A Case-Based Framework for Self-Healing Paralyzed Components in Distributed Software Applications

Muksitul M. Tanim Hasan

BSSE 0301

A Thesis

Submitted to the Bachelor of Science in Software Engineering (BSSE) Program Office

Of the Institute of Information Technology, University of Dhaka

In Partial Fulfillment of the

Requirements for the Degree

BACHELOR OF SCIENCE IN SOFTWARE ENGINEERING

Institute of Information Technology

University of Dhaka

DHAKA, BANGLADESH

To Dr. Kazi Muheymin-Us-Sakib, my research supervisor

Abstract

Open source cloud computing provides free, on-demand, rapidly scaling, ubiquitous computing and data storage services that promises a significant number of prospective customer base. However, the largely distributed nature and growing demand for self-healing makes the infrastructure available for 24 x 7 Self-healing is the ability of the software to detect faulty modules at execution time and replace or recover those without affecting other components. This paper proposes a framework for self-healing of Distributed Software System (DSS). Monitoring component is used to detect and record failures of DSS. Healing system will replace the paralyzed components with healthy ones which will be initiated from the information given by Monitoring system to the proposed Reviver process. A failed case table is required to match the real life failures with it for identification of the solution. Distance between the failed case table and the recorded failures need to be calculated using exclusive OR since the solution closest to the fail can then be determined. Afterwards, the minimum distance between those is used to resolve the failure. This recovery is achieved through replacement of the faulty modules with redundant components in the DSS. Performance evaluation shows a desirable time consumption of less than the standard 0.7 seconds for component replacement in all the experimental iterations.

Preface

This thesis is submitted to the Bachelor of Science in Software Engineering (BSSE) Office of the Institute of Information Technology, University of Dhaka, in partial fulfillment to the requirements for the Degree. It contains the work done from January 2013. My supervisor on the research has been Dr. Kazi Muheymin-Us-Sakib. The thesis has been made solely by the author; some of the viewpoints are based on the research of others, and references to those sources are provided.

The thesis concludes with a discussion of the research and the additional research questions that have come up in the course of this research. The author conveys gratitude to the supervisor of this research for the continuous support and reviews. At the same time the author is grateful to all the members of the Institute of Information Technology, University of Dhaka and Panacea Systems Ltd, the software industry that provided the hardware for conducting experiments during the research.

Acknowledgements

This thesis is about implementing a software self-healing for paralyzed distributed components in distributed web applications. It has been mainly possible for the constant effort of my thesis supervisor Dr. Kazi Muheymin-Us-Sakib, whose continuous support has enabled me to attain international publications in this topic. It all began in 2014 when my research supervisor Dr. Sakib advised me to pursue research in Software self-healing for distributed web application. It was at Panacea System where I got my first flavors of Distributed computing and there was no looking back from then on. I am extremely grateful to Dr. Sakib for lending me this opportunity. From that moment I started to learn about the cloud using Google to good effect. I connected to many people who deal with this technology, electronically and face to face. Since it is impossible for me to thank them all, I will take this opportunity to mention the names of those without whom this study would have never been completed. Dr. Sakib was my supervisor in this research, and from that moment he has been guiding me in the research field, identifying my mistakes by going through every line of my write ups and my experimental results, investing significant portions of his valuable time to my development. At the same time he was always concerned about keeping my motivation high and maintaining a high spirit within the research team. I am also very grateful to the Director of the Institute of Information Technology, University of Dhaka, for allowing me pursue my research in a healthy and competitive atmosphere of IIT. I will also take this opportunity to thank Dr. Mohammad Shoyaib and his panel for the effective feedback during the second presentation of my thesis.

As I said earlier, I first learnt about the cloud at Panacea. Hence I convey my heartfelt gratitude to Mr. Mafinar Rashid Khan, Product Manager of Panacea, for keeping faith in me and enabling me to learn the concept of distributed computing also some development tools related for implementation as an intern at his company. Also, my heartiest thanks to Mr. Redwanul Karim Ansari, Chief Executive Officer (CEO) of Panacea Systems Ltd, for appointing me as the Software Developer at his company. Hence I was fortunate enough to be provided with the powerful cloud servers of Panacea where I executed the tests and obtained the results of the thesis experiment finally I would like to thank my family and friends who have directly and indirectly helped me in the research. Special thanks go to my mother and father, who have supported me. My elder brother M was always a support and source of confidence. I would like to thank all the students of BSSE01 batch who are seeking completion of the Bachelor of Science in Software Engineering (BSSE) program through their participation in research and projects. Thanks go to the staffs of IIT, who have been a continuous source of help. The members of IIT family have really made this research fun for me and made the moments very enjoyable

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Self-healing is a mechanism that autonomously detects failures and responds to those at run-time without human intervention. Component based self-healing in DSS make the necessary adjustments to restore the application towards normal operation by reviving the depended processes, components or whole system. Software system has become too complex to manage and fix manually. Applications are now designed with so many depended processes and distributed components that it is hard to immediately react to failures manually. Naturally in distributed environments, depended processes may run in different virtual machines of an application. When a process fails, other processes have no idea why application or system gets down. In this case, a system administrator is required to detect failures manually and take initiatives as per demand, leading to increased recovery time and expenses. Early detection of paralyzed components at run-time is highly important to sustain the errors and prevent those from propagating to other components of the application [1], [2]. Ensuring fault tolerance in modern distributed software requires the components of those to self-heal that is to detect causes of component paralysis at the execution time.

Redundant components must be used to replace the paralyzed ones together with state restoration in short time and without affecting the healthy ones. The complex nature of distributed applications present significant challenges to ensure self-healing of those. Research needs to be conducted to detect key causes of software paralysis at run-time and replace the failed components of the application with new ones. Legacy applications were injected with self-healing primitives to ensure recovery of software modules after failure [3]. Run-time errors were identified as attributes that were recovered through the proposed primitives. The methodology is applicable to codes specific to Java applications. Byte- codes were analyzed to detect run-time issues, and modules suffering from run-time errors were recovered through reinitiating. However, the mechanism proposed by the authors involved termination and restart of the modules from initial state, hence the importance of ensuring state restoration of critical processes were not considered. Security perspective of cloud based applications were taken under consideration and malicious attacks on cloud components were detected in [4]. However, the authors did not consider the importance of redundant components for fast recovery of paralyzed modules in cloud based software.

The proposed mechanism is a three layered framework to detect faults in software components and recover from those in short time. The first layer is called the Monitor that provides a failure case table containing predefined software fault conditions for components together with the solutions identified for those. The Monitor module also analyses real life software components to determine and list failures. A paralyzed component is the one that does not perform its target functionality due to certain failures. Next, Healing module of the framework determines the distance between the detected faults and the ones listed in the failure case table using exclusive OR. The minimum distance found in this way will yield the closest solution to the current problem [5]. The entry in the failure case table that has the minimum distance is considered as the solution to the paralysis problem, hence the solution prevalent for that case is applied to the paralyzed component. This failure case table is used to compare with real life cases and determine solution

for those. Afterwards, based on the solution the Reviver module replaces the faulty components with redundant ones preserving the last active state. Redundant copies of healthy components are stored in additional cloud virtual machine (vm) instances that are used to replace the paralyzed ones. Empirical analysis of performance yields desirable results based on a case study resembling real life scenario of an e-978-1-4799-6399-7/14/$31.00 c 2014 IEEE commerce website. Each component of the website resides in individual vm-instances of the cloud, hereby ensuring decentralization of services. The proposed framework is appended into the e-commerce software as additional components. Next, faulty iterations of the application were executed and faults collected for those. Upon calculation of exclusive OR between the failed test case and real life cases, minimum distance of detected faults that are determined within the range of 0-2 units which is desirable. Time consumption for replacing paralyzed component with a health component in cloud vm-instance was found to be less than 0.7 seconds in four iterations, resulting in desirable time complexity specified in [5].

1.1 Motivation for the Research

In distributed systems system ensuring reliability and availability is a key issue. In order to accomplishing this task, in traditional model we use system administrator who monitor this task and later took decision as per situation demands for system availability. However manual handling is error prone and human error can occur in and between. Therefore Self-healing mechanism can be incorporated in distributed component based web application for making the system available 24 X 7 fashion. However such mechanism is cost effective. Self-healing is a mechanism that independently locates disappointments and reacts to those at run-time without human mediation.

Component based repairing in DSS make the vital changes in accordance with restore the application towards ordinary operation by restoring the depended techniques, components or entire framework. Programming framework has ended up excessively perplexing to oversee and alter physically. Applications are presently planned with such a large number of depended methods and conveyed segments that it is difficult to instantly respond to disappointments physically. Characteristically in disseminated situations, depended courses of action may run in distinctive virtual machines of an application. At the point when a methodology falls flat, different courses of action have no clue why application or framework gets down. For this situation, a framework overseer is obliged to discover disappointments physically and take activities according to request, prompting expanded recuperation time and costs. Early discovery of deadened parts at run-time is exceedingly paramount to support the blunders and keep those from spreading to different segments of the application [1], [2]. Guaranteeing deficiency resistance in advanced conveyed programming requires the parts of those to self-recuperate that is to catch reasons for part loss of motion at the execution time.

From background study and real life experience on distributed system we found such motivation to conduct research on this field. Such motivation leads us to identify the primacy research problem.

1.2 Addressing the Research Questions

Based on the motivation stated in the previous section, the importance for ensuring system availability and reliability we need to incorporate self-healing mechanism in component based distributed web applications.

- How to implement self-healing mechanism for components in distributed system for web application?

  A methodology has been proposed for implementation of self-healing mechanism for components in distributed system applications
- How to detect software faults in distributed system for multiple components?

  Fault localization reason have been identified for this purpose
- How to incorporate self-healing architecture for paralyzed software components in distributed systems?

  An architecture for component monitor and revving mechanism is used for incorporating the self-healing mechanism for paralyzed distributed components

1.3 Research Contributions

Earlier self-healing mechanism only available in single application. However, proposed research is to self-heal the paralyzed distributed components for distributed applications. In this proposed research an algorithm is also proposed for self-recovery of the paralyzed distributed components. Along with an architecture for implementing the systems is also proposed. With the explanation of result analysis architecture and implementation of self-healing mechanism in component based distributed application can be described as well.

1.4 Thesis Organization and structure

Chapter 2: In chapter background study that need to conduct this research has been discussed
Chapter 3: In chapter literature review and related that need to conduct this research has been discussed
Chapter 4: In chapter proposed methodology for Case-Based Framework for Self-Healing Paralyzed Components in Distributed Software Applications has been discussed
Chapter 5: Case study, Experimental Setup and result analysis is discussed in this chapter

Chapter 2 Background study

2.1 RPC

In computer science, a remote procedure call (RPC) is an inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction. In the proposed research RPC is used to send the command to the virtual machine. However there are some more ways to communicate with instances though for our experimental result we found RPC more convenient

2.2 Twisted Protocol

Twisted Networking protocol is used to implement network protocol parsing and handling for TCP servers (the same code can be reused for SSL and Unix socket servers).

This protocol handling class will usually subclass ***twisted.internet.protocol.Protocol***. Most protocol handlers inherit either from this class or from one of its convenience children. An instance of the protocol class is instantiated per-connection, on demand, and will go away when the connection is finished. This means that persistent configuration is not saved in the Protocol. The persistent configuration is kept in a Factory class, which usually inherits from ***twisted.internet.protocol.Factory.*** The ***buildProtocol*** method of the Factory is used to create a Protocol for each new connection.

It is usually useful to be able to offer the same service on multiple ports or network addresses. This is why the Factory does not listen to connections, and in fact does not know anything about the network. See the endpoints documentation for more information, or ***twisted.internet.interfaces.IReactorTCP.listenTCP***, and the other ***IReactor*.listen*** APIs for the lower level APIs that endpoints are based on.

Prototype

```
from twisted.internet.protocol import Protocol

class Echo(Protocol):

  def dataReceived(self, data):

    self.transport.write(data)
```

This is one of the simplest protocols. It simply writes back whatever is written to it, and does not respond to all events. Here is an example of a Protocol responding to another event:

```
from twisted.internet.protocol import Protocol
```

```
class QOTD(Protocol):


  def connectionMade(self):
    self.transport.write("Connection is fine\r\n")
    self.transport.loseConnection()
```

2.3 PostgreSQL

For our experimental result and case study we use PostgreSQL. Which is used to store data spatially. It can manage workloads running from minimal single-machine applications to broad Internet-going up against applications with various concurrent customers. Late structures similarly give replication of the database itself for openness and versatility.

PostgreSQL, frequently just "Postgres", is an item social database administration framework (ORDBMS) with a stress on extensibility and principles agreeability. As an issue server, its essential capacity is to store information, safely and supporting best practices, and recover it later, as asked for by other programming applications, be it those on the same machine or those running on an alternate machine over a system (counting the Internet). It can deal with workloads running from little single-machine applications to extensive Internet-confronting applications with numerous simultaneous clients. Late forms likewise give replication of the database itself for accessibility and adaptability.

PostgreSQL executes most of the Sql:2011 standards ACID-agreeable and value-based (counting most DDL proclamations) abstaining from locking issues utilizing multi version concurrency control (MVCC), gives invulnerability to grimy peruses and full serializability; handles complex SQL questions utilizing numerous indexing systems that are not accessible in different databases; has updateable perspectives and appeared perspectives, triggers, outside keys; backings works and put away methods, and other expandability and has countless composed by outsiders. Notwithstanding the likelihood of working with the real restrictive and open source databases, PostgreSQL helps movement from them, by its far reaching standard SQL help and accessible relocation apparatuses. Also if exclusive expansions had been utilized, by its extensibility that can copy a lot of people through some implicit and outsider open source similarity augmentations, for example, for Oracle.

2.4 Nginx

nginx [engine x] is an HTTP and reverse proxy server, as well as a mail proxy server, written by Igor Sysoev. For a long time, it has been running on many heavily loaded Russian sites including Yandex, Mail.Ru, VK, and Rambler. According to Netcraft, nginx served or proxied 20.41% busiest sites in November 2014. For our case study we hosted the application in nginx server.Also in order to deploy the e-commerce application we have gone through the documentation of nginx serverS.

Chapter 3 Literature Review

In this chapter, the existing research issues in self-healing for distributed system, self-adaption technique and how to ensuring distributed system availability have been highlighted. At the same time the scope of research in this growing field are highlighted. With regard to the rapid growth of distributed systems and self-healing software system will ensure the systems availability in 24X7 fashion without human interventions. Hence the scope of this work is manifold. Following sections identifies how self-healing mechanism can be effective for ensuring system obtainability.

- ***A model driven approach for self-healing computing system," in Computational Intelligence and Security (CIS):*** Application based failure detection model has been proposed in [6]. Sensors are implemented into the existing codes that trigger alarms when a failure of code execution occurs. Information about the causes of failure is detected and a solution is suggested to the users from a solution set that is predefined. The suggestions are used by the clients to recover from the failed state. The components are stacked into one global module that helps the analysis of failures in those. However, the increased affinity to failure caused by a single global component has not been considered here. Additionally the importance of using distributed component architecture to ensure uptime of other services when one fails has been considered to a minimum extent.

- ***Self-healing on the cloud: State-of-the-art and future challenges," in Quality of Information and Communications Technology (QUATIC):*** Rule based methods to detect software faults based on programmer induced failures have been presented in [7]. Pre-specified failure conditions have been considered by the authors and fails were detected based on those. Failure attributes like buffer overflows and coding errors were searched for in the code and possible code blocks were identified and suggested for correction. However, the problem due to sudden process termination caused by component failure was not considered.

- ***Biologically-inspired preventive mechanism for self-healing of distributed software components:*** Self-healing mechanism of critical sections of components has been proposed in [8]. A set of components, connectors and healing parts are installed at each node. At the same time, the components place a copy of the failure condition rules in the description repository of the proposed framework. Policy based analysis have been used to generate dependency graphs. Based on those, the self-healing policy is chosen that involves removing the failed processes and recovering those from initial state. Since the

policy enforces the failed processes to start from scratch, the importance of restoring components from the last working state have not been taken under consideration.
The authors proposed Flogger that is a novel file-centric logger suitable for accessing logs in both private and public cloud environments [8]. The authors stated that through logging of file accesses and transfers, trust in the cloud can be ensured. The authors proposed a model that records file centric accesses and transfer information from the kernel levels of both private and public clouds. Through ensuring kernel level recording of file access logs, total transparency and accountability can be ensured in the cloud [8]. Hence Flogger would work to increase the trust of the people on the cloud. The process of obtaining critical information from the kernel level logs to make inference on malicious activities have not been stated.

- ***Proactive and reactive runtime service discovery: a framework and its evaluation:*** Market based heuristics have been used to ensure adaptation of cloud virtual machine (vm) instances to changing scenarios [9]. Self-adaptation has been regarded as a critical Quality of Service (QoS) requirement for the cloud. Continuous double action algorithm has been proposed to allow services to choose from the components available. In addition, how cloud computing minimizes cost through self- healing has been identified in [10]. The choice is based on the resource allocation on various vm-instances, where the vm with resources nearest to satisfying the process requirement is selected and the component is configured in it. However, the self-adaptation mechanism is active during system initiation phase only and does not take into consideration the self- recovery required during system runtime.

- ***Formalizing a methodology for design-and runtime self-healing***: Unanticipated changes to the source code that violate assumptions and internal structure of the programs at compile time have been addressed in [11] The system is divided into two parts, firstly the functional part that implements the expected requirements and secondly the self-healing part that defines the policies of automatic failure detection and recovery. The two modules are defined separately and the failure module is used to encapsulate the functional and self- healing components. The fault model defines a pre-determined set of faults. However, the faults defined in the fault model belong to compile time issues of software only, runtime faults have been considered to a minimal extent by the authors.

- ***A reconfiguration framework for self-healing software:*** Mechanism for translating anomalies in software based on pre-defined failure states haven proposed in [12]. The model specifies the characteristics and goals of the system through analysis of the inputs and outputs that are generated after processing those. Next, state rules are created based on the internal and external constraints of the system. On the basis of the defined state rules, a dependency graph is created that is used to detect what components are affected

by a failed process. However, monitoring the system to identify faults at run time was not taken under consideration.

- ***Self-Healing on the Cloud:State-of-the-art and Future Challenges :*** The research on the issue that render impracticable or ineffective the usage of the development-time. Spectrum Based Fault-Localization (SFL) is used to solve these issues. SFL is a lightweight statistic based fault localization technique that takes as its input an execution trace abstraction, call program spectrum and produces a list of possible fault candidates sorted according the possibility of the being the real failure justification.[7]
  The proposed methodology does the work in three steps (1) detect errors/failure, (2) locate the faulty components and (3) target such components with an appropriate remedy. Although the research on fault localization in run-time so SFL creates to localize faults during development-time and with the help of light-weight architecture it detects the faults
  This paper describes about how they detect failures and errors along with the procedure of fault localization. Proposed solution also perform proactive maintenance in order to enhance survivability. However, about how the whole system will fit into generalized cloud system and its

- ***SHelp: Automatic Self-healing for Multiple Application Instances in a Virtual Machine Environment:*** ASSURE is their proposed solution, using rescue point and error virtualization techniques it provides the self-healing system. Proposed Methodology does the work in 4 steps (1) Fault localization (2) Error virtualization (3) identify rescue point (4) rescue the system. This paper proposed the solution by identifying the rescue points then make the healing action. Shelp extends the solution to a virtualized computing environment. It adopts a two-level storage hierarchy to manage rescue points and the associated weight values in a centralized manner to accelerate the fault discovery process. However author considered this solution to a limited extent in terms of Component based solution. Our methodology proposes component monitor which used to deal the actions regarding fault detection and status checker.[2]

- ***Rule Based Self-healing:*** Rule based methods used to detect software faults based on programmer prompted failures have been presented in this paper. Pre-specified failure conditions have been considered by the authors and failures/errors were detected based on those. Failure characteristics like buffer overflows and coding errors were investigated in the code and possible code blocks were identified and suggested for correction.
  However, the problem caused due to sudden termination of processes caused by component failure was not considered. Investigating all the codes of an application will not be possible for every commercial application also. Although author did not consider the self-healing on component based distributed system.

- ***A Modeling Framework for Self-Healing Software Systems:*** This paper proposes a modeling framework to facilitate the development of self-healing software systems. Their proposed solution and Code generation platform supports automated generation of self-healing enabled software systems.[23]
  This paper presents a generic framework for modeling self-healing software systems. Modeling paradigms are used to capture software faults and their detection and purposes to transform the system from faulty states to functioning states specified in the model. The classification of software faults and the modeling of fault detection and resolution facilitate the construction of self-healing software systems. Self-healing is achieved by transforming the self-healing models into platform-specific implementation, which is then composed with the base module to form an integrated software system that provides failure resolutions to mitigate the effect of software faults.
  Although this paper proposes a modeling framework for self-healing system. Code generation platform is used towards making an automatically generated self-healing enabled software system. However, how the code generation will support every application was not considered by the author. Although the proposed solution did not consider the healing system for components in distributed system.

- ***A Self-Healing Approach for Object-Oriented Applications:*** This paper present an approach and architecture for fault analysis and self-healing of interpreted object oriented applications. By combining aspect-oriented programming, program analysis, artificial intelligence, and machine learning techniques, it can heal a significant number of failures of real interpreted object-oriented applications.[4]

  This paper does the work in 3 steps (1) Identify the errors (2) Select techniques (3) Apply appropriate technique for a particular situation. However, the problem caused due to sudden termination of processes caused by component failure was not considered. Also how the whole system will fit into distributed component based web application is not discussed

Summary

This table is highlighted the existed solution and their methodology. Also highlighted the limitation and comparison of the approach along with the proposed research

Number	Name of the publication	Proposed solution	Methodology	Limitations
1	***Self-Healing on the Cloud: State-of-the-art and Future Challenges***	The research on the issue that render impracticable or ineffective the usage of the development-time. Spectrum Based Fault-Localization (SFL) is used to solve these issues. SFL is a lightweight statistic based fault localization technique that takes as its input an execution trace abstraction, call program spectrum and produces a list of possible fault candidates sorted according the possibility of the being the real failure justification.	The proposed methodology does the work in three steps (1) detect errors/failure, (2) locate the faulty components and (3) target such components with an appropriate remedy. Although the research on fault localization in run-time so SFL creates to localize faults during development-time and with the help of light-weight architecture it detects the faults and the make the healing action. Healing action in the cloud paradigm where applications tend to self-adapt to environment changes by increasing or decreasing the number of components in order to be able to deliver the expected performance while reducing expenses	This paper describes about how they detect failures and errors along with the procedure of fault localization. Proposed solution also perform proactive maintenance in order to enhance survivability. However, about how the whole system will fit into generalized cloud system and its methodology has considered to a limited extent. Although our proposed solution has also considered the component based self-healing in distributed system.

2	***SHelp: Automatic Self-healing for Multiple Application Instances in a Virtual Machine Environment***	ASSURE is their proposed solution, using rescue point and error virtualization techniques it provides the self-healing system	Proposed Methodology does the work in 4 steps (1) Fault localization (2) Error virtualization (3)identify rescue point (4) rescue the system	This paper proposed the solution by identifying the rescue points then make the healing action. Shelp extends the solution to a virtualized computing environment. It adopts a two-level storage hierarchy to manage rescue points and the associated weight values in a centralized manner to accelerate the fault discovery process. However author considered this solution to a limited extent in terms of Component based solution. Our methodology proposes component monitor which used to deal the actions regarding fault detection and status checker.

3	***Rule Based Self-healing***	Rule based methods used to detect software faults based on programmer prompted failures have been presented in this paper.	Pre-specified failure conditions have been considered by the authors and failures/errors were detected based on those. Failure characteristics like buffer overflows and coding errors were investigated in the code and possible code blocks were identified and suggested for correction.	However, the problem caused due to sudden termination of processes caused by component failure was not considered. Investigating all the codes of an application will not be possible for every commercial application also. Although author did not consider the self-healing on component based distributed system.

4	***A Modeling Framework for Self-Healing Software Systems***	This paper proposes a modeling framework to facilitate the development of self-healing software systems. Their proposed solution and Code generation platform supports automated generation of self-healing enabled software systems.	This paper presents a generic framework for modeling self-healing software systems. Modeling paradigms are used to capture software faults and their detection and purposes to transform the system from faulty states to functioning states specified in the model. The classification of software faults and the modeling of fault detection and resolution facilitate the construction of self-healing software systems. Self-healing is achieved by transforming the self-healing models into platform-specific implementation, which is then composed with the base module to form an integrated software system that provides failure resolutions to mitigate the effect of software faults.	This paper proposes a modeling framework for self-healing system. Code generation platform is used towards making an automatically generated self-healing enabled software system. However, how the code generation will support every application was not considered by the author. Although the proposed solution did not consider the healing system for components in distributed system.

5	***A Self-Healing Approach for Object-Oriented Applications***	This paper present an approach and architecture for fault analysis and self-healing of interpreted object oriented applications. By combining aspect-oriented programming, program analysis, artificial intelligence, and machine learning techniques, it can heal a significant number of failures of real interpreted object-oriented applications.	This paper does the work in 3 steps (1) Identify the errors (2) Select techniques (3) Apply appropriate technique for a particular situation	However, the problem caused due to sudden termination of processes caused by component failure was not considered. Also how the whole system will fit into distributed component based web application is not discussed

In this chapter the related research in the field of Self-healing mechanism and implementing those with effective analysis models have been highlighted. The motivation of the research in Self-healing for paralyzed component in distributed application and effective representation of Self recovery of the components with appropriate architecture for ensuring reliability and availability of the distributed component based web application have been specified. The significant research contributions have been identified in this chapter.

CHAPTER 4 Methodology: Component Based Self-Healing Mechanism In Distributed System

4.1 Introduction

The rising complication of software systems demands innovative ways of control over the systems. Systems should be able to adapt dynamically to changes in the environment and components. An emerging methodology to overcome this problem is software self-healing. Architecture for component based self-healing mechanism in distributed system is proposed in this section and depicted in Figure 1. If an application goes down due to the process that are running on VM, then system admin needs to be involved to revive the component, which is costly and less efficient. The challenge of the objective is to detect failures and revive them at run-time in distributed system. Proposed framework of self-healing systems are classes of software systems that exhibit the ability to detect failures at run-time and revive the whole system autonomously. Life cycle of component-based software self-healing in distributed system consists of four major activities:

- Monitoring the components and depended process
- Detecting the failures issues
- Notifying the main the application about the defective components
- Reviving the components as per policy

4.2 Overview

During the design of the architecture, attention is given to component based self-healing as well as fitting the framework in distributed environments. DSS provides the outcomes to users with the help of multiple components while the depended components in different virtual machines are not aware of other components operating properly or not.

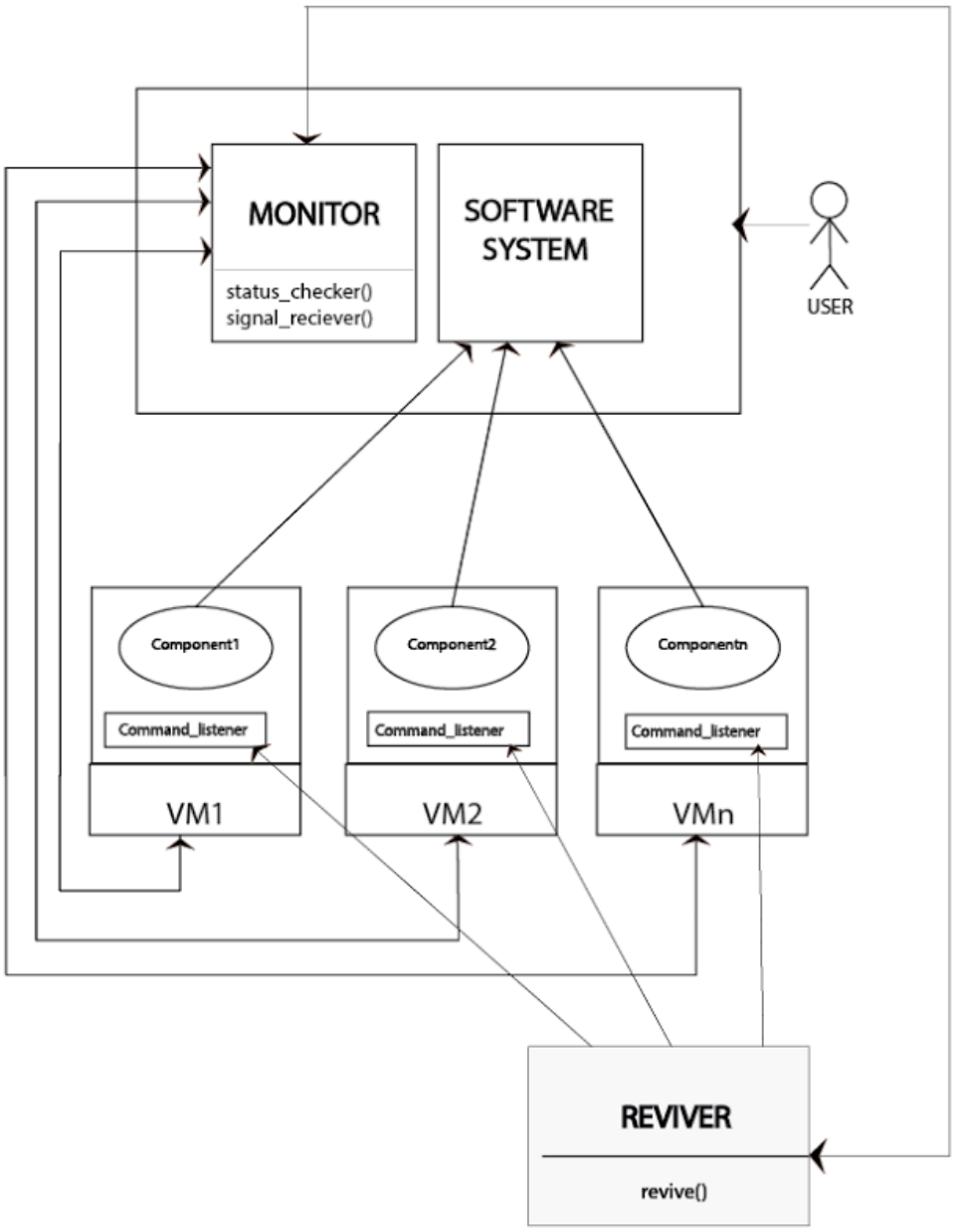

Figure 1 : Communication of the modules in proposed methodology

The architecture performs the healing action with a Component Monitor program and transmits the information to the ***Process Reviver***. However, the virtual machines have command listener to heal the components as per policy and to send the status to monitor program. The top-level overview of the proposed architecture is shown in Fig 1. The architecture is divided into three segments while the whole component based self-healing for DSS stands on three core tasks that are:

- Component Monitor
- Reviver process
- Healing system

The first is ***Component Monitor***, which will run along with main application. ***Component Monitor*** is designed such that it will store all the depended components and processes information (e.g. process number, process id, VM information, etc.) by checking regular status. Next, it will send the faulty processes or components that responds to the ***Reviver*** to the ***Healing system***.

The next one is ***Reviver***, which will collect the process's or component's information that needs healing from component Monitor. It sends the request to the Healing system with the faulty process information and restoration data. After healing or recovering that faulty process it will send feedback message to the component monitor.

The final segment is ***Healing system***, which will respond to Revivers request and heal the processes for performing normal operation. All three segments are designed such that it should fit in complex DSS.

4.3 Component Monitoring system and error detection

Component Monitoring system will monitor the processes or components. With a failure detection methodology it will identify the failures then send it to the reviver to re-initiation. Anomalies that were considered can happen to software system are process crash in computing, run-time-error of a depended process, communicating connections problem and resource over allocation. The four major failures stated here are identified by Monitoring system. Each of the failure and its identification procedure are discussed in this section.

Process crashes when it performs an operation that is not allowed by the Operating System (OS). If a process is attempting to access (read or to write) at a specific memory block which is not allowed for that specific process, the process get crashed which is called segmentation fault. Also while attempting to execute invalid instructions, to perform inaccessible I/O operations and passing invalid arguments to system calls can lead a process to dump [13].

In some critical conditions, process attempts to execute machine instructions with bad arguments, for example divide by zero, function on DE normalization or NULL values can make

the process crash. Monitoring system will identify this process crashes by identifying the error messages and checking the time duration that has elapsed. If a process does not respond to main application within the threshold time then it will be considered as a paralyzed depended process. Segmentation fault and unauthorized faults will be detected from the error messages. Run-time error occurs during the execution of a program [14]. Such as running out of memory is a run-time error. By setting up a threshold value for each parameter like load time, memory, response time etc. monitoring system can identify the run-time error.

Use of self-healing system and process reviver software system can get rid of the run-time error. Error detection and status checker method of reviver will identify this failures, then it can initiate the paralyzed process once again by healing system. Communicating issues in distributed system is one of the major problems [15]. It can be identified by sending request from monitoring system, if virtual machine does not respond to the request within the threshold time then monitoring system will consider this as communicating connection problem. Resource over allocation can be happened in terms of using over memory, bandwidth, operating the process for long time without conveying any error message [16]. On this occasion again proposed monitoring system will identify those by checking with threshold value.

Above stated failures can happen in Distributed Software (DS), monitoring system can identify those failures and take initiatives to revive the process. Reviver will initiate the process with necessary information and if possible with previous data also. Proposed component monitoring system will consider only the four major failures although there are some other issues like functional problem, logical problem etc. will not be under consideration, because those faults are design problems and internal program errors.

4.4 Proposed self-recovery algorithm

Reviver component of the proposed framework collects all the faulty process information and stores those in a journal as shown in Algorithm 1. Reviver sends the request to Healing system to heal the process and receives to feedback from Monitor. Initially there will be an empty ***Proc_list*** and ***Proc_info*** register. Monitoring system will store all the process information (vms ip , hardware address, port, process resource allocation, etc). Since it is initialized at the initiation, next it will push both the ***Pro_cid*** and ***Pro_cinfo*** into the empty list. To get the ***Proc_info GetProcessInfo(Proc[i])*** method has been introduced. This method will take Procid as an argument and return all the information of that particular process including vm's basic information.

Algorithm 1 Algorithm for self-healing component

```
1: procedure SELF-HEAL(P_List, Proc_info)
2:     Proc_list ← []
3:     Proc_info ← []
4:     Proc_list and Proc_info
5:     while i = num_DependedProc do
6:         SignalReciever(P_id, Proc_info[i])
7:         If Proc_[i] == IsFaultyStatProc[i]
8:         Proc_info ← GetProcessInfo(Proc[i])
9:     end while
10:    while i = Proc_list do
11:        Else return Process is not paralyzed
12:        If (P_INFO[I] ==IsFaulty(Proc_status[i])), then
13:        Feedback = Reviver(Proc_info, VM_info, Policy)
14:        Else return Message Process is ok
15:    end while
16:    IsFaulty_StatProc[i]
17:    return Faulty = Error_Message_vm
18:    If Status and Proc_info == Faulty
19:    Else return False
20:    Proc_list.push(Proc[i])
21:    Proc_info = Get_Proc_info(process[i])
22:    feedback = Reviver(stringProc_info, vm_info,
23:    Policy)
24:    return true
25: end procedure
```

Although this method call will request vm's, those will respond to the request by collecting information from components including process information. ***SignalReciever()*** is another method, which takes ***Process_id*** and ***Proc_info*** as arguments. It will return the status of that process or components of the vm. If the return value matches faulty codes then a reviver will send the process with the information for healing. In this process all the journals will be stored as a feedback in the Monitoring system.

If the process is operating correctly it will generate a positive message and notify the administrator. ***IsFaulty()*** is a method that will verify the response from vm whether the process performs as desired. The fault detection will be done based on the threshold value set for the system and error messages from the OS.

4.5. Procedure of healing system

The proposed framework revive or recover faulty processes using this healing system. The mechanism is provided with the necessary information given by the Reviver. It needs the depicted information to heal the process as:

- Faulty process information
- Previous restoration data
- VM's information

Healing will send the feedback message to the Reviver if it can complete the action. It will store the most recent three healthy components to the vm which had faulty ones. In the case of replacing the components, difference between the stored-set and detected failed-set is determined using exclusive OR between the two. Redundant copies of healthy components are stored in that vm which had faulty components. The stored- set that has the minimum distance is considered as the solution to the paralysis problem, hence the solution is prevalent. The solution included preserving the last working state of the faulty component, replacing it with a healthy one, and restoring the state into the new module. Afterwards the new component is integrated into the software architecture and begins functioning in co-ordination with the other active components.

Chapter 5 Case Study

The proposed methodology can be tested through concrete experimentation of the self-healing framework depicted in Figure 2. The realization of the performance can be conducted through implementation of the proposed algorithms in a real life case. The test environment should consist of a web based electronic commerce (e-commerce) site implemented in cloud. The system consists of a user-friendly web page where customers can register, post adverts of the product, browse new products and book those for purchase. Hence a web-service component is required for the operation of the e-commerce website. In this case Nginx web server is used coupled with a PostgreSQL database that is configured in a separate vm-instance. The PostgreSQL database server stores and accepts queries for all user registrations, posting adverts and booking items through the application web service [17].

5.1. Environmental Setup

In order to get the experimental result on "Case based self-healing in paralyzed components on distributed system" we followed the below procedure

- Environmental setup should support the architecture which allows the component monitor to send and receive the instructions
- Fixed IP of each virtual machine is necessary to communicate with them as per necessity
- The whole system will be consists of one master machine and other slave one as virtual machines
- A network protocol named *twisted is configured on main machine*

- ***apt-get install python-setuptools***
  apt-get update
  pip install twisted
  [if the commands fail please use sudo for permission, you can also run this on virtual environment]

- After configuring put component monitor on the main machine and inside the project which you want to be self-healed

- List out the process and components that are depended with the system to the ***settings.py***

- For component monitor we use the following code for importing necessary library that we used to implement the component monitor system

```
import os from subprocess

import check_output as co from twisted.internet

import reactor from twisted.internet

import task from settings
```

- In component monitor we use our algorithm to identify the faults and running process from settings
- Sample code for implementing one module

```
def is_running(proc):
   return len(filter(None, [proc.lower() in i.lower() \
                for i in co(["ps", "aux"]).split("\n")])) > 0
def revive(path):
   return os.system(path)
def f():
   for i in processes_to_watch:
      if not is_running(i["name"]):
         print "{} is not running".format(i["name"])
         notify_auth(i["name"])
         output = revive(i["cmd"])
         if output == 0:
            print "{} has been revived".format(i["name"])
      else:
         print "{} is running fine!!!".format(i["name"])


t = task.LoopingCall(f)
t.start0.01)
reactor.run()
```

5.2. Integration of Monitor Module

Upon receiving a failure condition cause due to external queries mainly from custom searches, the Monitor module detects the issue, sorts the problem and assigns it to a specific problem group that is recorded in the data storage. Based on the rules set against each problem the Monitor identifies the faulty component running in the paralyzed vm instance, records the last working state and dependencies of the component using recovery orientation. The Monitor algorithm is implemented in Python and requires Twisted to execute in distributed environments.

Recovery procedures include isolation of the paralyzed component. Next a new vm from the redundant array of vms is allocated and the state of the paralyzed vm is restored in it. The new vm becomes functional when it is integrated with the remaining vms running the components of the e-commerce web site [18]. Hence, the issue of failed components is self-healed using the redundant array of vm-instances in the cloud environment. The cloud framework that can be used for experimentation is OpenStack Icehouse running on CentOS 6.5 servers.

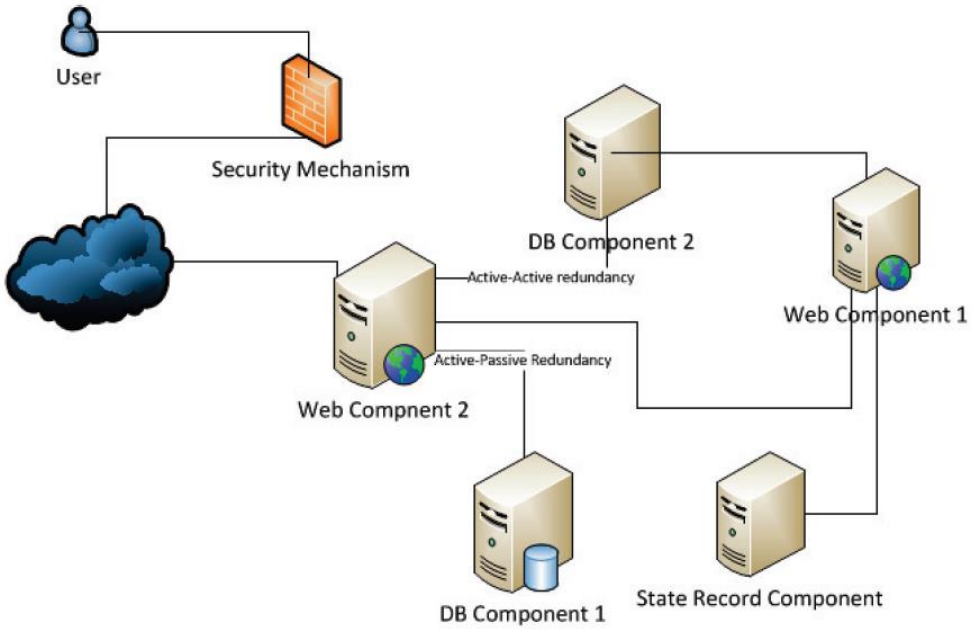

Figure 2 Self-healing components with redundancy for e-commerce application

TABLE 1 DECLARATION OF PREDIDINED FAILURE CASES IN DISTRIBUTED SOFTWARE

Failure Cases	Nginx Conf	NginxComm	PermitErr	Registration Err	PostgreSQL Conf	Solution
User_Registration	Yes	No	Yes	Yes	Yes	Restart Ngix, allocate vm and set permission
Post Advert	No	Yes	Yes	No	Yes	Allocate new vm and reload states
Browse Page	No	Yes	No	No	Yes	Trigger configuration upon restart of Nginx
Book Item	No	No	Yes	No	Yes	Allocate new vm and restart PostgreSQL
Visit Home	Yes	No	No	No	Yes	Allocate new vm and load Nginx state

Table 2 Minimum Distances of the test iteration from Predefined Cases

Failure Cases	NginxConf	NginxCom	PermitErr	RegistrationErr	PostgresSQL Conf	Minimum Distance
Iteration-1	1	0	0	0	0	1
Iteration - 2	1	0	1	0	1	1
Iteration-3	0	0	1	0	0	2
Iteration-4	0	0	1	0	0	2

5.3 Two layer distributed component

At the application layer, individual components are configured in separate vm-instances of the cloud. Hence distributed component architecture is achieved where separate services are designations to separate vms. The e-commerce website is hosted in the cloud using a 2-layer

implementation mechanism [19]. The top layer consists of vm containing the web server and pages. Hence the Domain Name Service (DNS) component is also configured in the vm-instance to ensure separation of services. The first tier consists of the vm that contains web pages with which customers can communicate and search for desired goods in the e-commerce website. The operations at the client end involves the component running at this vm-instance that include registering, posting new adverts and selling products.

The second vm consists of PostgreSQL database that stores data entered by users. Two tables are added primarily to the database namely User Information, ItemList and Price. All the components are implemented in cloud datacenter running OpenStack Icehouse on CentOS 6.5. The vm-instances are allocated Random Access Memory (RAM) of 1 GigaByte (GB) each and 4 virtual Central Processing Units (vCPU). The application tier has 10 GB of disk drives and the DB tier has 30 GB of hard disk storage allocate using the cloud.

5.4. Interaction of distributed components

The paper takes into consideration that the wide array of operations like registration of new users, searching for products, adding new products for sale, reporting items, etc are conducted by the e-commerce website under consideration in the case study. Hence the results that can be obtained from the experimental test bed are applicable to the entire class of standard e-commerce websites.
The proposed Monitor plugin is developed and implemented in the e-commerce components to track and diagnose failure of those. Algorithm 1 identifies the Monitor mechanism and states the inputs, observations and notification measures. More specifically, the service tracked by Monitor are summarized as: User Registration, Post Advert, Browse Page, Book Item, and V isit Home. Under the given scenario, the following failures are identified to occur frequently in software components.

- ***Web-server failure:*** The request overflow problem can occur in the web server.

- ***Server to Application Communication failure:*** This issue occurs when the webserver fails to communicate with the application component due to network service termination. In case of Service Oriented Architecture (SOA) this issue may occur when one vm component is manually shut down and the other vm's cannot communicate with it.

- ***PostgreSQL failure:*** PostgreSQL configuration issues may result in failure of the database component, which occurs during incorrect configuration of the DB vm.

- ***File permission failure:*** This failure case occurs when the users do not have permission to execute database write operations in the DB component of the e-commerce website.

The User Registration activity will require the component to access he UserList table in the DatabaseBase (DB) component, whereas Browse Page require access to the ItemList table, Visit Home is a simple command to load the home page of the website and Book Item requires access to the Price table in the DB component. Out of all the requests identified, a call to the DB

5.5 Result analysis

The experimentation is conducted through saving the failure cases in a case table as shown in Table I. The first row identify different component failures that are monitored and the first column highlights the user activity in the e-commerce web pages [20]. The failure cases are fed into the monitor vm that matches the component vm's for the availability if defined failures.

Table 3 Time Consumption and VM allocation Redundant Components

Failed Case	Time(sec)	VM-allocation	State-Status
Iteration-1	0.44	0	Not-Preserved
Iteration-2	0.60	6	Preserved
Iteration-3	0.68	7	Preserved
Iteration-4	0.34	4	Preserved

The experiments contained four iterations where the user conducted a number of pre-defined activities and Monitor was configured in each of the components to detect failures. In 4 iterations of the experiment the failure conditions were triggered with respect to the four criteria described in previous section. Table II highlights the output of the failures in 4 iterations during experiment. The 1 shows an occurrence of a failure in the pre-specified attribute and 0 shows that failure did not occur.

After detection of the failures in Table II, distance between the obtained iteration and pre-defined failed cases are calculated. Exclusive OR is used to identify the distances between each saved failed case and failures of iteration. Among those the minimum distance is calculated to identify the closed failed case. When the closest case is determined, the solution of that case is implemented for the target iteration and the time consumed for system restoration and self-recovery are recorded. Also time required for vm-allocations are recorded to identify the number of active cloud vm's from the resource pool.

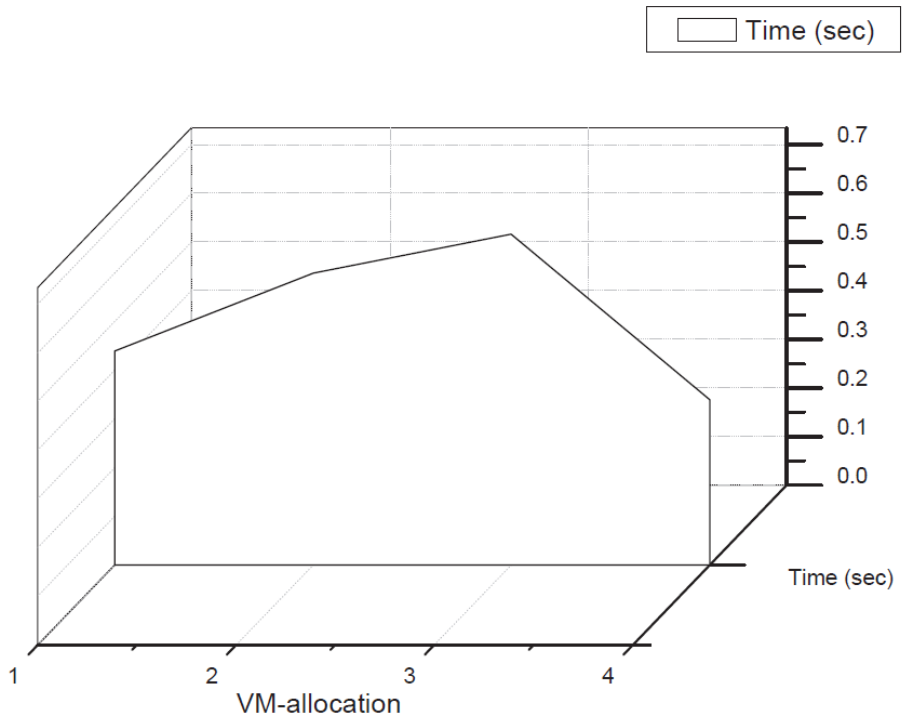

Figure 3 Time-consumption representation for self-healing in the tested iterations

The time consumed for vm allocation and self-healing has been shown in Table III. For every iteration, the time was less than the predefined standard of 0.7 seconds [16]. The table shows that no vm is allocated for Iteration 1 and that no state was needed to be preserved since it was closest to the case of homepage loading for the e-commerce sight, thereby does not require critical saving of states. The performance of the system in terms of high speed self recovery has been highlighted in Figure 3

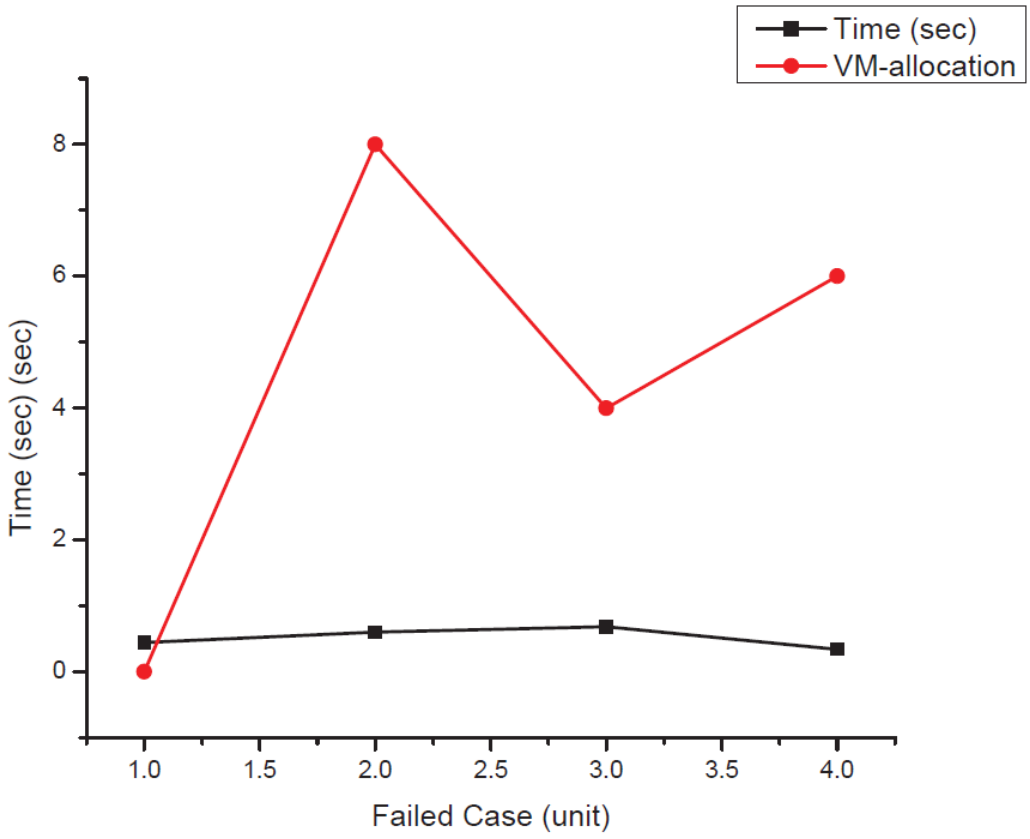

Figure 4 : Vm-allocation and Time-requirement comparison

The time-consumption is respond to vm-allocation has been represented graphically in Figure 4. The results show the desirable low time overhead of the proposed self- healing framework and state preservation of failed software components.

Case based result for small program which is running on different virtual machine: This component monitor will always get the health status of the component and will take action as situation demands. For test purpose we have set few programs that works fine. By setting up a threshold value like memory over allocation, runtime-error we crashed those program and reviver healed those components in a moment. Expected time for reviving is

Table 4 Reason form and revive time

Program	Reason for crash	VM number	time
Forever.py **factorial program that runs upto infinite** **Threshhold value set to – 500KB for memory over allocation**	The file size increases to 501 KB which passed the threshold value	1	0.3s expected 0.1 s
Imortal.py **It send 'hi I am working fine' to the component monitor every 2 seconds** **A programmatically set function manipulates the time to 4 s**	Monitoring system did not get the message on 2s . As it found that respond time is above to 2s it automatically take action for revivg process	2	0.2s expected 0.1s
Forever.py **this programs runs and print 1**	Manually stops the program "CTRL + C " with thi s command in that specific vm's terminal	3	0.1s expected 0.1s

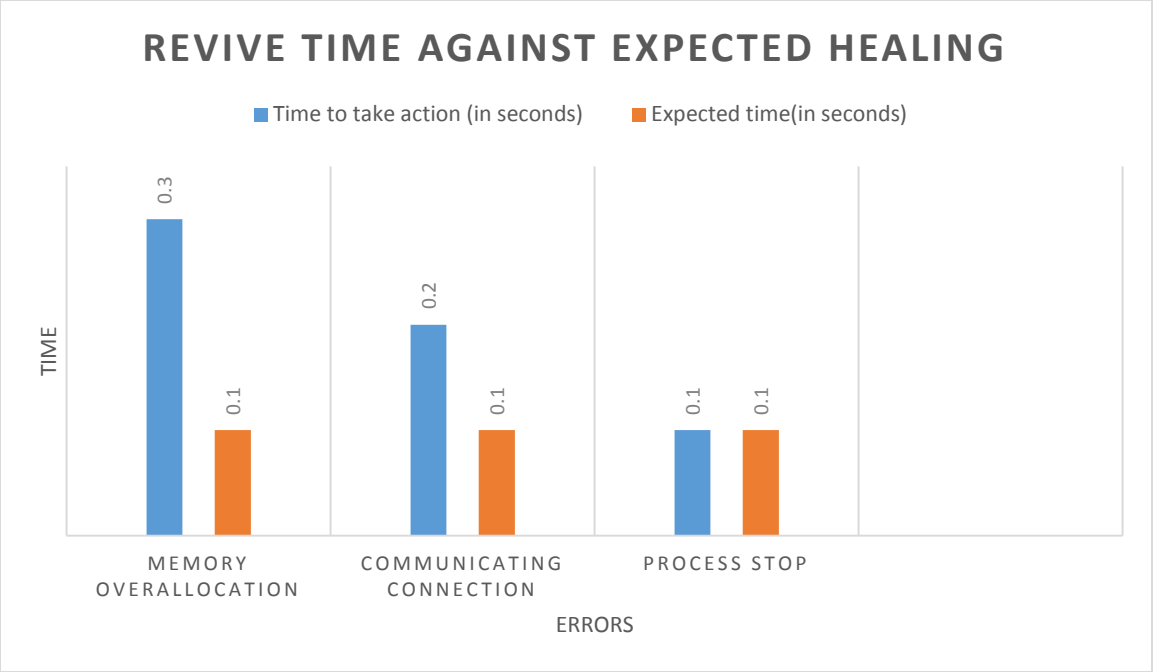

Figure 5 Revive time against expected healing

Three program executed on three different virtual machines with their private ip. A component monitoring system will send command to the healing system for reviving. For memory over allocation t took .3 second for revving whereas .1s is expected. This delay happened due to network latency. Also for communicating connection problem network latency. When process stop error healing system done the job in expected time.

However the whole system can be more improved by reducing network latency. Improvement of the whole architecture and recovery can improve the result.

Although the result is satisfied for primary level of implementation. More optimized recovery algorithm can be proposed for improvement of the result.

Chapter 6. CONCLUSION

This paper has proposed a framework for self-healing mechanism in complex distributed applications. Through the identification of the failures in the processes and components, the proposed framework regenerated the paralyzed components to operate seamlessly. The framework aimed to revoke the paralyzed or faulty components by the Reviver module, which initiated the components with the previous state and necessary information provided by the Monitor system. The proposed architecture compared detected faults with pre-defined failure case table through exclusive OR that enabled obtainment of distance between those in each case. The solution of the predefined entry in the failure case table that had the minimum distance with the detected failure was applied since it is the closest to recover from the obtained failed condition. Empirical evaluation of the experimental iterations show that time required for self-healing of the components is below 0.7 seconds in all the four test cases. At the same time minimum distance was calculated as 0-2 units in all the iterations, showing similarity of the cases to real life scenarios. Since cloud is used to aid in rapid allocation of vm-instances, redundancy can be achieved. As a result the distributed soft- ware applications can be made more fault tolerant through amalgamation of the proposed self-healing mechanism.

As stated earlier, the proposed framework ensures effective self-healing of distributed components using redundant vm- instances from the cloud. Association of learning methods for the framework to sprawl instances in accordance to component requirements in real time is an area of future research interest.

Bibiography

[1] V. Nallur and R. Bahsoon, “A decentralized self-adaptation mechanism for service-based applications in the cloud,” Software Engineering, IEEE Transactions on, vol. 39, no. 5, pp. 591–612, 2013.

[2] A. Imran, A. U. Gias, R. Rahman, A. Seal, T. Rahman, F. Ishraque, and K. Sakib, “Cloud-niagara: A high availability and low overhead fault tolerance middleware for the cloud,” in Computer and Information Technology (ICCIT), 2013 International Conference on. IEEE, 2013, pp. 231–237.

[3] S. Sidiroglou, O. Laadan, C. Perez, N. Viennot, J. Nieh, and A. D. Keromytis, “Assure: automatic software self-healing using rescue points,” ACM SIGARCH Computer Architecture News, vol. 37, no. 1, pp. 37–48, 2009.

[4] M. M. Fuad, D. Deb, and M. J. Oudshoorn, “Adding self-healing capabilities into legacy object oriented application.” in ICAS, vol. 6, 2006, pp. 51–51.

[5] S. Montani and C. Anglano, “Achieving self-healing in service delivery software systems by means of case-based reasoning,” Applied Intelligence, vol. 28, no. 2, pp. 139–152, 2008.

[6] L. Wei, Z. Yian, M. Chunyan, and Z. Longmei, “A model driven approach for self-healing computing system,” in Computational Intelligence and Security (CIS), 2011 Seventh International Conference on. IEEE, 2011, pp. 185–189.

[7] N. Cardoso and R. Abreu, “Self-healing on the cloud: State-of-the-art and future challenges,” in Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the. IEEE, 2012, pp. 279–284.

[8] M. Bisadi and M. Sharifi, “A biologically-inspired preventive mechanism for self-healing of distributed software components,” in Advanced Engineering Computing and Applications in Sciences, 2008. ADVCOMP’08. The Second International Conference on. IEEE, 2008, pp. 152-157.

[9] A. Zisman, G. Spanoudakis, J. Dooley, and I. Siveroni, “Proactive and reactive runtime service discovery: a framework and its evaluation,” Software Engineering, IEEE Transactions on, vol. 39, no. 7, pp. 954–974, 2013.

[10] A. Imran, A. U. Gias, and K. Sakib, “An empirical investigation of cost-resource optimization for running real-life applications in open source cloud,” in High Performance Computing and Simulation (HPCS), 2012 International Conference on. IEEE, 2012, pp. 718–723.

[11] G. Jung, T. Margaria, C. Wagner, and M. Bakera, “Formalizing a methodology for design and runtime self-healing,” in Engineering of Autonomic and Autonomous Systems (EASe), 2010 Seventh IEEE International Conference and Workshops on. IEEE, 2010, pp. 106–115.

[12] J. Park, G. Yoo, and E. Lee, “A reconfiguration framework for self-healing software,” in Convergence and Hybrid Information Technology, 2008. ICHIT’08. International Conference on. IEEE, 2008, pp. 83–91.

[13] S.-W. Cheng and D. Garlan, “Stitch: A language for architecture-based self-adaptation,” Journal of Systems and Software, vol. 85, no. 12, pp. 2860–2875, 2012.

[14] C. Parra, X. Blanc, A. Cleve, and L. Duchien, “Unifying design and runtime software adaptation using aspect models,” Science of Computer Programming, vol. 76, no. 12, pp. 1247–1260, 2011.

[15] H. Song, G. Huang, F. Chauvel, Y. Xiong, Z. Hu, Y. Sun, and H. Mei, “Supporting runtime software architecture: A bidirectional- transformation-based approach,” Journal of Systems and Software, vol. 84, no. 5, pp. 711–723, 2011.

[16] G. Salvaneschi, C. Ghezzi, and M. Pradella, “Context-oriented pro- gramming: A software engineering perspective,” Journal of Systems and Software, vol. 85, no. 8, pp. 1801–1817, 2012.

[17] A. Imran, A. U. Gias, R. Rahman, and K. Sakib, “Provintsec: a provenance cognition blueprint ensuring integrity and security for real life open source cloud,” International Journal of Information Privacy, Security and Integrity, vol. 1, no. 4, pp. 360–380, 2013.

[18] R. De Lemos, H. Giese, H. A. M¨uller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel et al., “Software engineering for self-adaptive systems: A second research roadmap,” in Software Engineering for Self-Adaptive Systems II. Springer, 2013, pp. 1–32.

[19] N. Huber, A. van Hoorn, A. Koziolek, F. Brosig, and S. Kounev, “Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments,” Service Oriented Computing and Applications, vol. 8, no. 1, pp. 73–89, 2014.

[20] H. Ehrig, C. Ermel, O. Runge, A. Bucchiarone, and P. Pelliccione, “Formal analysis and verification of self-healing systems,” in Fundamental