be terminated once the operation is done. Due to this nature, the resource consumption of Depot Agent is insignificant to the local computing resource and thus it is not being recorded and documented.

5.1.2 Service Container Resource Consumption

CPU Usage (in %)	Memory Usage (in MB)			Disk I/O (Kb/sec)
	Private Bytes	Sharable Bytes	Working Set	
1.5	20.99	662.7	683	1429.7

Table 5-4: Service Container Resource Consumption

Service container or in another word virtual machine is where the monitoring service is being hosted. While RMS is running, it constantly logging the monitoring data into database, and thus the disk I/O of the service container is relatively high.

5.1.3 Network Consumption

A handful of data transfer operations between local or remote hosts within the network is involved in Navigator mechanism. For instance backing up the disk image and transferring the disk image to other host. It is worthwhile to examine the network consumption during those processes.

Operation	Network Utilization (in %)	Actual Transfer Rate (Mb/sec)	Time Taken(in Minute)
Transfer disk image to remote host	66%	66	2:34
Database synchronization between primary and secondary service container	0.1%	0.1	0:11

Table 5-5: Network Consumption between different operations

5.1.4 I/O Consumption

Backup disk image is one of the essential step in data consistency management. Before the service container is started for execution, a backup copy will be made and for service placement usage in later time. Figure

Operation	Transfer Rate (Mb/sec)	Time Taken (in minute)
Backup disk image	96	1.43

Table 5-6: I/O consumption of backup disk image operation

5.1.5 Failover and Fallback

Time taken to complete the failover and fallback process is important as it directly affect the efficiency and effectiveness of the high-availability framework. Table 5-5 shows the time taken data for various operations.

Operation	Time Taken (in minute)
Failover for primary service container's failure	1:09
Failover for secondary service container's failure	3.45
Fallback for primary service container's failure	5:26

Table 5-7: Time taken of various Failover and fallback process

If the secondary service container is failed, basically the failover process will be the primary Pilot Agent will required to find another suitable candidate to host the secondary service container. The Process is consider done once the secondary service container is up and running. Thus in this case, the failover and fallback process for secondary service container belongs to same process and thus, there is no data recorded for fallback for secondary service container’s failure.

Based on the result tabulated in Table 5-7, the failover for secondary service container is much slower compare to failover primary service container. This is so because is secondary service container’s availability is less important, the primary Pilot Agent will do the checking every 5 minute compare to checking on primary Service Container which is every 5 sec.

5.1.6 Benchmarking

To investigate the performance deterioration of hosting a service container on a workstation, PassMark Performance Test 7 (PassMark 1998) is used to benchmark the performance of the workstation under different workload and the performance when hosting a service container. To benchmark the performance of the pc, PassMark will carry out a series of Performance Test include CPU Tests, graphics Tests, Disk Tests and Memory Tests.

We used a single-threaded CPU-bound program called FiboPrime to simulate workload. FiboPrime is a program which identifies prime Fibonacci number. Since the workstation used is a quad-core machine, by executing one FiboPrime.exe, it will able to simulate 25% workload. Workload can be increase simply by executing more Fibo.exe at the same time. For example executing 2 FiboPrim.exe at the same time will simulate 50% workload of CPU.

CPU Utilization	PassMark Score
Idle	861.8
25%	823.5
50%	792.7
75%	757.2
Hosting a Service Container	854.3

Table 5-8: PassMark Score with different workload

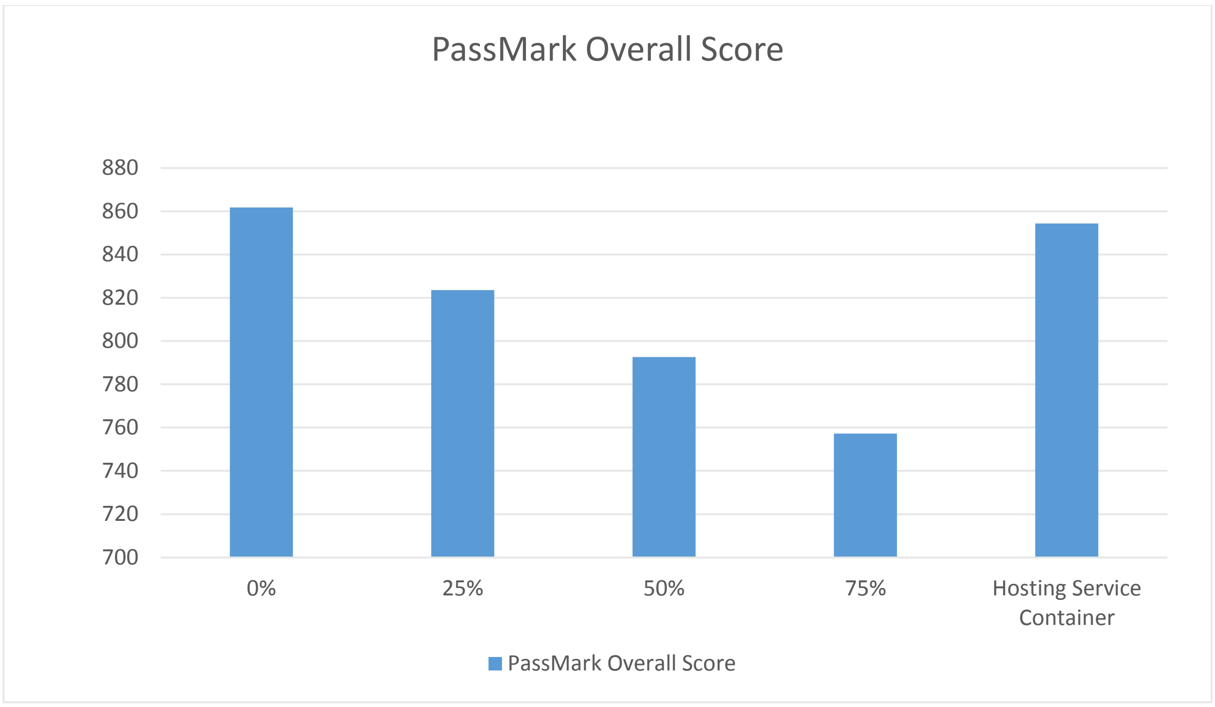


Figure 5-1: PassMark Overall Score

By referring the data tabulated in Table 5-8 and Figure 5-1, the benchmarking result indicates that when a particular physical host is hosting a service container, the host's performance degradation only around 0.7% as compare to 38% when the host on 25% CPU utilization. It proved that the service container and the High-Availability framework did not over-consume the resources of the local physical resources even though the particular host is not dedicated as a component of the high-availability.

5.2 Overall Service Availability and Service Placement Activity

To further examine the usability of Navigator, we have deployed the Navigator framework under an ICT teaching laboratory which having 32 interconnected workstation. ICT teaching laboratory is chosen because every day the workstation is being used by different people. Power on and off the workstation is very often and the activity is very random. Under such environment, we can exam the survivability of the RMS under Navigator framework and to what extend Navigator can secure the service availability

5.2.1 Summary of Experimental Result

We deployed both Depot Agent and Service Depot on 32 computers. We deployed the service container at one of the computer randomly and the experiment is started with execution of the Primary Pilot Agent. The experiment has been carry out for 12 days in total. However, we faced some constraints during this experiment and one of it is the policy of the ICT teaching laboratory is all the computers will be turned off at the end of the day which is 6pm. Thus in the experimental result, the longest period of the RMS can survive is until the end of the day. The detail of Navigator activities at each of the day is documented in Appendix A.

Generally, the experimental result is not as expected. There is very little movement of Navigator and so with the failover process. Most of the day, primary and secondary service container will survive until end of the day when all the computers being turn off. We able to record two failover activities for primary service container however we can’t capture any failover for secondary service container. Apart of that, the resource monitoring is failed once during mid of the day when both the computer is being shut down at about the same time. This indicates that the failover process is still not fast enough to cover immediate failure from primary to secondary service container.

Due to lack of informative data is being collected, there is no conclusion can be made can be make regarding to the usability and reliability of the Navigator prototype. However, since there is a case that RMS failed to perform failover during the mid of the day, it means that the Navigator is still not reliable enough to be use in production. One of the critical issue which led to failure of RMS is the process of data synchronization, backup and transfer keep very long time and its result in the RMS can’t resist to consecutive failure in short interval. Hence, further improvement and optimization need to be done in order to improve the usability and reliability of the system.

CHAPTER 6 PROJECT REVIEW

In this project, a novel Automated High-Availability Resource Monitoring Framework is being devised and developed. In this chapter, the project will be reviewed in different perspective including strength and weakness opposed by the proposed solution. In later section, discussion will be done on identifying potential improvements and future works of the project.

6.1 System Review

6.1.1 Self-Management

The essence of autonomic computing systems is self-management, the intent of which is to free system administrators from the details of system operation and maintenance and to provide user with a machine that runs at peak performance 24/7. (Kephart and Chess 2003, p. 41–50) Align with this motivation, Navigator framework automated most of the configuration and maintenance operation assures the availability of the RMS with the cooperation between three agents: Depot Agent, Service Depot and Pilot Agent.

Most of the conventional High-Availability technique lack of portability due the problem of there is always two different set of configuration for primary and secondary services. However, it is difficult to construct a set of configuration for both primary and secondary service simply because primary and secondary their role is different. One should remain idle while another is active. Hence, in order to improve the scalability of the High-Availability framework, all different set of configuration is encapsulate into Pilot Agent. Pilot Agent will execute appropriate configuration based on which service container it is responsible to.

This is done by ping the IP address the primary and secondary service container. If primary is not exist in the landscape yet, Pilot Agent will take up the role of primary Pilot Agent. Similarly, if primary service container is exist while secondary service container is not, the Pilot Agent will become secondary. Basically Pilot Agent will always be aware which role it should take up in the framework. Portability is improved with such mechanism without the need of any human intervention.

6.1.2 Service Encapsulation

Encapsulate the whole RMS indeed give us much advantage on portability of the service. Under Navigator framework, Pilot Agent can moves the disk images to anywhere and just start up the service from there. The dependency on dedicated machine is greatly reduced in this case. However, the size of disk image produced often very huge. For instance in this project, the disk image produced is 1.2G and the size is growing along with the growth of database size which reside inside the virtual machine. This situation imposed a threat to Navigator framework, if the size of virtual machine is too large, it will directly affect the transfer time and backup time of the disk image. If the disk image is too large, the critical period before fallback is completed will become unreasonably wrong.

6.1.3 Data Synchronization

Similar with conventional high-availability configuration problem, data synchronization is often required two different set up for secondary and primary. In order to eliminate such limitation, we decided to use mysqldump to back up all the configuration and historical data to the secondary service container periodically. The data consistency is still maintained in this way however the process can be intrusive if the database is growing bigger in term of size. One of the possible solution is to only backup the configuration data. But the tradeoff here will be losing the historical data which is often intolerable to user.

6.1.4 Failover and Fallback

The experimental result in Chapter 5 has proved Navigator able to handle multiple hardware failure with about 10sec downtime. This is so because once the Pilot Agent can’t ping the service container, failover process will be initiated immediately. This approach is efficient for hardware failure, however this is not so with network failure. Image if the network is not available, Pilot Agent will not able to ping service container even though the service container is up and running. If the Pilot Agent initiate another instance of service container at this moment, there will be duplicate either primary of secondary service container inside the network.

6.1.5 Awareness

Due to the lack of communication between Pilot Agent and Service Depot, Service Depot have no idea there is a service container reside under it local resource. Service Depot should be aware if a service container is running on its own local resource. If let says the local resources is overloaded, at least the Service Depot is aware and proceed with negotiation with Pilot Agent to migrate the resource to any other vacant resources.

Besides, Pilot Agent resolves the IP address of secondary service container to become the primary IP address during failover. This approach is not robust as changing IP on-the-fly is only meant for temporary purpose. The IP might be inaccessible after a long duration. The primary Pilot Agent should be aware of this and carry appropriate step to handle such error.

6.2 Future Work

6.2.1 Data Management

The size of database is growing over time. Thus the idea of encapsulate the RMS together with database is not feasible as the size of database directly affect the dynamic service placement performance. Thus in future, the historical data should be managed in a distributed fashion. One of the option here is Hadoop(Apache Software Foundation. 2005). Hadoop is a framework that allows for the distributed processing of large data set across clusters of Computer. Rather rely on hardware to deliver high-availability, Hadoop is designed to detect and handle failures at the application layer. These features are similar to Navigator framework, thus Navigator can be integrated with Hadoop, allows Hadoop to do the data management and Navigator remains on safeguarding the availability of the RMS.

6.2.2 Service Encapsulation

It is efficient to encapsulate the RMS into a virtual machine since it able to eliminate almost all the dependency of the service to hardware. However this technique is prone to few problem which mentioned in section 6.1.2. In the future, we should try on other approach of service encapsulation like Operating-system level virtualization. OS level virtualization produce lighter service container but the challenge here will be how to solve the dependency of OS level virtualization to the kernel dependency.

We can install multiple identical RMS under one landscape. We can configure each of RMS to monitor different host or different metrics under the landscape. So at the end of the day, the component that differentiates these RMS is the set of configuration. An approach we used here is, instead of encapsulate the service into a container together with the configuration, we can encapsulate the configuration and migrate it when it is necessary. The potential problem of this mechanism is how exactly the RMS load the configuration. If the configurations need to be pre-loaded, then we will be facing difficulty to migrate the configuration only.

6.2.3 Scalability of High-Availability

Although Navigator framework able to withstand unlimited failure as long as there is vacant computing resource to host the primary and service container, however there is critical time where the service cannot be failed. Although the critical time is short, but it still posting threat to the RMS. Especially if the RMS is deployed under a very large scale computing system which have high failure rate.

In large scale computing system, multiple redundancy is justifiable and thus, Navigator framework can be improved in a way that allow user to create more redundancy for failover. If there is at least two redundancy is standby for failover, basically the critical time will be greatly reduce.

6.2.4 Failover

As mentioned in section 6.1.4, current failover process is not robust. Instead of hiding the failover process to the other node in the network, Navigator can choose to propagate the failover notification to all the nodes in the network. In this case, if all the nodes is aware about the failover, there will be no failure of failover since everybody is aware of it and everybody will refer the new service container as the primary service container. For instance, we broadcast a message to every single node in the network about the event. If broadcasting is intrusive, we can use protocol such as Peer-to-Peer protocol to achieve the message propagation to all the nodes.

6.2.5 Navigator Agents

To achieve seamless automation, all the Navigator agents is ought running 24/7 to manage the Navigator framework. However, for the current design, all agents only able to handle failure of RMS but not error which also often occur in computing system and also to the agents itself. For instance, if the agents encounter error, it might terminate itself and will not have self-healing process will be executed. Thus, Navigator agents need to have more sense of “cognitive” in order to handle various kind of failure and error.

6.2.6 RMS

We used Zabbix as our RMS so that we do not need to reinvent the RMS. Unfortunately, very often, due to Zabbix is not dedicated to use under Navigator framework, we face a numbers of obstacles to realize the Navigator architecture. For example, to scale Zabbix, we have to host Zabbix in a better hardware specification environment. This issue will cause the dynamic service placement process become more challenging as the mechanism need to find an optimal host for Zabbix to reside.

The fact here is Zabbix operates under a client-server architecture. Zabbix’s performance can be deteriorate if too much movement involved. Thus, in future we have to design a distributed RMS which do not rely on client-service architecture and integrate it into Navigator framework.

6.2.7 Optimal Service placement

Navigator uses the up time of the computing resource as the indicator on selecting candidate to host the service container. But up time alone is not enough to identify the best candidate to host the service container. In future, different algorithm which taking account of different indicator or even user behavior can be used to analyze and determine the optimal candidate to host the RMS.

BIBLIOGRAPHY

Apache Software Foundation. 2005, Hadoop, viewed 5 April, 2014, <http://www.freebsddiary.org/jail-6.php>.

Clark, C, Fraser, K, Hand, S, Hansen, JG, Jul, E, Limpach, C, Pratt, I and Warfield, A 2005, Live migration of virtual machines, *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pp. 273–286.

Docker Inc. n.d., Homepage - Docker: the Linux container engine, viewed 5 April, 2014, <https://www.docker.io/>.

DVL Software Ltd. n.d., The FreeBSD Diary -- Jails under FreeBSD 6, viewed 5 April, 2014, <http://www.freebsddiary.org/jail-6.php>.

Icinga 2009, Home - Icinga: Open Source Monitoring, viewed 29 May, 2013, <https://www.icinga.org/>.

Kephart, JO and Chess, DM 2003, The vision of autonomic computing, *Computer*, 36, (1), pp. 41–50.

Li, Y, Li, W and Jiang, C 2010, A Survey of Virtual Machine System: Current Technology and Future Trends, IEEE, pp. 332–336.

linuxcontainers.org 2014, Linux Container, *linuxcontainers.org*, viewed 5 April, 2014, <http://www.solarwinds.com/>.

LogicMonitor 2008, Network Monitoring, SNMP monitoring software, Server Monitoring | LogicMonitor, viewed 30 June, 2013, <http://www.logicmonitor.com/>.

Massie, ML, Chun, BN and Culler, DE 2004, The ganglia distributed monitoring system: design, implementation, and experience, *Parallel Computing*, 30, (7), pp. 817–840.

Monitis.com 2006, Network & IT Systems Monitoring | Monitis - Monitor Everything, viewed 21 February, 2014, <http://www.monitis.com/>.

Nagios 2002, Nagios - The Industry Standard in IT Infrastructure Monitoring, viewed 29 May, 2013, <http://www.nagios.org/>.

Opsview Ltd. 2014, Opsview | Enterprise IT Monitoring for Networks, Applications, Virtual Servers & the Cloud, viewed 5 April, 2014, <http://www.opsview.com/>.

OpvenVZ.org 2005, OpenVZ Linux Containers Wiki, viewed 5 April, 2014, <http://openvz.org/Main_Page>.

Oracle Corporation 2008, MySQL :: MySQL 5.1 Reference Manual :: 4.5.4 mysqldump — A Database Backup Program, viewed 5 April, 2014, <https://dev.mysql.com/doc/refman/5.1/en/mysqldump.html>.

PassMark 1998, PassMark Software - PC Benchmark and Test Software, viewed 5 April, 2014, <http://www.passmark.com/index.html>.

Rackspace, US Inc 2014, Technical Details of Cloud Monitoring, viewed 21 February, 2014, <http://www.rackspace.com/cloud/monitoring/techdetails/>.

Van Renesse, R, Birman, KP and Vogels, W 2003, Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining, *ACM Transactions on Computer Systems (TOCS)*, 21, (2), pp. 164–206.

SolarWinds 2003, Solarwinds: The Power to Manage IT, viewed 5 April, 2014, <http://www.solarwinds.com/>.

T. Ylonen 2006, RFC 4251 - The Secure Shell (SSH) Protocol Architecture, viewed 29 March, 2014, <http://tools.ietf.org/html/rfc4251>.

The OpenNMS Group Inc. 2002, Get OpenNMS « The OpenNMS Project, viewed 21 February, 2014, <http://www.opennms.org/get-opennms/>.

Tucker, A and Comay, D 2004, Solaris Zones: Operating System Support for Server Consolidation., *Virtual Machine Research and Technology Symposium*.

Zabbix 2001, Homepage of Zabbix :: An Enterprise-Class Open Source Distributed Monitoring Solution, viewed 29 June, 2013, <http://www.zabbix.com/>.

Zabbix.org 2014, Docs/howto/high availability - Zabbix.org, viewed 9 April, 2014, <https://www.zabbix.org/wiki/Docs/howto/high_availability#Successful_failover>.

Zenoss, Inc. 2005, Transforming IT Operations | Zenoss, viewed 21 February, 2014, <http://www.zenoss.com/>.

APPENDIX A

◀ Feb 2014	~ March 2014 ~					Apr 2014 ▶
Sun	**Mon**	**Tue**	**Wed**	**Thu**	**Fri**	**Sat**
						1
2	**3**	**4**	**5**	**6**	**7**	**8**
9	**10**	**11**	**12**	**13**	**14** Service is up and experiment is started	**15** Service is up
16 Service is up	**17** Service is up until the end of day	**18** Service is up until the end of day	**19** x	**20** Service is up until about 1pm and the resource monitoring is unavailable anymore	**21** x	**22** x
23 x	**24** x	**25** x	**26** Service is up until the end of day	**27** Service is up until the end of day	**28** x	**29** x
30 x	**31** Service is up until the end of day	**Notes:** X indicates no experiment is being carried out.				

◄ Mar 2014	~ April 2014 ~					May 2014 ►
Sun	**Mon**	**Tue**	**Wed**	**Thu**	**Fri**	**Sat**
		1 Failover once (failover after primary service container is unavailable and service is up until the end of day)	**2** Service is up until the end of day	**3** Failover once (failover after primary service container is unavailable and service is up until the end of day)	**4** Service is up until the end of day and experiment is stop at the same day	**5**
6	**7**	**8**	**9**	**10**	**11**	**12**
13	**14**	**15**	**16**	**17**	**18**	**19**
20	**21**	**22**	**23**	**24**	**25**	**26**
27	**28**	**29**	**30**	**Notes:**		