8-6-2011

A model-based approach for automatic recovery from memory leaks in enterprise applications

Zimin Wang

A MODEL-BASED APPROACH FOR AUTOMATIC RECOVERY FROM MEMORY LEAKS IN ENTERPRISE APPLICATIONS

By

Zimin Wang

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

August 2011

A MODEL-BASED APPROACH FOR AUTOMATIC RECOVERY FROM MEMORY LEAKS IN ENTERPRISE APPLICATIONS

By

Zimin Wang

Approved:

Sherif Abdelwahed Assistant Professor of Electrical and Computer Engineering Department (Major Advisor)	Randolph Randy Follett Assistant Professor of Electrical and Computer Engineering Department (Committee Member)
Bryan A. Jones Assistant Professor of Electrical and Computer Engineering Department (Committee Member)	James E. Fowler Professor and Graduate Program Director of Electrical and Computer Engineering Department (Graduate Coordinator)
Sarah A. Rajala Dean, Bagley College of Engineering	

Name: Zimin Wang

Date of Degree: August 6, 2010

Institution: Mississippi State University

Major Field: Electrical Engineering

Major Professor: Sherif Abdelwahed

Title of Study: A MODEL-BASED APPROACH FOR AUTOMATIC RECOVERY FROM MEMORY LEAKS IN ENTERPRISE APPLICATIONS

Pages in Study: 104

Candidate for Degree of Master of Science

Large-scale distributed computing systems such as data centers are hosted on heterogeneous and networked servers that execute in a dynamic and uncertain operating environment, caused by factors such as time-varying user workload and various failures. Therefore, achieving stringent quality-of-service goals is a challenging task, requiring a comprehensive approach to performance control, fault diagnosis, and failure recovery.

This work presents a model-based approach for fault management, which integrates limited lookahead control (LLC), diagnosis, and fault-tolerance concepts that: (1) enables systems to adapt to environment variations, (2) maintains the availability and reliability of the system, (3) facilitates system recovery from failures. We focused on memory leak errors in this thesis. A characterization function is designed to detect memory leaks. Then, a LLC is applied to enable the computing system to adapt efficiently to variations in the workload, and to enable the system recover from memory leaks and maintain functionality.

DEDICATION

I would like to dedicate this research work to my beloved parents, Mr. Jianping Wang and Mrs. Sulan Li for their love, my advisor Dr. Sherif Abdelwahed for his continuous support, and my cherished friends Han Zhang, Rui Jia and Rajat Mehrotra for their generous help.

ACKNOWLEDGEMENTS

I take great pleasure in expressing my gratitude and sincere thanks to my academic advisor, Dr. Sherif Abdelwahed, for his valuable guidance and support that enabled me to complete my research work in the stipulated time. I would also like to thank Dr. Randolph Follett and Dr. Bryan Jones for being on my graduate program committee and evaluating my thesis work. I also take this opportunity to express my thanks to Mr. Rui Jia for his suggestions regarding LLC. I am thankful for Mr. Rajat Mehrotra, who helped me setup the DayTrader system required for the research study. I am expressing my thanks to all the graduate students in my lab at Mississippi State University for their assistance and cooperation during the course of my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Motivation and Research Objectives

During the past ten years, service orientation (SO) has become a new system design paradigm for building distributed systems. Service Oriented Architecture (SOA) is a paradigm for designing a software system, which provides service to either applications or other services. It is a system design philosophy, which is independent of any specific technologies, *e.g.*, Web Services (WS) or J2EE [1].

The main structure blocks of SOA include the service provider, the service registry, and the service consumer. Service providers are applications that provide service to the service consumer via a response SOAP (Simple Object Access Protocol) message. Service consumers have access to the service by sending a request SOAP message. Both the service consumer and service provider can share the same software application at the same time.

Although clients always want to have stable and reliable service from web service providers, sometimes service providers may not be able to provide their service at the level the consumers expect because of faults and failures. Faults can happen in different places at different stages; for example, faults can occur in the software application, in middleware, or in hardware components.

Compared to traditional client-server applications, a Web-based SO distributed system can be made up of a number of services which are typically hosted by different providers across the Internet with distant geographical locations. Without centralized control over all the constituent services, which are often geographically distributed far away, there is no guarantee about the reliability of the underlying service. If any of the constituent services fail, the service may be seriously affected. Therefore, the unreliability of any constituent service could cause Quality of Service (QoS) degradation of the composite service, even if all the other constituents remain stable and reliable all the time.

The general approach to fault management techniques can be viewed as a loop of monitoring, storing, analyzing and remedying. For these SO distributed systems, fault management first requires identifying and classifying faults that may occur during service execution and then specifying a set of strategies to handle them. This work mainly addresses memory leak faults, which is one of the most common faults that may happen in any distributed system.

Researchers have recently investigated different approaches to design fault management systems. An efficient and stable fault management system is necessary and important for a stable and reliable web service system.

The objective of this work is to construct a fault management system by applying an online Limited Lookahead Controller (LLC) to detect, handle and correct a memory leak fault in a general class of SO distributed systems. This developed management structure addresses the detection, isolation, and recovery from a memory leak fault.

1.2 Problem and Approach

Various control approaches have been developed and applied in different application environments to meet user-specific requirements, such as efficiency and load balancing. In the proposed design, a model-based controller is developed to automatically manage the performance and availability of a distributed system, which has a memory leak fault. A memory leak is one of the most common faults that can happen in any computer system. In the proposed system, memory leak problems exist either outside the service application running parallel in the same system or inside the web service system, which will affect the performance of the application since the hosting server start working. In addition, the service application running on our distributed platform is a Java-based application where the Java garbage collector also has an impact on the overall performance. To manage memory leaks in distributed systems, a LLC is designed to initiate effective recovery actions when such a fault is detected.

Before the controller takes action, a Java program, “Memory Monitor”, that is a crucial functioning part in the system, will characterize the system performance, analyze it and return diagnosis related information to the controller.

LLC can be applied to a general class of hybrid system with mixed discrete event and continuous dynamics and a finite set of control inputs, - typical characteristics of many computing systems [110]. In this online control structure, system model parameters are estimated and used by the system to forecast future behavior. The controller will analyze the current parameters and select the best control input for the system. In the proposed management structure, the most important requirements are power consumption and response time. Under memory leaks, the system will have a relative performance

criteria based on which LLC is able to choose the best control action that will balance the power consumption and response time to meet certain Quality of Service (QoS) objectives. Basically, the controller estimates a set of future states from the current step then selects the path that can minimize a cost function and also satisfy both the state and input constraints.

The objective of the LLC structure is to optimize the system utility function (characterizing the system performance) using the available control options (inputs). Generally, controller performance depends on a couple of highly correlated control factors including prediction horizon, control set, and sampling time.

We show that, using the designed management structure a benchmark distributed system is able to detect, diagnose and recover from the memory leak fault. Two basic scenarios are implemented. In the first scenario, the process with the memory leak is an independent process outside of the web service application process. The controller is able to detect the process with the memory leak, get the PID (Process ID), and then take action on the process. For the second scenario, when the memory leak fault happens inside the service application, a special function called "Memory Monitor" is designed to detect the existence of memory leak and predict the intensity of the memory leak. According to the result from the Memory Monitor, LLC will take action to correct the memory leak. The action can be either to restart the main system or switch to the backup system. Based on a given utility function, the LLC evaluates the current memory leak intensity and chooses one of these two actions. Finally, the controller will bring the system back to the correct working state.

1.3 Thesis Organization

This thesis is organized as follows. Chapter II introduces the background and gives a common notation of distributed systems and fault management. Chapter III presents the literature review of the various fault management methods, fault detection and diagnosis methods in distributed systems. This chapter discusses the recent research results of fault management, including the classification and comparison of those methods, and also shows the analysis results about their contributions and limitations. Chapter IV proposes the experimental approachs, methods and results of the memory leak simulation and characterization. This chapter introduces the application platform and methods that are used in the proposed system, and also shows the experimental results and analysis of the garbage collector. Chapter V introduces the LLC controller and the implementation of the LLC in the target distributed system. Three different experimental scenarios are discussed and related experimental results and analysis are presented in detail. Finally, Chapter VI summarizes the conclusions and discusses future work.

CHAPTER II

BACKGROUND

This chapter introduces the technical terms used and related background on distributed systems and web service. Classifications of fault and failure and the definition of a memory leak is also presented.

2.1 Distributed System

A distributed computing system is generally defined as a collection of independent computers that appears to its users as a single coherent system [2]. There are two additional points about this definition. The first one is that the components of a distributed system are autonomous. The other aspect is that the whole system always appears as a single system to the users.

Basically, distributed systems have several characteristics. First, interactions between distributed system components are hidden from users. To users, any distributed system always appears as a single system. Another characteristic is that distributed systems can interact with users and applications in a uniform and consistent way. Also, it is desirable that distributed systems should be continuously available, even if some parts of the distributed systems have problems and are temporarily out of order.

2.1.1 Types of Distributed Systems

Generally, there are three types of distributed systems: distributed computing systems, distributed information systems, and distributed pervasive systems. Usually, distributed computing systems are designed for high-performance computing. Distributed information systems are found in organizations that include complex networked applications. Distributed pervasive systems are usually small, battery-powered, mobile and only have a wireless connection [2].

2.1.2 Distributed System Architecture

It is crucial to organize the distributed system well, because the components of some complex software can physically be far away. Mostly, organization of the distributed system addresses the software components and their interactions. This organization is called software architecture.

2.1.2.1 Architecture Styles

The architecture style is defined by the different components that make up the system, the way these components communicate with each other, and how they are physically connected to each other. There are several classified architecture styles including layered architecture, object-based architecture, data-centered architecture and event-based architecture as illustrated in Figure 2.1.

Layered architecture has all the components structured in several levels where any component at a certain level is only allowed to call components at a lower level. Object-based architecture is organized in a more flexible way. In this structure, objects

correspond to components, and they interact through remote call procedures. This software structure is in the same style as the client-server system architecture. The SO distributed systems discussed in this work are of this type. As for the data-centered architecture, the communications between all the processes are based on a common passive repository. In event-based architecture, processes communicate through the propagation of events, which sometimes carry data. Event-based architectures have an advantage in that processes do not need to strictly relate to each other. The Combination of event-based architecture and data-centered architecture creates a structure known as shared data spaces. This work mainly discusses object-based architecture.

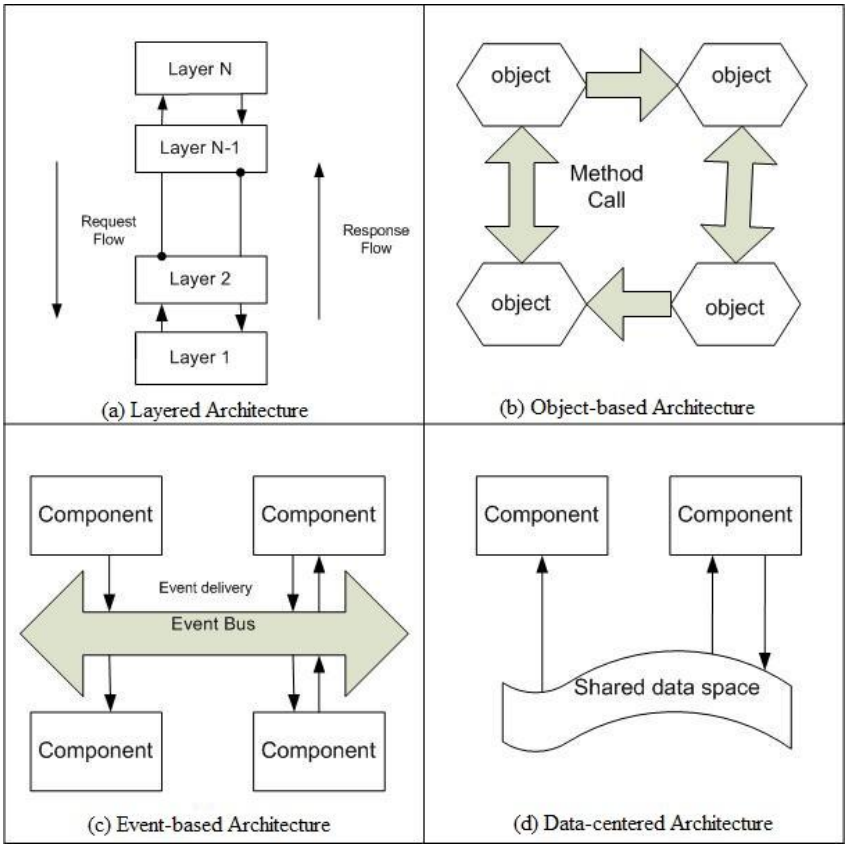

Figure 2.1 Architecture styles of distributed system

2.1.2.2 System Architecture

System architecture addresses software components, their connections, as well as their physical placement. System architecture can be either centralized or decentralized. This work only discusses the centralized type of distributed systems. The client-server model is a typical centralized architecture.

Organizing distributed applications in terms of clients and servers is a useful approach. Client processes can go through different user interfaces that can be very simple displays or advanced interfaces, which can handle compound documents. Client software is aimed at providing distribution transparency by putting the communication details in the structure. Servers are much more complicated than clients and they can be iterative or concurrent and can include one or more services.

In client-server architecture, the client process requests a service from a service providing process called a server by communicating with request and reply as shown in Figure 2.2. This kind of mechanism is also known as request-reply behavior.

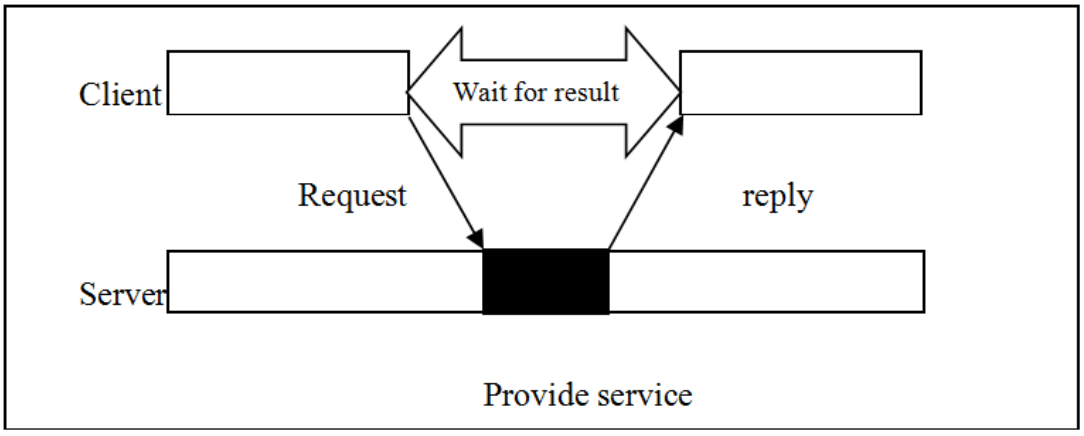

Figure 2.2 General interaction between a client and a server.

2.1.3 Self-management in Distributed Systems

Distributed computing systems, comprising a large number of networked hardware and software components, host a wide variety of enterprise, communication, and scientific applications with stringent reliability and quality-of-service requirements. These systems typically operate in a dynamic environment and are subject to uncertain operating conditions caused by multiple factors such as time-varying workload, hardware degradation, and software failure. To operate such systems efficiently and reliably, multiple performance related settings, such as resource provisioning and relative priority between applications, must be dynamically tuned to match the time-varying environment and changing conditions. As computing systems increase in scale and complexity, meeting their QoS and reliability requirements via manual tuning is not just tedious and error-prone, but also infeasible. Coping with this complexity requires systems to become capable of managing themselves given high-level objectives from administrators, users, and system engineers.

Self-management is compromised of the following main aspects :

1. Self-configuration, which provides techniques so that the service architecture can work properly even if nodes are added or removed during execution.
2. Self-healing, which provides the techniques to make sure the system can work properly when nodes or communications fail. It also provides techniques to support the reconfigurations of nodes.
3. Self-optimizing, which enables the system to balance the workload and manage overloads automatically to achieve optimal usage of resources and minimize operating cost.

4. Self-protection, which is responsible for detection and mitigation of security attacks and intrusion attempts.

2.1.4 Fault Tolerance

Reliability is one of the most important aspects of practical distributed systems. Reliable operation of distributed systems requires that the system can automatically recover from most failures without significantly affecting the functionality of the system and the underlying services.

2.1.4.1 Basic Concepts

For distributed systems, fault tolerance is highly correlated with the concept of dependability. Dependability covers a number of features, which are required by the distributed systems including availability, reliability, safety and maintainability. First of all, Availability refers to the possibility that the system works properly at any given moment and is always ready to provide the service based on the users' requirements. As for Reliability, it represents the property that a system can run continuously without major failures. Safety indicates the situation that if a system temporarily fails to work properly, nothing serious will happen such as the whole system crashing. Finally, Maintainability is how easily a failed system can be recovered.

Basically, all distributed systems are designed to provide services to their users whose requirements are different from each other. The system is said to fail when it cannot provide one or more promised services. An error is a part of the working status for a system, which may cause failure. The problems that lead to a failure are called a fault.

2.1.4.2 Process Resilience

Redundancy is the key technique for fault tolerance. To design a fault tolerant distributed system, organizing a number of identical processes into groups is the key approach. The most important feature for organizing groups is that when a group gets a message, all the members of the group will receive it. So, when a failure happens to one or more of the members in the group, hopefully other members can still work properly and take over for it.

Usually, it is necessary that processes in the same group be highly correlated, so that communication between members can be reliable and adhere to stringent ordering. The structure of organizing groups can be classified into two types: flat and hierarchical. In flat groups, all the processes are equal, and there is no dominant process to control other members. On the other hand, some processes serve as coordinators and have overall control, and all the other processes work as fellow workers in a hierarchical group. Both of these two types of groups are widely used in modern distributed systems and depend on different requirements. The experimental system in this work is the flat group type with all the processes being equal.

Organizing identical processes into groups enables us to mask one or more processes that have faults in that group. Basically, there are two approaches to carry out the masking or replicating: by means of primary-based protocols, or through replicated-write protocols.

In order to properly mask failures, distributed systems should detect failures before we take action to recover from failures. In general, for processes in the same

group, the members without faults should be able to identify which members are still working properly, and which ones have errors.

2.1.4.3 Reliable Communication

For all distributed systems, faults not only happen to processes but to communications as well. Most of the failure models and concepts mentioned above apply for communication faults as well. Faults like crashes, omissions, timing, and arbitrary failures are common for communication channels. In particular, constructing reliable communication channels needs more focus on mitigating crash and omission failures. For arbitrary failures in communication channels, they probably happen because of duplicate messages. Messages may be stored in the buffer for a relatively long time, and when it is re-sent to the network, the original sender has already made a retransmission.

Because it is very important to achieve process resilience by replication, reliability of group communication is also crucial for constructing fault tolerant distributed systems. For simple small groups, reliable communication is feasible. However, as the size of the groups keep growing, the scalability of reliable communication becomes hard to achieve. To maintain the scalability, the key issue is to reduce the number of those messages reporting information about the receipt of the messages by the receivers.

2.1.4.4 Recovery

When a fault happens, it is important and sometimes necessary for the system to recover from the error state back to the original correct state to remain functionality.

Basically, the approach to recover a system from a fault is to replace the error state with a correct state. Generally, there are two types of error recovery. First, backward recovery refers to bringing the system from the current erroneous state back to the previous correct state. In this type of recovery, the system has to keep a record about every state from the beginning, and whenever an error occurs, the system will be brought back to the last correct state based on the record. Obviously, every state from the beginning has to be recorded which means a checkpoint has to be made. The second type of recovery is forward recovery. In this case, the system tries to skip the erroneous state when it occurs and bring itself to a correct new state instead of going back to the previous checkpoint state. Both of these two types of recoveries have their advantages and disadvantages.

Generally, backward recovery is more widely used in modern distributed systems as a general approach to recover failures. The major advantage of this type is that it can be applied to any specific system or process. However, it also has three main problems: First, bringing the system or process back to the previous state is a relatively expensive operation. Second, because the backward recovery is independent from the environment where it is used, there is no guarantee that the same fault will not immediately happen again after recovering from it. Finally, for some states it is sometimes impossible to bring the system back to the checkpoint, though a complete record about the system has been made. To improve performance, some systems combine checkpointing with message logging.

To make the system recover to the previous correct state, a complete record about every state from the beginning is needed. Therefore, stable and safe storage is also

important for distributed systems. The most basic forms of storage can be classified into three subcategories: ordinary RAM memory, disk storage, and stable storage. The proposed system in this work uses ordinary RAM memory to record every working state.

2.1.5 Quality of Service (QoS)

An important requirement for a SO distributed system application is to maintain stability and to provide consistent services that reach the quality standard. This requirement for applications is not only included in the functional services, but focuses on the quality of the environment that hosts the services as well. QoS describes the ability of the application to respond to expected invocations and to provide services at a level that satisfies both the customers and providers.

The distributed system applications used by enterprises are becoming increasingly complex and dependent on other distributed system applications. QoS is becoming the most important requirement or consideration for both service customers and service providers.

2.2 Distributed System Service Management

This section introduces the architectural characteristics, structure, and responsibilities of the Distributed System (DS) management and support infrastructure.

2.2.1 Terms Related to Management of Distributed Systems

Michael P. Papazoglou mentions, "A managed or manageable resource in a distributed environment could be any type of a hardware or software component that can

be managed and that can maintain embedded management related metadata. A managed resource could be a server, storage unit, database, application server, service, application, or any other entity that needs to be managed." [1] In distributed systems, the management structure takes charge of controlling and monitoring applications through the whole function cycle from initializing to collecting results to making sure applications work correctly and properly [3]. Manageability can be defined as the ability to take actions of administration and superposition and receive feedback about those actions on a system or component [4]. At the same time, manageability should be under constraints that the target application has to work correctly, be active and be able to check the application performance from time to time. Based on his definitions, manageability can be viewed as three different functioning parts: monitoring, tracking and control. The first part, monitoring, represents the ability to make a record of the current and previous performance from some components, and these records can be used for further reporting and analyzing. The second part, tracking, refers to the ability to observe multiple features of the same unit from multiple resources. The third part, control, indicates the ability to send an alert message about the current behavior of the target application to which the management system is applied.

According to M. P. Papazoglou [1], distributed management solutions should have components for user experience: monitoring, infrastructure monitoring, transaction monitoring and resource provisioning. Before applying this to distributed systems, any management system should specify four necessary requirments [5]. First of all, the specific applications or components that need to be managed. Second, the characteristics

of the target applications. Then, the changes needed to meet the requirement for the specific application. The last requirement, the relationships among managed components.

Most modern enterprise management systems apply various management mechanisms including fault management, security management, performance management, configuration management and accounting management [6].

2.2.2 Features and Functional characteristics of DS management

Distributed system applications need to be managed at least two levels. The first is the operation level. Usually the administrator controls the distributed system to start or stop from this level. Also, work information such as how many processes or applications are running in a service system can be observed from this level. The other level is the business management level. This level provides access for monitoring and analyzing the operating performance, so that business administrators have access to watch over the business operations, identify opportunities and recover problems. In this way, administrators can make sure the distributed system is working correctly and me all the requirements for specific business tasks.

The first management level – the operational management dimension – can be defined as the mechanism for checking the existence, availability, stability, performance and the maintenance of a distributed system application. The second management level – the business management dimension – offers end-to-end visibility and can control the whole system including all of the processes across multiple applications on one or more enterprises. This management dimension is highly correlated to the concept of business management.

A distributed system management framework includes a number of components that enable monitoring, troubleshooting, and service for application management. Four activities are considered as the responsibilities of an end-to-end distributed system management framework. First, the DS management framework is responsible for measuring end-to-end levels of service for work that is delivered to users. Second, management framework also needs to break work units into components that can be identified and be measured. Third, management framework attributes end-to-end service levels and the required resources to those work units. Finally, the framework should be able to identify current problems and predict future requirements.

To perform the activities mentioned above, the management framework should take into account several other system management aspects, including: performance indicators, auditing, monitoring, troubleshooting, service redeployment, service life cycle and state management, *etc.*

2.3 Controller Background

This section introduces basic control approaches and notation approaches. As control theory has been around for decades, different strategies and approaches have been developed to adapt dynamic systems to meet various operational requirements. One classical control technique, feedback control, is the most widely used control method for the past decades due to its simplicity and effectiveness. The term feedback seems to have been first in use in 1920 by Bell Telephone Laboratories. Following basic feedback control, more sophisticated control methods have been developed to handle complex dynamics and requirements, such as adaptive control [7], stochastic control [8], fuzzy

control [9], and optimal control [10]. All of these control techniques have their own advantages and disadvantages depending on the different environments to which they are applied.

2.3.1 Introduction to Common Automatic Control Approaches

Classical control theory was initially developed to address physical process control. Subsequent control techniques usually have the same basic framework and components though these techniques may be applied to different systems.

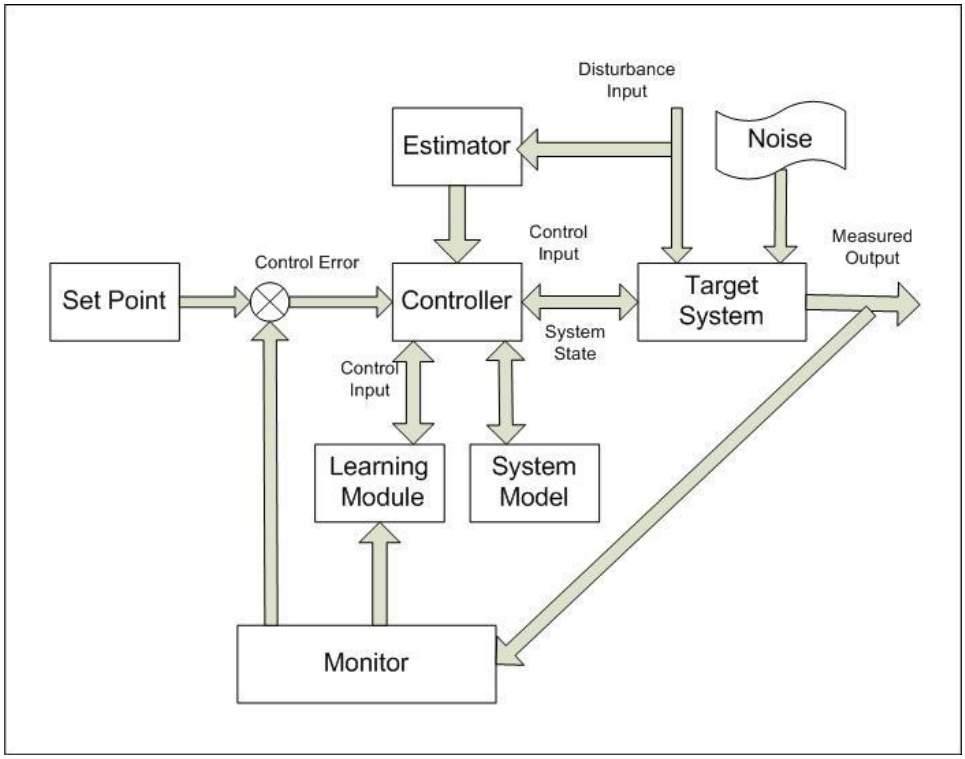

Figure 2.3 Basic structure of a control system

The basic structure of a typical control system, shown in Figure 2.3, includes the following parts:

1. The set point is also referred to as desired output or reference input. It is a system state or the system response that a system is designed to reach. Usually, by achieving the desired requirements, a stable controlled system will reach the set point value.

2. The control error refers to the difference between the desired output and the actual output.

3. The control input represents a lot of parameters all of which will affect the performance of the designed system. However, these parameters are not always the same, and they can be adjusted dynamically. A lot of relative parameters, such as the CPU frequency, can affect the system.

4. The controller is a component designed to make the target system reach the desired output by using a set of control inputs, which are also processed by the controller based on the information it receives.

5. The disturbance refers to any external input that will affect the performance of the controlling process. Due to disturbance, the actual output will be different from the desired one.

6. The measured output represents the actual output that can be measured, such as response time and memory utilization.

7. The estimator is a function of the observable sample data used to estimate unknown parameters.

8. The system state is the intermediate variable that can be used to define the relationships between control inputs, performance variables and measurements.

2.3.2 Common Control Techniques

Based on the basic structure and main components introduced above, several control approaches are developed and widely applied in various control management problems.

2.3.2.1 PID Feedback Control

PID Feedback Control represents proportional-integral-derivative control (PID controller), and is a feedback control loop mechanism that is widely used in control applications. The aim of this type of controller is to correct the difference between the actual output and the desired output while maintaining system stablility. To correct the error, the controller calculates the difference and then takes action, sometimes directly taking a corrective action. Sometimes the output is a control input that contains information about adjusting the process. The control input of the PID controller is determined by a weighted sum of three parts: the Proportional value, the Integral and the Derivative. The reaction to the current error is determined by the proportional value. The integral comes from the sum of all recent errors and also takes action based on this. Finally, the derivative decides the reaction to the rate of the error change. A PID controller can be changed to a PI, P or I controller if any of the other parts is omitted.

PID controllers have been widely used in various computing systems and application environments. A digital PI controller is applied to measure server utilization

as mentioned by Abdelzaher *et al.* [11]. Abdelzaher and Bhatti [12] used a PI controller to present QoS maintenance in web server resource management. As for real time scheduling problems, Lu *et al.* [13] proposed a feedback control real-time scheduling (FCS) framework based on QoS control, and Lu *et al.* [14, 15] provided a method to maintain a low deadline miss-ratio in unknown environments by using feedback control. For service management, Dovrolis *et al.* [16] and Lu *et al.* [17] proposed an approach for different service classes on web servers to guarantee a relative delay, and a saturated integral controller is applied to the evaluation of controllers used for service level management of a software system by Parekh *et al.* [18]. In addition, Steere *et al.* [21] presented a CPU allocation feedback controller based on proportion and period, and Lu *et al.* [19] provided a feedback controller design problem to cache resource allocation, and feedback control theory was applied to analyze a congestion control algorithm on IP routers by Hollot *et al.* [20].

2.3.2.2 Feed-Forward Control

Feed-forward control [22] takes direct action according to the predicted behavior so that the system can react to a disturbances before they affect the system. Feed-forward control requires a model that predicts the effect of system inputs. For this prediction model, a number of theoretical techniques can be used including real-time scheduling theory and queuing theory. Because the feed-forward controller maintains the system at the operating point, a linearized model with relatively small deviation from operating point is enough for the feedback control. Sha *et al.* [23] presented the prediction of the future queuing delay using queuing theory based on request arrival rate and service rate.

With request arrival rate and service rate, the queuing delay in the steady state can be calculated with a simple formula according to Kleinrock [24]. This predictor is designed to be able to respond to sudden and transient changes of the workload [25]. The performance influence on the latency of future requests is predicted by heuristic approximation. Applying this heuristic approximation, Lu *et al.* [26] developed relative delay maintenance in a web server. Chandra *et al.* [27] introduced resource allocations required to meet some service level objectives that are calculated according to the predicted workloads by the online measurements of three parts including the request arrival process, service demand distribution and queue length. Xu *et al.* [28] proposed an approach that directly sets the actuation level for the next control period based on the desired output value and the predicted related variable. The possibility of using resource utilization metrics to predict future resource demands is studied.

2.3.2.3 Adaptive Control

The main problem addressed by an adaptive controller is to modify the control law used by a controller from time to time in response to changes in the system dynamics and structure. Taking the network server as an example. The workload of a network will keep changing according to the changing demands over time. In this case, the control law needs to adjust itself to specific conditions that can change at any time. Lu *et al.* [29] introduced an adaptive pole placement control applied to QoS-aware web caching, supporting proportional differentiation on the average hit rate of the different content classes. With parameters updated online based on a linear approximation, the controller fits into a model to dynamically tune its function. To deal with modeling inaccuracies and

load disturbances, a Queueing-Model-Based Adaptive Control that is made up of an online parameter estimator and a control law from the known parameter is proposed by Lu *et al.* [29]. For the parameter estimator, Recursive Least Squares estimators are normally used. Liu *et al.* [30] presented an adaptive control of resource containers among shared servers. As for the indirect self-tuning adaptive controller, the dynamic model is estimated from online measurements, and its parameters come from the calculation with the current estimated model by the recursive least-squares method and pole placement. A direct self-tuning adaptive controller is used to ensure the performance for storage access by Karlsson *et al.* [31]. In most application environment, controllers look at the system as a "black box" which is completely unknown to the controllers. To gain access to information about the system, controllers can usually use monitors to watch the system. Adaptive control approaches have been widely applied in nonlinear systems [32]. The adaptive control method has also been used to develop an intelligent fault-tolerant control system by Diao and Passino [33]. When MIMO adaptive optimal controller is combined with a nonlinear optimizer, this system is able to increase availability of the computer service when multiple customers exist [34].

2.3.2.4 Fuzzy Control

Fuzzy control systems come from fuzzy logic with qualitative decision-making specifications [36]. In fuzzy logic, there is the concept of a fuzzy set that is described by the membership function. According to this function, all the members in a certain set are related to the numbers 0 and 1. Therefore, any element, x, in a set, A, has a value in a continuous interval [0:1]. The most famous applications of fuzzy logic are the design of

fuzzy rule-based systems where fuzzy IF-THEN rules are used. In these systems, fuzzy rules are used to represent control strategies, and previous generations and future generations also use fuzzy logic statements. Actually, a fuzzy model is a qualitative model made up with a set of fuzzy rules to characterize the relationship between the input and output of the system [37]. A fuzzy system model is applied to describe the relationship between the workload of the system and the resource demand from its input-output data even if the controller is completely unfamiliar with the target system by Talukder *et al.* [38]. This control system aims to achieve the desired application performance with minimal resource consumption. An event based control optimization formulation of the resource management problem was presented and a method to adaptively change desired system performance of the sensor network in response to events was discussed in this paper.

2.3.2.5 Model Predictive Control

Model predictive control is one of the most widely applied control approaches in industrial process systems, which has a predictor model to watch over a finite future period, and then the system will choose the best action that can both achieve target performance requirements and optimize a cost function under certain constraints [39]. Lu *et al.* [40] used this approach to control CPU utilization in distributed real-time systems. Like a constrained optimization problem, it requires an online solution and requires a lot of overhead in real-time systems.

2.3.2.6 Stochastic Control

Stochastic Control is a subcategory of control theory that aims at predicting and reducing the impact of the random deviations and disturbances of a dynamic system. Deviation of the system's performance from the system's desired course often happens when random noise and disturbances exist. A number of methods have been developed to reduce uncertainty and to improve control performances. Several applications about stochastic control of computing systems are carried out. Shalom *et al.* [41] focused on applying stochastic control theory to resource allocation under uncertainty. To transmit deadline-constrained data over time with various channels aiming at minimizing the total transmission energy consumption, an approach is developed using continuous time formulation and stochastic control theory by Zafer and Modiano [42].

2.3.2.7 Optimal Control

An optimal controller is widely used when the system is trying to find a control law and to meet some optimal requirements. The paths of the control variables that can minimize the functional cost are described by different equations. An optimal control policy is just a set of these equations. In order to maintain the schedulability of real-time systems and also the quality of service, a dynamic optimal control-model-based queuing theory is provided by Lu *et al.* [26]. With the performance cost criteria, optimal controllers can guarantee the control performance and also make up for delays in communication networks [45, 46]. Combining optimal control with stochastic control, this methodology could apply the linear quadratic Gaussian controller with quadratic cost [46].

2.4 Fault and Failure in Distributed Systems

The definitions of fault and failure are presented in the IEEE Standard Glossary of Software Engineering Terminology [47]. According to the IEEE, a fault is one of the following two situations: 1.) A defect in a hardware device or component; for example, a short circuit or broken wire; 2.) An incorrect step, process, or data definition in a computer program.

According to the IEEE, the definition of failure is the inability of a system or element to perform its desired functions with specific performance requirements. The fault tolerance discipline distinguishes between a human action (a mistake), its manifestation (a hardware or software fault), the result of the fault (a failure), and the amount by which the result is incorrect (the error). In a nutshell, mostly, a fault is the cause of failure and failure is the result of fault.

2.4.1 Classifications

Six different types of faults that may occur in distributed system are classified by Huda *et al.* [50]. These six groups are further divided into different sub classes are shown below.

Table 2.1 Fault classification in distributed systems

Fault Type	**Example**
Application and OS faults	Memory leak Resource unavailability
Hardware fault	Memory chips fault CPU fault Storage Disks faults
Network fault	Node failure Packet loss Corrupted packets
Response fault	Value fault Byzantine error
Software fault	Un-handled exception Unexpected input
Timeout fault	Faults caused by time out.

According to Ardanga *et al.,* failures in the DS management can be classified into five categories: functional failures, operational failures, semantic failures, privacy failures and security failures [49].

Table 2.2 Failures in DS management

Failure type	Cause of failure	Example
Functional failure	Software bug, hardware faults in the system	Computer hard drive crashed
Operational failure	System or some participant service is unavailable due to communication problem or unpredictable load	Heavy load in an air ticket booking DS make it unable to accept new requests
Semantic failure	Interacting operations between two participants are not compatible due to different ontology. Message exchanged between two services are incompatible.	Hotel reservation DS and car rental DS communicates between each other and does not have same time format.
Privacy failure	Service or data are inaccessible because they are privacy sensitive and DS would not disclose information to everybody.	Hotel reservation DS may not provide age of the customer to Car rental DS
Security failure	Data is accessed without enough credentials or authority, or without a special secure link.	The car rental Web Service only accepts SOAP messages over the secure HTTP, while the WSMS sends SOAP message over standard HTTP.

2.4.2 Memory Leak

A memory leak or leakage in computer science is a particular type of memory consumption by a computer program where the program is unable to release memory it has acquired. When a system does not properly manage its memory allocations, it is said to leak memory. A memory leak is a bug. Symptoms can include reduced performance and failure.

A memory leak can affect the performance of a computing system by reducing the amount of available memory space. Eventually, in the worst-case, part of the system or device may stop working properly because too much available memory is allocated. Sometimes, the system may slow down unacceptably or even crash suddenly.

Memory leaks may not be serious in common computing systems. In modern operating systems, the memory used by an application will be released when the application terminates, assuming that for programs that only run for a short time, memory leak errors may not be noticed and are rarely serious.

Memory leaks can be more serious in other cases:

1. When a program, such as background tasks on servers, runs for a very long time and consumes additional memory over time, the memory leak will be increasingly serious over time. For example, in embedded devices some background tasks may run for many years.
2. For systems where new memory is allocated frequently, a memory leak may be serious. For example, rendering the frames of a computer game or animated video requires allocation of new memory frequently.

Generally, it is more robust and convenient for developers to use automatic memory management, because they do not need to be concerned about the sequence when cleanup is performed or worry about whether or not an object is still referenced. However, automatic memory management may impose a performance overhead, and there is no guarantee that automatic memory management can detect all errors that cause memory leaks.

In the Java programming environment, the garbage collector is a typical automatic memory management tool. In the proposed system, the Java garbage collector plays an important role, which the following chapters will discuss in detail.

CHAPTER III

LITERATURE REVIEW: FAULT MANAGEMENT IN DS

This chapter reviews the fault management methods in distributed systems including various diagnostic methods, detection approaches, and fault management models.

3.1 Fault Diagnose and Detection

Fault detection has been studied for a long time in computing systems. There are some basic traditional techniques that have been frequently used to check whether or not physical machines or applications are still running, such as ping, heartbeat [51], and HTTP error code monitors [52]. Moreover, some methods based on statistical learning, which detect application-level service failures, are also widely applied. For example, Barham *et al.* [54] applied stochastic context free grammar to model the request's control flow among multiple machines aiming at detecting component failures and localizing performance bottlenecks as well. Idé and Kashima [53] presented an approach that treats a web-based system as a weighted graph and uses graph mining techniques to monitor the graph sequences for failure detection. The Pinpoint project mentioned by Chen *et al.* [55] proposed two statistical techniques that can be applied in the web-based systems with the purpose of analyzing the request path shapes and component interactions for detecting

failures. The project shown by Jiang *et al.* [56] applies a time series analysis to observe the dependency relationships among system variables for fault detection.

Kang *et al.* [57] proposed a tool for fault detection and diagnosis, especially suitable for virtualized consolidation systems. Because every application usually reveals itself in multiple instances in the data center, this tool extracts the correlated characteristics among multiple application instances by applying a statistical technique that is a canonical correlation analysis (CCA). According to the correlated characteristics, this tool is able to watch the system and detect the fault, then send back alarms. Compared to traditional fault detection techniques, the proposed tool is robust to system dynamics; therefore it can avoid a lot of false alarms.

Moreover, expert systems are widely applied in fault detection. Relevant recent research work includes Forbus and Falkenhainer [91], Oyeleye, Finch and Kramer [92], Dvorak and Kuipers [93]. In these knowledge-driven techniques, it is important for numeric computations to provide information for the final decision-making, though the governing elements are symbolic. Various combinations of knowledge-based techniques with numerical techniques have been proposed. Frank [70, 71] and Patton *et al.* [72] all considered effective combinations of both methods will become appropriate solutions that can be widely used in different situations.

Neural network techniques are also frequently used in fault detection. Recent researches on neural networks in online fault detection processes include Hoskins and Himmelblau [73], Kramer and Leonard [74], Zhang and Morris [75], Gan and Danai [76], etc. Some neural network techniques have obvious disadvantages that many scientists have noticed. For example, the back propagation network is recommended for online

fault diagnosis applications. It is widely known that one of the most obvious drawbacks for online fault detection is that high accuracy of measurements is needed for calculation of the evolution of faults. Fault detection usually takes measurements from instruments, which are possibly not sensitive enough or may have noisy data. Under these circumstances, the neural network will not be able to identify faults successfully. To improve the quality of identifying faults, systems usually do some pre-processing to data in order to let the meaningful parameters be presented to the system while the other noisy data does not appear to the system.

There are a number of research works on qualitative simulation for online fault detection, including qualitative reasoning by De Kleer and Brown [77], the qualitative process theory of Forbus [78] and a lot of research work in diagnostics by Weld and De Kleer [79], Herbert and Williams [80], *etc.* Compared to other methods, especially neural network methods, the main advantage of this approach is that resource-consuming functions are not needed.

In addition, scientists have also researched online expert systems. Recent online diagnostic systems research areas usually combine quantitative methods with qualitative methods for fault detection. This combination enables the evaluation of all available information and knowledge about the system for fault detection. Compared with traditional expert systems, the combination system has a database that contains information about the current state of the process. Applying online connections with sensors, signal analysis methods, model-based strategies and deep reasoning techniques, Angeli and Atherton [137] proposed an online expert system to detect faults in electro-hydraulic systems.

There has been much research on fault diagnosis in computing systems in recent years. Event correlation is probably the most frequently applied method [58,59]. A number of commercial tools have been designed to aid problem determination, like HP's OpenView [60] and IBM's Tivoli [61]. Basically, these techniques are based on either human-generated rules or some known dependency models about the system. However, because of the complexity and the frequently changing characteristics of the distributed system, it is not easy to construct a meaningful model for diagnosis. To solve these concerns, Cohen *et al.* [62] provided a simplified Bayesian network to model the dependency relationships of system attributes, and then achieved automatic correlation among variables and the service level agreement (SLA) violations. Aguilera *et al.* and Brown *et al.* adaptively construct the dependency knowledge for diagnosis based on dynamic observations [63,64]. In addition, Brodie *et al.* [65] presented an intelligent probing approach that can dynamically locate system failures with a Bayesian based inference technique.

Mirgorodskiy and Miller [66] proposed a diagnostic approach for distributed systems with self-propelled instrumentation. This approach can be divided into three steps. First, there is a novel execution monitoring technique that can inject a fragment of code into an application process dynamically. Second, an algorithm is used in the system that separates the trace into flows, which indicates user-meaningful activities. This step simplifies the manual examination and makes the automated analysis of the trace possible. The last step is automated root cause analysis that compares the flows to distinguish the anomalous flow and identify the specific function that probably caused the failure.

3.2 Fault Management in a Distributed System

For fault management in a distributed system, the key technique is making use of redundancy. By applying redundancy, only a relatively small number of components need to be duplicated. In the case of component failure, the redundant components can take over the responsibilities. Moreover, redundancy can also be used in the fault detection step through comparing results from each duplication.

There are two basic types of faults that may happen in distributed systems. The first type is a software bug or a hardware fault that occurs in centralized components, such as a storage node or management node. This type of failure could affect the entire system. The other type of failure is a crash of software on one of the computation nodes in the distributed system. The reasons causing these faults include a software bug in an application or a problem in the operating system local to the node. This type of failure causes the application on a certain node to no longer run properly while the other nodes can still work properly.

The standard technique for handling application failures is to check the system state periodically, and mark a checkpoint so that it can be used in case of a failure. In the case of a process failure, the application state is restored from the most recent set of checkpoints. There are a variety of protocols that have been developed to determine when processes should record checkpoints and to restore the application state. Replicating the functionality for a centralized system component is widely applied to achieve fault tolerance. Basically, there are two types of replication: active replication and passive application. For the first type, there is a backup machine, which will run exactly the same system as the primary node. Doubling the number of these nodes will double the cost of

the system, while it does not increase the capacity of the system [82, 83]. For the other type of replication, a backup machine always resides in an idle or powered off state and has all the primary system software. In case of a failure, the backup machine will replace the primary machine.

The most basic form of fault tolerance for parallel applications is combining checkpointing and rollback recovery. Rollback recovery techniques, is a kind of state preservation. To communicate over a network, rollback recovery techniques model a message-passing application sending messages among a fixed number of processes in a distributed system. There is an assumption that all the processes have access to some stable storage, which can still work properly even when a failure occurs. In the case of process failures, based on stored computational states, the application's process can be rolled back to the most recent checkpoint state where there is no fault [84].

In systems with virtual processors, each virtual processor is mapped to a physical processor, and each physical processor is responsible for one or more virtual processors. Without affecting the overall execution of the application, virtual processors can be transferred from one physical processor to another, which can also be used for fault tolerance. If a computation node happens to have failures, the virtual processors can be completely transferred to other nodes with the overall system working properly [85].

Fault tolerant solutions can be implemented in a number of different forms, including software libraries, special programming languages, compiler or preprocessor modifications, operating system extensions and system middleware. Each of these implementations has its own advantages and disadvantages in the forms of power, portability, and ease of use.

Fault management of Web Service (WS) is a relatively a new research field. Most of the presented work by scientists is still ongoing. Not much work with practical experimental results has been done on this topic.

He [49] proposed a structure to implement failure recovery capabilities in WS management systems including interface, main modules, supporting modules and web service. The proposed architecture is shown in Figure 3.1.

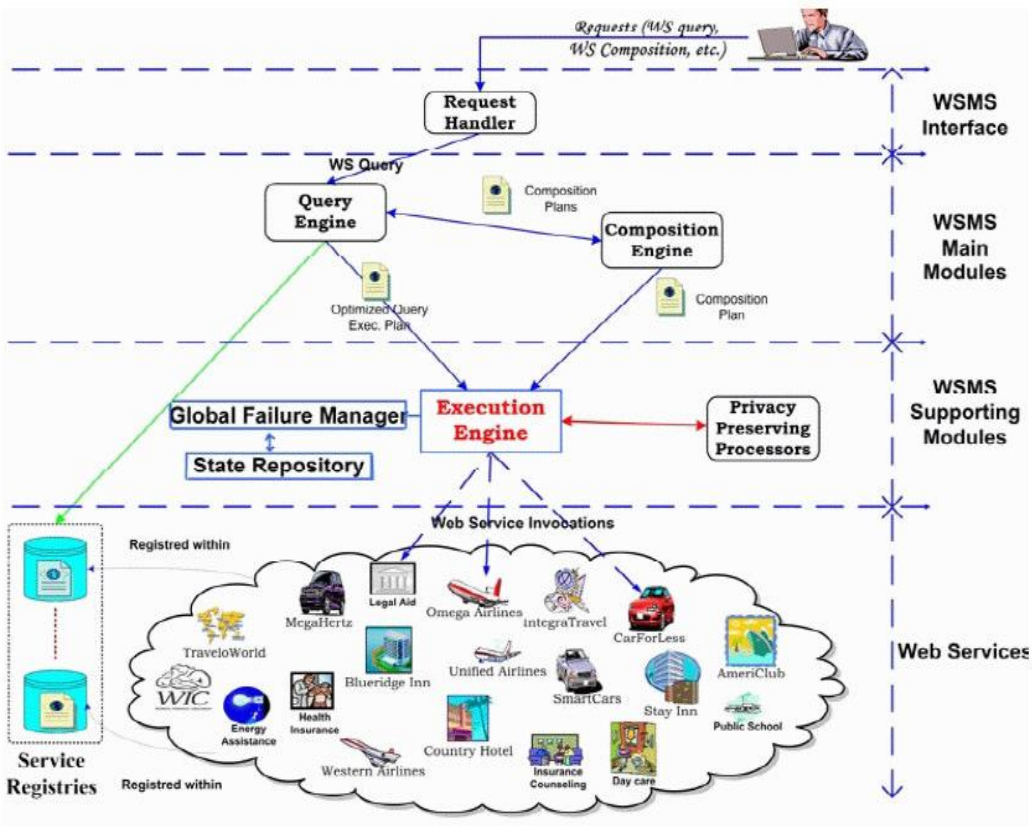

Figure 3.1 Reliable WS application structure [49]

In this proposed structure, the Request Handler receives all the requests from users. The Request Handler is also responsible for interpreting requests and checking

their syntactic and semantic validity. Then the composite service and query service make a connection between the Request Handler and execution engine. After that, the processed results from these two components come to the execution engine stage and are processed by it. At the same time, the privacy preserving processor takes care of maintaining all privacy requirements either from users, web services or data perspectives. Another very important function component in the system is the Global Failure Manager (GFM), which keeps a record of all executions and repairs of the system. The GFM is the main functioning part of fault management in this system, and it is responsible for monitoring faults, conducting failure diagnosis and recovering coordination.

Benharref *et al.* [86] provided an architecture that is based on web service for fault management. The main dominant component of the architecture is an observer controlled by a manager. Once the observer gets any fault or the observation period expires, it feeds the result back to the manager. Figure 3.2 shows the fault detection structure in this paper.

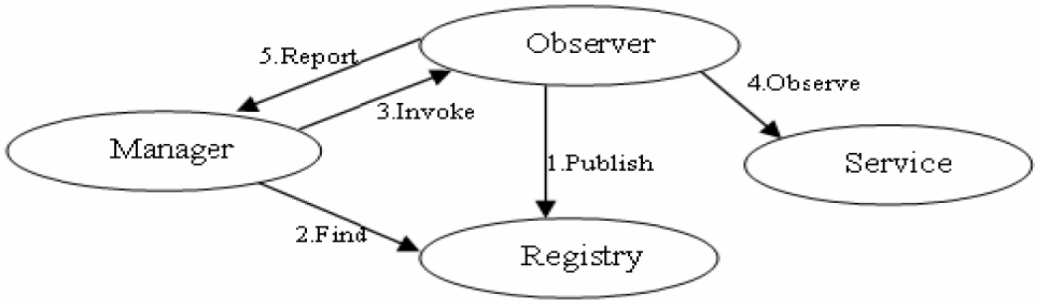

Figure 3.2 Fault detection structure [86]

Experimental results were not shown, though a complete structure was provided in this paper. In addition, when there is deadlock in the system, the system will not work. In contrast, the proposed approach will detect memory leaks and recover the target system from memory leaks successfully which can be validated by experimental results.

Mansouri *et al.* presents a functional monitoring model for distributed systems [50]. This monitoring model has a number of functions, such as generating monitoring information, processing the information, disseminating the information, and presenting the information. They only showed the monitoring model but no overall management solutions or any practical experiment results. Marcus and Stern [52] proposed an independent system architecture that is based on Remote Method Invocation (RMI) and JAVA for a distributed fault management system. However there are no actual experimental results about this architecture.

Kumar mentioned the best practices that can be used for distributed system management by web services [87]. He defined the best practices using different principles, including service orientation has to be properly thought through, design for evolution, write wrappers and handle failure conditions gracefully. He only showed a system on the theoretical level, and there are no actual experimental results provided.

A classification of faults in web services and a self-healing platform for the recovery of faults are presented by Ardanga *et al.* [48]. His work provides a self-healing platform that implements run-time service oriented fault analysis and recovery actions. However, they also only showed the platform structure on the theoretical level and no experimental data is included.

Dynamo has been proposed as a framework that provides a solution to extend the current Business Process Execution Language (BPEL) process with self-healing ability [88]. The use of supervision rules defines what functional expectations and non-functional expectations a process needs to meet during run-time execution. Reaction strategies are used to recover from erroneous situations. VieDAME presents a non-intrusive monitoring approach to monitor BPEL processes based on QoS, using Aspect-Oriented Programming (AOP) techniques [89]. Recovery is based on the placement of existing partner services. Subramanian *et al.* proposed a model extension to the BPEL engine to make it monitor and recover the parts other than composition [90]. The aim of this work is to handle functional failures of components during run-time in order to facilitate self-healing. Aghdaie and Tamir describe a client transparent fault tolerance model for Web servers in a distributed system [91]. It detects server errors and routes requests to a standby backup server for reducing service failures. One of the actions in our proposed LLC system is also forwarding the request to the backup server when errors happen to the main server. Our work realized similar fault tolerance performance using different models.

A fault tolerance model for grid services is proposed by Zhang *et al.* [92]. The model uses the passive replication technique. A QoS taxonomy for distributed systems is provided by Sabata *et al.* [93]. The taxonomy allows the QoS specification of the components in a distributed system from resources to applications. The passive replication technique is explored by Liang *et al.* [94]. To achieve fault tolerance, some modifications are proposed for the Web Service Definition Language (WSDL) and Simple Object Access Protocol (SOAP) standards with the purpose of allowing the

specification of Web service replicas and the redirection of service requests. Dialani *et al.* [95] discussed that distributed computing and grid computing applications could be designed with a fault tolerance to achieve robustness. It focused on error recovery and did not deal with the management of multiple service replicas. The active replication technique used in WS is provided by Santos *et al.* [96]. To support consumer transparent fault tolerance, the architecture has a central forwarding component that includes the mechanisms of managing replicas. To deal with the fault of the forwarding component, the architecture has a backup component.

As for the N-version model proposed by Looker *et al.* [97], in order to avoid errors caused by specification and implementation problems, different versions of replicas are developed. This implementation requires the consumer to obtain the WS status. WS-Replication that is provided by Salas *et al.* [98] achieves consistent replication of Web services by totally ordering all incoming requests to the replicated WS. WS-Replication uses separate proxy and dispatcher processes to capture and multicast clients' requests, receive multicast messages from JGroup and forward the requests to the replicated WS respectively. Thema reported a Byzantine fault tolerant framework for WS [99]. It is constructed on a consensus-based replication algorithm.

CHAPTER IV

MEMORY LEAK SIMULATION AND CHARACTERIZATION

Memory leak simulation and characterization is introduced in this chapter. First, the experimental environment is presented. In the second part of this chapter, the simulation of a memory leak is carried out with two different scenarios. Java garbage collection that has a significant influence on the simulation result is also discussed in this part. In the third part, the characterization implementation and relative results are proposed.

4.1 Introduction to the Implementation Environment

The Implementation environment in the target system is a Linux system with Xen Hypervisor, and the DS application used in this design is the DayTrader system.

DayTrader is a benchmark application originally developed by IBM and donated to the Apache Geronimo project in 2005. It is built around the paradigm of an online stock trading system aimed at serving as both a functional example of a full-stack J2EE 1.4 application and as a test bed for running performance tests. A core set of Java EE technologies are used in this application, including Java Servlets and JavaServer Pages (JSPs) for the presentation layer and Java database connectivity (JDBC), Java Message Service (JMS), Enterprise JavaBeans (EJBs) and Message-Driven Beans (MDBs) for the

back-end business logic and persistence layer. Figure 4.1 shows a high level overview of this application architecture constituted by Java technology blocks [99].

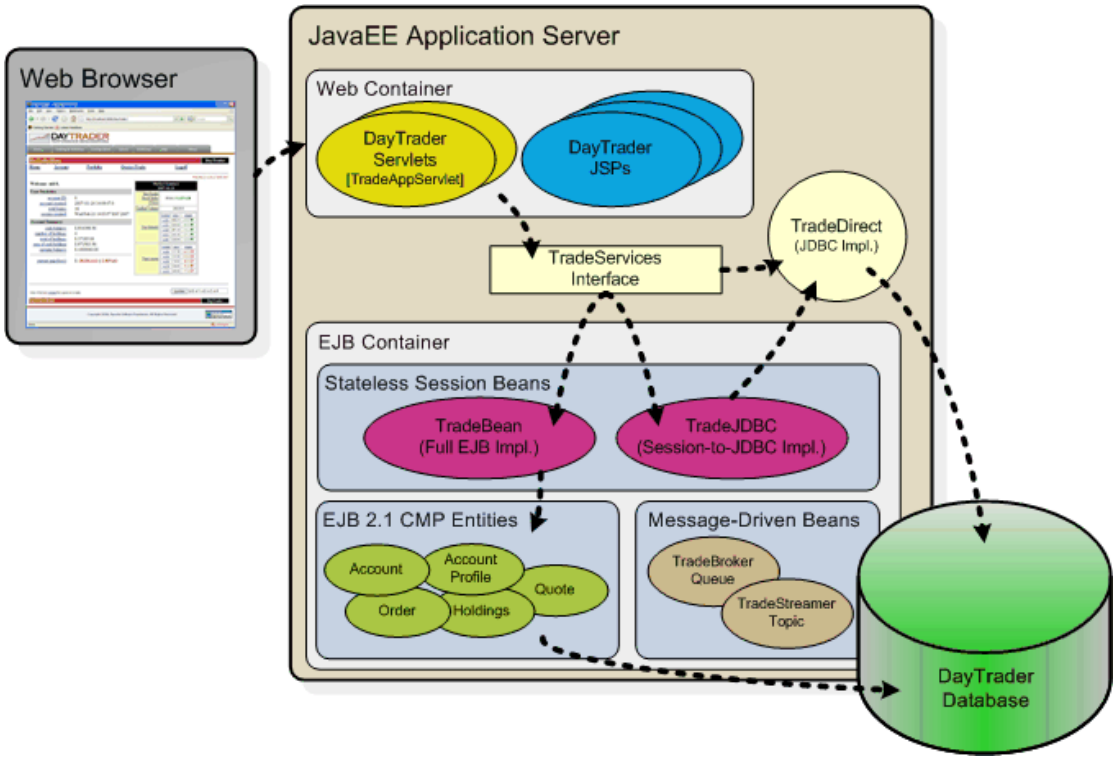

Figure 4.1 Overview of DayTrader [100]

In the above figure, a Model-View-Controller (MVC) design pattern holds several Java Servlets and JSPs. TradeAppServlet is the major controller servlet that is responsible for receiving incoming client requests, triggering the desired business logic, and forwarding responses to the appropriate Java Server Pages(JSP). In addition, some other servlets and JSPs are used to manage the supporting database and configure the DayTrader runtime options.

The business logic and persistence layer forms the bulk of the DayTrader application. The TradeServices interface provides a number of important business operations including register, login, get holdings, buy, complete order, logout, *etc.* Based on DayTrader, there are three different implementations of these services, corresponding to three commonly used JavaEE application design patterns. They are TradeDirect, TradeJDBC and TradeBean. These implementations can be easily switched changing the runtime mode on the configuration page.

The TradeDirect implementation performs CRUD (create, read, update, and delete) operations directly on the supporting database by applying custom JDBC code. Database connections, commitment, and rollbacks in the code can be manually controlled. As for the Trade JDBC stateless session bean, it serves as a wrapper for TradeDirect. The session bean assumes control of all transaction management while TradeDirect remains responsible for handling the JDBC operations and connections. This implementation shows the most commonly used JavaEE application design pattern.

Another crucial component of the persistence layer is the Java Messaging Service (JMS). JMS is responsible for two specific functions. First, JMS is used for asynchronously processing orders. Second, publishing quote price updates is also based on JMS [100].

In the proposed design, random request rates are used as client request rates, which refers to the number of requests sent from the client to the server every 30 seconds. The random request rates come from observations about an online trading system. Random request rates may change with time, and have a sharp increase during busy periods and reach the limit. During other periods, such as at night when clients rarely use

the server, request rates have an obvious decrease. The overall observation of a typical random request rate is shown in Figure 4.2, which comes from the workload characterization of the 1998 world cup web site [110].

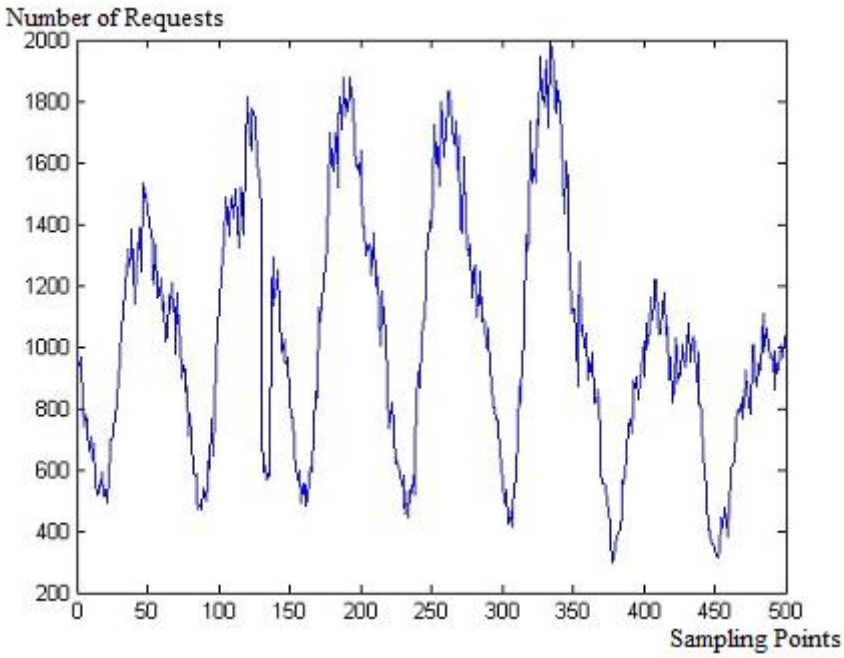

Figure 4.2 Typical random request rate

In the experimental system, the total memory of the system where the DayTrader application works is 1GB, and the platform is named nop06. The database is located at another machine named nop05, and the client is on nop08. The DayTrader application is also called the web server in this work.

4.2 Parallel Memory Leak Simulation Implementation

One of the advantages of using Java is that there is no need to carefully consider allocating and freeing memory. Programmers simply create objects, and Java is able to take care of freeing them when the program no longer requires the objects. This mechanism in Java programming is known as Garbage Collection (GC). However, GC will not reclaim a heap chunk if there are any references to it. If a program holds a reference to a heap chunk that will not be used during the rest of its life, this situation is considered a memory leak.

A memory leak was simulated successfully with the following Java code and this Java program ran in parallel with the DayTrader application on nop06 with a total memory of 1GB. This program is called Memory Leak (ML) in this work.

Table 4.1 Memory leak simulation code

```
import java.io.IOException;
import java.util.HashSet;
import java.util.Random;
import java.util.Vector;

public class LeakExample {
   static Vector myVector = new Vector();
   static HashSet pendingRequests = new HashSet();

   public void slowlyLeakingVector(int iter, int count) {
      for (int i=0; i<iter; i++) {
         for (int n=0; n<count; n++) {
            myVector.add(Integer.toString(n+i));
         }
         for (int n=count-1; n>98; n--)
            myVector.removeElementAt(n);
         }
      } //allocates 99 vectors for every 100 vector requests in every iteration.
     }

   public void leakingRequestLog(int iter) {
      Random requestQueue = new Random();
      for (int i=0; i<iter; i++) {
         int newRequest = requestQueue.nextInt();
         pendingRequests.add(new Integer(newRequest));
      }
     }   //all coming requests are kept in a hash table till it completed

   public void noLeak(int size) {
      HashSet tmpStore = new HashSet();
      for (int i=0; i<size; ++i) {
         String leakingUnit = new String("Object: " + i);
         tmpStore.add(leakingUnit);
      }   //leak most memory,but all gets garbage collected
  }

   public static void main(String[] args) throws IOException {
      LeakExample javaLeaks = new LeakExample();
      for (int i=0; true; i++) {
         try { // sleep to slow down leaking process
            Thread.sleep(1000);
         } catch (InterruptedException e) { /* do nothing */ }
         System.out.println("Iteration: " + i);
         javaLeaks.slowlyLeakingVector(1000,100);
         javaLeaks.leakingRequestLog(20000);
         javaLeaks.noLeak(100000);
      }
   }
}
```

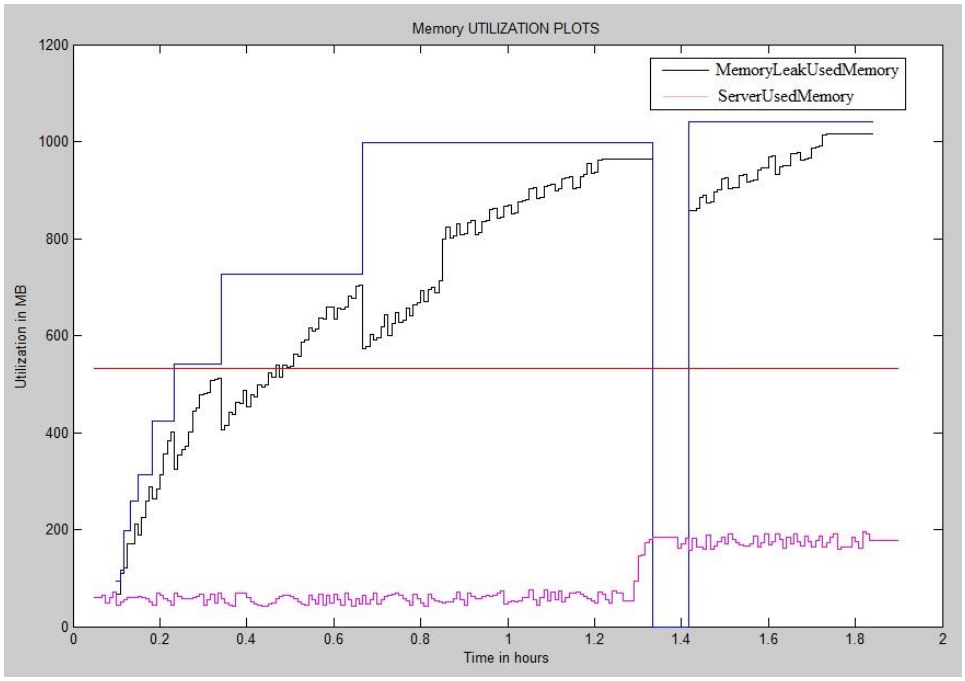

Figure 4.3 Memory utilization of both Memory Leak and web server

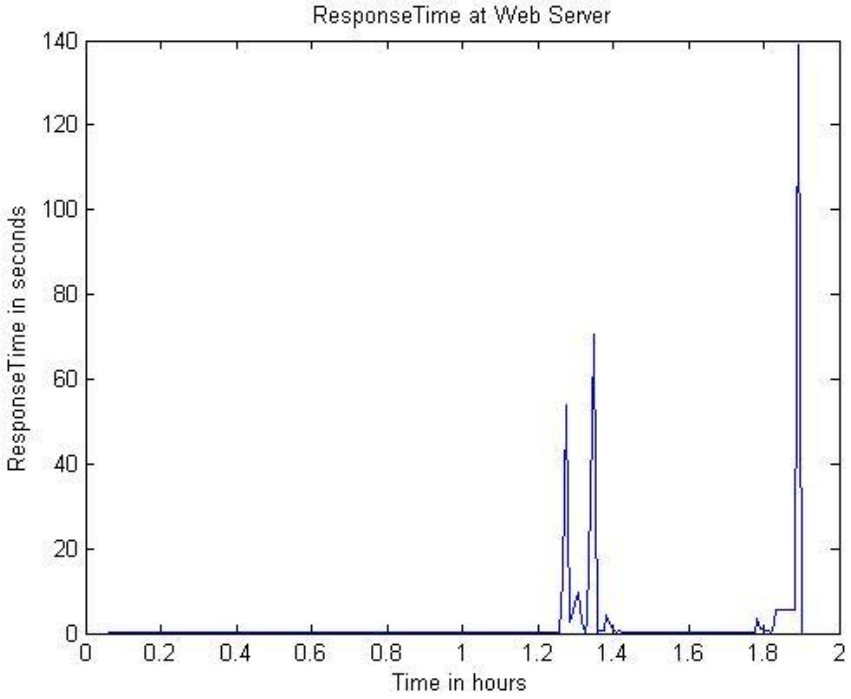

Figure 4.4 Response Time on the Web Service

Figure 4.3 shows the web server working properly for the first 1.3 hours with memory consumption of around 60 MB. After 1.3 hours, the memory utilization of the web server had a sudden increase and remained at this high memory consumption of around 200 MB. The ML kept taking memory after it started running. After 1.3 hours, the total memory consumption of ML and the web server reached the system limit of 1GB. This point is called the memory collision point in this work, and the ML crashed at this point. Moreover, as shown in Figure 4.4, the response time of the web server stayed at a very small value until the memory collision happened. Because of this collision, the response time of web server increased to a high value of over 200 seconds, indicating that the web server processed requests from clients at a slower speed compared to the normal working state.

The following graphs show the queue level and the error rate of the web server. The queue level refers to the size of the queue holding the requests from clients that were ready to be processed. This parameter is highly correlated with the response time of the web server. When the server needs more time to process each request, the queue where the waiting requests stay also increases. The error rate presents the number of requests that is missed by the server for every sampling period.

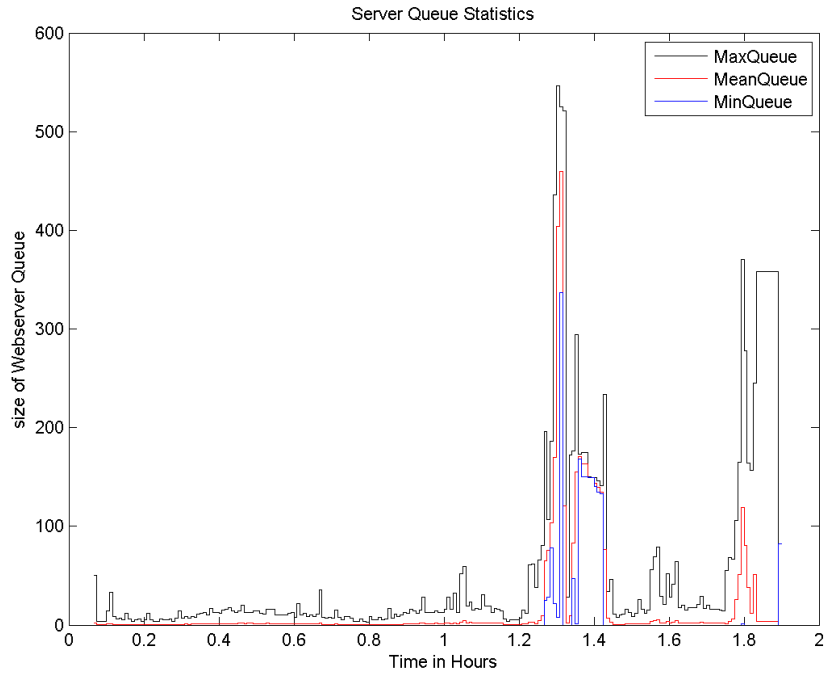

Figure 4.5 Queue level of web server

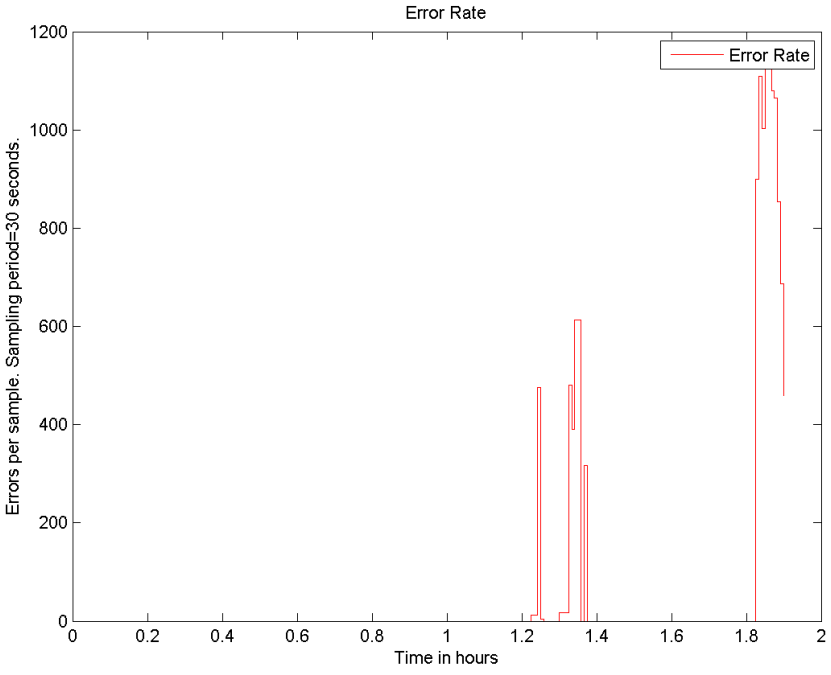

Figure 4.6 Error rate

The impact of a memory leak is also reflected from the server queue and the error rate as shown in Figure 4.5 and Figure 4.6. The memory collision made it difficult for the web server to quickly process requests. Therefore, these two parameters increased to high values when the memory collision happened.

4.3 Simulation of a Memory Leak inside the Web Server

Garbage collection can help the Java application to control memory. Considering the possibility that the garbage collector may not run during an application's lifetime, there is no guarantee as to when or if the JVM will invoke the garbage collector even if a program explicitly calls System.gc(). Actually, the garbage collector does not automatically run until a program needs more than the current available memory from the system. At this point, the JVM will first attempt to make more memory available by invoking the garbage collector.

DayTrader has a function called singleton that all requests have to go through and record logs. A Memory leak code integrated into this function makes it possible to simulate a memory leak inside DayTrader. In this simulation, DayTrader will leak an amount of memory when each request is processed. The strength of the memory leak can be adjusted. In this proposed system, we use index: 100, 200, 300, 400, 500, 600 to represent the exact leak rates of 3MB/min, 6MB/min, 9MB/min, 12MB/min, 15MB/min and 18MB/min respectively.

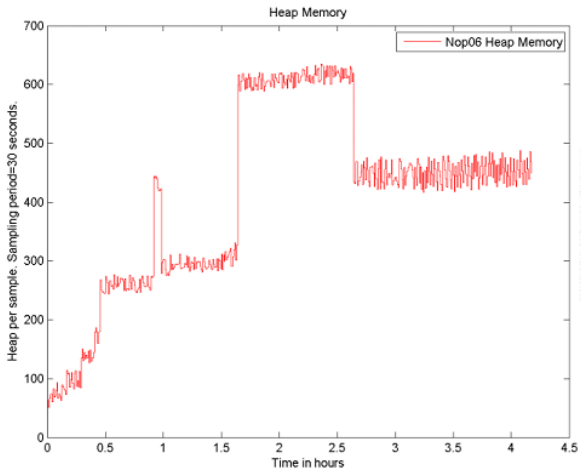

Figure 4.7 Memory utilization with rate 200

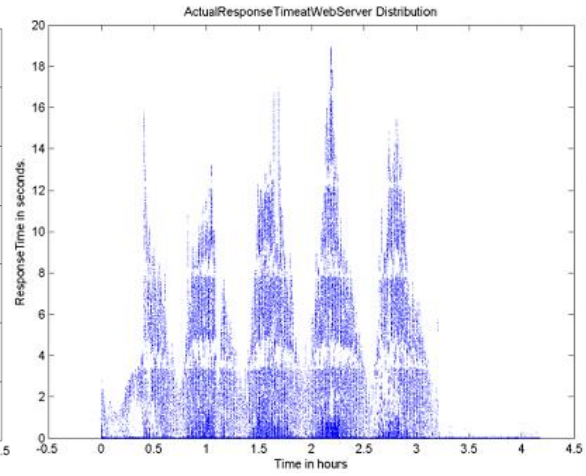

Figure 4.8 Response time with rate 200

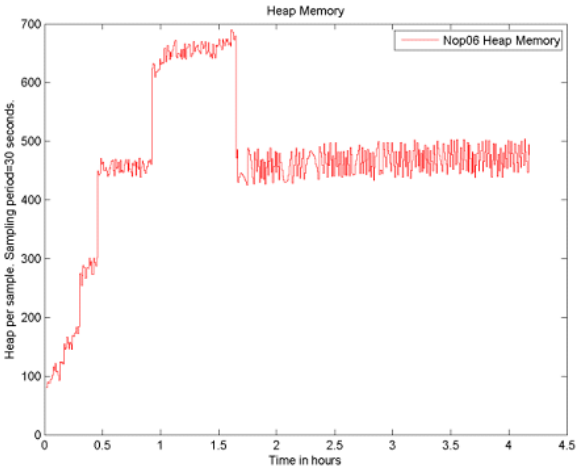

Figure 4.9 Memory utilization with rate 400

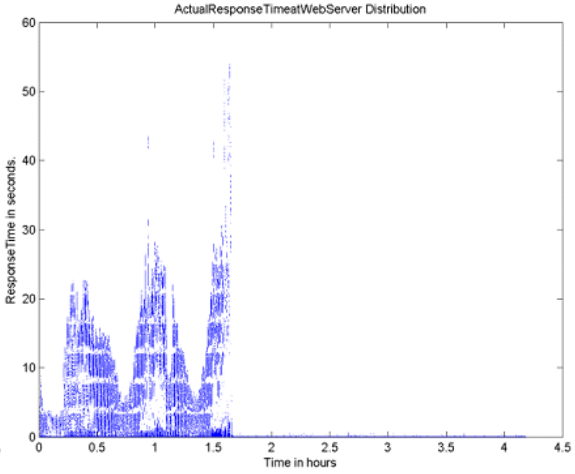

Figure 4.10 Response Time with rate 400

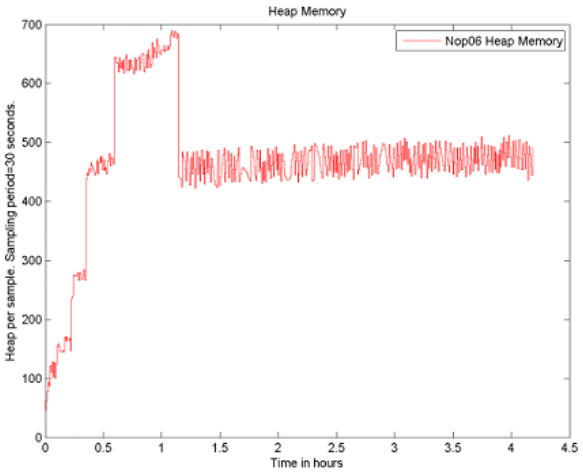

Figure 4.11 Memory utilization with rate 600

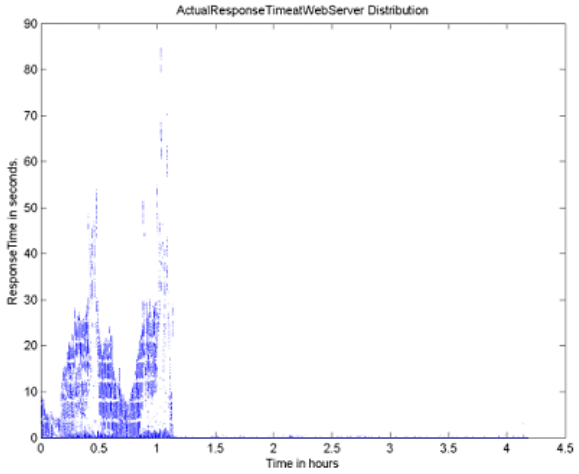

Figure 4.12 Response Time with rate 600

Three sets of simulations are carried out with memory leak strengths of 200, 400, and 600 respectively. The above graphs show memory utilization and response time in these experiments.

Comparing memory utilization graphs, three plots have similar shapes that memory utilization keeps increasing to a very high value around 650 MB and then goes back to around 500 MB. The point where memory utilization reaches the summit and goes down is called the cutoff point in this work. The time periods for the memory utilizations to reach the cutoff points are different and correlate with the rates of the memory leak. When the leak strength is adjusted to 200, 400, 600, it takes 2.5 hours, 1.7 hours, and 1.1 hours to reach cutoff points respectively. Therefore, the slopes of theses memory utilization plots are highly correlated with the leak strength.

According to the response time plots, they all remain at very high values from the beginning of the experiments and lasted for 3.3, 1.7 and 1.1 hours corresponding to leak strengths of 200, 400 and 600 respectively. The average of these high value periods are

10 seconds, 20 seconds, and 40 seconds for the leak strengths of 200, 400, and 600 respectively. Therefore, the average response time is also highly correlated with the strength of the memory leak.

4.4 Garbage Collector Analysis

This section introduces the Java garbage collector from the basic framework and algorithm to the characteristics of the garbage collector. Moreover, the impact of the garbage collector in the experimental system is discussed.

4.4.1 Garbage Collector Introduction

A Java program creates objects and stores them in the JVM's heap. Java's "new" operator creates objects, and at the same time memory for new objects is allocated to the heap. Garbage collection is the process that automatically frees objects no longer referenced by the program. The objects that are no longer needed by the program are viewed as "garbage" and can be collected. Therefore, the garbage collector must somehow determine which objects are no longer referenced by the program and free the occupied space to provide available heap space.

Garbage collection frees programmers from the responsibility of freeing allocated memory. It is difficult to know when to explicitly free allocated memory. JVM has an advantage in this job over programmers in that it helps ensure program integrity. Garbage collection is an important part of Java's security strategy. Java programmers are unable to crash the JVM by incorrectly allocating memory [109].

A potential disadvantage of a garbage-collected heap is that it may add overhead, which will affect the performance of the Java program. The JVM is responsible for keeping track of which objects are being referenced by the current program, and it will detect and free unreferenced objects in the run time. This activity will probably require more CPU time than CPU consumption in the situation that the program explicitly freed unnecessary memory. In addition, programmers working with the garbage collector have less control over the scheduling of CPU time devoted to freeing those no longer needed objects.

Nowadays, very good garbage collection algorithms have been developed, and adequate performance can be achieved for most applications. Because Java's garbage collector runs on its own thread, it will run transparently alongside the execution of the current Java program. Moreover, if programmers want to explicitly call a garbage collection at some point, System.gc() or Runtime.gc() can be requested.

Another important characteristic for garbage collectors is that they run finalizers on objects. Generally, it is not possible to predict exactly when unreferenced objects can be garbage collected. It is also not possible to predict when object finalizers will be run.

A lot of work has been done in the area of garbage collection algorithms. A number of different techniques have been developed that could be applied to a JVM. Any garbage collection algorithm has two basic responsibilities. First, it must detect garbage objects. Second, it has to reclaim the heap space used by the objects that is no longer needed and make it available for later use by the system. Typically, defining a set of roots and determining reachabilities from these roots accomplishes garbage detection. An object is regarded as reachable if there is a path from the roots by which the executing

program can access the object. The roots are always accessible to the program. The objects that are reachable from the roots are considered live. The objects that are not reachable are regarded as garbage and will be collected by the garbage collector.

Usually, there are two basic approaches to distinguish live objects from garbage; reference counting and tracing. Reference counting garbage collectors are able to distinguish live objects from garbage objects by keeping a count for each object on the heap and tracking the number of references to that object. As for the second type, tracing garbage collectors actually trace out the graph of references starting with the root nodes. For those objects that are encountered during the trace, they are marked as live objects by the garbage collector. Therefore, after the trace is complete, unmarked objects are considered to be "garbage" and will be collected [109].

4.4.2 Impact of the Garbage Collector

From the previous analysis of the memory utilization graphs (Figure 4.7, Figure 4.8, Figure 4.9) in the simulation experiments, memory utilization by the web server dropped to around 500 MB after reaching the peak memory consumption. In this Java based experimental environment, the garbage collector keeps monitoring the running processes to collect memory space that is no longer needed. The reason for dropping memory utilization after reaching its peak could be garbage collection in the experimental system. However, there is no guarantee that the garbage collector will work on the memory leak, and when it will start working is also unknown to the system users.

Moreover, comparing these memory utilization plots, as the strength of the memory leak increases, the time when the garbage collector starts working is earlier. In

the experimental system, the garbage collector seems to always start working after the memory utilization reaches 600MB. The exact value of this amount is unknown to the system users.

4.5 Characterization Implementation

This section discusses the characterization of memory leaks as identified from simulation results. With the characterization results, the LLC can reason about and compute the appropriate response to the memory leak. According to the analysis in sections 4.2 and 4.3, memory utilization and response time are highly correlated with the strength of the memory leak. Therefore, these two parameters are used for characterization analysis.

A low pass filter is used to smooth the fluctuation of the memory utilization plots. To extract useful information for the following prediction step, a linear fit is carried out on results that are processed by the low pass filter. Then, a two-step prediction method is applied to determine the existence of the memory leak and predict the strength of the memory leak. The overall framework is shown as follows. The function part of this characterization is called Memory Monitor in this proposed system. As shown in Figure 4.13, the Memory Monitor part includes both a low pass filter and a linear fitting function.

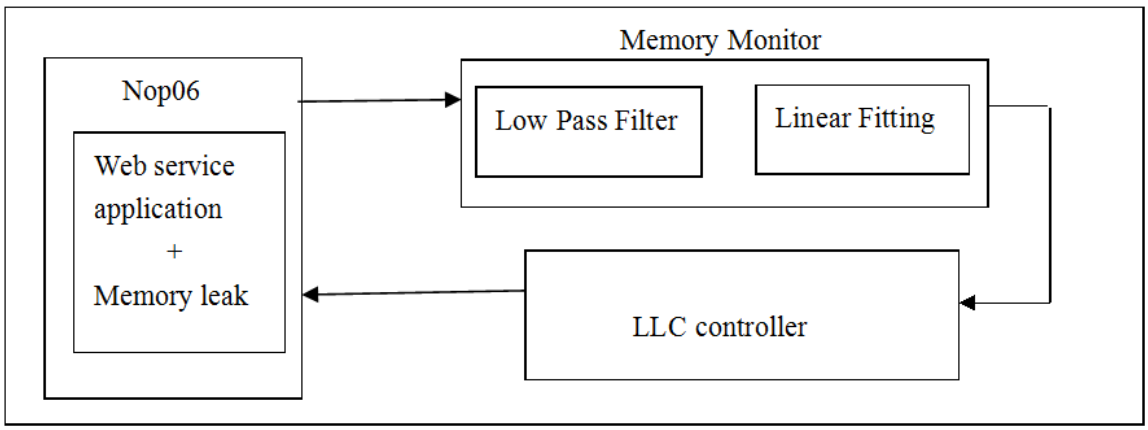

Figure 4.13 The framework of characterization

4.5.1 Low Pass Filter

To construct a proper low pass filter, several relative parameters have to be determined such as the stop band and the order of the low pass filter. Therefore, a FFT (Fast Fourier Transform) is applied to the process memory utilization dataset with MATLAB. The following graphs(Figure 4.14, Figure 4.15, Figure 4.16) show the results of FFT overa few memory utilization datasets in the experiments.

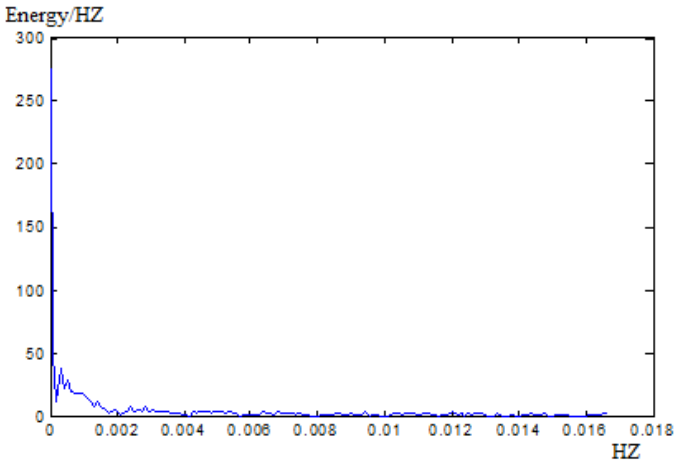

Figure 4.14 FFT for heap memory--memory leak100

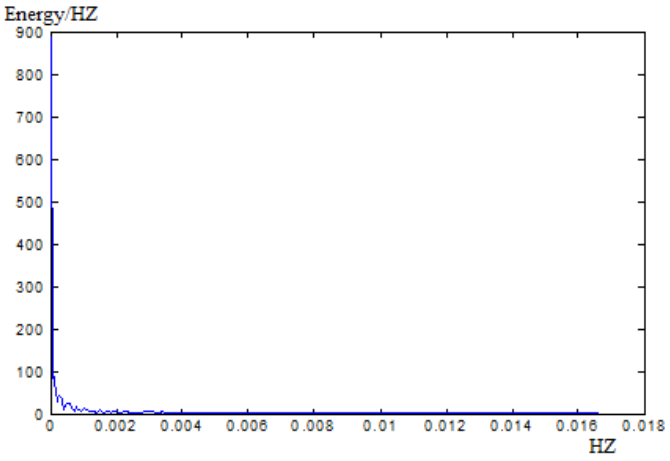

Figure 4.15 FFT for heap memory--memory leak 200

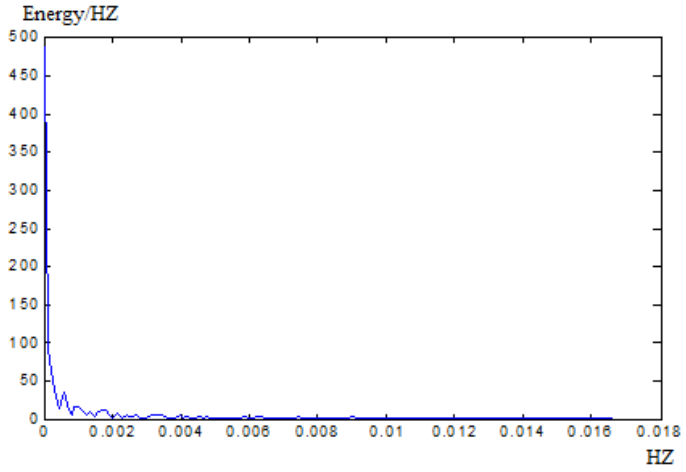

Figure 4.16 FFT for heap memory--memory leak 400

According to the above FFT results of the memory utilization datasets, the stop band is chosen at point 0.0002, and the order of the low pass filter was selected to be 2. With the stop band and order decided, a Butterworth low pass filter is constructed by applying the command [b, a]=butter(n,Wn,'ftype') in Matlab. Then, relative parameters, a and b, are calculated by Matlab.

By applying the appropriate parameters, the low pass filter is constructed in Java. After the processing of this low pass filter, all the heap memory plots in the experiments become smooth curves. The following graphs are the memory utilization plots after using the low pass filter for different strengths of memory leaks. According to the analysis in section 4.3, as the memory leak strength increases, the slope in the corresponding memory utilization plots increase. Therefore the slope of the heap memory graphs after the LPF (low pass filter) is a good parameter for characterization.

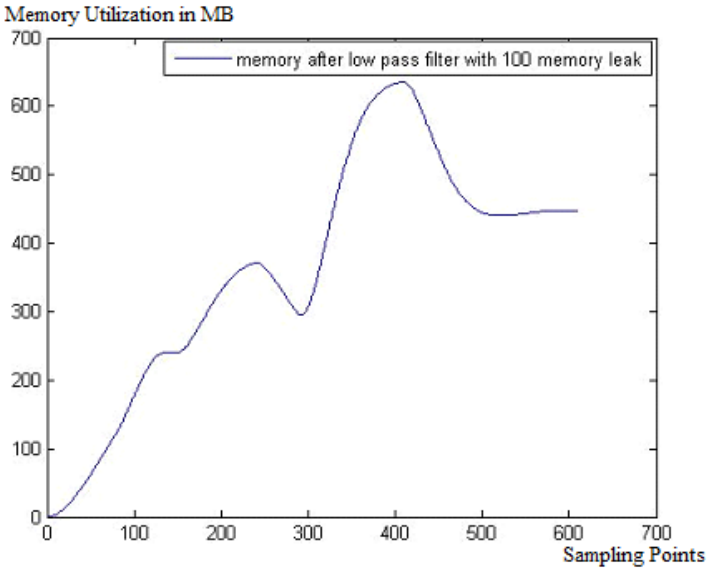

Figure 4.17 Memory utilization after LPF with 100 memory leak

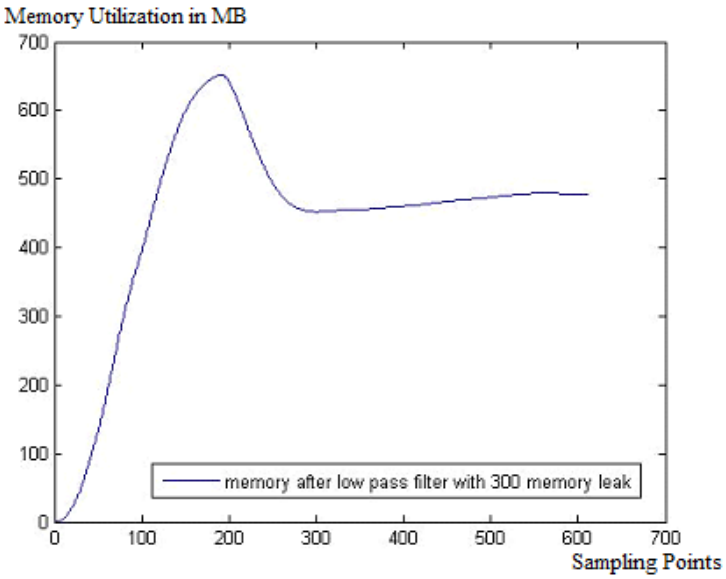

Figure 4.18 Memory utilization after LPF with 300 memory leak

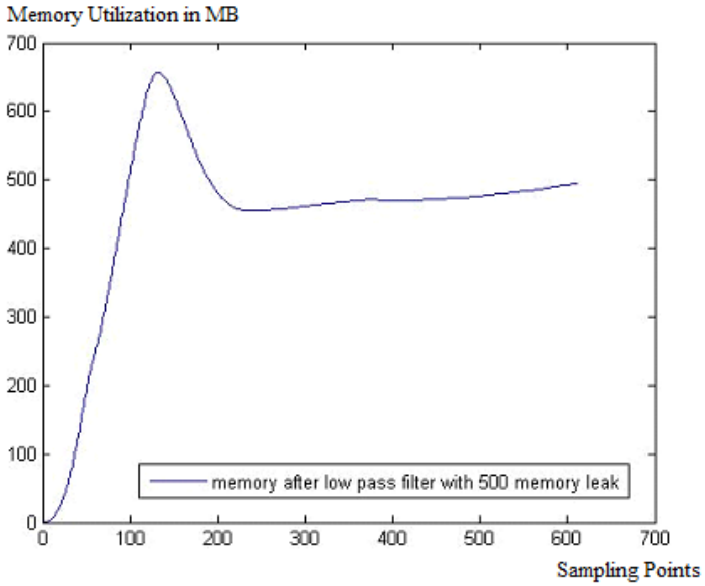

Figure 4.19 Memory utilization after LPF with 500 memory leak

From the above set of graphs, when the memory leak strength is larger than 300, the plot of memory utilization after the LPF becomes smooth curves without obvious

fluctuations. In this case, the strength of the memory leak can be predicted directly based on slopes. However, for the small memory leak strength cases, such as the 100 memory leak strength case, the first peak in the memory utilization after the LPF plot (Figure 4.17) only indicates the existence of a memory leak in the system. In order to accurately predict the exact memory leak strength, in the cases of small strength, another slope has to be taken after the memory utilization plot becomes more stable and complete.

4.5.2 Linear Fitting

When the web server is working, data about the current memory consumption by the web server is added to the memory utilization dataset at every sampling interval, which is every 30 seconds in the proposed design. Therefore, the memory utilization dataset is updated every 30 seconds in the target system. This memory utilization dataset, which is also called the heap memory dataset, contains all the data about memory consumption until the current working state.

In this work, the Matlab linear fitting function polyfit is applied to take slopes on smooth memory utilization plots that have been processed by the low pass filter. In order to have the most updated slopes of the memory leak plots, the memory leak utilization plots have to be processed by linear fitting every 30 seconds, because the heap memory dataset is updated every 30 seconds. Also, the low-pass filter is required to process the heap memory dataset every 30 seconds before linear fitting is carried out.

Based on the algorithm of the Matlab function polyfit, a corresponding Java program that has the same function as the polyfit command is made to process the low pass filter results in the target environment that is Java based. At this point, the slopes of

the heap memory dataset graphs can be calculated every 30 seconds. The differences between the current slope and the previous slope is called delta in this system.

The analysis of the characterization results is out carried in two steps. The first step is monitoring the slope and delta in the plots of memory utilization after the LPF. When the delta value becomes zero that refers to the first peak point in the plots of the memory utilization after LPF (Figure 4.16, Figure 4.17, Figure 4.18); Memory Monitor outputs the existence of a memory leak. The second step is to continue watching the slopes and deltas. When the memory utilization reaches 400 MB, Memory Monitor will output the prediction result about the strength of the memory leak and send it to the LLC.

4.5.3 Prediction of Memory Leak Strength

The implementation of characterization and the analysis of characterization results are provided in the previous sections (4.4.1, 4.4.2). With the characterization results, the strength of the memory leak can be predicted through analysis of the response time. From the results of memory leak simulation experiments, the average response time for each request with different memory leak strengths are shown in the following table. A linear relationship can be calculated.

Table 4.2 Average response time

Memory Leak Intensity	Response Time (millisecond)
100	7.932
200	8.759
300	9.869
400	10.434
500	11.295
600	12.075

By applying linear fitting over the average response time data (Table 4.2), the prediction function is constructed as follows, where R refers to the average response time in milliseconds.

$$Strength = 118.8342 * R - 838.9462 \quad (4.1)$$

The experimental results show the memory leak strength can be accurately predicted by this function of Memory Monitor. Then, the characterization results and the prediction result about the memory leak strength are sent to the LLC for further evaluation regarding CPU frequency adjustment and recovery actions.

CHAPTER V

FAULT ADAPTIVE CONTROL DESIGN

5.1 LLC background

This section introduces the developed fault adaptive Limited Lookahead Controller (LLC) and the underlying design, algorithm and characteristics.

5.1.1 Hybrid System Model

The proposed limited lookahead control is designed for a class of hybrid system with a finite control input set. This type of system is described as below with a discrete time state-space equation.

$$x(k+1) = f(x(k), u(k), \omega(k)) \tag{5.1}$$

Where k is time index, $\omega(k) \subset \Omega \subset R^r$ $x(k) \subset X \subset R^n$ and are sampled forms of environmental parameters, and the continuous system state at time k, and $\mu(k) \subset U \subset R^m$ represents the control input. X denotes the system state space; U denotes the input set, and Ω denotes the environment input. We assume U to be finite and X to be continuous and compact. X is referred to as the system's operating domain. The model, f, describes the relationships among system parameters, especially with the ones relating the QoS specifications to the control inputs. The above model can capture the

dynamics of many computing systems since they typically have a limited finite set of tuning options.

When operating in dynamic and open environments, the computer system's inputs to the controllers are created by external uncontrolled sources. It has been observed that most e-commerce workloads of interest have strong time-of-day variations [102, 103, 104]. Among these variations, some of the crucial workload characteristics like request arrival rates can have considerable changes in a few minutes. Most of the time, the variations can be well predicted with forecasting methods such as Kalman filters [105], and as the Box-Jenkins ARIMA approach [106]. Through analyzing and simulating relevant parameters of the underlying system environment, a forecasting model is generated with system input.

$$\hat{\omega}(k) = \phi(\underline{W}(k-1, m), p(k)) \tag{5.2}$$

Where $\hat{\omega}(k)$ is the estimated value, $\underline{w}(k-1, m)$ is the set of observed environmental vectors $\omega(k-1)$, ... , $\omega(k-m-1)$, and $p(k) \subset R^p$ represents related estimation parameters. We can obtain the parameters by training the model with test data that represents real field situations. We measure that ϕ is differentiable among every argument. The estimation parameters are assumed to be constant to simplify the model and keep generality. It is assumed that ϕ is not periodically re-tuned and $m = 1$. Therefore, $\hat{\omega}(k) = (\omega(k-1), e(k))$, and $e(k)$ reflects the estimation error effect, and it is a bounded random variable. In the proposed design, parameter $e(k)$ is not required.

Since the current value of $\omega(k)$ needs to be measured until the next sampling instant, models with uncertain parameters can capture the system dynamics using the following equation in which $e(k)$ is the only uncertain parameter.

$$x(k+1) = f(x(k), u(k), \omega(k)) = f(x(k), u(k), \phi(\omega(k-1), e(k))) \tag{5.3}$$

In the proposed design, ω denotes the request arrival rate, R, at the server, which is estimated by a simple ARMA model.

$$R(k+1) = R(k-1) * 0.2 + R(k) * 0.8 \tag{5.4}$$

The CPU utilization of the target system in this work is also predicted with this approach by the following equation.

$$U(k+1) = U(k) * R(k+1) / R(k) \tag{5.5}$$

5.1.2 QoS Specifications

Generally, specific QoS objectives can only be achieved by computing systems if they can satisfy the determined operating constraints. QoS specifications are mainly divided into two groups. The first group is the set point specification, which means that key operating parameters should be maintained at a specific level. There are several examples for this group, including response time, system utilization levels, *etc*. Therefore, the controller is designed with the purpose of driving the system to a state close to the target state $x^* \in X$, and this driving process should be completed within infinite time, and the system will be maintained there at the target state. The second group is the performance specification in which relevant measures such as mode switching and power consumption have to be minimized.

Transient costs may also be taken into account as a part of the requirements for operation, indicating the fact that we prefer those trajectories towards the target state than others due to their cost or utility for the system. Sometimes, the cost from control inputs should probably also be considered by these performance measures.

The main purpose of applying the LLC controller is to make the computing system achieve the desired state x^* in a finite time period. Another important objective of the controller is to minimize the function for transient cost $J'(x,u)$ in the process of pushing the system toward x^*.

5.2 Controller Design

The structure of an online controller is shown in Figure 5.1. The system model predicts future operating states with a look-ahead horizon by estimating relevant operating environmental parameters, like patterns of arrival workload. The controller tries to optimize the forecast behavior through choosing the best control inputs for the system [107]. The look-ahead controller can be viewed as a function that maps QoS objectives to a set of control actions. The controller constructs a sequence of possible potential operating states up to a certain prediction horizon, N, based on the current state. Then, it chooses the best trajectory within the specified horizon, which minimize the cost function and at the same time satisfy the constraints of both state and input. After the evaluation, the control input chosen from the sequence as the best control action will be applied as the next control action to the system. This process repeats at each time instance.

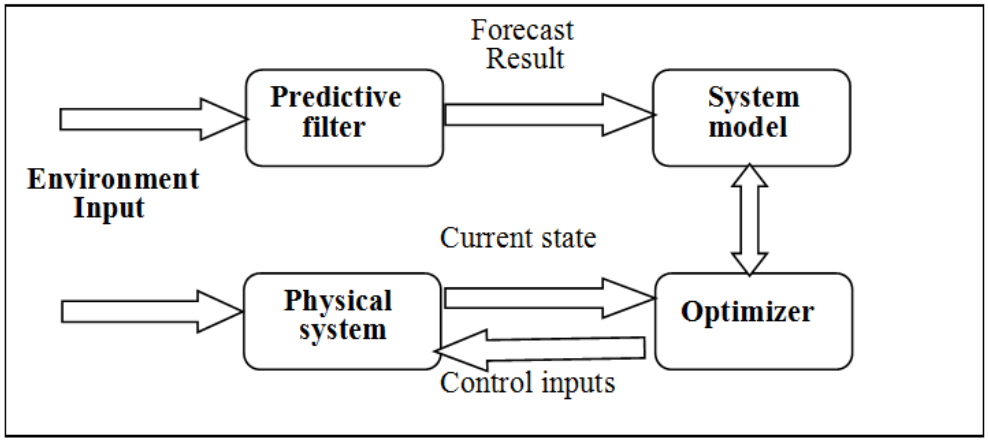

Figure 5.1 Online controller overall structure

There are some key ideas about the design of this controller. First, for future system states, $\hat{x}(k+j)$, within the prediction horizon, there are $j=1...N$ steps that are evaluated at each time instant, k, applying the behavioral model. When these predictions are made, the controller has to take both the current state measurement and future control signals into account. Second, a set of control signals $u(k+j)$ is calculated at each step within the prediction horizon through optimizing the specification about QoS. The only control input applied to the system at time instant k is the $u^*(k)$, which corresponds to the first input in the sequence, at the same time all the other control inputs in the sequence are discarded. For the next time instant, with $x(k+1)$ known, the whole process will begin again, though the predicted state at time k may not be the same as the actual observed state.

The control specification used in this system is the set point specification in which those important operating parameters should be maintained at a specified level. Therefore, the controller has the purpose of driving the system to a state close to the

desired set-point state $x^* \in X$ in an acceptable time, and the system will be maintained there. As shown in Figure 5.2, the next control action is chosen according to a distance map, $D(x) = \| x - x^* \|$, indicating the distance from the current state to the desired set point in the LLC approach. In this work, the distance map is the utility function J.

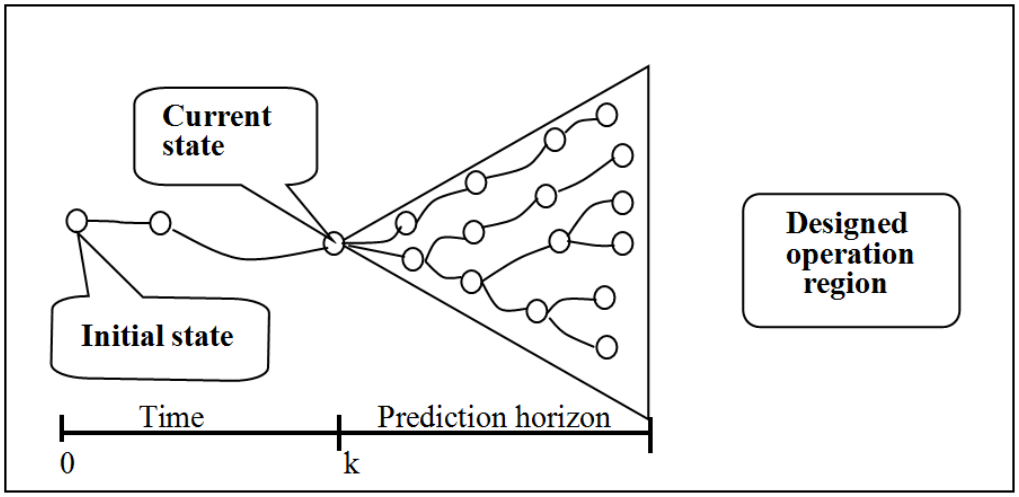

Figure 5.2 Distance map

5.3 Controller Algorithm

Table 5.1 shows the algorithm of the online control. It processes the input of the current state, $x(k)$, at each time instant, k, and returns the control input, $u^*(k)$, that minimizes the distance function. The controller constructs a tree containing all the possible future states to a certain prediction depth. For example, given the current $x(k)$, the algorithm will generate all the possible future system states using all control inputs from the valid current set, U. Before generating the states, we can estimate the

parameters of the operating environment. After generating the states, the cost function, which regards each estimated state, will be computed. When the prediction horizon exploration finishes, the system will generate a sequence of all states that can be reached $\hat{x}(k+1),...,\hat{x}(k+N)$ along each path in the tree. The optimal path will then be selected based on the utility function (distance map) and the first input along this path will be selected as the optimal input at time k.

The LLC algorithm chooses inputs from discrete values in a computational system and evaluates all the possible paths exhaustively to choose the best control input optimizing the system performance according to the utility function. Therefore, as the number of inputs increases, the search tree on which the algorithm evaluates all the operating states grows exponentially. Given U denoting the input set size and N indicating the prediction depth, the number of possible states to be explored is $\sum_{j-1}^{N} |U|^{j}$. This is not a major concern for those systems that only have a few control options. On the other hand, for the situation that a system has a lot of control input options, the corresponding control overhead will not be suitable for real-time application.

Table 5.1 The LLC Algorithm

1 **OLC**(x(k)) /* x(k) := current state measurement */
2 S_k := x(k); Cost(x(k)) = 0
3 **for all** k within prediction horizon of depth N **do**
4 Forecast environment parameters for time k + 1
5 $S_{k+1} := \phi$
6 **for all** $x \in S_k$ **do**
7 **for all** $u \in U$ **do**
8 $\hat{x} = \Phi(x,u)$ /* Estimate state at time k + 1 */
9 Cost($\hat{x}$) = Cost(x) + J($\hat{x}$)
10 $S_{k+1} := S_{k+1} \cup \hat{x}$
11 **end for**
12 **end for**
13 k := k + 1
14 **end for**
15 Find $x_{\min} \in S_N$ having minimum Cost(x)
16 $u^*(k)$:= initial input leading from x(k) to $x_{\min}$
17 **return** $u^*(k)$

5.4 Fault management using LLC

Using the system configuration described earlier in Chapter IV, the LLC is implemented on nop03, with nop06 working as the main system and nop09 working as the backup system. The LLC in this system not only responds to memory leak faults, but also adjusts the CPU frequency according to current the workload to optimize the system performance.

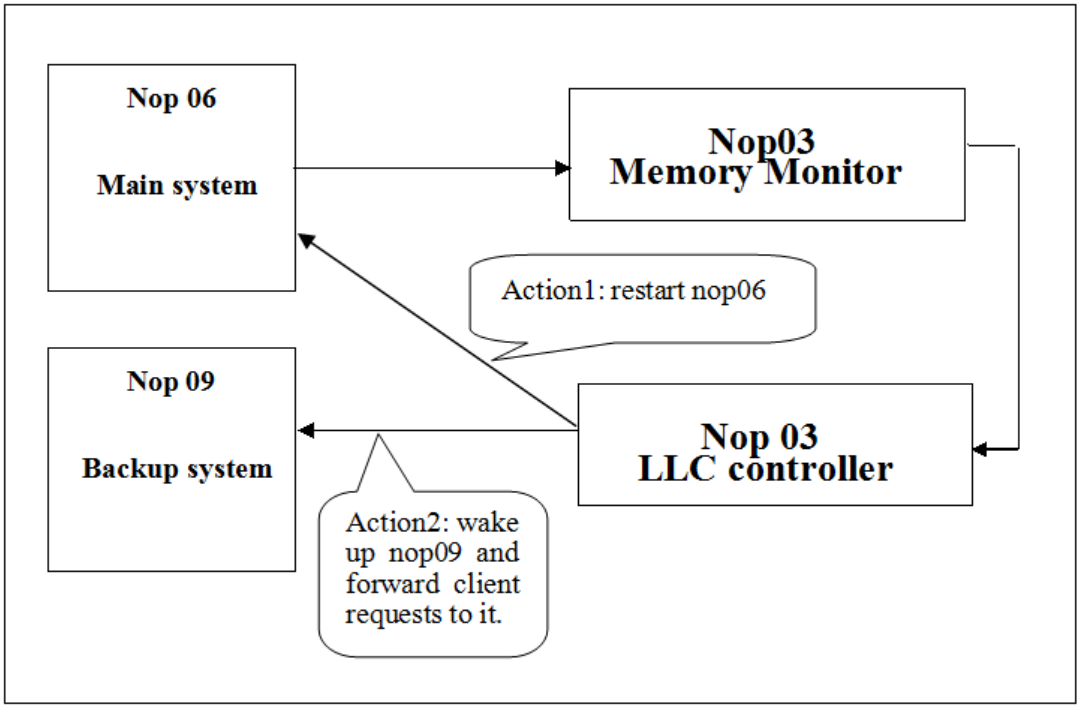

Figure 5.3 Fault management using LLC for scenarios 1 & 2

The LLC is responsible for adjusting the CPU frequency according to the current workload. The utility function of the LLC is as follows.

$$Utility = \alpha_1 * R(k) + \alpha_2 * Q(k) + \alpha_3 * E(k) \tag{5.6}$$

where $R(k)$ indicates the current response time in milliseconds, $Q(k)$ refers to the current queue level, $E(k)$ represents the current power consumption in Watts, and α_1, α_2 and α_3 are their weights that can be adjusted according to different requirements. In this work, the utility function considers three highly correlated parameters and combines them with different weights: 0.25 for response time, 0.6 for queue level and 0.4 for power consumption. Note that these weights can be adjusted to fit user defined priorities. Using this function, the LLC can evaluate the most proper CPU frequency for the current

working state and change the CPU frequency of the system to maintain optimal performance according the utility function.

The LLC is designed to maintain the CPU frequency according to the current client request rate that indicates the workload in this distributed system. The CPU frequencies in the target system can be 2GHz, 1.7GHz, 1.4GHz, 1.2GHz and 1GHz.

The second responsibility of the LLC is to respond to memory leak faults. Two recovery actions are designed for the LLC. The first is to start the main system on nop06. The second is to switch to the backup system on nop09. The backup system nop09 is a mirror system of the main server with the same configuration as nop06 and in a sleep state ready to be woken up by the controller. The LLC evaluates two available actions by applying the Memory Leak Cost function as follows:

$$\mathrm{Re}start\_C = (0.045 * S - 3.8) * \delta * 0.5 \tag{5.7}$$

$$Switch\_C = \beta * 0.5 \tag{5.8}$$

where S denotes the prediction result about the strength of the memory leak, 112 refers to the average restart delay in seconds and 135 refers to the power consumption by waking up the backup system from the sleep state. All the other numbers in the above functions are constants and are evaluated based on the analysis of memory leaks described in Chapter IV. When the controller obtains the memory leak strength from the Memory Monitor, the LLC compares the two cost values of the above functions to choose the action with a smaller cost value.

Both of these two actions have advantages and disadvantages. Restarting the main server consumes less power but has a longer delay because it usually takes more than one

minute to restart the nop06. On the other hand, if the LLC chooses to switch systems, more power is consumed, and there is less delay, because the backup system, nop09, has to be woken up from a sleep state, and the extra system will start to work. In such a circumstance, using the backup system will lead to more power consumption though the web server can continue offering its service temporarily when the controller chooses to switch. Therefore, depending on the type and intensity of memory leak errors, the LLC will evaluate both of these two actions by applying the Memory Leak Cost function and choose the action with minimal cost value.

In this design, the power consumption has non-linear features with respect to CPU frequency and the CPU utility. Therefore, a series of experiments were conducted to measure the power consumption by setting the CPU utility from 0% to 100% under all five possible frequencies. Estimation of power consumption can be found by checking the look-up table [108].

5.4.1 The first scenario

In the first scenario, the web server on nop06 has a memory leak with intensity of 300-level, and the LLC responded to this memory leak. Then, the memory leak with intensity of 300-level occurred again in the server on nop06, and the LLC reconfigured the web server to make it work properly. In this case, after the Memory Monitor detected the memory leak and predicted the strength of the memory leak, the LLC evaluated both actions of restarting and switching and chose to restart the system.

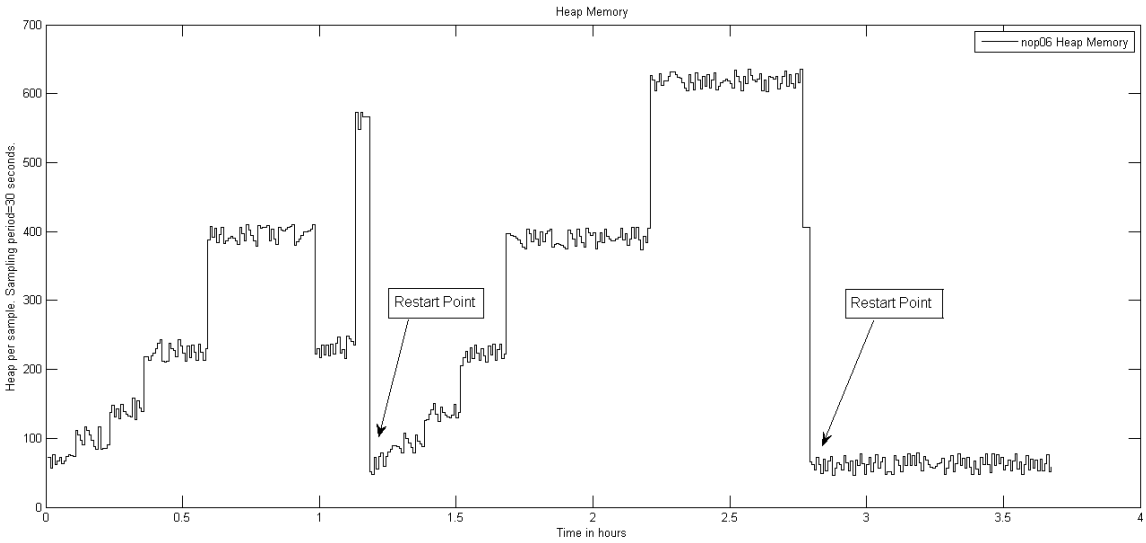

Figure 5.4 Heap memory for scenario 1

According to the memory utilization graph from this scenario, the main system restarted at 1.1 hours and continued working. However, the 300-memory-leak problem still existed in the system, and the Memory Monitor detected, predicted and took action again at 2.8 hours. This time the controller restarted the nop06 and reconfigured the web server again to make sure it worked fine without a memory leak problem.

At the first action point, which is point 1.1 hours (132nd sampling point), the LLC evaluated the two actions by applying the memory leak cost function (5.7, 5.8). With the predicted result about the memory leak strength from the Memory Monitor, the LLC evaluated two cost values of Restart_C=34.72 and Switch_C=67.50. Comparing these two values, the LLC chose to restart the main system, nop06. At the second action point of 2.73 hours (328th sampling point), the LLC evaluated the two actions of restarting and switching in the same manner as at the first action point. Because this time the strength of

memory leak was still 300, Restart_C and Switch_C have the same values of 34.72 and 67.50. Therefore, LLC still chose to restart the nop06.

After the second restart point, which is about 2.73 hours, the web server worked in a normal correct state.

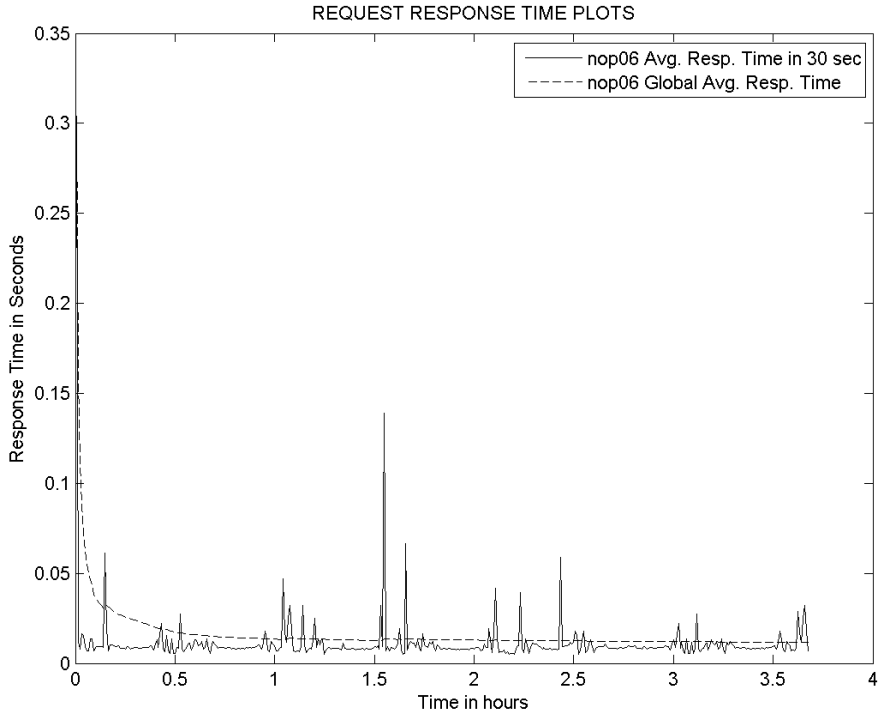

Figure 5.5 Response time for scenario 1

Figure 5.5 shows that the average response time from the beginning to the first restart point was similar to the average response time from the first restart point to the second restart point. The response time in both of these two phases was larger than the average response time in the phase from the second restart point to the end. Therefore, the global response time kept decreasing from the beginning the experiment.

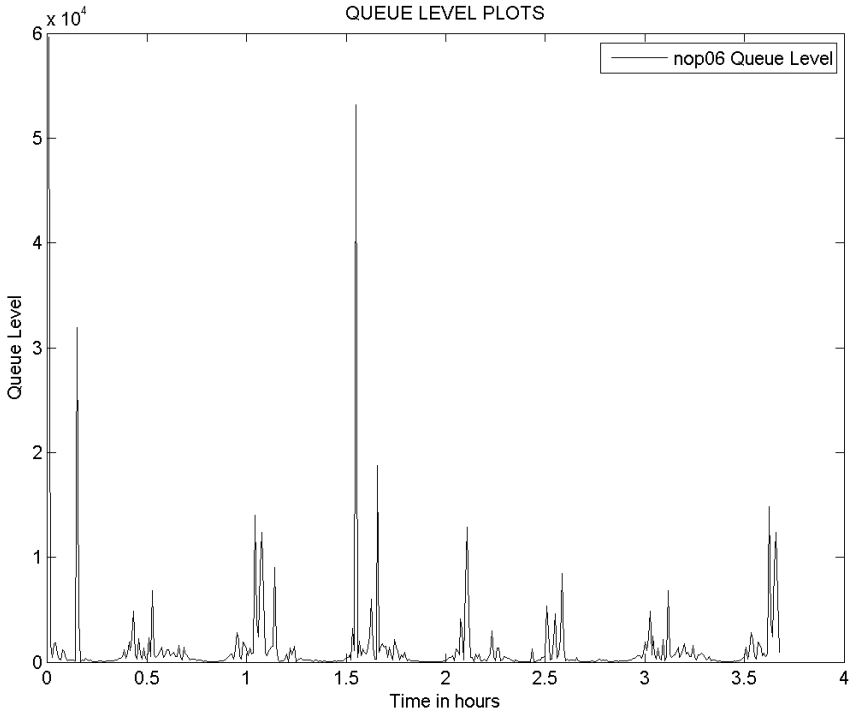

Figure 5.6 Queue level for scenario 1

The queue level (Figure 5.6) of the web server shows a similar trend with respect to request arrival rate (Figure 5.7). This is because when there are more requests on the server the corresponding queue size will be larger. Also, the queue level (Figure 5.6) highly correlates with the response time of the web server (Figure 5.5). As mentioned in Chapter IV, if the server processes requests at a slower speed, more requests will wait in the queue.

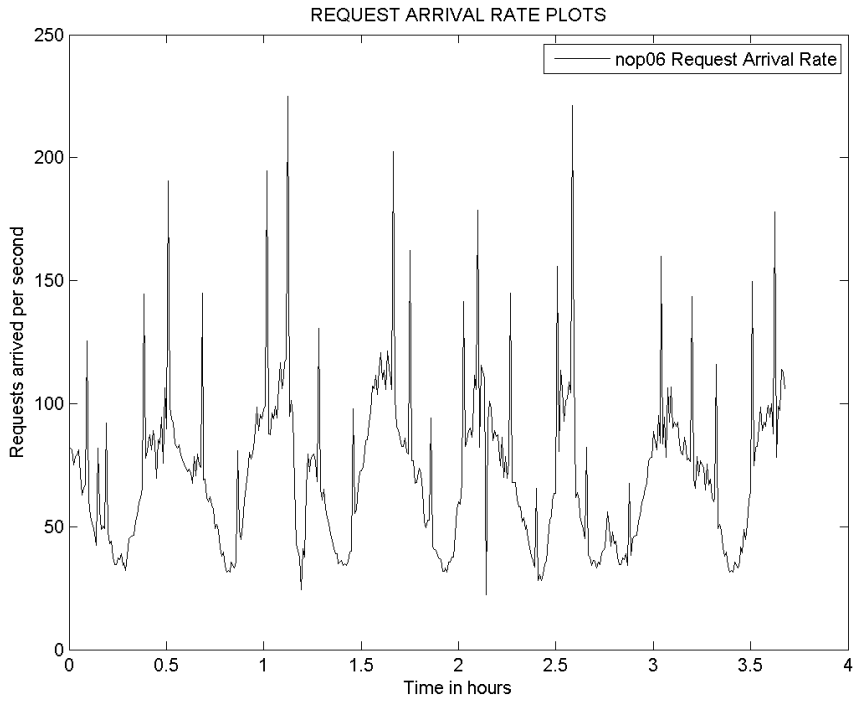

Figure 5.7 Request arrival rate for scenario 1

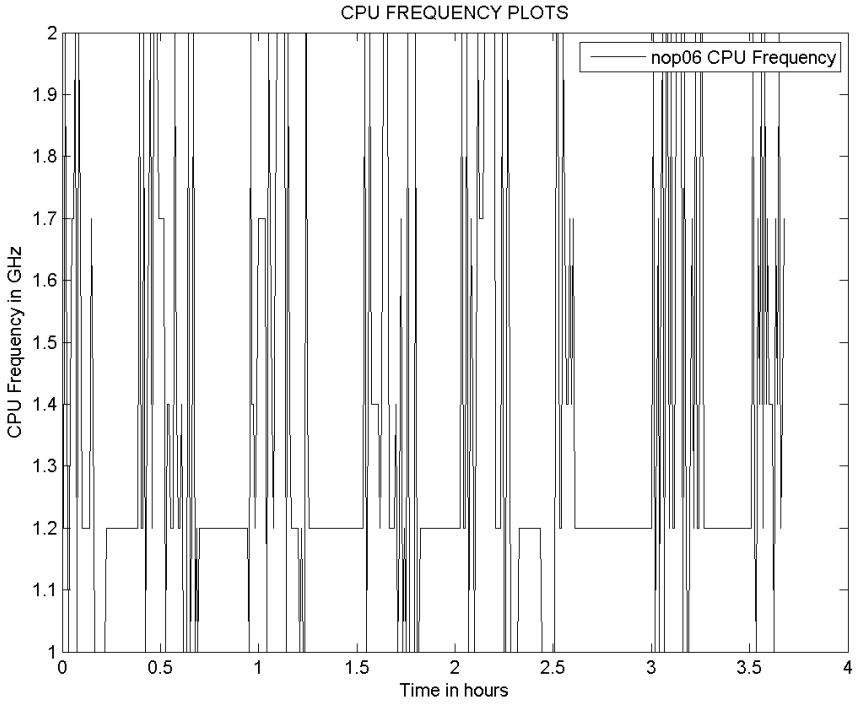

Figure 5.8 CPU frequency for scenario 1

Comparing the CPU frequency plot (Figure 5.8) with the Request arrival rate plot (Figure 5.7) it shows that the CPU frequency was automatically adjusted by the LLC according to the request arrival rate. When the request rate was at a relatively high value of over 100/s, the CPU frequency was also adjusted to a relative high value of 2GHz.

5.4.2 The second scenario

In the second situation, the web server on nop06 had a memory leak with strength of 300 at first, and the LLC chose to restart the main server nop06. Then, a memory leak with strength of 500 occurred in the web server on nop06 again. This time the controller evaluated utility functions (5.7, 5.8) again for this memory leak and chose to switch to the backup system on nop09.

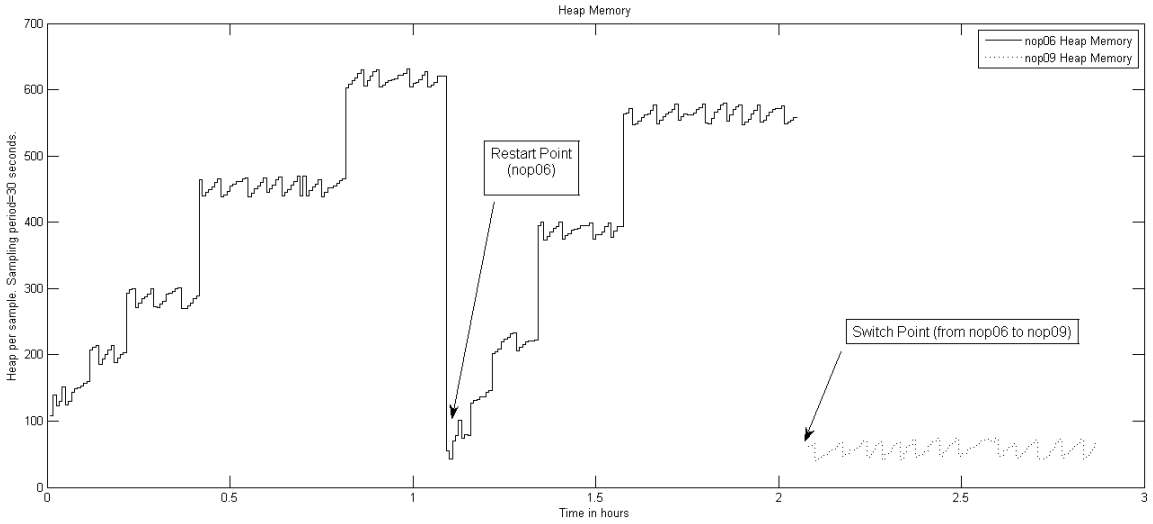

Figure 5.9 Heap memory for scenario 2

In this case, a memory leak with strength of 300 happened to the web server on the main system, nop06. The Memory Monitor predicted the strength of 300 and forwarded the result to the LLC. Based on the utility function, the LLC chose to restart the main system, nop06, at point 1.1 hours. After the LLC's first action at 1.1 hours, a memory leak with strength of 500 occurred in the web server on nop06 again. With the Memory Monitor's prediction result, the LLC evaluated the memory leak cost equations (5.7, 5.8) which have values of Restart_C= 110.38 and Switch_C=67.50. After evaluation of these cost values, the LLC chose to wake up the backup system and switch to it.

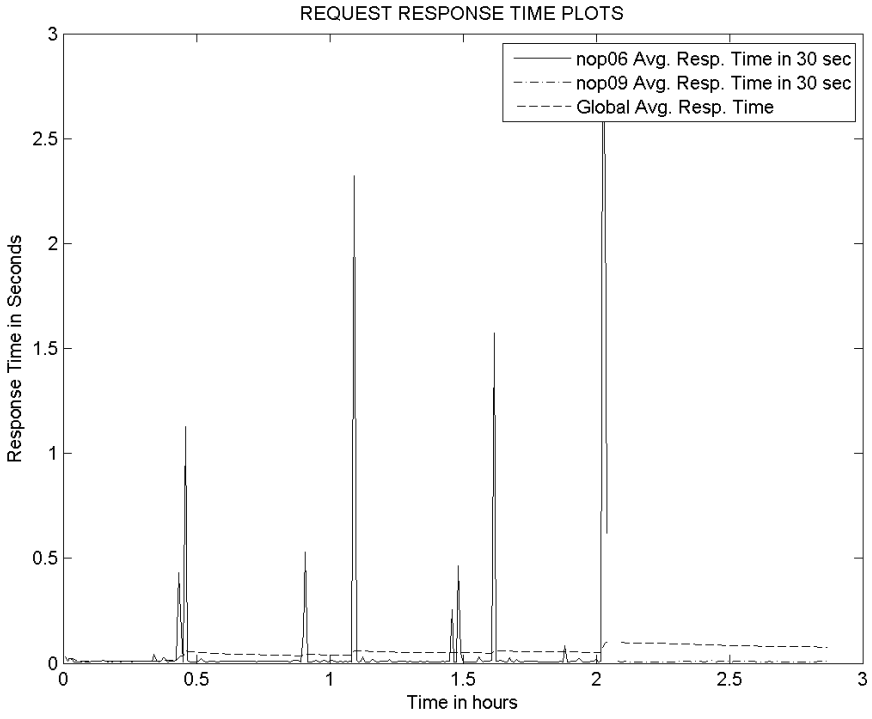

Figure 5.10 Response time for scenario 2

According to the above response time graph (Figure 5.10) of the web servers on nop06 and nop09, the response time on nop09 only has values after the switch point, because nop09 was kept in the sleep state before the LLC chose to switch the system to nop09. After nop09 was woken up by the LLC and took over the workload from nop06, the web server on nop09 worked in a normal and correct state. Therefore, the response time of the web server on nop09 was very small compared to the response time on nop06, which had a memory leak. As for the response time of the web server on nop06, it changed with the requests arrival rate (Figure 5.12), staying at a relatively high value. From the global average response time, which combines the response time on both nop06 and nop09, the average response time in the phase of memory leak 300 is smaller than the average response time in the phase of memory leak 500. Moreover, because both nop06 and nop09 run in the system after the switch point and the global average response time combines the response time from two systems, the global average response time is kept at a relatively high value compared to the global average response time in the phase before the switch point.

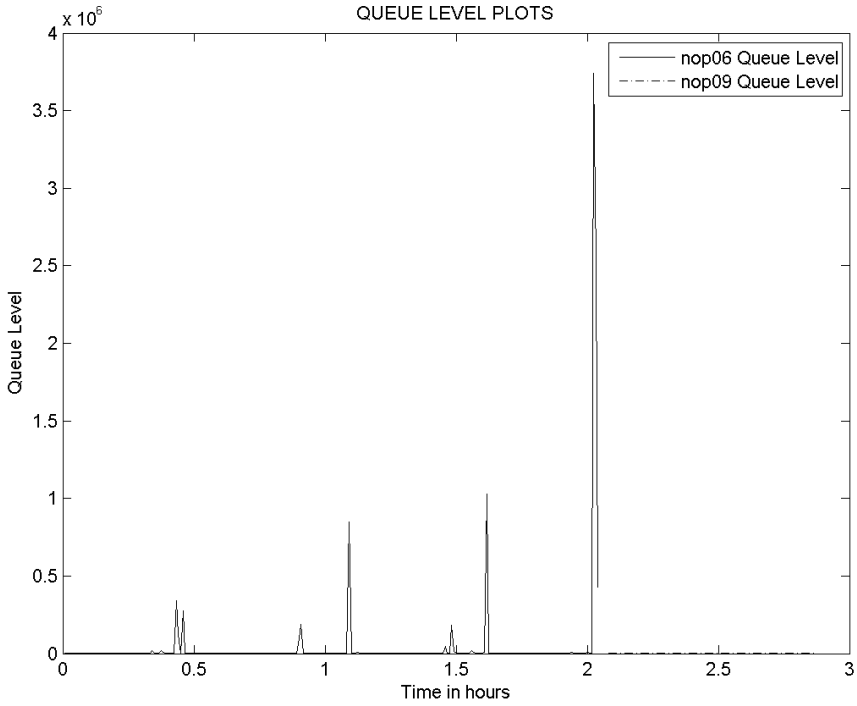

Figure 5.11 Queue level for scenario 2

The queue level (Figure 5.11) of the web server is clearly correlated with the request arrival rate (Figure 5.12) and the response time (Figure 5.10).

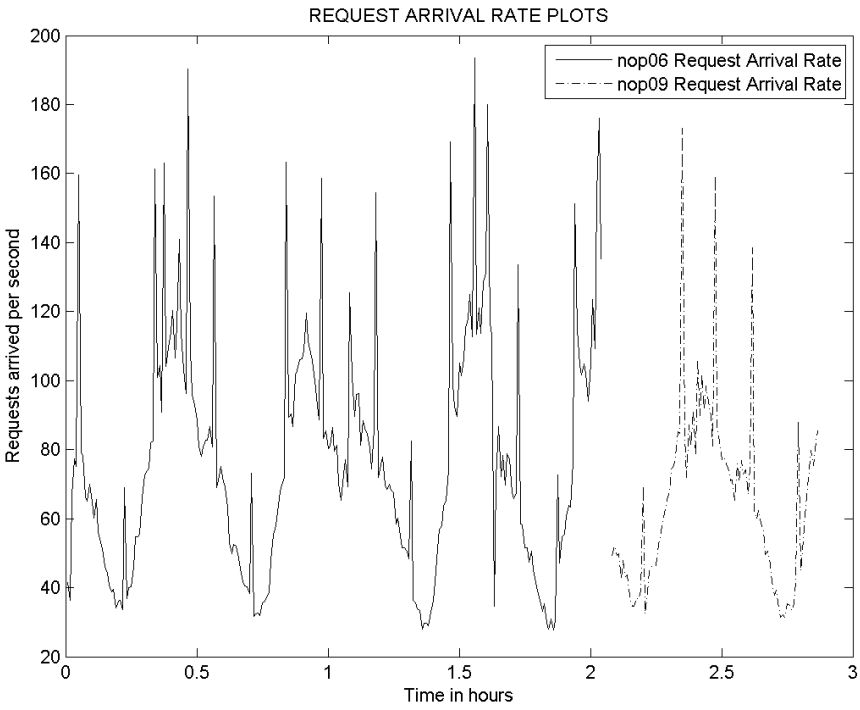

Figure 5.12 Request arrival rate for scenario 2

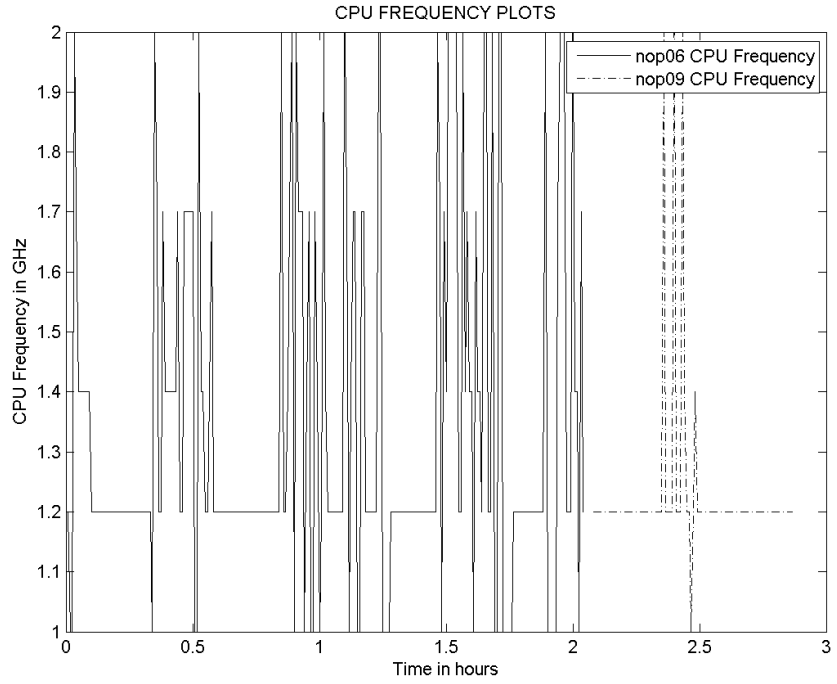

Figure 5.13 CPU frequency for scenario 2

According to the CPU frequency plot (Figure 5.8) and the request arrival rate plot (Figure 5.13), the CPU frequency on both nop06 and nop09 were automatically adjusted by the LLC based on the change of request arrival rate.

5.4.3 The third scenario

The third scenario is the parallel situation in which a memory leak occurs outside the web service application and two processes run in parallel. Under this circumstance, the controller keeps watching the total memory of the system, detects errors and takes action to recover the target system from the errors. The implementation structure of this scenario is shown in Figure 5.14. When the total memory consumption of all the processes on nop06 reaches 350 MB, the controller will compare the increasing rate of all the running processes and select the one closest to the increasing rate of the total memory. In this implementation, all the processes except for the web service application are assumed to not be fatal to the overall system and can be killed when errors appear. Therefore, in the experiment, after the controller detected the process that has a memory leak, the LLC killed this process. The following graphs show the relative parameters.

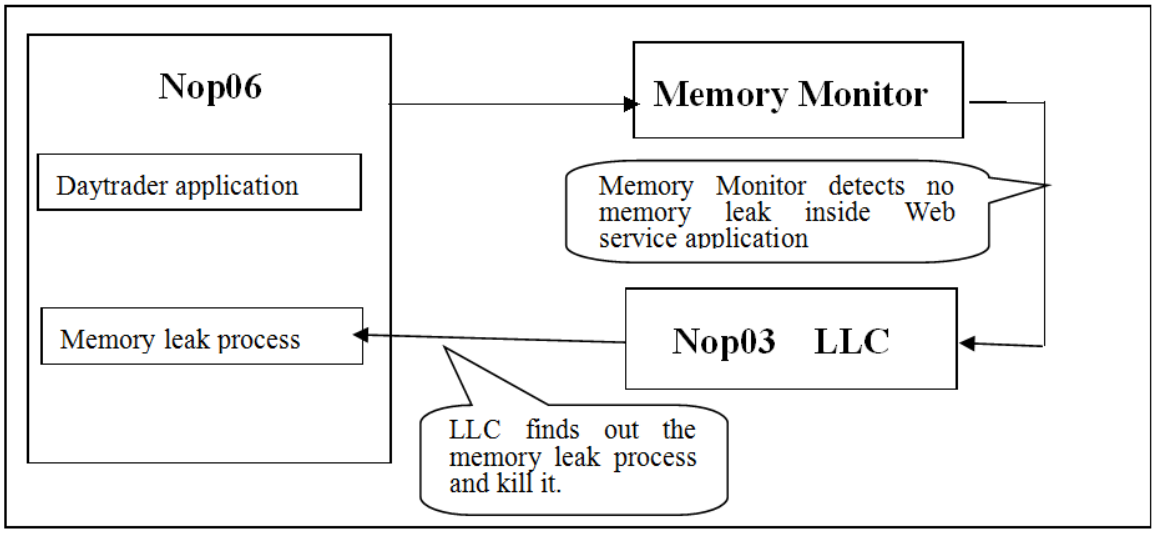

Figure 5.14 Fault management using LLC for scenario 3

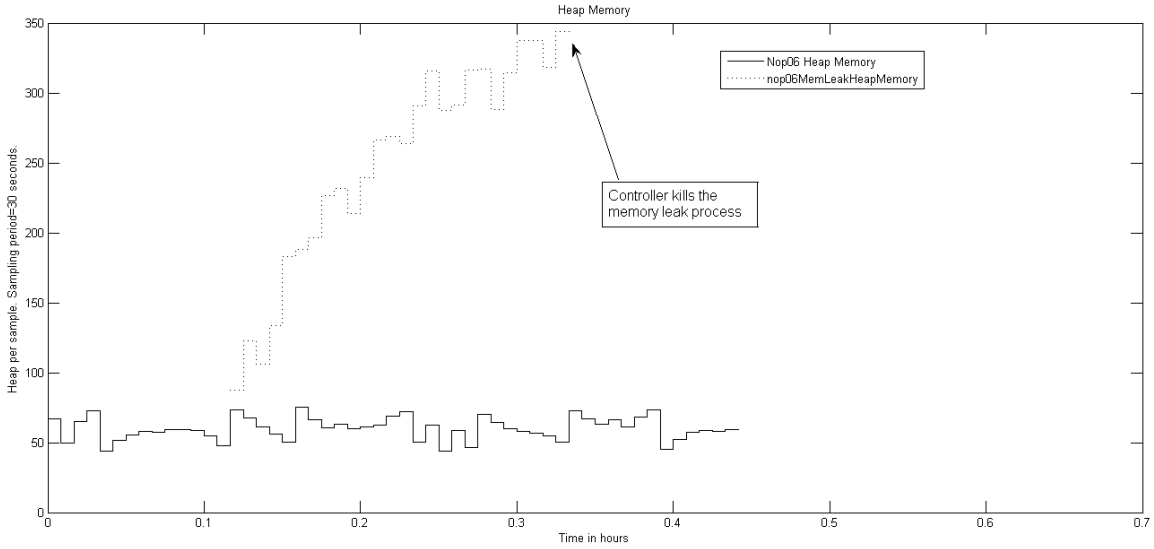

Figure 5.15 Heap memory for scenario 3

According to the above memory plot (Figure 5.15), the web server ran at a normal state for the first 0.1 hours, and after 0.1 hours, a program with a memory leak started to run on the same system of nop06. At 0.35 hours, the LLC detected and terminated the

process with the memory leak. The heap memory of the web server remained unaffected throughout the experiment.

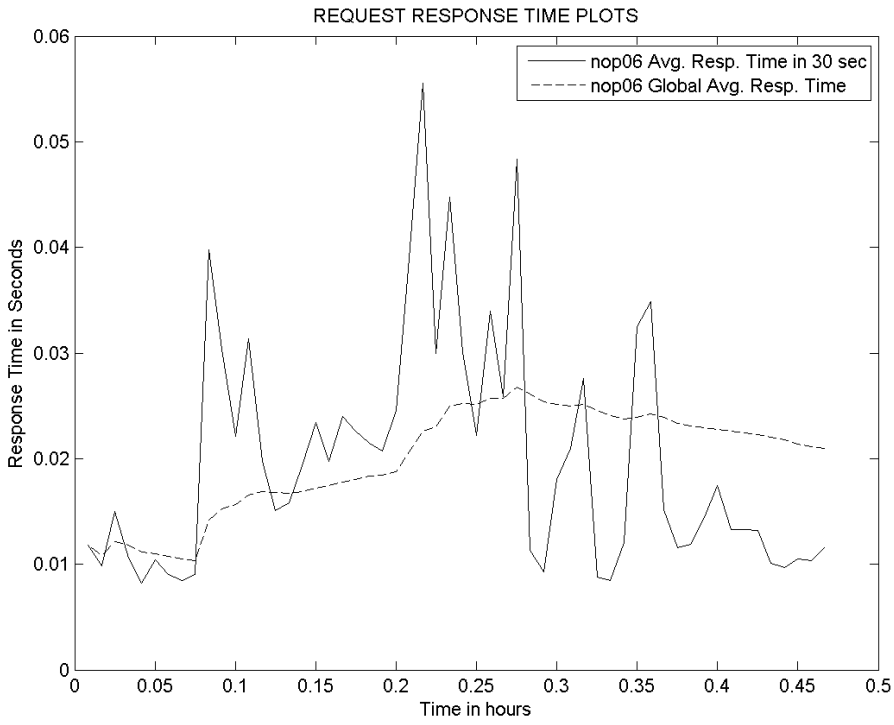

Figure 5.16 Response time for scenario 3

From the response time plot (Figure 5.16), the response time of the web server remained at a normal low value before the program with the memory leak started working. During the phase from point 0.07 hours to point 0.27 hours when the web server and the program with the memory leak ran parallel, the response time of the web server remained at a relatively high value of around 0.03 seconds. After the program with the memory leak was killed by the LLC at 0.35 hours, the response time of the web server went back to the normal low value that is about 0.01 seconds.

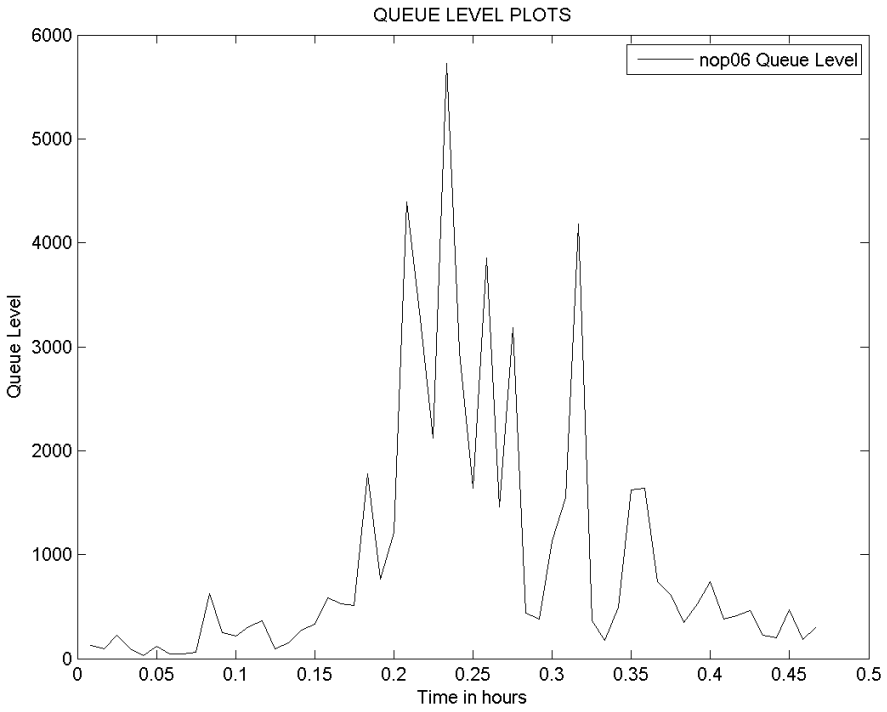

Figure 5.17 Queue level of web server for scenario 3

The queue level of the web server (Figure 5.17) has a similar trend as the response time of the web server (Figure 5.16) and the request arrival rate (Figure 5.18).

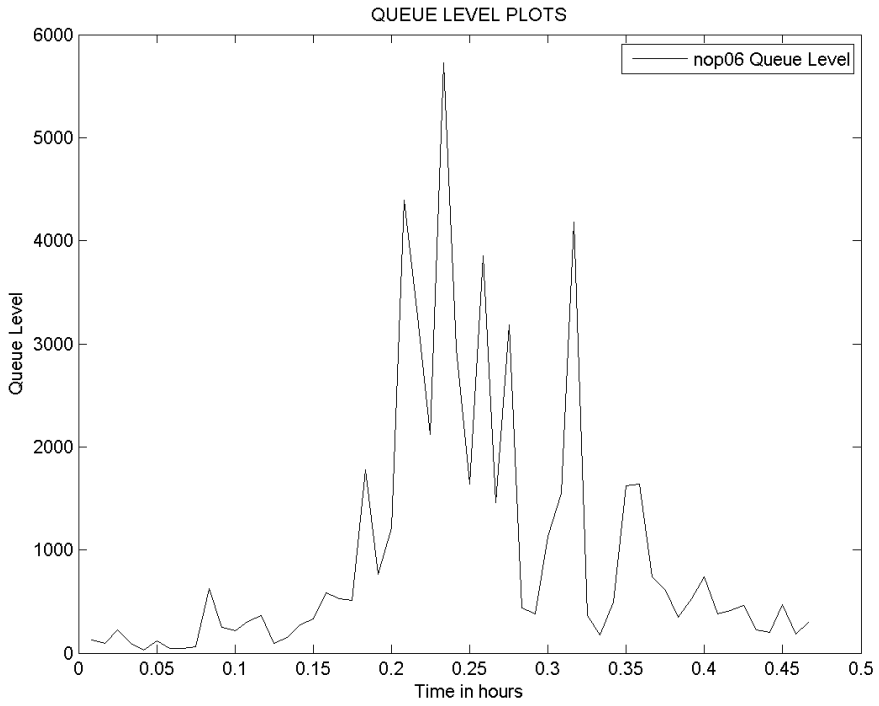

Figure 5.18 Request arrival rate for scenario 3

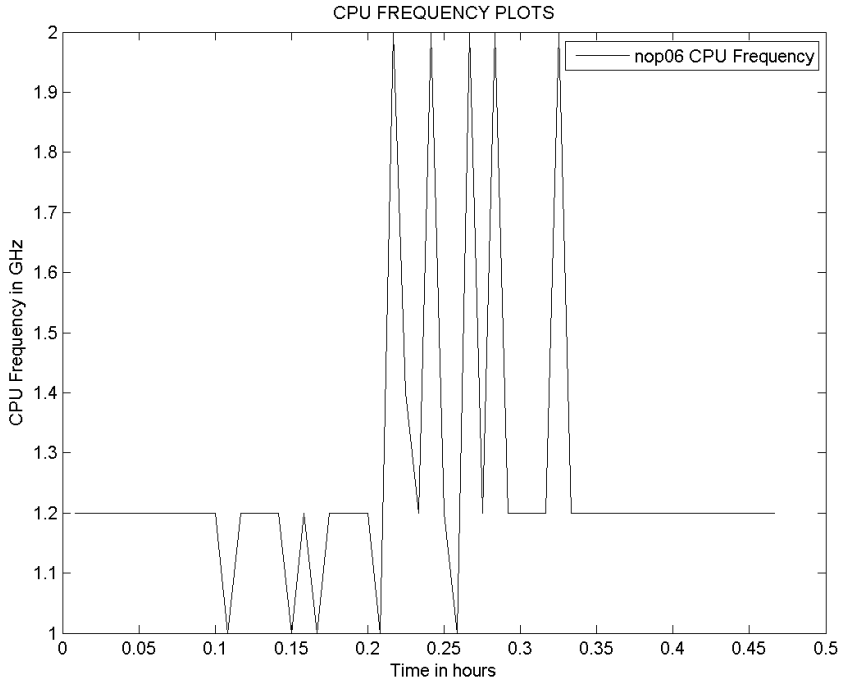

Figure 5.19 CPU frequency for scenario 3

The CPU frequency plot (Figure 5.19) and the request arrival rate plot (Figure 5.18) show that the CPU frequency was automatically adjusted by the LLC according to the request arrival rate.

CHAPTER VI

CONCLUSION

This thesis presentes an implementation of a limited lookahead controller (LLC) for fault management in a distributed system. A benchmark online trading application, Daytrader was used as the service application. This implementation of the LLC is conducted in three phases. The first phase is the simulation of a memory leak in the target system. In the second part, the memory leak simulation results are characterized. In the third phase, the LLC maintains the CPU frequency and takes three actions to respond to the memory leak to recover the system. When the process containing memory leak fault is not a critical process, LLC choose to directly kill the corresponding process. In the other cases, LLC takes two available actions to recover the target system, and they are are restarting the main system and switching to the backup system. This is based on the intensity of memory leak from the result of the characterization.

When a memory leak was simulated inside the web server, the memory utilization of the web server increased to a high value of around 650 MB and decreased to around 500 MB due to Java garbage collection. The experiments showed that as the intensity of the memory leak increased, the time for the web server's memory utilization to reach a peak value dropped significantly, and the web server's response time increased. The average value of the high response time increases with the increase of the memory leak

intensity. However, the high response time lasted for a shorter time with a smaller intensity of memory leak. In addition, the queue level of the web server had a similar trend as the response time, because when the web server needed a longer time to process requests from a client, there would be more requests waiting in the queue. On the other hand, when the memory leak was simulated outside the web server, running in parallel with the web server program, the web server remained unaffected until the total memory utilization of the system reached the system limit of 1 GB and the memory leak simulation program crashed, and all the related working parameters of the web server increased to very high values.

In the second part, the Memory Monitor was designed to characterize the memory leak simulation results. A low pass filter was designed to process the memory utilization dataset to smooth the curve of the plot. Then, the Memory Monitor analyzed the processed memory utilization dataset by applying a linear fitting analysis to get the memory utilization slope that indicates the intensity of the memory leak. Moreover, analyzing the average response time of the web server can also show the intensity of the memory leak. To predict the memory leak characterization results, the slope's increment was monitored by the Memory Monitor every 30 seconds. When the increment was zero or below zero the Memory Monitor output the existence of a memory leak. Following that, the Memory Monitor predicted the intensity of the memory leak by analyzing the average response time when the memory utilization reached 400 MB. The prediction results showed that the intensity of the memory leak could be approximately predicted.

The implementation of the LLC in the target system is conducted in three different scenarios. Results from all of the scenarios show that the LLC is able to adjust

the CPU frequency according to the changes of the client request rate by evaluating the utility function which balances both the system response time and power consumption. In the first two scenarios, the target system had a 300-level memory leak at first; the LLC chose to restart the main system nop06 after evaluating the memory leak utility function. Then, when the web server contained a 500-level memory leak, the LLC chose to switch to the backup server on nop09. The memory leak was implemented outside the web server running in parallel in the third scenario; the LLC detected and killed the corresponding process that had a memory leak to ensure that the web server remained unaffected. All of the scenarios have similar results, indicating that the LLC can recover the target system from a memory leak fault by taking the corresponding best action. With the memory leak utility function, the LLC is able to choose the best action that balances both the response time and power consumption.

In future research, this implementation could also be used to control other types of faults such as resource unavailability in a distributed system by adjusting the characterization part. More research is needed in characterization to improve the accuracy of predictions. Furthermore, in the Java garbage collection, the exact point was not established when the garbage collection started working in the target system. More studies need to be performed in order to investigate the garbage collection working characteristics. In addition, new control actions, besides restarting and switching used in this thesis, can be added to recover the system from faults. Example actions can be setting an upper limit of the memory utilization for certain processes before the processes start working.

REFERENCES

[1] M. P. Papazoglou, “Web services: Principles and technology.” 2007, Prentice Hall.

[2] A.S.Tanenbaum, M.V.Steen, *Distributed system principles and paradigms,* PHI, 2009

[3] M. P. Papazoglou, W. J. van den Heuvel, "Web services Management: A survey", *IEEE Internet Computing*, Vol 9, No.6, Nov-Dec, pp 58-64, 2005.

[4] J. Murry, "Designing Manageable Applications", *Web Developer’s Journal*, October 2002, available at http://www.webdevelopersjournal.com/articles/design_man_app/

[5] T. Mehta, “Adaptive Web Services Management Solutions", *Enterprise Networks and Servers*, vol. 17, no. 5, available at http://www.enterprisenetworksandservers.com/monthly/toc.php?35

[6] D. Kakadia *et al.*, "Enterprise Management Systems: Architectures and standards", *Sun Microsystems*, April 2002, available at http://www.sun.com/blueprints/0402/ems1.pdf

[7] W. Tu, C.J. Sreenan, W. Ji, "Worst-case delay control in multigroup overlay networks", *Transactions on Parallel and Distributed Systems*, Vol.18, No. 10, pp. 1407-1419, 2007.

[8] N. Gui, C. Wu, S. Chen, J. Wang, "A stable stateless fair bandwidth allocation algorithm using stochastic control," *Communications, Circuits and Systems Proceedings International Conference*, 25-28 Jun., Guilin, pp. 1722-1726, 2006.

[9] A.K. Moharana, K. Panigrahi, B.K. Panigrahi, and P.K. Dash, "Vsc based hvdc system for passive network with fuzzy controller," *Power Electronics, Drives and Energy Systems, PEDES '06. International Conference*, 12-15 Dec., New Delhi, pp. 1-4, 2006.

[10] T. Chai, "A hybrid intelligent optimal control method for the whole production line and applications," *ICIT '07. IEEE International Conference*, 20-24 Mar., Beijing, pp. 14-15, 2007.

[11] T. F. Abdelzaher, K. G. Shin, N. Bhatti, "Performance guarantees for web server end-systems: a control-theoretical approach," *In Parallel and Distributed Systems IEEE*, Vol.13, No. 1, pp. 80-96, 2002.

[12] T.F. Abdelzaher, N. Bhatti, "Web server qos management by adaptive content delivery," *In Quality of Service, IWQoS '99, Seventh International Workshop*, 31 May-04 Jun., London, pp. 216-225, 1999.

[13] C. Lu, J. Stankovic, G. Tao, and S. Son. "Feedback control real-time scheduling: Framework, modeling, and algorithms," *Journal of Real-Time Systems*. Vol. 23, No. 1-2, pp. 85-126, 2002.

[14] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley, "Performance specifications and metrics for adaptive real-time systems", *Proceedings of the 21st IEEE conference on Real-Time Systems,* 2-4 Jul., Orlando, pp. 13-23, 2000.

[15] C. Lu, J.A. Stankovic, G. Tao, and S. H. Son, "Design and evaluation of a feedback control edf scheduling algorithm"*, Proceedings of the 20th IEEE conference on Real-Time Systems Symposium*, Phoenix, pp. 56-67, 1999.

[16] C. Dovrolis, D. Stiliadis, and P. Ramanathan, "Proportional differentiated services: Delay differentiation and packet scheduling," *ACM SIG-COMM Computer Communication Review*, Vol. 29, No. 4, pp. 109-120, 1999.

[17] C. Lu, Y. Lu, T.F. Abdelzaher, J.A. Stankovic, and S. Son, "Feedback control architecture and design methodology for service delay guarantees in web servers," *Transactions on Parallel and Distributed Systems*, Vol. 17, No. 9, pp. 1014-1027, 2006.

[18] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus, "Using control theory to achieve service level objectives in performance management", *IEEE/IFIP International Symposium on Integrated Network Management Proceedings,* Seattle*,* pp. 841-854, 2001.

[19] Y. Lu, A. Saxena, and T.F. Abdelzaher, "Differentiated caching services; a control-theoretical approach", *21st International Conference on Distributed Computing Systems*, 16-19 Apr., Mesa, pp. 615-622, Apr. 2001.

[20] C.V. Hollot, V. Misra, D. Towsley, and W. Gong, "A control theoretic analysis of red," *Proceedings INFOCOM 2001, 20th Annual Joint Conference of the IEEE Computer and Communications Societies,* 22-26 Apr., Anchorage, pp.1510-1519, 2001.

[21] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," *Proceedings of the third symposium on Operating systems design and implementation*, Berkeley, pp. 145-158. , 1999.

[22] K. Astrom and T. Hagglund, *PID Controllers: Theory, Design, and Tuning*, Instrument Society of America, 2nd edition, 1995.

[23]L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queueing model based network server performance control," *23rd IEEE on Real-Time Systems Symposium,* pp. 81-90, 2002.

[24]L. Kleinrock, *Queueing Systems Theory*, John Wiely & Sons, January 1975.

[25]D. Henriksson, Y. Lu, and T. Abdelzaher, "Improved prediction for web server delay control," *Proceeding of 16th Euromicro Conference on Real-Time Systems*, pp 61-68, Jul., 2004.

[26]Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu, "Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers," *Proceedings of the 9th IEEE on Real-Time and Embedded Technology and Applications Symposium*, 27-30 May, pp. 208-217, 2003.

[27]A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 1 June, New York, 2003.

[28]W. Xu, X. Zhu, S. Singhal, and Z. Wang, "Predictive control for dynamic resource allocation in enterprise data centers," *Network Operations and Management Symposium, NOMS 2006. 10th IEEE/IFIP*, 3-7 Apr., Vancouver, pp 115-126, 2006.

[29]Y. Lu, T. Abdelzaher, C. Lu, and G. Tao, "An adaptive control framework for qos guarantees and its application to differentiated caching," *2002 Tenth IEEE International Workshop on Quality of Service*, pp. 23-32, 2002.

[30]X. Liu, X. Zhu, S. Singhal, and M. Arlitt, "Adaptive entitlement control of resource containers on shared servers," *IM 2005, 9th IFIP/IEEE International Symposium on Integrated Network Management*, 15-19 May, Nice, pp. 163-176, 2005.

[31]M. Karlsson, C. Karamanolis, and X. Zhu, "Triage: performance isolation and differentiation for storage systems," *ACM Transactions on Storage,* Vol. 1, No. 4, pp. 457-480, Nov. 2005.

[32]Y. Diao and K.M. Passino, "Adaptive neural/fuzzy control for interpolated nonlinear systems," *IEEE Transactions on Fuzzy Systems*, Vol. 10, No. 5, pp. 583-595, Oct. 2002.

[33]Y. Diao and K.M. Passino, "Stable fault-tolerant adaptive fuzzy/neural control for a turbine engine," *IEEE Transactions on Control Systems Technology,* Vol. 9, No. 3, pp. 494-509, May 2001.

[34]M. Karlsson, "Maximizing the utility of a computer service using adaptive optimal control," *Proceedings of the 2006 IEEE International Conference on Networking, Sensing and Control*, 23-25 April, Ft. Lauderdale, pp. 89-94, 2006.

[35]M. Orlovich and R. Rugina, "Memory leak analysis by contradiction," Computer Science, Vol. 41, No. 34, pp. 405-424, 2006.

[36]L. A. Zadeh, "fuzzy sets," *Information and Control*, Vol. 8, pp. 338-353, 1965.

[37]M. Sugeno and T. Yasukawa, "A fuzzy-logic-based approach to qualitative modeling," *IEEE Transactions on Fuzzy Systems*, Vol. 1, No. 1, pp. 7, Feb. 1993.

[38]A. Talukder, R. Bhatt, T. Sheikh, R. Pidva, L. Chandramouli, and S. Monacos, "Dynamic control and power management algorithm for continuous wireless monitoring in sensor networks," *, 29th Annual IEEE International Conference on Local Computer Networks*, 16-18 Nov., pp. 498-505, 2004.

[39]E.F. Camacho and C. Bordons, *Model Predictive Control, Advanced Textbooks in Control and Signal Processing,* Springer-Verlag, 2004.

[40]C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 16, No. 6, pp. 550-561, 2005.

[41]Y. B. Shalom, R. Larson, and M. Grossberg, "Application of stochastic control theory to resource allocation under uncertainty," *IEEE Transactions on Automatic Control*, Vol. 19, No. 1, pp. 1-7, Feb 1974.

[42]M. Zafer and E. Modiano, "Minimum energy transmission over a wireless fading channel with packet deadlines," *2007 46th IEEE Conference on Decision and Control*, 12-14 Dec., New Orleans, pp.1148-1155, 2007.

[43]M. Zafer and E. Modiano, "Delay-constrained energy efficient data transmission over a wireless fading channel," *Information Theory and Applications Workshop*, Jan. 29- Feb 2., La Jolla, pp 289-298, 2007

[44]X. Chen, Q. Zhu, Y. Liao, P. Kuang, and G. Xiong, "Dynamic optimal control for aperiodic soft real-time systems," *2006 International Conference on Communications, Circuits and Systems Proceedings,* 25-28 June, Guilin, pp. 2796-2800, 2006.

[45]B. Lincoln and B. Bernhardsson, "Optimal control over networks with long random delays*", Proceedings of the International Symposium on Mathematical Theory of Networks and Systems,* 2000.

[46]L. Xie, W. Zhao, and Z. Ji, "Lqg control of networked control system with long time delays using δ -operator," *Sixth International Conference on Intelligent Systems Design and Applications*, Jinan, pp.183-187, 2006.

[47]IEEE Standard Glossary of Software Engineering Terminology. lEEE Std 610.121990 (Revision and reddgnation of IEEEstd7921983) available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00159342

[48]D. Ardanga *et al.*, "Faults and recovery actions for self healing web serives," available at ftp://ftp.elet.polimi.it/users/Barbara.Pernici/ws-diamond/Faults-Recovery-Actions-Poliminov05.
Pdf

[49]W. He, "Recovery in Web service applications", *IEEE International Conference on e-Technology, e-Commerce and e-Service, 28-31 Mar., Taipei,* pp. 25 -28, 2004.

[50]M.T. Huda, H. W.Schimdt, I. D.Peake, *An agent oriented dynamic fault tolerant framework for Grid computing*, Monash University, Melbourne, 2005.

[51]M. K. Aguilera, W. Chen, and S. Toueg, "Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks," *Theoretical Computer Science, special issue on distributed algorithms,* Vol. 220, pp. 3 − 30, 1999.

[52]E. Marcus and H. Stern, *Blueprints for High Availability,* John Wiley and Sons, Inc., New York, NY, 2000.

[53]T. Idé and H. Kashima, "Eigenspace-based anomaly detection in computer systems," *10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* Seattle, WA, pp. 440−449, August 2004.

[54]P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, "Magpie: real-time modelling and performance-aware systems," *9th Workshop on Hot Topics in Operating Systems (HotOS IX),* 8-10 May, Napa Valley, pp. 85-90, May 2003.

[55]M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic systems," *In International Performance and Dependability Symposium,* 23-26 Jun., pp. 595-604, Washington, 2002.

[56]G. Jiang, H. Chen, and K. Yoshihira, "Discovering likely invariants of distributed transaction systems for autonomic system management," *In Proceedings of the 3rd International Conference on Autonomic Computing (ICAC),* June, Dublin, pp. 199−208,, 2006.

[57]H. Kang, H. Chen, G. Jiang, "PeerWatch: a Fault Detection and Diagnosis Tool for Virtualized Consolidation Systems," *Proceedings of 7th International Conference on autonomic Computing (ICAC),* Jun., Washington, 2010.

[58]A. Bouloutas, S. Calo, and A. Finkel, "Alarm correlation and fault identification in communication networks," *IEEE Trans. Communications,* Vol.4, No. 2, pp. 523−533, 1994.

[59]I. Rouvellou and G. W. Hart, "Automatic alarm correlation for fault identification," *In Proceedings of the INFOCOM,* pp. 553−561, 1995.

[60]H. Corporation. H.P. Openview. http://www.openview.hp.com/.

[61]IBM. Tivoli business system manager. http://www.tivoli.com/.

[62]I. Cohen, S. Jeffrey, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," *In Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI),* San Francisco, CA, December 2004.

[63]M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed systems of black boxes," *In Proceedings of the nineteenth ACM symposium on Operating systems principles,* pp. 74−89, Bolton Landing, NY, 2003.

[64]A. B. Brown, G. Kar, and A. Keller, "An active approach to characterizing dynamic dependencies for problem determination in a distributed environment," *In Proceedings of the Seventh IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Seattle, WA, May 2001.

[65]M. Brodie, I. Rish, and S. Ma, "Intelligent probing: A cost-efficient approach to fault diagnosis in computer networks," *IBM Systems Journal*, Vol. 4, No. 1, pp. 372−385, 2002.

[66]A.V. Mirgorodskiy and B. P. Miller, "Diagnosing Distributed Systems with Self-propelled Instrumentation," *In Proceedings of the IFIP International Federation for Information*, pp. 82−103, 2008.

[67]B. Falkenhainer, "Self-explanatory simulations: an integration of qualitative and quantitative knowledge," *In Proceedings of the AAAI*, pp. 380-387, 1990.

[68]F. Finch and M. Kramer, "A robust event-oriented methodology for diagnosis of dynamic process systems," *Computers and Chemical Engineering*, Vol. 14, No. 12, pp. 1379-1398, December 1990.

[69]D. Dvorak and B. Kuipers, "Process monitoring and diagnosis: a model-based approach," IEEE Expert, Vol. 6, No.4, pp. 67-74, Jun 1991.

[70]P. Frank, "Fault Diagnosis in Dynamic Systems Using Analytical and Knowledge based Redundancy - A Survey and Some New Results," *Automatica*, Vol. 26, No. 3. pp 459-474, May 1990.

[71]P. Frank, "Analytical and qualitative model-based fault diagnosis: A survey and some new results," *Europian Journal of control*, Vol. 2, No. 3, pp. 6-28, 1995.

[72]R.J.Patton, P.M. Frank, and R.N. Clark, *Fault diagnosis in dynamic systems, Theory and application*, Prentice Hall.

[73]D. M.Himmelblau, "Fault detection and diagnosis - Today and tomorrow," *In Proceedings of the 1st IFAC Workshop on fault detection and safety in chemical plants*, pp. 95-105, Kyoto, Sept. 1993

[74]M. A. Karamer and J. A. Leonard, "Diagnosis using backpropagation neural networks; analysis and criticism," *Computer and Chemical Engineering,* Vol.14, No. 12, pp. 1323-1338, December 1990.

[75]J. Zhang and A. Morris, "On-line process fault diagnosis using fuzzy neural networks," *Intelligent Systems Engineering,* Vol. 3, No. 1, pp. 37-47, 1994.

[76]C. Can, and K. Danai, "Fault Diagnosis with a Model-Based Recurrent Neural Network," *In Proceedings of the Safeprocess 03*, Washington, U.S.A. pp.699-704, 2003.

[77]J. D. Kleer and J. Brown, "A qualitative physics based on confluences," *Artificial Intelligence,* Vol. *24*, No. 1-3, pp. 7-84, Dec. 1984.

[78]J. Collins and K. Forbus, "Reasoning about fluits via molecular collections," Readings in Qualitative reasoning about physical systems Eds. D. Weld and J. Kleer, M. Kaufman Inc. pp 503-507.

[79]J. D. Kleer and J. Kurien, "Fundamentals of Model-Based Diagnosis," *In Proceedings Safeprocess 03,* Washington, U.S.A. pp.25- 36, 2003.

[80]R. Herbert and G. Williams, "An initial evaluation of the detection and diagnosis of power plant faults using a deep knowledge representation of physical behaviour," *Expert Systems,* Vol. *4*, No. 2, pp. 90-99, May 1987.

[81]C. Angeli, and D.P Atherton., "A model based method for an on-line diagnostic knowledge-based system," *Expert Systems* Vol. 18, No. 3, pp. 150-158, 2001.

[82]E. Shokri, P. Crane, J. Dussault, K. Kim, and C. Subbaraman, "ROAFTS: A CORBA-based middleware for real-time object-oriented adaptive fault tolerance support," *In IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, San Francisco, CA, December 1997.

[83]D. Goldberg, M. Li, W. Tao, and Y. Tamir, "The design and implementation of a fault-tolerant cluster manager," *Technical Report Computer Science Department Technical Report* CSD-010040, University of California, Los Angeles, CA, October 2001.

[84]E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Comput. Surv.*, Vol. 34, No. 3, pp. 375−408, 2002.

[85]M.Treaster, "A Survey of Fault-Tolerance and Fault-Recovery Techniques in Parallel," available at http:/systems.arXiv:cs/0501002v1 [cs.DC] 1 Jan 2005

[86]A. Benharref , R. Glitho and R. Dssouli, "Mobile Agents for Testing Web Services in Next Generation Networks". Available at: http://www.springerlink.com/content/y67740g27mjk5x87/

[87] P. Kumar, "Web services and IT management", Queue - Volume 3 , Issue 6, pp 44 - 49 , available at: http://portal.acm.org/citation.cfm?doid=1080862.1080876

[88]L. Baresi, and S. Guinea, "Dynamo and Self-Healing BPEL Compositions," *In ICSE Companion*, pp. 69-70. IEEE Computer Society, 2007.

[89] O. Moser, F. Rosenberg, and S. Dustdar, "Non-Intrusive Monitoring and Service Adaptation for WS-BPEL," *In Proceedings of the 17th International Conference on World Wide Web*, pp. 815-824, New York, NY, USA, 2008.

[90]S. Subramanian, P. Thiran, N.C. Narendra, G.K. Mostefaoui, and Z. Maamar, "On the Enhancement of BPEL Engines for Self-Healing Composite Web Services," *IEEE/IPSJ International Symposium on Applications and the Internet,* pp. 33-39, 2008 W.M.P. van der Aalst, A.H.M.

[91]N. Aghdaie and Y. Tamir, "Client-transparent fault-tolerant Web service," *In Proceedings of the IEEE International Conference on Performance, Computing, and Communications*, pp. 209-216. IEEE Computer Society, 2001.

[92]X. Zhang, D. Zagorodnov, M. Hiltunen, K. Marzullo, and R. D. Schlichting, "Fault-tolerant grid services using primarybackup: feasibility and performance," *In Proceedings of the IEEE Intl. Conf. on Cluster Computing*, pp. 105-114. IEEE-CS, 2004.

[93]B. Sabata, S. Chatterjee, M. Davis, J. J. Sydir, and T. F.Lawrence, "Taxonomy for QoS specifications," *In Proceedings of the Workshop on Object-Oriented Real-Time Dependable Systems,* pp. 100-107. IEEE Computer Society, 1997.

[94] D. Liang, C.-L. Fang, C. Chen, and F. Lin, "Fault tolerant Web service," *In Proceedings of the Tenth Asia-Pacific Software Engineering Conference*, pp. 310−319. IEEE-CS, 2003.

[95]V. Dialani, S. Miles, L. Moreau, D. D. Roure, and M. Luck, "Transparent fault tolerance for Web services based architectures," *In Proceedings of the 8th International Euro-Par Conference on Parallel Processing*, pp. 889–898. Springer, 2002.

[96]G. T. Santos, L. C. Lung, and C. Montez., "Ftweb: A fault tolerant infrastructure for Web services," *In Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, pp. 95−105, IEEE Computer Society, 2005.

[97]N. Looker, M. Munro, and J. Xu, "Increasing Web service dependability through consensus voting," *In Proceedings of the 29th Annual International Computer Software and Applications Conference,* pp. 66−69. IEEE Computer Society, 2005.

[98]J. Salas *et al.*, "WS-Replication: A framework for highly available Web services," *In Proceedings of the 15th International Conference on World Wide Web, Edinburgh, Scotland,* pp. 357−366, May 2006.

[99]M. Merideth *et al.*, "Thema: Byzantine-fault-tolerant middleware for Web services applications," *In Proceedings of the IEEE Symposium on Reliable Distributed Systems,* pp. 131−142, 2005.

[100]"Apache Geronimo v2.0 daytrader- a more complex application". Available at: https://cwiki.apache.org/GMOxDOC22/daytrader-a-more-complex-application.html

[101]"Apache Geronimo v2.0 DayTrader". Available at: https://cwiki.apache.org/GMOxDOC20/daytrader.html

[102]D. Menasce *et al.*, "In search of invariants for e-business workloads," *In Proceedings of ACM Conference Electronic Commerce*, pp. 56-65, 2000.

[103]M. F. Arlitt and C. L. Williamson, "Web server workload characterization: the search for invariants," *SIGMETRICS Perform. Eval. Rev.*, Vol. 24, No. 1, pp. 126-137, 1996.

[104]M. Arlitt and T. Jin, "Workload characterization of the 1998 world cup web site," *Technical report hpl-99-35r1, Hewlett-Packard Labs,* Septermber 1999.

[105]K. Brammer and G. Siffling, *Kalman-Bucy Filters,* Norwood MA: Artec House, 1989.

[106]S.A. DeLurgio, *Forecasting Principles and Applications,* McGraw-Hill,1998.

[107]T. Abdelzaher, Y. Lu, R. Zhang, and D. Henriksson, "Practical application of control theory to web services," *In Proceedings of the 2004 American Control Conference,* vol.3, pp. 1992-1997, 2004.

[108]R. Mehrotra, A. Dubey, S. Abdelwahed, A. Tantawi, "Model Identification for Performance Management of Distributed Enterprise Systems," *Technical Report Electrical and Computer Engineering Department technical* report ISIS-10-104, Mississippi State University, April, 2010

[109]JavaWolrd, "Java's garbage collecte heap", available at http://www.javaworld.com/javaworld/jw-08-1996/jw-08-gc.html.

[110]S. Abdelwahed, G. Karsai, and G. Biswas, "Online Safety Control of a Class of Hybrid Systems," *In Proceedings of the IEEE Conf. on CDC*, Nashiville, Vol. 2, pp. 1988-1990, 2002.

[111] M. Arlitt and T. Jin, "Workload characterization of the 1998 world cup web site," *Technical Report HPL-99-35R1*, Hewlett-Packard Labs, September 1999