Reliable Web Services by Fault Tolerant Techniques: Methodology, Experiment, Modeling and Evaluation

Term Paper (Spring 2006)

By

Chan Pik Wah

Supervised by Prof. Michael R. Lyu

Department of Computer Science and Engineering

The Chinese University of Hong Kong Shatin, N.T., Hong Kong

Abstract

With ever growing use of Internet, Web services become increasingly popular and their growth rate surpasses even the most optimistic predictions. Services are self-descriptive, self-contained, platform-independent and openly-available components that interact over the network. They are written strictly according to open specifications and/or standards and provide important and often critical functions for many business-to-business systems. Failures causing either service downtime or producing invalid results in such systems may range from a mere inconvenience to significant monetary penalties or even loss of human lives. In applications where sensing and control of machines and other devices take place via services, making the services highly dependable is one of main critical goals. Currently, there is no experimental investigation to evaluate the reliability and availability of Web services systems. In this paper, we identify parameters impacting the Web services dependability, describe the methods of dependability enhancement by redundancy in space and redundancy in time and perform a series of experiments to evaluate the availability of Web services. To increase the availability of the Web service, we use several replication schemes and compare them with a single service. The Web services are coordinated by a replication manager. The replication algorithm and the detailed system configuration are described in this paper.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

With service-oriented computing becoming a reality, there is an increasing demand for dependability. Service-oriented Architectures (SOA) are based on a simple model of roles. Every service may assume one or more roles such as being a service provider, a broker or a user (requestor).

This model simplifies interoperability as only standard communication protocols and simple broker-request architectures are needed to facilitate exchange (trade) of services. Not surprisingly, the use of services, especially Web services, became a common practice. The expectations are that services will dominate software industry within the next five years. As services begin to pervade all aspects of life, the problems of service dependability, security and timeliness are becoming critical and appropriate solutions need to be found.

Several fault tolerance approaches have been proposed for Web services in

the literature [1, 2, 3, 4, 5, 6], but the field still requires appropriate theory, appropriate models, effective design paradigms, a practical implementation, and an in-depth experimentation for building highly-dependable Web services.

In this paper, we propose such experimental setting and offer a roadmap to dependable Web services. We compose a list of parameters that are closely related to evaluating quality of service from the dependability perspective. We focus on service availability and timeliness and consider them a cornerstone of our approach. Security, which can also be viewed as a part of dependability, is beyond the scope of this paper.

1.2 Research Objective

There are many fault-tolerant techniques that can be applied to Web services including replication and diversity. Replication is one of the efficient ways for creating reliable systems by time or space redundancy. Redundancy has long been used as a means of increasing the availability of distributed systems, with key components being re-executed (replication in time) or replicated (replication in space) to protect against hardware malfunctions or transient system faults. Another efficient technique is design diversity. By independently designing software systems or services with different programming teams, diversity provides an ultimate resort in defending against permanent software design faults. In this paper, we focus on the systematic analysis of the replication techniques when applied to Web services. We analyze the performance and the availability of the Web services using spatial and tem-

poral redundancy and study the tradeoffs between them. A generic Web service system with spatial as well as temporal replication is proposed and experimented with.

1.3 Contribution

In the paper, we have the following main contributions:

- Survey on reliability methodologies
- Survey on Web Services reliability
- Proposed an architecture for dependable Web Services
- Develop a reliability model for the proposed scheme

1.4 Structure of Report

The rest of the paper is organized as follows. Related work and the background is presented in Chapter 2. Failure respond stage of Web Services is introduced in Chapter 3. We then review methodologies for reliable Web services, present our approach with a list of key parameters, and propose a roadmap for further development and experimentation in Chapter 4. In Chapter 5 we describe the architecture and system configuration, and then we document execution of Web service experiments and present the results. Reliability Model of the proposed system in Chapter 6. Finally, in Section 7 we draw some conclusions and sketch the outlook.

Chapter 2

Background

Web service is a self-contained, modular application built on deployed network infrastructure including XML and HTTP. It uses open standards for description (Web Service Definition Language, WSDL), discovery (Universal Description, Discovery, and Integration, UDDI) and invocation (Simple Object Access Protocol, SOAP). Web service becomes more popular and fault tolerance becomes essential property for a Web service. There are different proposed approaches for improving the reliability of the web services, including N-version programming, replication, reliable messaging, message ordering and duplicate elimination.

Let first have brief introduction of Web Service, an overview of state-of-the-arts technologies in reliability and a literature review of current reliable Web Service systems.

2.1 Web Services

Web Services are self-contained business function that operate over the Internet. They are written to strict open specifications to work together and with other similar kinds of components.

Web Services are useful to business as they enable systems in different companies to interact with each other, more importantly, in a far easier way than before. With business needing closer operation between suppliers and customers, engaging in more joint ventures, and facing the prospect of more mergers and acquisitions, companies need the capability to link up their established systems quickly and efficiently with other companies. Thus Web Services give companies the capability do more e-business, with more potential business partners, in more and different ways than before, and at reasonable cost.

The recent growth in use of the www on the internet has cased a significant increase in the demand on web services Web services have gained high popularity in the development of distributed application systems. Some critical applications also consider using web services paradigm due to the benefit of interoperability, reusability, and adaptability. To support critical applications, existing web service model needs to be extended to assure survivability.

2.1.1 Technologies in Web Services

A Web Service is programmable application logic accessible using standard Internet protocols. Web Services combine the best aspects of component-

based development and the Web. Like components, Web Services represent black-box functionality that can be reused without worrying about how the service is implemented. Unlike current component technologies, Web Services are not accessed via object-model-specific protocols, such as DCOM, RMI, or IIOP. Instead, Web Services are accessed via ubiquitous Web protocols (such as HTTP) and data formats (such as XML).

A Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service performs a specific task or a set of tasks. A Web service is described using a standard, formal XML notation, called its service description, that provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location. Web service descriptions are expressed in WSDL.

A service provider creates a Web service and its service definition and then publishes the service with a service registry based on a standard called the Universal Description, Discovery, and Integration (UDDI) specification.

Once a Web service is published, a service requester may find the service via the UDDI interface. The UDDI registry provides the service requester with a WSDL service description and a URL (uniform resource locator) pointing to the service itself. The service requester may then use this information to directly bind to the service and invoke it.

The architecture of Web Service is shown in Figure 2.1

Properties of Web Services:

- Perform encapsulated business functions using request/reply as well as

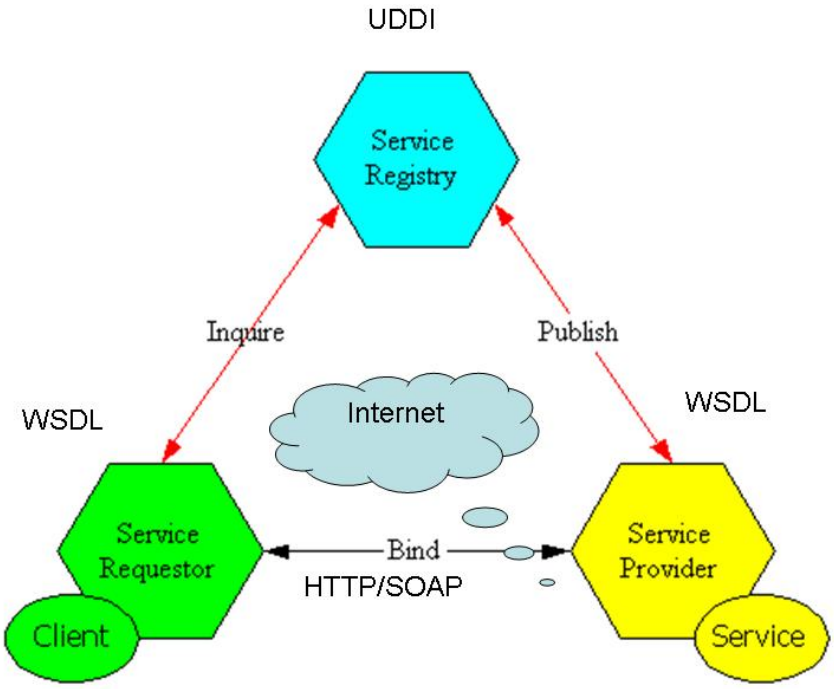

Figure 2.1: Architecture of Web Service

business process interactions

- Looser coupling via less reliance on pre-defined interfaces
- Can be mixed and matched to create complete process
- Enable dynamic integration through embedded capability of service discovery and binding

2.1.2 Problem of current Web Services

Transaction

Atomicity is not provided. The key point is that HTTP is stateless, however, business processes or transactions are useful.

Security

Add-on measures such as Encryption are needed to deal with the insecure Internet transportation

Interoperability

IBM, Microsoft, intel, BEA and other companies formed the Web Services Interoperability Organization (WS-I), a non-profit organization for promoting Web Services standard. The idea behind WS-I was not create new standards, but rather to assemble "profiles" of standards from the W3C, OASIS and others. The profiles are sets of related standard against which conformance tests and certifications can be established.

Reliability

The Internet is inherently unreliable. Currently there is no singel underlying transport protocols (HTTP, FTP, SMTP) addresses all reliability issues, namely guaranteed delivery, ordered delivery, and duplicate elimination. For collaborative e-business and e-transaction scenarios, message reliability becomes a criticak issues.

2.2 Methodologies for reliable Web Services

2.2.1 Introduction

Reliability is a measure of the success with which the system conforms to some authoritative specification. Reliability engineering provides the theoretical and practical tools whereby the probability and capability of parts, components, equipment, products and systems to perform their required functions for desired periods of time without failure, in specified environments and with a desired confidence, can be specified, designed in, predicted, tested and demonstrated. [25]

Reliability includes:

- Guaranteed delivery: ensure that all information to be sent actually received by the destination or error reported.
- Duplicate elimination: ensure that all duplicated information can be detected and filtered out
- Ordering: communication between parties consist of several individual

message exchanges. This aspect ensure that Message Exchanges are forwarded to the receiver application in the same order as the sender application issued.

- Crash tolerance: ensures that all information prescribed by the protocol is always available regardless of possible physical machine failure
- State synchronization: if the MEP is cancelled for any reason then it is desirable for both nodes to set their state as if there were no communication between the parties.

There are number of method can be applied in the reliability issues, including:

- Redundancy
- Diversity

The following sections will discuss these techniques.

2.2.2 Redundancy

It is a well-known fact that fault tolerance can be achieved via spatial or temporal redundancy, including replication of hardware (with additional components), software (with special programs), and time (with the repetition of operations) [8, 9, 10].

Spatial redundancy can be dynamic or static. Both use replication but in static redundancy, all replicas are active at the same time and voting takes place to obtain a correct result. The number of replicas is usually odd as

and the approach is known as n-modular redundancy. For example, under a single fault assumption, if services are triplicated and one of them fails the remaining two will still guarantee the correct result. The associated spatial redundancy cost is high (three copies plus a voter). The time overhead of managing redundant modules such as voting and synchronization is also considerably large for static redundancy. Dynamic redundancy, on the other hand, engages one active replica at one time while others are kept in an active or in standby state. If one replica fails, another replica can be employed immediately with little impact on response time. In the second case, if the active replica fails, a previously inactive replica must be initialized and take over the operations. Although this approach may be more flexible and less expensive than static redundancy, its cost may still be high due to the possibility of hastily eliminating modules with transient faults. It may also increase the recovery time because of its dependence on time-consuming error-handling stages such as fault diagnosis, system reconfiguration, and resumption of execution.

Redundancy can be achieved by replicating hardware modules to provide backup capacity when a failure occurs, or redundancy can be obtained using software solutions to replicate key elements of a business process.

2.2.3 Diversity

In any redundant systems, common-mode failures (CMFs) result from failures that affect more than one module at the same time, generally due to a common cause. These include design mistakes and operational failures that

may be caused externally or internally. Design diversity has been proposed in the past to protect redundant systems against common-mode failures [11, 12] and has been used in both hardware and software systems [13, 14]. The basic idea is that, with different designs and implementations, common failure modes will probably cause different error effects. One of the design diversity techniques is N-version programming, and another one is Recovery Blocks. The key element of N-version programming or Recovery Block approaches is diversity. By attempting to make the development processes diverse it is hoped that the independently designed versions will also contain diverse faults. It is assumed that such diverse faults will minimize the likelihood of coincident failures.

2.3 Related Work

A Web Services Reliable Messaging Protocol is proposed in [1], which using flooding and acknowledgement ensures that messages are delivered reliably between distributed applications in the presence of software component, system, or network failures.

This specification (WS-ReliableMessaging) describes a protocol that allows messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures. The protocol is described in this specification in a transport-independent manner allowing it to be implemented using different network technologies. To support interoperable Web services, a SOAP binding is defined within this specification. The protocol defined in this specification depends upon other Web services

specifications for the identification of service endpoint addresses and policies. How these are identified and retrieved are detailed within those specifications and are out of scope for this document.

WS-FTM (Web Service-Fault Tolerance Mechanism) is an implementation of the classic n-Version model for use with Web Services [2] which can easily be applied to systems with a minimal change. The Web Services are implemented in different versions and the voting mechanism is in the client program.

FT-SOAP [3] is aimed at improving the reliability of the SOAP when using Web service. The system includes different function replication management, fault management, logging/recovery mechanism and client fault tolerance transparency.

FT-SOAP is based on the work of FT-CORBA [24], a fault-tolerant SOAP based middleware platform is proposed. There are two major targets in FT-SOAP: 1) to define a fault-tolerant SOAP service standard recommendation, and 2) to implement an FT-SOAP service prototype.

FT-Grid [7] is another design, which is a development of design-diverse fault tolerance in Grid. It is not specified for Web services but the techniques are applicable to Web Services. The FT-Grid allows a user to manually search through any number of public or private UDDI repositories, select a number of functionally-equivalent services, choose the parameters to supply to each service, and invoke those services. The application can then perform voting on the results returned by the services, with the aim of filtering out any anomalous results.

2.3.1 Summary

Although a number of approaches have been proposed for increasing the Web service reliability, there is a need for systematic modeling and experiments to understand the tradeoffs and verify reliability of the proposed methods.

Chapter 3

Failure Response Stages of Web Services

Web services will go through different stages when failure occurred and the failure response of Web services can be classified into different stages [15]. When failure occurred, the Web service is confined Fault detection techniques are applied to find out the failure causes and the failed components are repaired or recovered. Then, reconfiguration, restart and reintegration will be done. The flow of the failure response of Web service is shown in Figure 3.1 and the details of each stage are described as follows:

3.1 Fault confinement

This stage limits the spread of fault effects to one area of the Web service, thus preventing contamination of other areas. Fault-confinement can be achieved through use of: fault detection within the Web services, consistency checks

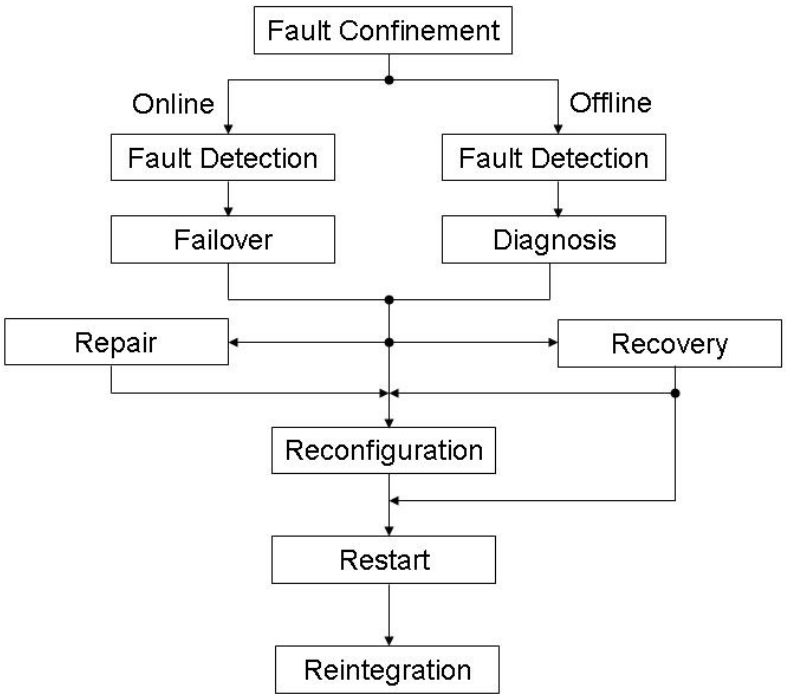

Figure 3.1: Flow of the failure response of Web services.

and multiple requests/confirmations.

3.2 Fault detection

This stage recognizes that something unexpected has occurred in a Web service. Fault latency is the period of time between the occurrence of a fault and its detection. Techniques fall in two classes: off-line and on-line. With off-line techniques, such as diagnostic programs, the service is not able to perform useful work while under test. On-line techniques, such as duplication, provide a real-time detection capability that is performed concurrently with useful work.

3.3 Diagnosis

This stage is necessary if the fault detection technique does not provide information about the fault location.

3.4 Reconfiguration

This stage occurs when a fault is detected and located. The Web services can be composed of different components. When providing the service, there may be a fault in individual components. The system may reconfigure its components either to replace the failed component or to isolate it from the rest of the system.

3.5 Recovery

This stage utilizes techniques to eliminate the effects of faults. Two basic recovery approaches are based on: fault masking, retry and rollback. Fault-masking techniques hide the effects of failures by allowing redundant information to outweigh the incorrect information. Web services can be replicated or implemented with different versions (NVP). Retry undertakes a second attempt at an operation and is based on the premise that many faults are transient in nature. Web services provide services through network; retry would be a practical approach as requests/reply may be affected by the state of the network. Rollback makes use of the fact that the Web service operation is backed up (checkpointed) at some point in its processing prior to fault detection and operation recommences from that point. Fault latency is important here because the rollback must go back far enough to avoid the effects of undetected errors that occurred before the detected error.

3.6 Restart

This stage occurs after the recovery of undamaged information.

- Hot restart: resumption of all operations from the point of fault detection and is possible only if no damage has occurred.
- Warm restart: only some of the processes can be resumed without loss.
- Cold restart: complete reload of the system with no processes surviving. The Web services can be restarted by rebooting the server.

3.7 Repair

At this stage, a failed component is replaced. Repair can be off-line or on-line. Web services can be component-based and consist of other Web services. In off-line repair either the Web service will continue if the failed component/sub-Web service is not necessary for operation or the Web services must be brought down to perform the repair. In on-line repair the component/sub-Web service may be replaced immediately with a backup spare or operation may continue without the component. With on-line repair Web service operation is not interrupted.

3.8 Reintegration

At this stage the repaired module must be reintegrated into the Web service. For on-line repair, reintegration must be performed without interrupting Web service operation.

Chapter 4

Dependable Web Services Architecture

4.1 Introduction

The previous chapter, the stages represent a general approach to system fault tolerance. In the following section, we propose a replication Web service system for reliable Web services. In our system, the dynamic hardware approach is considered and its architecture is shown in Figure 4.1.

4.2 Scheme details

In this system, the Web service is replicated on different machines, but only one Web service provides service at a time, which is called primary Web service. The Web service is replicated identically on different machines; therefore, when the primary Web service fails, the other Web services can imme-

diately provide the required service. The replication mechanism shortens the recovery time and increases the availability of the system.

The main component of this system is the replication manager (RM), which acts as a coordinator of the Web services. The replication manager is responsible for:

1. Creating a Web service.
2. Choosing (with anycasting algorithm) the best (fastest, most robust, etc.) Web service [13] to provide the service which is called the primary Web service.
3. Registering the Web Service Definition Language (WSDL) with the Universal Description, Discovery, and Integration (UDDI)
4. Continuously checking the availability of the primary Web service by using a watchdog.
5. Selecting another service if the primary service fails to ensure fault tolerance.

When the primary Web service fails, the replication manager selects the best suitable Web service again to continue providing the service. The replication manager maps the new address of the new primary Web service to the WSDL, thus the clients can still access the Web service with the same URL. This failover process is transparent to the users. The detail procedure is shown in Figure 4.1.

The work flow of the replication manager is shown in Figure 4.2. The replication manager is running on a server, it keeps checking the availability

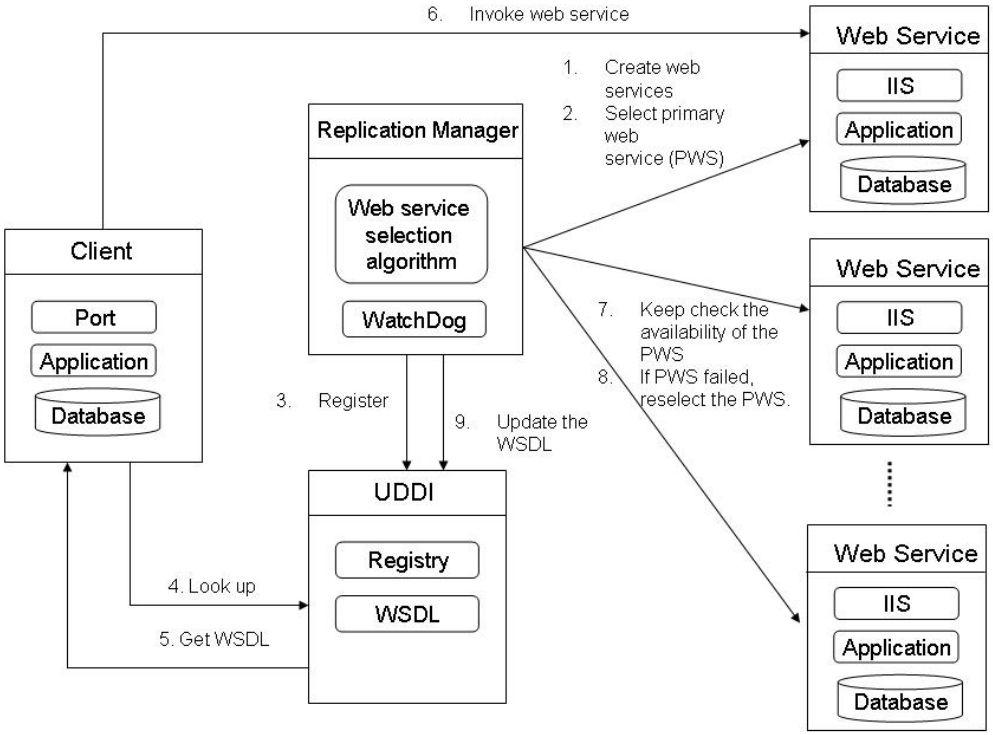

Figure 4.1: Proposed architecture for dependable Web services.

of the web services by the polling method. It sends messages to the web services periodically. If it does not get the reply from the primary Web Service, it will select another Web service to replace the primary one and map the new address to the WSDL. The system is considered failed if all the Web Services have failed.

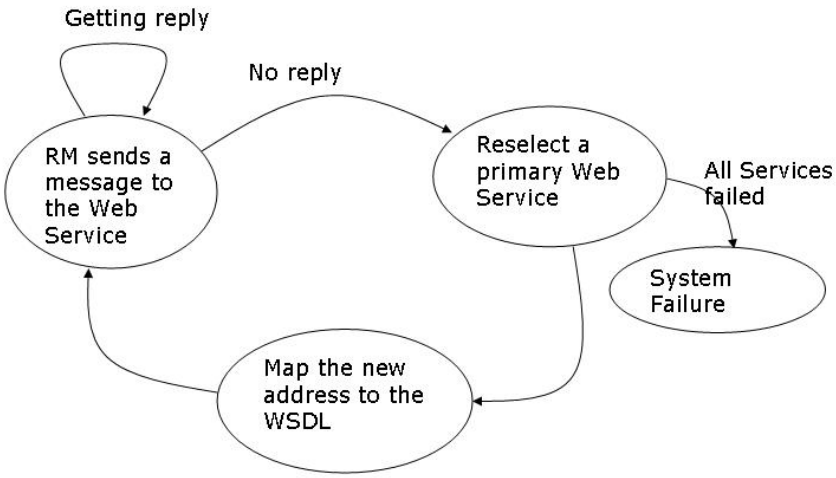

Figure 4.2: Workflow of the Replication Manager

4.3 Roadmap for Experimental Research

We take a pragmatic approach by starting with a single service without any replication. The only approach to fault tolerance in this case is the use of redundancy in time. If a service is considered as an atomic action or a transaction where the input is clearly defined, no interaction is allowed during its execution and the termination has two outcomes: correct or incorrect. In this case, the only way to make such service fault tolerant is to retry or reboot it. This approach allows tolerance of temporary faults, but it will not be sufficient for tolerating permanent faults within a server or a service. One issue is how much delay can a user tolerate, and another issue is the optimization of the retry or the reboot time; in other words, deciding when a current request should be timed out. By handling services as atomic transactions, exception handling does not help in dealing directly with inherent problems within a service, it is only possible to perform it at incorrect termination points or at timeout by re-execution using a retry or a reboot.

If redundancy in time is not sufficient to meet dependability requirements or time overhead is unacceptable, the next step is redundancy in space. Redundancy in space for services means replication where multiple copies of a given service may be executed sequentially or in parallel. If the copies of the same services are executed on different servers, different modes of operations are possible:

1. Sequentially, meaning that we await a response from a primary service and in case of timeout or a service delivering incorrect results, we invoke a back up service (multiple backup copies are possible). It is often called

failover.

2. In parallel, meaning that multiple services are executed simultaneously and if primary service fails, the next one takes over. It is also called a failover. Another variant is that the service whose response arrives first is taken.

3. There is also a possibility of majority voting using n modular redundancy (NMR), where results are compared and the final outcome is based on at least n/2 + 1 services agreeing on the result.

4. If diversified versions of different services are compared, the approach can be seen as either a Recovery Block (RB) system where backup services are engaged sequentially until the results are accepted (by an Acceptance Test), or an N-version programming (NVP) system where voting takes place and majority results are taken as the final outcome. In case of failure, the failed service can be masked and the processing can continue.

NVP and RB have undergone various challenges and vivid discussions. Critics would state that the development of multiple versions is too expensive and dependability improvement is questionable in comparison to a single version, provided the development effort equals the development cost of the multiple versions. We argue that in the age of service-oriented computing, diversified Web services permeate and the objections to NVP or RB become obsolete. Based on market needs, service providers competitively and independently develop their services and make them available to the market.

With abundance of services for specific functional requirements, it is apparent that fault tolerance by design diversity will be a natural choice. NVP should be applied to services not only for dependability but also for higher performance.

Finally, a hybrid method may be used where both space and time redundancy are applied and depending on system parameters a retry might be more effective before switching to the back up service. This type of approach will require a separate study.

We also need to formulate several additional quality-of-service parameters to service customers. We propose a number of fault injection experiments showing both dependability and performance with and without diversified Web services. The outlined roadmap to fault-tolerant services leads to ultra reliable services where hybrid techniques of spatial and time redundancy can be used for optimizing cost-effectiveness tradeoffs. In the next section, we describe the various approaches and some experiments in more detail.

Chapter 5

Experiments

A series of experiments are designed and performed for evaluating the reliability of the Web service, including single service without replication, single service with retry or reboot, and a service with spatial replication. We will also perform retry or failover when the Web service is down. A summary of five (0-4) experiments is stated in Table 5.1.

Our experimental system is implemented with Visual Studio .Net and runs with .Net framework. The Web service is replicated on different machines and the primary Web service is chosen by the replication manager.

Table 5.1: Summary of the experiments

		None	Retry/Reboot	Failover	Both(hybrid)
0	Single service, no retry	0	–	–	–
1	Single service with retry	–	1	–	–
2	Single service with reboot	–	2	–	–
3	Spatial replication	–	–	3	4

Table 5.2: Parameters of the experiments

	Parameters	Current setting/metric
1	Request frequency	1 req/min
2	Polling frequency	5 ms
3	Number of replicas	5
4	Client timeout period for retry	10 s
5	Failure rate λ	number of failures/hour
6	Load (profile of the program)	percentage or load function
7	Reboot time	10 min
8	Failover time	1 s

In the experiments, faults are injected in the system and the fault injection techniques are similar, for example, to the ones referred in [4, 20]. A number of faults may occur in the Web service environment [18, 19], including network problem, resource problem, entry point failure, and component failure. These faults are injected in the experimental system to evaluate the reliability of our proposed scheme. Our experimental environment is defined by a set of parameters. Table 5.2 shows the parameters of the Web service in our experiments.

5.1 Experimental results

We compare five approaches for providing the Web services. The details of the experiments are described in as follows:

0. Single service without retry and reboot

   The Web service is provided by a single server without any replication. No redundancy technique is applied to this Web service.

1. Single service with retry

   Web service provides the service and the client retries another Web service when there is no response from the original Web service after timeout.

2. Single service with reboot

   Web service provides the service and the Web service server will reboot when there is no response from the Web service. Clients will not retry after timeout when there is no response from the service.

3. Spatial replication with failover

   We use a generic spatial replication: The Web service is replicated on different machines and it is transferred to another machine when the primary Web service fails (failover). The replication manager coordinates among the replicas and carries out a failover in case of a failure. Clients will only submit the request once and will not retry.

4. Spatial replication with failover and retry (hybrid approach)

   Similar to the Experiment 3 where the Web service is replicated on different machines and it is transferred to another one (failover) when the primary Web service fails. But the client will retry if there is no response from the Web service after timeout.

The Web services were executed for 360 hours generating a total of 360x60 req/hr = 43200 requests from the client. A single failure is counted when the system cannot reply to the client. For the approach with retry, a single

Table 5.3: Experimental results

Experiments over 360 hour period (43200 reqs)	Normal	Resource Problem	Entry Point Failure	Network Level Fault Injection
Exp0	4928	6130	6492	5324
Exp1	2210	2327	2658	2289
Exp2	2561	3160	3323	5211
Exp3	1324	1711	1658	5258
Exp4	1089	1148	1325	2210

failure is counted when a client retries five times and still cannot get the result. A summary of the results is shown in Table 5.3 and the Figures 5.1 to 5.4 show the number of failures as the time increases.

Reliability of Web services are tested under different scenarios, including normal operation, resource problem by increasing the load of the server, entry point failure by rebooting the server periodically and a number of faults that are injected to the system by fault injection techniques.

In the fault injection, WS-FIT fault injection is applied. The WS-FIT fault injection method is a modified of Network Level Fault Injection. WS-FIT differs from standard Network Level Fault Injection techniques in that the fault injector decodes the SOAP message and can inject faults into individual RPC parameters, rather than randomly corrupting a message, for instance bit-flipping. This enables API-level parameter value modification to be performed in a non-invasive way as well as standard Network Level Fault Injection.

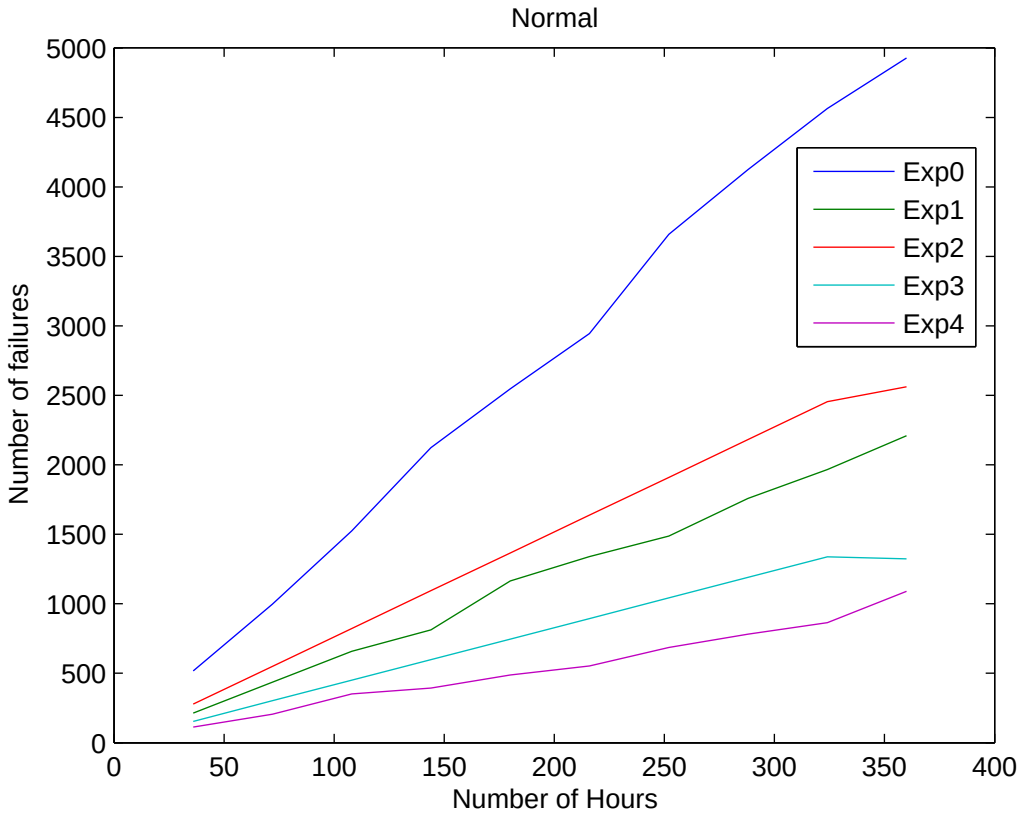

Figure 5.1: Number of failures when the server operates normally

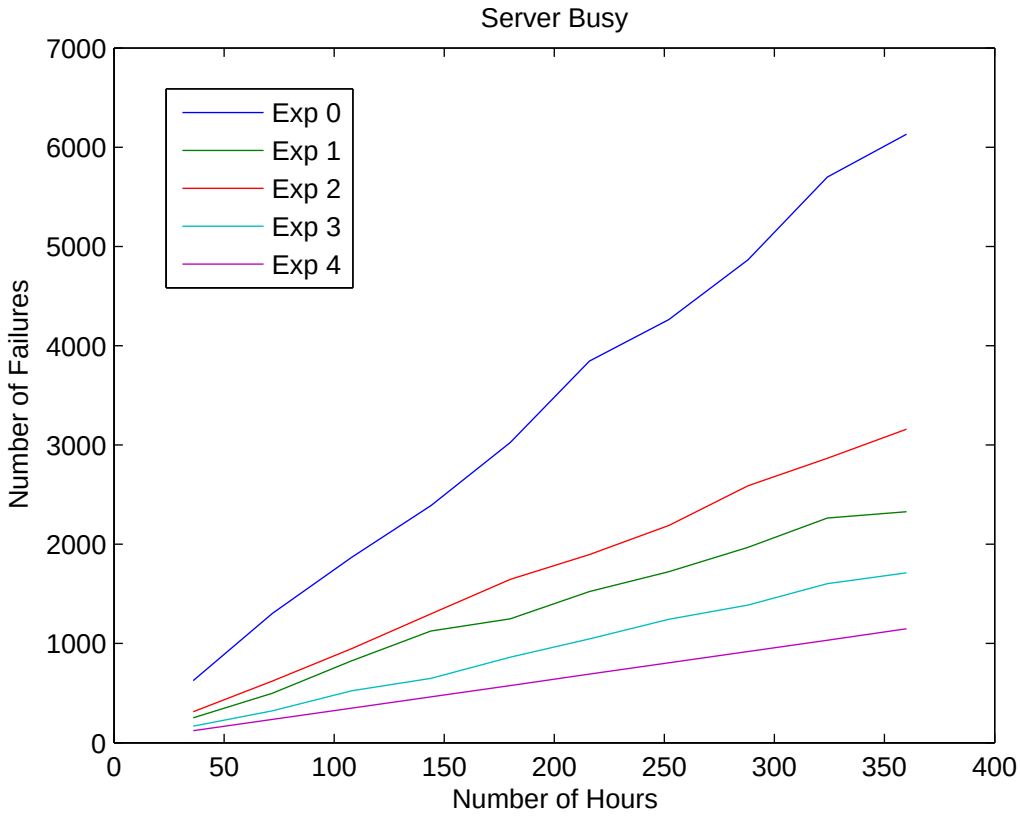

Figure 5.2: Number of failures under resource problem

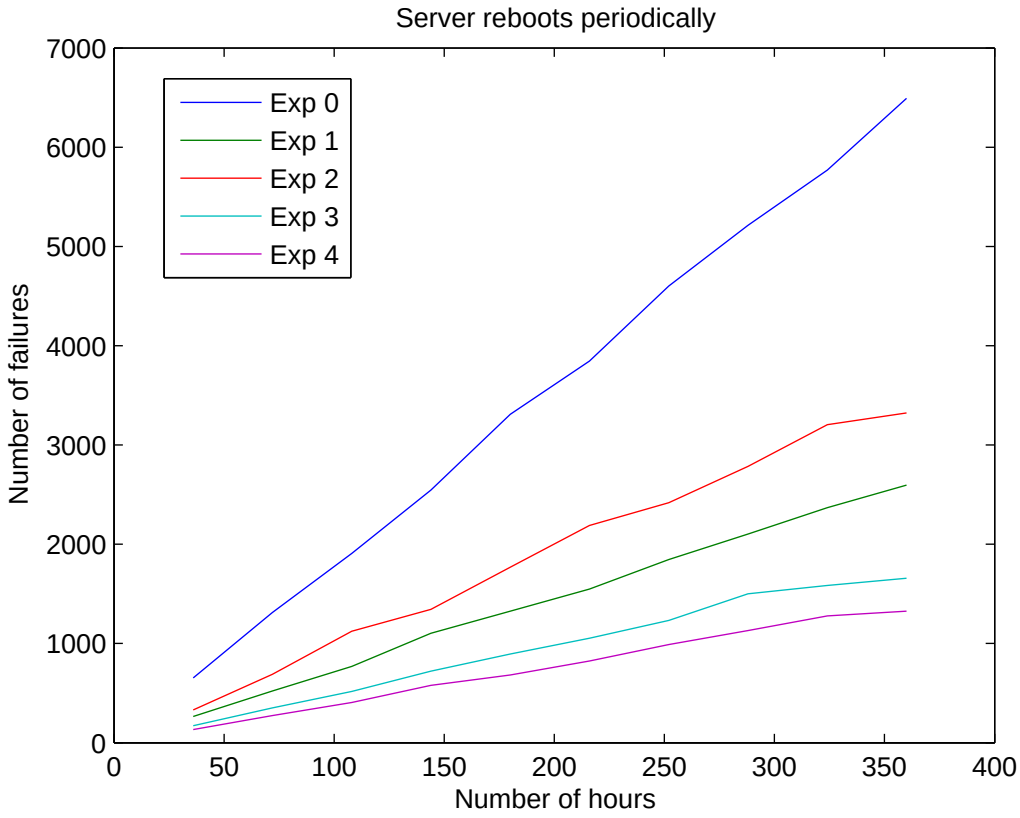

Figure 5.3: Number of failures under entry point failure

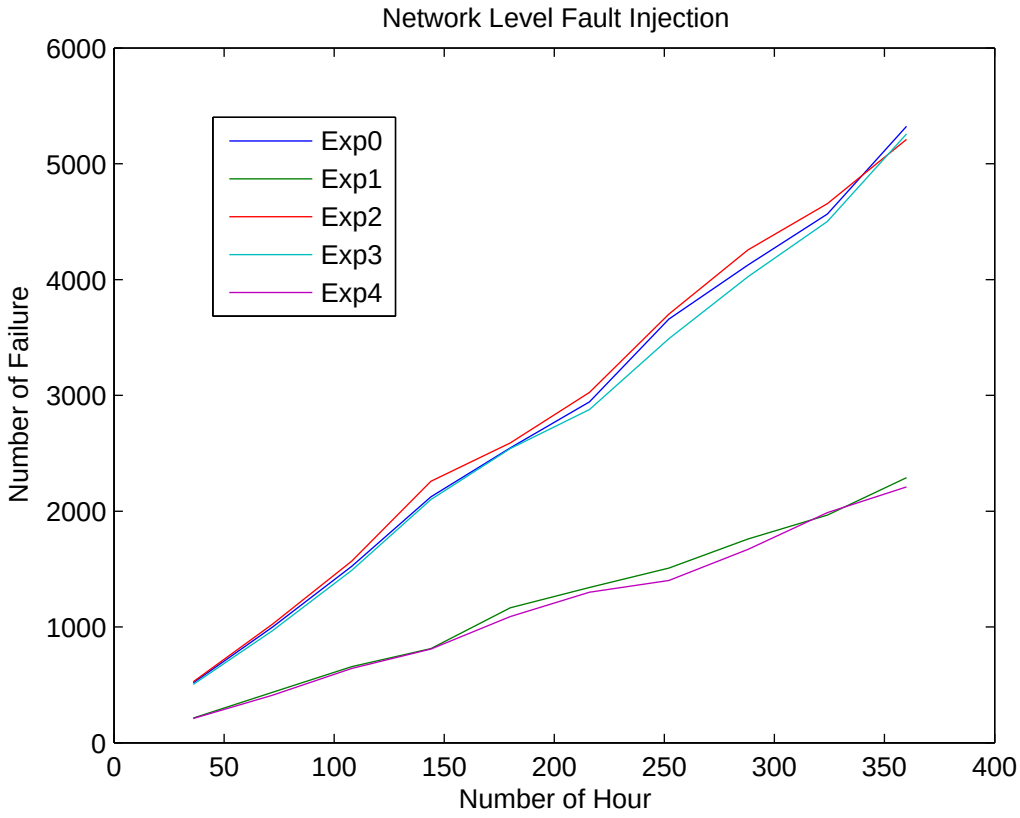

Figure 5.4: Number of failures under Network Level Fault Injection

5.2 Discussion

When there is no redundancy techniques applied on the Web service system (Exp 0), it is clearly shown that the failure rate of the system is the highest. Consequently, we try to improve the reliability of the Web service in two different ways, including spatial redundancy with replication and temporal redundancy with retry or reboot.

5.2.1 Resource Problem and Entry Point Failure

Under different situations, our approach improves the availability of the system differently. When the system is under resource problem and entry point failure, the experiment shows that the temporal redundancy helps to improve the availability of the system.

For the Web service with retry (Exp 1), the percentage of failures of the system (number of failed requests/total number of requests) is reduced from 11.97% to 4.93%. This shows that the temporal redundancy with retry can significantly improve the availability of the Web service. When there is a failure occurrence in the Web service, on the average, the clients need to retry twice to get the response from the web service.

Another temporal redundancy is Web service with reboot (Exp 2). For the experimental result, it is found that the failure rate of the system is also reduced: from 11.97% to 6.44%. The improvement is good, but not as substantial as the temporal redundancy with retry. It is due to the fact that when the Web service cannot be provided, the server will take time to reboot.

Spatial redundancy is applied in the system in Exp 3, which is the ap-

proach we have proposed. The availability of the system is improved even more significantly: the failure rate of the system is reduced from 11.97% to 3.56%. The Web service performs failover when the Web service cannot respond. The replication manager keeps checking the availability of the Web services. If the primary service fails, the replication manager selects another Web service to provide the service. The replication manager sends a message to the Web server to check the availability every 5 ms. It shortens the potential downtime of the Web service, thus the failure rate is reduced. In the experiment, on the average, the replication manager detects that there are around 600 failures in the Web services and performs the failovers.

To further improve the reliability of the Web service, both spatial and temporal redundancy is applied in the system in the Experiment 4. The failure rate is reduced from 11.97% to 2.59%. In the experiment, the Web service is replicated on five different machines and the clients will retry if they cannot get response correctly from the service. It is demonstrated that this setting results in the lowest failure rate. This shows that spatial and temporal redundancy (a hybrid approach) achieve the highest gain in reliability improvement of the Web service.

5.2.2 Network Level Fault Injection

When the system is under network level fault injection, the temporal redundancy reduces the failure rate of the system from 12.32% to 5.12%. When there are fault injected into the SOAP message, the system cannot process the request correctly, which will cause error in the system. However, with

temporal redundancy, the clients can resubmit the result to the system when there is fault injected into the previous message, thus, the failure rate of the system is reduced. However, the spatial redundancy approach cannot improve the availability of the system. . It is because even the message has injected faults and it will not trigger a failover of the system.

5.2.3 Failure Rate

The failure rate of a system is defined as:

$$\lambda = \lim_{\Delta \longrightarrow 0} \frac{F(t + \Delta t) - F(t)}{\Delta t} \tag{5.1}$$

The failure rate of our system, using a very specific scenario has improved from 0.114 to 0.025. The reliability of the system can be calculated with

$$R(t) = e^{-\lambda(t)t} \tag{5.2}$$

and Figure 5.5 shows the reliability of the discussed system.

Availability of the system is defined as:

$$A = \frac{MTTF}{MTTF + MTTR} \tag{5.3}$$

And in our case

$$\begin{aligned} MTTF &= \frac{1}{\lambda(t)} \\ &= \frac{1}{0.025} \\ &= 40 \end{aligned}$$

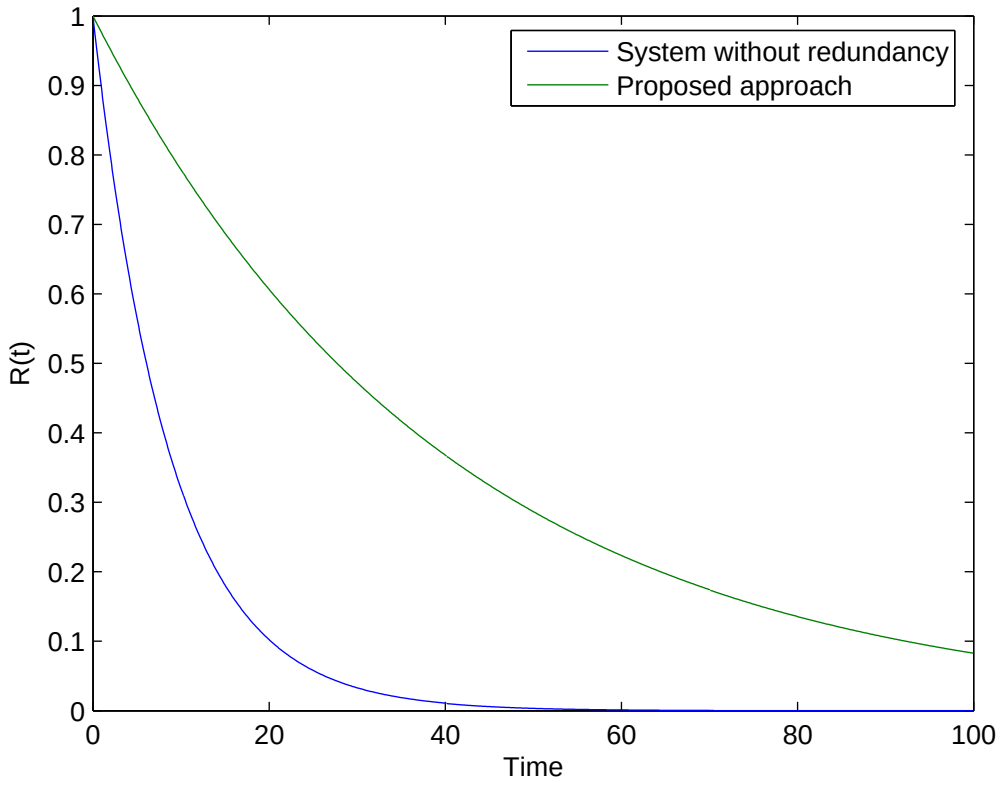

Figure 5.5: Reliability of the system over time

$$MTTF = 3s$$

$$\begin{aligned} A &= \frac{40}{40+3} \\ &= 0.93 \end{aligned}$$

which is quite of an improvement from $A = 0.75$ but still not up to standards of today's expectations.

Chapter 6

Reliability Modeling

We develop the reliability model of the proposed Web service paradigm using Markov chains[22, 21]. The model is shown in Figure 6.1.

The reliability model is verified through using the tool SHARPE [23].

In Figure 6.1(a), the state s represents the normal execution state of the system with n Web service replicas. In the event of an error, the primary Web service failed, the system will either go into the other states, i.e $s - j$ which represents the system with $n - j$ working replicas remaining, the replication manager responds on time, or it will go to the failure state F with conditional probability c_1. λ^* denotes the error rate at which recovery cannot complete in this state and c_1 represents the probability that the replication manager responds on time to switch to another Web service.

When the failed Web service is repaired, the system will go back to the previous state, $s - j + 1$. μ^* denotes the rate at which successful recovery is performed in this state, c_2 represents the probability that the failed Web service server reboots successfully. If the Web service is failed, it goes to

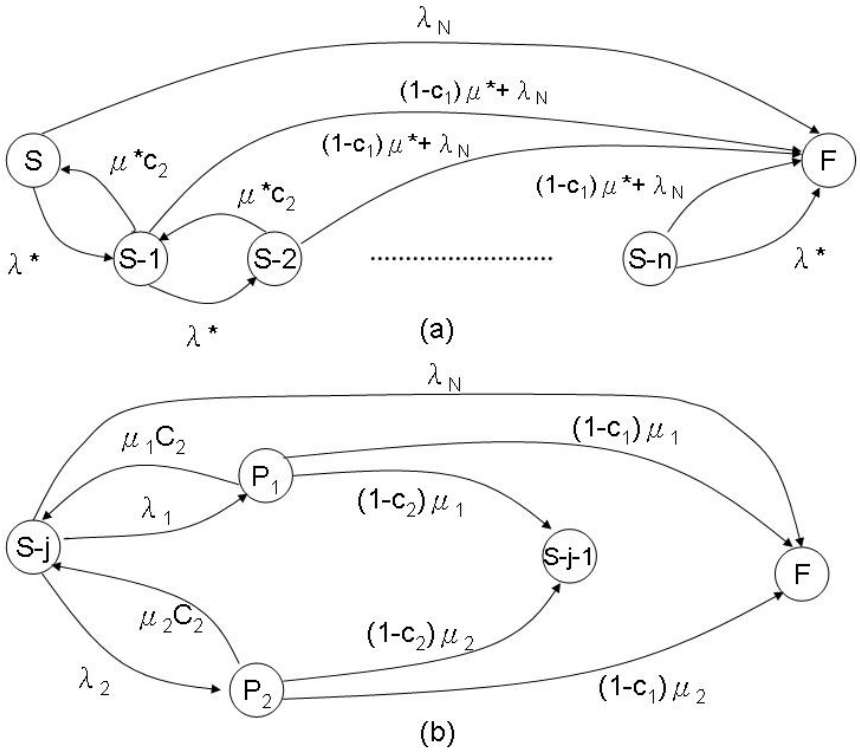

Figure 6.1: Markov chain based reliability model for the proposed system

another Web service. When all Web services are failed, the system enters the failure state F. λ_n is the network failure rate.

$s-1$ to $s-n$ represents the working state of the n Web service replicas and the reliability model of each Web service is shown in Figure 6.1(b). There are two types of failures simulated in our experiments, they are P_1 recourses problem (server busy) and P_2 entry point failures (server reboot). If failure occurred in the Web service, either the Web service can be repaired with μ_1 or μ_2 repair rate with conditional probability c_1 or the error cannot be recovered, it will go to the next state $s - j - 1$ with one less Web service replica available. If the replication manager cannot response on time, it will go to the failure state. From the graph, two formulas can be obtained:

$$\lambda^* = \lambda_1 \times (1 - C_2)\mu_1 + \lambda_2 \times (1 - C_2)\mu_2 \tag{6.1}$$

$$\mu^* = \lambda_1 \times \mu_1 + \lambda_2\mu_2 \tag{6.2}$$

From the experiments, we obtained the error rates and the repair rates of system, the values are shown in Table 6.1:

Table 6.1: Model parameters

ID	Description	Value
λ_N	Network failure rate	0.02
λ^*	Web service failure rate	0.025
λ_1	Resource problem rate	0.142
λ_2	Entry point failure rate	0.150
μ^*	Web service repair rate	0.286
μ_1	Resource problem repair rate	0.979
μ_2	Entry point failure repair rate	0.979
C_1	Probability that the RM response on time	0.9
C_2	Probability that the server reboot successfully	0.9

Chapter 7

Conclusion and Future Work

In the paper, we surveyed replication and design diversity techniques for reliable services and proposed a hybrid approach to improving the availability of Web services. Furthermore, we carried out a series of experiments to evaluate the availability and reliability of the proposed Web service system. From the experiments, we conclude that both temporal and spatial redundancy is important to the availability improvement of the Web service. In the future, we plan to test the proposed schemes with a wide variety of system, environment and fault injection scenarios and analyze the impact of various parameters on availability. Moreover, we will also evaluate the schemes with design diversity techniques.

Acknowledgement

Prof. Michael R. Lyu and Prof. Miroslaw Malek provide valuable suggestions and comments and these give significant direction to my research.

Bibliography

[1] R. Bilorusets, A. Bosworth et al, "Web Services Reliable Messaging Protocol WS-ReliableMessaging," EA, Microsoft, IBM and TIBCO Software, http://msdn.microsoft.com/library/enus/dnglobspec/html/ws-reliablemessaging.asp, Mar. 2004.

[2] N. Looker and M. Munro, "WS-FTM: A Fault Tolerance Mechanism for Web Services," University of Durham, Technical Report, 19 Mar. 2005.

[3] D. Liang, C. Fang, and C. Chen, "FT-SOAP: A Fault-tolerant Web Service, " Institute of Information Science, Academia Sinica, Technical Report 2003.

[4] M. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan, "Thema: Byzantine-Fault-Tolerant Middleware for Web-Service Application," Proc of IEEE Symposium on Reliable Distributed Systems, Orlando, FL, Oct. 2005.

[5] A. Erradi and P. Maheshwari, "A broker-based approach for improving Web services reliability", Proc of IEEE International Conference on Web Services, vol. 1, pp. 355-362, 11-15 Jul. 2005.

[6] W . Tsai, Z. Cao, Y. Chen, Y and R. Paul, "Web services-based collaborative and cooperative computing," Proc of Autonomous Decentralized Systems, pp. 552-556, 4-8 Apr. 2005.

[7] P. Townend, P. Groth, N. Looker, and J. Xu, "Ft-grid: A fault-tolerance system for e-science," Proc. of the UK OST e-Science Fourth All Hands Meeting (AHM05), Sept. 2005.

[8] "The effect of statically and dynamically replicated components on system reliability," D. Leu, F. Bastani and E. Leiss, IEEE Transactions on Reliability, vol.39, Issue 2, pp.209-216, Jun 1990.

[9] "Reliability analysis of real-time controllers with dual-modular temporal redundancy," Byung Kook Kim, Proc. the Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA) 1999, pp.364-371, 13-15 Dec. 1999.

[10] "On the increase of system reliability by parallel redundancy," K. Shen and M. Xie, IEEE Transactions on Reliability, vol.39, Issue 5, 99.607-611 Dec. 1990.

[11] A. Avizienis, and L. Chen, "On the implementation of N-version programming for software fault-tolerance during program execution," Proc. of First International Computer Software and Applications Conference, pp. 149-155, 1977.

[12] A. Avizienis, and J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," IEEE Transactions on Computer, pp. 67-80, Aug. 1984.

[13] J. Lala, and R. Harper, "Architectural principles for safety-critical real-time applications," Proc. of the IEEE, vol. 82, no. 1, pp. 25-40, January, 1994.

[14] R. Riter, "Modeling and Testing a Critical Fault-Tolerant Multi-Process System," Proc. 25th International Symposium on Fault-Tolerant Computing, pp.516-521, 1995.

[15] M. Lyu and V. Mendiratta, "Software Fault Tolerance in a Clustered Architecture: Techniques and Reliability Modeling," Proc of 1999 IEEE Aerospace Conference, Snowmass, Colorado, vol.5, pp.141-150, Mar. 6-13 1999.

[16] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek, "Selection algorithms for replicated web servers," Proc. of Workshop on Internet Server Performance 98, Madison, WI, Jun. 1998.

[17] N. Looker, M. Munro, and J. Xu, "Simulating Errors in Web Services," International Journal of Simulation: Systems, Science and Technology, vol. 5, pp 29 - 38, 2004.

[18] Y. Yan, Y. Liang and X. Du, "Controlling remote instruments using Web services for online experiment systems," Proc. IEEE International Conference on Web Services (ICWS) 2005, 11-15 Jul. 2005.

[19] Y. Yan, Y. Liang and X. Du, "Distributed and collaborative environment for online experiment system using Web services," Proc. the Ninth International Conference on Computer Supported Cooperative Work in Design 2005, vol.1, pp.265-270, 24-26 May 2005.

[20] N. Looker and J. Xu, "Assessing the Dependability of SOAP-RPC-Based Web Services by Fault Injection," 9th IEEE International Workshop on Object-oriented Real-time Dependable Systems, pp. 163-170, 2003.

[21] "Failure correlation in software reliability models," K. Goseva-Popstojanova and K. Trivedi, IEEE Transactions on Reliability, vol.49, Issue 1, pp.37-48, Mar. 2000.

[22] "Reliability estimation for statistical usage testing using Markov chains," H. Le Guen, R. Marie and T. Thelin, Proc. 15th International Symposium on Software Reliability Engineering (ISSRE) 2004, pp.54-65, Nov. 2-5 2004.

[23] R. Sahner, K. Trivedi, and A. Puliafito, "Performance and Reliability Analysis of Computer Systems," An Example-BasedApproach Using the SHARPE Software Package.Kluwer, Boston, MA (1996).

[24] Deron Liang, Cheng-Liang Fang and S.-M. Yuan, "A Fault-Tolerant Object Service on CORBA," Journal of Systems and Software, Vol. 48, pp. 197-211, 1999.

[25] Kececioglu, Dimitri, Reliability Engineering Handbook, Prentice Hall, Inc., Englewood Cliffs, New Jersey, Vol. 1, 1991.