A Partial Self-healing System to Remove Software Faults

A Project Report Submitted in the partial fulfillment of the requirements for the award of degree of

Master of Technology
in
Computer Science

By
Vishnu Varma P

Department of Computer and Information Sciences
University of Hyderabad
Hyderabad, India

April, 2010

CERTIFICATE

This is to certify that the project work entitled "**A Partial Self-healing System to Remove Software Faults**" being submitted to University of Hyderabad by **Vishnu Varma P** (Reg. No. 08MCMT29), in partial fulfillment for the award of the degree of Master of Technology in Computer Science, is a bonafide work carried out by him under my supervision.

Ms Rukma Rekha
Project Supervisor,
Department of CIS,
University of Hyderabad

Prof. Arun Agarwal
Head of Department,
Department of CIS,
University of Hyderabad

Prof. T. Amarnath
Dean,
School of MCIS,
University of Hyderabad

To,

My Parents and Friends.

Acknowledgments

I would like to express my sincere gratitude to **Ms Rukma Rekha**, my project supervisor, for valuable suggestions and keen personal interest throughout the progress of my course of research. She encouraged, supported, corrected and guided me during the project. The project has been a learning and growing experience for me. It is a good piece of work done in her guidance and a great opportunity to learn many things.

I am thankful to my lab mate, M.Venu and my friends for their advices and discussions with me. I would also like to thank the Open Source Community who provided the free software and documentation to work with.

I am extremely grateful to our Head of the Department, **Prof. Arun Agarwal**, for providing excellent computing facilities and a nice atmosphere for doing my project. I convey my heartfelt thanks to AI Lab staff for their help in completing the project work successfully.

I would like to take this opportunity to thank my friends who have been morale boosters for me, encouraged me to take up this course and supported me throughout this course with their love and affection. My special thanks to my wonderful parents who have always supported me in all my decisions.

Vishnu Varma P.

Abstract

Now a days information systems are becoming more complicated and most of todays information systems are networked, these systems include many resources and communication facilities which will be allocated over the network. The user wants to be able to access wide variety of services.

The self healing action can be implemented with a sequence of interactions between service manager and corresponding service when there is a failure in the service. We propose a service replacement technique for the failed network services.

Also we have calculated the System elapsed time to evaluate the performance of the system by taking the networked sorting services as a case study.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

INTRODUCTION

Autonomic Computing systems manage itself according to reach the administrator's goals. The term autonomic is derived from human biologyautonomic nervous system. ANS monitors heartbeat, body temperature, without any conscious effort on ones part. The goal of autonomic computing is to use appropriate solutions based on current state and specified policies. This autonomic concept was introduced by IBM in 2001 to further progress of IT industry is a looming software complexity crisis [7]. The need to integrate several heterogeneous environments into corporate-wide computing systems, and to extend that beyond company boundaries into the Internet, introduces new levels of complexity. Automating the management of computing resources is not a new problem for computer scientists. For decades system components and software have been evolving to deal with the increased complexity of system control, resource sharing, and operational management. Autonomic computing is just the next logical evolution of these past trends to address the increasingly complex and distributed computing environments of today.

Its time to design and build computing systems capable of running themselves, adjusting to varying circumstances, and preparing their resources to handle most effectively the workloads we put upon them.

These autonomic systems must anticipate needs and allow users to concentrate on what they want to accomplish rather than figuring how to rig the computing systems to get them there. (IBMs vision of autonomic computing, launched by IBMs senior vice president of research, Paul Horn).

With the growth of computer industry, no table examples with highly efficient networking hardware and powerful CPU's, autonomic computing has become a means to cope with the rapidly growing complexity of integrating, managing, and operating computer systems .

Four aspects of self-management, which Table 1 summarizes. Early autonomic systems may treat these aspects as distinct, with different product teams creating solutions that address each one separately. It has defined five evolutionary levels stating - Level 1 is the basic level that presents the current situation where the machines are managed manually. Level 2 - 4 is a partial autonomic machine embedded with some automated management functions. Level 5 represents the ultimate goal of Autonomic Computing i.e the self managing machine. And that's the reason of Autonomic Computing to be also called as self managing computing. Self Management is a process by which computer systems shall manage their own operation with out human intervention. The essence of autonomic computing systems is self-management, the intent of which is to free system administrators from the details of system operation and maintenance and to provide users with a machine that runs at peak performance 24/7. Autonomic systems will maintain and adjust their operation in the face of changing components, workloads, demands, and external conditions and in the face of hardware or software failures, both innocent and malicious. Self-managing systems and devices will seem completely natural and unremarkable, as will automated software and middle ware upgrades.

Self-management means different things in different fields: The number of computing devices in use is forecast to grow at 38% per annum and the average complexity of each is increasing. Currently this volume and complexity is managed by highly skilled humans; but the demand for skilled IT personnel is already outstripping supply, with labor costs exceeding equipment costs by a ratio of up to 18:1. Computing systems have brought great benefits of speed and automation but there is now an overwhelming economic need to automate their maintenance.

In The Vision of Autonomic Computing, Kephart and Chess warn that the dream of interconnectivity of computing systems and devices could become the nightmare of pervasive computing in which architects are unable to anticipate, design and maintain the complexity of interactions. They state the essence of autonomic computing is system self-management, freeing administrators from low-level task management while delivering more optimal system behavior [8].

A general problem of modern distributed computing system is that their complexity, and in particular the complexity of their management, is becoming a significant limiting factor in their further development. Large companies and institutions are employing large-scale computer networks for communication and computation. The distributed applications running on these computer networks are diverse and deal with many different tasks, ranging from internal control processes to presenting web content and to customer support.

Additionally, Mobile computing is pervading these networks at an increasing speed: employees need to communicate with their companies while they are not in their office. They do so by using laptops, PDAs, or mobile phones with diverse forms of wireless technologies to access their companies data.

This creates an enormous complexity in the overall computer network which is hard to control manually by one or more human operators. Manual control is time-consuming, expensive, and error-prone. The manual effort needed to control a growing networked computer system tends to increase very quickly. 80% of such problems in infrastructure happen at the client specific application and database layer. Most 'autonomic' service providers guarantee only up to the basic plumbing layer (power, hardware, operating system, network and basic database parameters).

1.1 Motivation

The world of Internet has given a programmer to do more complex and demanding software solutions. As networks and distributed systems grow and change, system deployment failures, hardware and software issues, and human error can increasingly hamper effective system administration. Autonomic Computing helps to address the complexity issues by using technology to manage technology. As we try to make the systems as autonomous as possible, new problems and new solutions emerge.

- Users want to be able to access the services from wide variety of devices.
- Applications are required to managed so that system will run with fewer user interventions.
- To achieve performance benefits of related services.
- The network traffic delays occurred in the previous proposed model.

- To evaluate the elapsed time for the proposed system also to get the performance benefits.

1.2 Objective

The objective of the work is using a service replacement technique for the failed network services. We have taken networked sorting services as a case study to develop the system which provides three sorting services. Also evaluated the system elapsed time.

1.3 Goal Of Autonomic Computing

An architecture for autonomic computing must accomplish three fundamental goals:

- First, it must describe the external interfaces and behaviors required of individual system components.
- Second, it must describe how to compose these components so that the components can cooperate toward the goals of system-wide self- management.
- Finally, it must describe how to compose systems from these components in such a way that the system as a whole is self-managing.

1.4 Structure Of Thesis

Chapter 2 gives the Background of new Autonomic Computing projecting its importance. This chapter also discusses about the autonomic attributes. The summary of this chapter touches the applications and challenges of the Autonomic Computing.

Chapter 3 describes the literature survey presenting the past work of the current research with the techniques used in past work and present technique and also mentions about the related work. It also stresses on the structure of the system described in current work. It will give the services provided by the system and the failure replacement technique working on the system.

Chapter 4 presents the architectural design of the System for better understanding, the design will be projected by considering an example as networked sorting services. This chapter completely deals with the features of the modules and working of current replacement algorithm, with the time complexity of failure replacement strategy and it will be compared with the previous work i.e pre-emption detection strategy. Therefore a complete view of the research is presented here.

Chapter 5 discusses the implementation details followed by the results in detail and
Chapter 6 presents the summary of the current work, conclusion and future work of the research.

Finally, the last section mentions the bibliography of the research work.

Chapter 2

BACKGROUND

2.1 Four Fundamental Elements of Autonomic Computing

In addition to possessing the eight key characteristics, a computer system must possess at least one of the four fundamentals elements of the following self- managing properties:

- **Self-Healing**: Systems discover, diagnose, and react to disruptions. For a system to be self-healing, it must be able to recover from a failed component by first detecting and isolating the failed component, taking it off line, fixing or isolating the failed component, and reintroducing the fixed or replacement component into service without any apparent application disruption. System will need to predict problems and take actions to prevent the failure from having an impact on applications. The self-healing objective must be to minimize all outages in order to keep enterprise applications up and available at all times. Developers of system components need to focus on maximizing the reliability and availability design of each hardware and software product toward continuous availability .

- **Self-Optimizing:** Systems monitor and tune resources automatically. The concept of self-optimization is when a computer environment manages resource allocations and workloads to meet the end-user needs without any or minimal human interference.

- **Self-Configuring:** Systems adapt automatically to dynamically changing environments. The idea of self-configuring is for hardware and software systems to have

the ability to define themselves at any given moment. Having the desired capabilities, allows new components to be dynamically added without any or minimal human interference.

- **Self-Protecting:** Systems anticipate, detect, identify, and protect themselves form the attacks from anywhere. The ability to detect unauthorized access, eliminates intrusions, reports such activities for each occurrence, and secured backup capabilities as the original source manager system.

2.2 Characteristics Of Autonomic Systems

An autonomic computing system consists of eight key characteristics as follows :

- To be autonomic, a computing systems needs to know itself-and comprise components that also possess a system identity.
- An autonomic computing system must configure and reconfigure itself under varying and unpredictable conditions.
- An autonomic computing system never settles for the status quo-it always looks for ways to optimize its workings.
- An autonomic computing system must perform something akin to healing-it must be able to recover from routine and extraordinary events that might cause some of its parts to malfunction.
- A virtual world is no less dangerous than the physical one, so an autonomic computing system must be an expert in self-protection.
- An autonomic computing system knows its environment and the context surrounding its activity, and acts accordingly.
- An autonomic computing system cannot exits in a hermetic (protected from outside influence) environment. Perhaps most critical for the user, an autonomic computing system will anticipate the optimized resources needed while keeping its complexity hidden.

In order to incorporate the above characteristics in self managing computing systems future autonomic computing systems will have four fundamental features.

Concept	Current computing	Autonomic computing
Self-configuration	Corporate data centers have multiple vendors and platforms. Installing, configuring, and integrating systems is time consuming and error prone.	Automated configuration of components and systems follows high-level policies. Rest of system adjusts automatically and seamlessly.
Self-optimization	Systems have hundreds of manually set, non-linear tuning parameters, and their number increases with each release.	Components and systems continually seek opportunities to improve their own performance and efficiency.
Self-healing	Problem determination is large, complex systems can take a term of programmers weeks.	System automatically detects, diagnoses, and repairs localized software and hardware problems.
Self-protection	Detection and recovery from attacks and cascading failure is manual.	System automatically defends against malicious attacks or cascading failures. It uses early warning to anticipate and prevent system wide failures.

Table 2.1: Four aspects of self-management as they are now and would be with autonomic computing.

2.3 Need Of Autonomic Computing

The growing complexity of modern networked computer systems is currently the biggest limiting factor in their expansion. The increasing heterogeneity of big corporate computer systems, the inclusion of mobile computing devices and the combination of different networking technologies like WLAN, cellular phone networks, and mobile ad hoc networks make the conventional, manual management very difficult, time consuming, and error-prone. Simply stated from above, managing complex systems has grown too costly and prone to error. People under such pressure make mistakes, increasing the potential of system outages with a concurrent impact on business. The following points will reveal more about need of autonomic computing.

It is now estimated that one-third to one-half of a company's total IT budget is spent preventing or recovering from crashes [5]. Aberdeen Group studies show that administrative cost can account for 60 to 75 percent of the overall cost of database ownership (this includes administrative tools, installation, upgrade and deployment,training, administration salaries, and service and support from database suppliers).

The autonomic computing reduces deployment and maintenance cost and increases stability of IT systems that are able to adopt and implement directives based on business policy, and are able to make modifications based on changing environment.

In a survey made on causes of outrages in four areas, most frequently found outrages are:

- For systems: operational error, user error, third party software error, internally developed software problem, inadequate change control, lack of automated processes.
- For networks: performance overload, peak load problems, insufficient bandwidth.
- For database: out of disk space, log file full, performance overload.
- For applications: application error, inadequate change control, operational error, non automated application exceptions.

The solution is Autonomic Computing systems that will have ability to configure, tune and even repair themselves. AC in approach to self managed computing systems that will work independently with out human intervention. Thus the result is the need of self managing systems and new development approaches that can deal with real life complexity and uncertainty. The goal of the Autonomic Computing is to create systems that run themselves, capable of high-level functioning while keeping the system's complexity invisible to the user. It's challenge is to produce practical methodologies and techniques for development of such self managing systems , so that they may be leveraged to deal with failure and recover easily.

2.4 Applications Of Autonomic Computing

Autonomic Computing promises to simplify the management of computing systems. But that capability will provide the basis for much more: From seamless e-sourcing and grid computing, to dynamic e-business and the ability to translate business decision that managers make to the IT processes and policies that make decisions reality.

One of the first applications of AC is e-sourcing, which is gaining importance now. E-sourcing is the ability to deliver IT as a utility, when you need it, in the amount you must have to accomplish the task at hand. Other applications include server load balancing, process allocation, monitoring power supply, automatic updating of software and

drivers, pre-failure warning, memory error-correction, automated system backup and recovery, etc. One area where Autonomic Computing can contribute significantly is Grid Computing. Grids, empowered with the self-managing capabilities of autonomics can revolutionize computing. And the applications are not just restricted to IT industry alone. For instance, a powerful grid that aims to bring advanced methods of breast cancer diagnosis and screening to patients, while reducing costs. The grid is a utility-like service delivered over the internet, enabling thousands of hospitals to store mammograms in digital form. The grid will provide analytical tools that help physicians diagnose individual cases and identify cancer 'cluster' in the population. Biometrics grid not only lead to produce new medicines for combat diseases but also develop more nutritious foods to feed the world's population. Today's software and hardware system components will evolve toward more autonomic behavior. For example- Web servers and software: Web servers can provide real-time diagnostic dashboard information, enabling the customers to more quickly become aware of resource problems, instead of relying on after the-fact reports to identify problems. problems. Once improved instrumentation is available, autonomic functions can be introduced that enable the Web server infrastructure to automatically monitor, analyze, and fix performance problems. As an example, suppose an application server is freezing-up intermittently, and no customer transactions are being processed for several seconds, thus losing thousands of dollars in business, as well as customer confidence and loyalty. Using real-time monitoring, predictive analysis, and auto tuning, the freeze-up is anticipated before it happens. The autonomic function compares real-time data with historical problem data (i.e., suggesting that the cache sizes were set too low). The settings are reset automatically without service disruption, and a report is sent to the administrator that shows what action was taken.

Database Management: New database software tools can use statistics from the databases, analyze them, and learn from the historical system performance information. The tools can help an enhanced database system automatically detects the potential bottlenecks as they are about to occur and attempt to compensate for them by adjusting tuning parameters. Query optimizers can learn the optimal index and route to certain data and automatically seek out that path based on the historical access patterns and associated response times.

Systems management: Systems management software can contain improved problem determination and data collection features designed to help businesses better diagnose and prevent interruptions (or breaches of security). Such systems management software must enable customers to take an end-to-end view of their computing environment across mul-

tiple, independently installed hardware and software elements. A bank transaction, for example, might touch a discrete database, another transaction, and Web application servers as it is processed across a network. If a problem occurs with processing on one of the individual components, lack of an integrated problem determination infrastructure makes it more difficult to determine what prevented that bank transaction from completing successfully. A consolidated view created by the system management software would enable the system and IT staffs to identify and quickly react to problems as they happen by providing an end-to-end view of the application. The end-to-end view of the environment allows companies to understand problems and performance information in the context of their business goals.

Servers: Computers can be built that need less human supervision. Computers can try to fix them selves in the event of a failure, protect themselves from hacker attacks, and configure themselves when adding new features. Servers can use software algorithms that learn patterns in Internet traffic or application usage, and provision resources in a way that gives the shortest response time to the task with the highest business priority. Server support for heterogeneous and enterprise workload management, dynamic clustering, dynamic partitioning, improved setup wizards, improved user authentication, directory integration, and other tools to protect access to network resources are all steps toward more autonomic functioning.

IBM hardware and software systems have already made significant progress in introducing autonomic computing functionality. But there is much more work ahead. The efforts to achieve cohesive system behavior must go beyond improvements in the individual components alone. These components must be federated, employing an integrating architecture that establishes the instrumentation, policy, and collaboration technologies so that groups of resources can work in concert, as for example, across systems in a grid. System management tools will play a central role in coordinating the actions of system components, providing a simplified mechanism for system administration and for translating business objectives into executable policies to govern the actions of the IT resources available.

2.5 Benefits of Autonomic Computing

Autonomic computing will produce many benefits for individuals, organizations and business. However, there will be many short term benefits and long term benefits through-

out the era of autonomic computing.

Short-term I/T related benefits:

- Simplified user experience through a more responsive, real-time system.
- Cost-savings - scale to use.
- Scaled power, storage and costs that optimize usage across both hardware and software.
- Full use of idle processing power, including home PC's, through networked system.
- Natural language queries allow deeper and more accurate returns.
- Seamless access to multiple file types. Open standards will allow users to pull data from all potential sources by re-formatting on the fly.
- Stability, High availability, High security system, Fewer system or network errors due to self-healing.

Long-term, Higher Order Benefits:

- Realize the vision of enablement by shifting available resources to higher-order business.
- Embedding autonomic capabilities in client or access devices, servers, storage systems, middleware, and the network itself. Constructing autonomic federated systems.
- Achieving end-to-end service level management.
- Collaboration and global problem-solving. Distributed computing allows for more immediate sharing of information and processing power to use complex mathematics to solve problems.
- Massive simulation - weather, medical - complex calculations like protein folding, which require processors to run 24/7 for as long as a year at a time.

2.6 Stages In Autonomic Computing Loop

- The monitor function provides the mechanisms that collect, aggregate, filter and report details (such as metrics and topologies) collected from a managed resource.

- The analyze function provides the mechanisms that correlate and model complex situations (for example, time-series forecasting and queuing models).
- These mechanisms allow the autonomic manager to learn about the IT environment and help predict future situations.
- The plan function provides the mechanisms that construct the actions needed to achieve goals and objectives. The planning mechanism uses policy information to guide its work.
- The execute function provides the mechanisms that control the execution of a plan with considerations for dynamic updates.

Chapter 3

LITERATURE SURVEY

3.1 Self-Healing System

Self-healing is one of the four features that characterize autonomic computing systems. The general view of self-healing systems is that they perform a reconfiguration step to heal a system having suffered a fault. Moreover a self-healing system should have the ability to modify its own behavior in response to changes in the environment, such as resource variability, changing user needs, mobility and system faults [5].

lifecycle of self-healing systems must consist of four major activities:

- Monitoring the system at runtime.
- Planning changes.
- Deploying the changes.
- Enacting the changes.

Self-Healing denotes the system ability to examine find, diagnose and react to system malfunctions. Self-healing components or applications must be able to observe system failures, evaluating constraints imposed by the outside, and to apply appropriate corrections. In order to automatically discover system malfunctions or possible futures failures, it is needful to know the expected system behavior. Autonomic systems must have knowledge about own behavior then they must have a knowledge in order to determine if the

actual behavior is consistent and expected in relation of the environment. In new contexts or in different scenarios, new system behaviors can be observed and the knowledge module must evolve with the environment.

3.1.1 Self-healing System Process

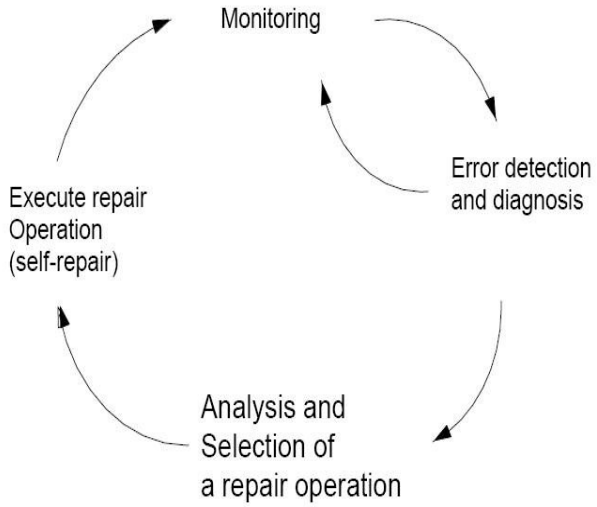


Figure 3.1: Self-healing System Process

Self-Healing systems basically endure a process in order to maintain satisfactory quality of service of the principal system during runtime in the presence of any fault. The first cycle is called the monitoring cycle. During the monitoring cycle, the systems monitor will inspect the computer environment for any improper conduct. After the monitors inspections are complete, it will send the data gathered through current observations to the next stage. The second phase of the cycle is called error detection and diagnosis; if the diagnosis reports that there is no fault in the system then it will loop back to the monitor for more observations. If there is an error detected by the monitor, the error detection cycle will report it to the next stage of the cycle. The third stage of the cycle is known as analysis and selection of a repair operation. At this stage, the fault is analyzed and a method of repairing is determined at this part of the cycle. After the repair operation is determined, the report is passed onto the finally phase of the cycle called execute repair and operation (self-repair). Any repairs that are needed are completed at this phase in the cycle. Once, the faulty areas

are self-repaired the cycle begins all over again. Since this cycle is a closed loop the, the process of self-healing environments will continuous heal itself as depicted in the above figure 3.1.

3.1.2 Disadvantages of Self-healing mechanism

The architectural models can be specialized to the particular style of the system such as reliability, performance or security. However, they have the following disadvantages:

- In system level, they are geared towards specific system topology, software applications [9].
- It is very difficult to model the target system with respect to the normal state.

3.2 Fault-tolerant system

Fault-tolerance or graceful degradation is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naively-designed system in which even a small failure can cause total breakdown. Fault-tolerance is particularly sought-after in high availability or life critical systems.

Fault-tolerance is not just a property of individual machines; it may also characterize the rules by which they interact. Recovery from errors in fault-tolerant systems can be characterized as either roll-forward or roll-back. When the system detects that it has made an error, roll-forward recovery takes the system state at that time and corrects it, to be able to move forward. Roll-back recovery reverts the system state back to some earlier, correct version, for example using check pointing, and moves forward from there. Roll-back recovery requires that the operations between the checkpoint and the detected erroneous state can be made idempotent. Some systems make use of both roll-forward and roll-back recovery for different errors or different parts of one error.
Differences between self-healing and fault tolerances:

Area	Description	Techniques
Fault tolerant computing	Ability of a system to respond gracefully to an unexpected hardware or software failure	Fault Avoidance Fault Detection Fault Masking
Self-healing systems	Automatically detects, diagnoses and repairs localized software and hardware problems	Monitoring, Diagnosis Planning, Repair Mechanism. (System evolution)

Table 3.1: Four aspects of self-management as they are now and would be with autonomic computing.

3.3 System Description

Consider different types of failing services: Services on which other services had dependencies and services that had no dependent services. We also considered the scalability of the scheme by considering varying levels of registered services in the lookup service. The services were all located in a local area network, so network delays were abstracted and therefore the results include only delays due to processing delays in the machines and the small delay incurred in LAN communication. The evaluation of the above model and associated software is not an easy task, as there is no straightforward way of evaluating distributed applications with ad hoc self-healing capabilities, nor are there any clear metrics or accepted benchmarks. However, we use elapsed time as a performance profile metric to outline the effect and overheads of ad hoc self-healing capabilities on systems performance.

For calibration purposes, prior to the evaluation a range of preliminary experiments have been conducted including:

- Running a number of trails to measure and determine the efficient operating range, tolerance and control rules applicable for instance to the sorting algorithm services.
- Defining an upper and lower performance limits for given applications, which will provide some type of knowledge for instance to guide the system to perform self-healing to maintain a specified overall system performance.
- Measuring the self-healing latency time, which is used as a nominal time tolerance measure.

- Measuring scalability of the system. Due to the complexities of networked environments, our experiments adopted a simplified model where the services were located over a Local Area Network, rather than across the Internet.

3.4 Pre-emption Detection Technique

With pre-emptive detection, the service manager checks, on a regular basis, that each of the services associated with the application is alive. If a service fails to respond to the service manager, it is assumed that the service has failed and the recovery process is started and the service manager then notifies the assembly service. The main advantage of the pre-emptive detection is that, as the service manager periodically polls the services, services may be found to be faulty prior to the moment when the application would wish to use them, therefore they can be replaced with zero replacement waiting time. On the other hand, the pre-emptive mechanism, although reducing the replacement waiting time, generates more network traffic, which may lead to congestion if there are large numbers of applications and services being used by these applications.

3.5 Pre-emption detection Algorithm

Algorithm 1: Pre-emption detection Algorithm

Step 1 : For all services S_i, i = 1,2.... n
Step 2 : int test = Generate a signal(P_i, SM)
Step 3 : For each time slice t sec.
Do Generate a signal (P_i, SM)
Step 4 : If (1500 $\leq$ input elements $\leq$ 20000)
Step 5 : If test == 1
Service is working.
BubbleSort(P_i, m)
Step 6 : ElseIf(2000 $<$ input elements $\leq$ 200000)
test = Generate a signal(P_i, SM)
if (test ==1)
Service is working.
QuickSort(P_i, SM)
Step 7 : Else if (input elements $>$ 200000)
test = Generate a signal(P_i. SM)
if (test == 1)
Service is working.
InsertionSort(P_i, SM)
Step 8 : Else if (test != 1)
Recovery(S_i, P_i)
Step 9 : DNS comparison.
If s.name(i)==s.name(j)
Replace function s.name(i) with s.name(j).
Step 10 : Goto Step 2
Step 11 : else Report to administrator
Step 12 : end if
Step 13 : end if
Step 14 : end if
Step 15 : end loop.

3.6 Case study example

As an example, we developed a JAVA application for sorting arrays of integers that may use different algorithms:

Bubble Sort, Quick Sort and Insertion Sort. Sorting algorithms are generally recognized as an important benchmark in scientific and commercial application. Each algorithm was implemented as a separate Jini based distributed service.

For each sorting algorithm, a different range of array sizes was used. This is because Bubble Sort displays much longer delays than Quick Sort and Insertion Sort for similar array sizes, and Quick Sort in turn displays much longer delays than Insertion Sort. Therefore we are not trying to compare sorting performance of the various algorithms, we are using their different performances as a means for triggering the reconfiguration of services. So we run the experiment for each service to find the ideal choice of an array size for each service. As expected, during the experiment it was noticed that the running time for each service dramatically changes when the size of the array increases.

For instance, for the Bubble Sort the suitable data set interval was from 1500 to 20,000, for Quick Sort was from 20,000 to 200,000 and for the arrays that have more than 200,000 elements Insertion Sort Service running time was the shortest.

This feature is exploited to test the self-healing property, that is, the performance drop of a given search algorithm will trigger the recover self-healing process. Implemented a set of QoS rules (policy) to trigger the self-healing process. This means for instance if the size of array exceeds 20,000 elements then the Service Manager will decide to switch the sort algorithm from Bubble sort to Quick Sort algorithm by invoking the QuickService then the system continues its operation. To measure the system latency due to self-healing startTime and endTime are invoked to measure respectively the start and end clock time of a given self-healing process.

Chapter 4

SYSTEM DESIGN

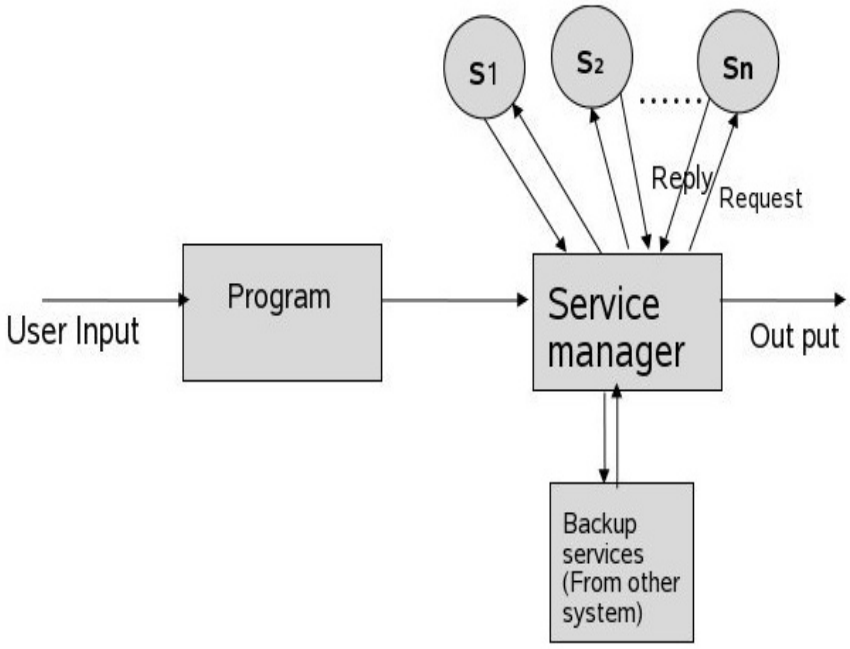
A system is nothing but a set of services. The following given system is providing n number of services. Also I have taken all these services are independent, therefore no single service is dependent on another service. This has to be taken to achieve parallel processing of services. If the input is given to the system then based on our input the service manager in the system request a particular service in the system such that the user request will be done in efficient manner. If user wants a particular service then this request will be given to service manager, then service manager checks all the services according to given request and if any failure occurred in the service then it will be reported to service manager. The service manager sends a request regarding failed service in the network then using DNS (Domain Name Search) alternative service will be replaced on failed service, therefore the system will continue its normal execution.

4.1 System Block Diagram

4.2 Service Manager

The purpose of the service manager is to verify which service is suitable to user request in the system, it also verify weather any failure is occurred in the system or not. If any abnormality in the system is occurred then it will be applied to restored such that normal process have to be done. The system is using the failure replacement technique to continue the system in normal process. Domain name based searching is carried to find the alternative of failed service.

Figure 4.1: System Design



While the system is running certain service and this service is failed while running then the service manager finds the failed service and a request is generated in the network and next system is verified the corresponding service using name based search in the system if is found in the system then it will give reply to the request. Therefore this new service is replaced in the place of failed one.

4.3 Service Replacement Technique

We have taken three networked sorting services as example in the system and proposed our model such that when the requester accessing one service then the self-healing system checks that particular service is alive or not based on the number of input elements. The assumption has been considered such that these network sorting mechanisms are independent.

If any service is found to be not working then the service manager generates a signal to replace this failed service with new service from the other system in the network. For example if bubble sorting is not working then service manager put a request on network then

then from the next system using domain name searching system will give reply, therefore this failed bubble sorting is replaced with the new one to continue the system execution.

The elapsed time of the system includes all system running time, failure detection time and service replacement time.

4.4 Explaining the system with Case study example

- Each sorting algorithm was implemented as an individual Jini service (BubbleService, QuickService and InsertionService).
- A randomly chosen size of array was passed for sorting and each service was invoked separately.
- The time in milliseconds for the service invocation and sorting was measured in each case.
- The measured elapsed time is used to identify the boundary range and algorithm intervals. For each algorithm, the measured elapsed time can indicate the time performance profile and thus the efficiency boundary of each algorithm. So the measurement of the elapsed time is used as measurement metric to generate the time limitation for each sorting process, as soon as one service reaches the limit another service is required to be invoked. Only for the purpose of the experiment and evaluation it is assumed that reaching the limit of one service causes the failure. So in order for the system to continue functioning the other service should be found and invoked.
- At this point the Service Manager is sent an event notification (using the Jini Remote Event) of which service is required to be invoked. The system does the service discovery through service name and then invokes this service. This process involves the following steps:

    1. Discover the service by name.
    2. Parse the document and get the invocation method.
    3. Pass this method to the Invocation Service.

- The latency (elapsed time) for all these processes is measure.

4.5 Service Replacement Algorithm

Algorithm 2: Service Replacement Algorithm

Step 1 : For all services S_i, i = 1,2.... n
Step 2 : If (1500¡= input elements $\leq$ 20000)
BubbleSort(p_i, m)
Step 3 : int test = Generate a signal(P_i, SM)
Step 4 : If (test ==1)
Service is working.
Step 5 : Else If(20000 l input elements $\leq$ 200000)
QuickSort(P_i, SM)
test = Generate a signal(P_i, SM)
if (test == 1)
Service is working.
Step 6 : Else if (input elements 200000)
InsertionSort(P_i, SM)
test = Generate a signal(P_i, SM)
if (test == 1)
Service is working.
Step 7 : Else if (test != 1)
Recovery(S_i, P_i)
Step 8 : DNS comparison.
If (s.name(i)==s.name(j))
Replace function s.name(i) with s.name(j).
Step 9 : Goto Step 2
Step 10: else Report to administrator
Step 11: end if
Step 12: end if
Step 13: end if
Step 14 : end loop.

Here,

S_1, S_2, S_3 . . . . . so on are services in the system.

P_1, P_2, P_3 . . . . . P_i are port numbers of service

SM is the service manager.

DNS is Domain Name System.

Chapter 5

IMPLEMENTATION AND RESULTS

To justify the performance benifits of the service replacement strategy here we have taken three sorting services bubble sort, quick sort ant insertion sort as example and the preformance of these individual services are given as tabular format below. Then by taking these services into the system again implemented the performance improvement of the system with the reference of the system elapsed time.

By taking average of muntiple number of observations system elapsed time is calculated. This elapsed is compared with individual services in the system. This elapsed time in milli seconds are taken in the following table to plot the system response graph.

No. of elements	BubbleService	QuickService	InsertionService	Service Replacement
2000	155	175	194	182
5000	380	412	486	407
10000	870	890	942	1027
20000	2500	1540	1660	1602
40000	6907	2357	3381	2399
80000	20742	4557	6990	4612
100000	20990	5650	7122	5659
200000	29980	9960	7920	8120
500000	606127	217391	11320	11425

Table 5.1: Output elapsed time(in milli seconds) for the different sorting services for various range of inputs and performance of the service replacement strategy.

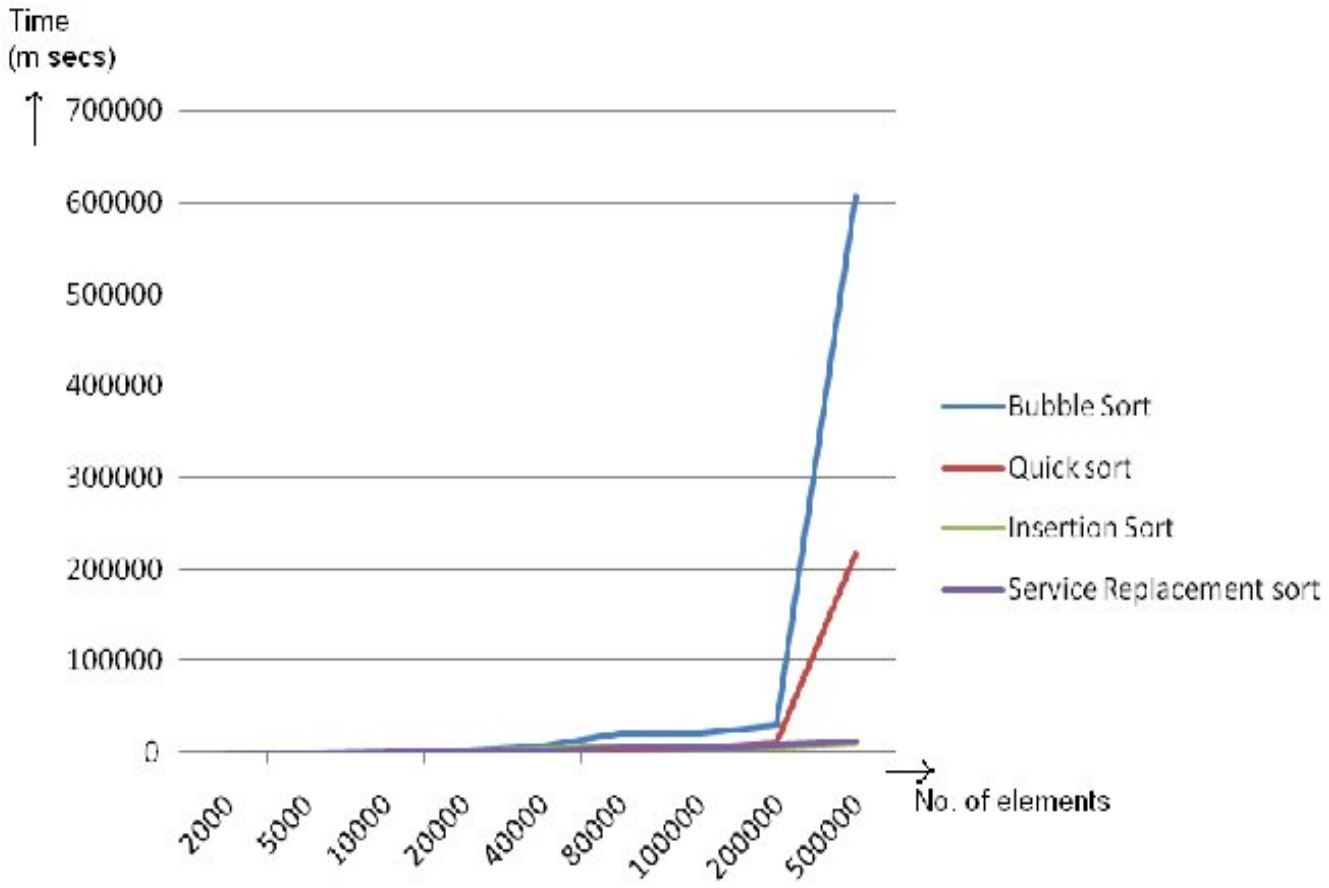


Figure 5.1: System response for different inputs i.e number of input elements Vs elapsed time.

5.1 Time Complexity Of Pre-emption Detection Technique

Since, the pre-emption detection technique checks the system in regular intervals of time Therefore,

Total elapsed time = System Running Time + (System Checking Time * n)
Here, n is Number of times that the checking of System done.

If any failure of a service is found in the System then the particular service will be replaced by the backup service. There fore,

Total elapsed time = System Running Time + (System Checking Time * n) + Service Replacement Time.

By taking case study example,

Best Case :

Total elapsed time = System Running Time + [O(n^2) + O(nlog(n)) + O(n))] * n + O(1)

Worst Case :

Total elapsed time = System Running Time + [(O(n^2) + O(nlog(n)) + O(n^2))] * n + O(1)

5.2 Time Complexity Of Service Replacement Strategy

Here the System continue its operation till the failure of any service occurred, if any service is found to be fail then the corresponding service will be replaced by using name based search in the system. Therefore, by considering service failure is occurred,

Total elapsed time = System Running Time + Alternative Service Checking Time + Service Replacement Time.

Here the System running time will vary according to the number of input elements. Because, the sorting service is varied according to the number of input elements.

If the System uses the Bubble sorting service Then,

Time complexity = O(n^2) + O(n) +O(1)

If the System uses the Quick sorting service Then,

Time complexity = O(nlog(n)) + O(n) +O(1)

If the System uses the Insertion sorting service Then,

For **Best Case :** Time complexity = O(n) + O(n) + O(1)

For **Worst Case :** Time complexity = O(n^2) + O(n) + O(1)

O(1) is the service replacement time.

O(n^2) is the time complexity of bubble sorting service.

O(nlog(n)) is the time complexity of quick sorting service.

5.3 Differences Of Both Techniques

Pre-emption Detection Technique	**Service Replacement Strategy**
With pre-emptive detection, the service manager checks, the system in regular intervals of time	Here the service if found to be failed then only service manager checks for alternative service to continue the system execution.
Networked traffic delays will be more. Because fault detection can be done in regular intervals of time.	Here the networked traffic delays are less but if any service failure occurred then traffic delays will be increased.
Here it will take zero service replacement waiting time.	Here service replacement waiting time is the time to search the alternative service.

Table 5.2: Differences Between Pre-emption Detection Technique and Service Replacement Strategy

Chapter 6

CONCLUSION AND FUTURE WORK

We discussed the benefits of service replacement approach to the development of the networked sorting system with different services. We showed the performance of bubble sort, quick sort, insertion sort and these are compared with our networked system. The performance benefits of service replacement approach are evaluated.

The System can be implemented by taking the dependent services in to the system as consideration.

Bibliography

[1] Sun Microsystems, Jini Network Technology, 2000. http://wwws.sun.com/software/jini.

[2] An Architectural Blueprint For Autonomic Computing, June 2006. http://www.ginkgo-networks.com/IMG/pdf/AC_Blueprint_White_PaperV7.pd

[3] Fault tolerant systems concepts, June 2006. http://www.wikipedia.org/wiki/Fault-tolerant_system.

[4] Ganek AG and Corbi TA. The drawing of the autonomic computing era. *IBM System Journal*, 2003.

[5] Pereira Grishikashvili.E, Pereira R.B, and Taleb-A Bendiab. Performance Evaluation For Self-Healing Distributed Services And Fault Detection Mechanisms. *Journal of Computer and System Sciences*, 72:1172–1182, 2006.

[6] Horn and Paul. Autonomic computing: Ibm's perspective on the state of information technology. *In IBM Research*, 2001.

[7] Jeffrey O. Kephart and David M. Chess. The Vision Of Autonomic Computing. *IBM Corporation*, 2001.

[8] P. Koopman. Elements of the self-healing system problem space. *In:ICSE WADS03*, 2003.

[9] Rajesh Kumar Ravi and Vinaya Sathyanarayana. Container based framework for self-healing software system. *the 10th IEEE International workshop on Future Trends of Distributed Computing system*, pages 306–310, 2004.

[10] M. Parashar H. Tianfiels Sterritt, Roy and R. Unland. A concise introduction to autonomic computing. *Advanced Engineering Informatics*, 2005.