Reliable Web Services: Methodology, Experiment and Modeling

Pat. P. W. Chan, Michael R. Lyu
Department of Computer
Science and Engineering
The Chinese University of Hong Kong
Hong Kong, China
pwhchan, lyu@cse.cuhk.edu.hk

Miroslaw Malek
Department of Computer
Science and Engineering
Humboldt University Berlin,
Germany
malek@informatik.hu-berlin.de

Abstract

We identify parameters impacting 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 Web services, we employ several replication schemes and compare them with a single service. The Web services are coordinated by a replication manager. It provides a round robin algorithm for scheduling the workload of the Web services and keeps updating the availability of each Web service. The replication algorithm and the detailed system configuration are described. Experiments are performed to evaluate the resulting service availability. Modeling on the Web services with Petri-net is constructed and verified through experiments with different applications. With the parameters obtained from the experiments, the proposed model can be engaged to demonstrate the characteristics of the Web service.

1. Introduction

As the use of Web services is growing, there is an increasing demand for dependability. Service-oriented Architectures (SOA) [5] 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).

The use of services, especially Web services, became a common practice. In Web services, standard communication protocols and simple broker-request architectures are needed to facilitate exchange (trade) of services, and this standardization simplifies interoperability. In the coming few years, services are expected to dominate software industry. As services begin to permeate all aspects of human society, the problems of service dependability, security and timeliness are becoming critical, and appropriate solutions need to be made available.

Several fault tolerance approaches have been proposed for Web services in the literature [2, 9], but the field still requires theoretical foundations, appropriate models, effective design paradigms, practical implementations, and in-depth experimentations for building highly-dependable Web services. We attack these issues in a unified approach.

The rest of the paper is organized as follows. Related work of dependable services is presented in Section 2, in which the problem statement about reliable Web services is stated. In Section 3, methodologies for reliable Web services and experimental results are described, in which we describe experimental settings and offer a roadmap to dependable Web services. Reliability modeling is presented in Section 4. Finally, conclusions are made in Section 5.

2. Related Work

Fault tolerance can be achieved via spatial or temporal redundancy, including replication of hardware (with additional components), software (with special programs), and time (with diversified operations) [7]. Spatial redundancy can be dynamic or static, both of which use replication. 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 and the approach is known as n-modular redundancy (NMR). Redundancy can be achieved by replicating hardware modules to provide backup capacity when a fault occurs, or redundancy can be obtained using software solutions to replicate key elements of a business process.

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 [1] and has been used in both firmware and software systems [6]. The basic idea is that, with different design and implementations, common failure modes can be reduced.

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.

Based on the discussed techniques, a number of reliable Web services techniques appeared in the recent literature. WS-FTM (Web Service-Fault Tolerance Mechanism) is an implementation of the classic N-version model with Web services [9] which can easily be applied to systems with a minimal change. The Web services are implemented in different versions and the voting mechanism is conducted in the client program.

FT-SOAP [8], on the other hand, 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, in which a fault-tolerant SOAP-based middleware platform is proposed.

FT-Grid [15] is another design, which is a deployment of design diversity for fault tolerance in Grid. It is not originally 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.

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

In the paper, we proposed a framework for the deployment of reliable Web services, and enhance a previous scheme [3] with Round-robin algorithm and N-version programming in the Web services. 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 temporal redundancy, and study the tradeoffs between them. A generic Web service system with spatial as well as temporal replication is proposed, its prototype is implemented as an experimental testbed.

3. Methodologies for Reliable Web Services

In the following section, we propose a replication Web service system for reliable Web services. In our system, the dynamic approach is considered and its architecture is shown in Figure 1.

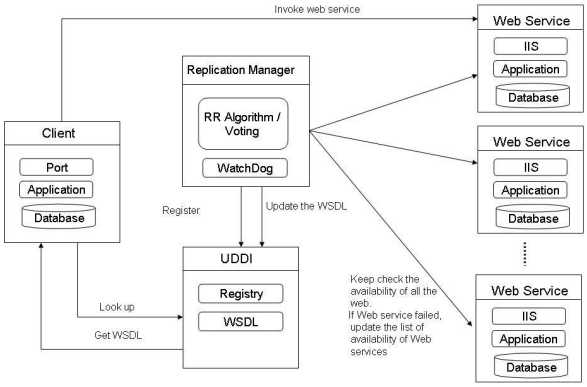

Figure 1. Proposed architecture for dependable Web services.

3.1. Scheme details

In the proposed system, we applied two different approaches for managing the spacial replication, including: Round-robin (RR) algorithm and N-version programming. We preformed different experiments to evaluate the reliability of the system.

In the first approach, the Web servers work concurrently and a Round-robin algorithm [14] is applied for scheduling the work among the Web services. The Web service is replicated on different machines. When there is a Web service fault, the other Web servers can immediately 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. Choosing (with an anycasting algorithm) the best (fastest, most robust, etc.) Web service [13] to provide the service which is called the primary Web service.

2. Keeping the availability list of the Web services.

3. Registering the Web Service Definition Language (WSDL) with the Universal Description, Discovery, and Integration (UDDI).

4. Continuously checking the availability of the Web services by using a watchdog.

5. Applying the Round-robin algorithm for the scheduling the workload of the Web service.

The replication manager schedules the work of the Web service with the Round-robin algorithm; therefore the resources of the system can be fully utilized. The replication manager distributes the work to different Web servers according to the availability of the servers. The requests are sent to different Web services accordingly. The replication manager maps the new address of the Web service which provides the 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 detailed procedure is shown in Figure 1.

The replication manager is running on a server, which keeps checking the availability 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.

In the second approach, different versions of the Web service are employed. The requests from the clients will be forwarded to all versions of the Web services. When all the results are ready, a voting algorithm is applied to obtain the majority result and return the answer to the corresponding client.

The architecture of the system is similar to the first approach. However, the functionality of the replication manager is different. The replication manager is responsible for:

1. Selecting the primary Web service for holding the voting procedure. Once the selected Web service gets the request, it will forward the request to all the Web services.

2. Keeping the availability list of the Web services.

3. Registering the Web Service Definition Language (WSDL) with the Universal Description, Discovery, and Integration (UDDI).

4. Continuously checking the availability of the Web services by using a watchdog.

3.2. 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.By handling services as atomic transactions, exception handling does not help in dealing directly with inherent problems of a service. Consequently, continuous service is only possible by performing re-execution using a retry or reboot at the termination points or after a timeout period.

If redundancy in time is not appropriate to meet dependability requirements or if the 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).

2. In parallel, meaning that multiple services are executed simultaneously and if the primary service fails, the next one takes over. 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, where results are compared and the final outcome is based on at least $\lfloor n/2 + 1 \rfloor$ 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 fault, 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 can be mitigated. 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 purpose.

Table 1. Summary of the experiments

Experiment ID	1	2	3	4	5	6	7	8
Spatial replication	0	0	0	0	1	1	1	1
Reboot	0	0	1	1	0	0	1	1
Retry	0	1	0	1	0	1	0	1

We 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.

3.3. Experiments

A series of experiments are designed and performed for evaluating the reliability of the Web service. In the system, we applied retry, reboot and spatial replication with Round-robin or N-version Web services. Table 1 shows all the combinations of the experiments.

3.3.1 Experiment Setup

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

In the experiments, we run different Web services in our system to evaluate the availability of the proposed fault tolerant techniques under different situations. Faults are injected in the system and the fault injection techniques are similar, for example, to the ones referred in [10]. A number of faults may occur in the Web service environment.The faults are also further divided into permanent fault (the server is down permanently once this fault occurs) and temporary fault (the fault only occurs randomly). Our experimental environment is defined by a set of parameters. Table 2 shows the parameters of the Web services in our experiments.

For both approaches described in Section 3.1, different experiments are performed. We compare eight approaches as shown in Table 1 for providing the Web services in each approach. The details of the experiments are described in as follows:

1. Single server 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.

Table 2. Parameters of the experiments

	Parameters	Current setting/metric
1	Request frequency	1 req/min
2	Polling frequency	10 per min
3	Number of replicas	5
4	Client timeout period for retry	1 mins
5	Max number of retries	5
6	Fault rate λ	number of faults/hour
7	Load (profile of the program)	78.5%
8	Reboot time	10 min
9	Failover time	1 s
10	Communication time to Computational time ratio	10:1
11	Round-robin rate	1 s
12	Temporary fault probability	0.01
13	Permanent fault probability	0.001

2. Single server with retry The Web service provides the service and the client retries another Web service when there is no response from the original Web service after timeout.

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

4. Single server with retry and reboot The Web service provides the service and the Web server will reboot when there is no response from the Web service. Clients will retry after timeout when there is no response from the service.

5. Spatial replication with Round-robin / N-version We use a generic spatial replication: For the first approach, the Web service is replicated on different machines and the requests are transferred to different Web services according to the scheduling of the workload by the replication manager. For the second approach, five different versions of the Web services are employed. The requests are sent to all version and majority answer is chosen by voting and sent back to the client.

6. Spatial replication with Round-robin / N-version and retry This is a hybrid approach which is based on the approach in Exp 5. However, the clients will retry after timeout when there is no response from the service.

7. Spatial replication with Round-robin / N-version and reboot This is similar to the Exp 5 where the Web service is replicated on different machines and the request is transferred to the server scheduled by the manager. In addition, the Web server will reboot when there is no response from the Web service.

8. Hybrid approach with spatial replication, retry and reboot This is the proposed hybrid approach. The Web service is replicated as described in Exp 5. The server will reboot when there is no response from the Web service. The client will retry after timeout.

Table 3. Experimental results with Round-robin algorithm

Experiments (number of fault / response time)	1	2	3	4
Normal case	0/183	0/192	0/190	0/187
Temp	705/190	0/223	723/231	0/238
Perm	6144/–	6337/–	1064/–	5/2578
Experiments (number of fault / response time)	5	6	7	8
Normal case	0/188	0/195	0/193	0/190
Temp	711/187	0/233	726/188	0/231
Perm	5637/–	5532/–	152/187	0/191

Table 4. Experimental results with N-version programming

Experiments (number of fault / response time)	1	2	3	4
Normal case	0/318	0/320	0/315	0/319
Temp	429/321	0/356	423/364	0/356
Perm	3861/–	3864/–	614/–	3/4027
Experiments (number of fault / response time)	5	6	7	8
Normal case	0/322	0/318	0/321	0/319
Temp	0/325	0/321	0/322	0/324
Perm	1544/323	1546/324	63/324	0/323

3.3.2 Experimental Results

The Web services were executed for 5 days for each experiment generating a total of 7200 requests from the client. A single fault is counted when the system cannot reply to the client. For the approach with retry, a single fault is counted when a client retries five times and still cannot get the result. A summary of the results with the Round-robin algorithm is shown in Table 3 and a summary of the results with the N-version programming is shown in Table 4.

Tables 3 and 4 show the improvement of the reliability of the system with our proposed paradigm. Under different situations, the availability of the system improved differently. In the normal case, there is no failure in different systems. For further investigation, we insert various kinds of faults into the system.

When no redundancy techniques are applied on the Web service system (Exp 1), it is clearly seen 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.

Single server with retry When the system is under temporary fault, the experiment shows that the temporal redundancy helps to improve the reliability of the system. For the Web service with retry (Exp 2), the number of faults is reduced to zero. This shows that the temporal redundancy with retry can significantly improve the reliability of the Web service. When there is a fault occurrence in the Web service, on the average, the clients need to retry twice to get the response from the Web service. However, the response time of the Web service is increased. When there is permanent fault, this scheme cannot reduce the number of faults in the system.

Single server with reboot Another temporal redundancy is Web service with reboot (Exp 3). For the experimental result, it is found that the fault rate of the system is reduced when there is permanent fault. When there is permanent fault, the server will try to reboot. Once the server finishes rebooting, it can provide the service again. The fault rate is reduced from 85.3% to 14.0%. For temporary fault, the improvement is 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.

Single server with retry and reboot With retry and reboot, the fault rate of both temporary and permanent cases are reduced. It takes the advantages of both algorithms. For temporary fault, the number of fault is reduced to zero. For permanent fault, the number of fault is significantly reduced from 85.3% to 1%; however, the response time is also greatly increased.

Spatial replication with Round-robin With the spatial replication in Exp 5, the fault rate in the permanent fault is reduced from 85.3% to 78.3%. The fault rate is reduced because there are more servers in the system. When server is failed, the replication manager will update the availability list and forward the requests to other servers. When all the servers are failed, the system will not be able to handle the requests from the clients.

Spatial replication with N-version programming With the spatial replication in Exp 5, the fault rate of the Web service is greatly reduced. When a fault occurs to a Web service, other Web services are still operating, and the majority result will be selected and returned to the client. Thus, the failure of a Web service will not affect the system. When permanent faults occur, the fault rate is reduced from 85.3% to 21.4% with this scheme. The fault rate is reduced because other versions of Web service are available and majority result is returned to the client. In the N-version approach, the fault rate is much lower than the that of Round-robin ap-

Table 5. Number of failure with varying time-out period for retry

Timeout period for retry	Number of failure in temporary fault	Number of failure in permanent fault
0	2	81
2	0	2
5	0	0

proach.

Spatial replication, retry or reboot In Exp 6 and 7, hybrid approaches with retry (Exp 6) or reboot (Exp 7) are conducted. We found that the fault rate is not much improved comparing with that in Exp 4. However, the average response time of the Web service is reduced.

Spatial replication with Round-robin / N-version, retry and reboot After performing the above experiments, we propose a hybrid approach for improving the reliability of Web service which includes spatial redundancy, retry and reboot. The reliability of the system is improved more significantly: the fault rate of the system is reduced from 85.3% to 0 and the average response time is short. The replication manager keeps checking the availability of the Web services. When there is a server failure, other servers are responsible to handle the requests. At the same time, the failed server will reboot. Thus, the response time for handling the requests are greatly reduced. In Exp 8, it is demonstrated that this setting results in the lowest fault rate. This shows that combining spatial and temporal redundancy in a hybrid approach achieves the highest gain in reliability improvement of the Web service.

3.3.3 Optimal Parameters

To evaluate the parameters in the system, we preform a set of experiments. Through the experiments, we obtained a set of optimal parameters setting for the Web service system. The parameters we examined include: number of tries, timeout period for retry, polling frequency, number of replicas and load of server. The results are shown in Table 5 to 7, respectively. In each experiment, a total of 9000 requests are handled in a 30-minutes duration.

In Table 5, as the timeout period for retry is too short, the replica cannot reboot on time, causing the number of failures to increase. Also, another cause of the failure is that the replication manager cannot respond on time to switch the primary Web service. There are five tries, and the reboot time for a server is around 50 seconds.

Table 6 shows the number of failures varies with polling frequency. If the polling frequency is low, the replication manager cannot respond on time and the request will still be sent to the failed server causing the failures in the system. When the polling frequency increases, the situation improves. The replication manager can respond on time and reduce the number of failures.

Table 6. Number of failure with varying polling frequency

Polling frequency (number of requests per mins)	Number of failure in temporary fault	Number of failure in permanent fault
0	0	7124
1	0	811
5	0	12
10	0	1
15	213	254
20	1124	1023

Table 7. Number of failure with varying load of the server

Load of the server	Number of failure in temporary fault	Number of failure in permanent fault
75%	0	0
80%	2	3
85%	10	14
90%	512	528
95%	3214	3126
99%	8792	8845

From Table 7, the optimal load of the Web server is 75%. If the load of the server is too high, the server is not able to handle the requests, which increases the failure rate of the system.

Also, we performed experiment on the number of failures changes with the number of retry. From the experiments, we found that the number of retry depends on the failure rate of the Web service. If the failure rate of the Web service is large, the number of retry is needed for the application. Another parameter we evaluate is the number of replicas. We found that three replicas are sufficient to reduce the number of failures nearly to zero.

4. Reliability Modeling

We develop a reliability model of the proposed Web service paradigm using Petri-Net [11] and Markov chains [4]. The model is shown in Figure 2(a), Figure 2(b) and Figure 3. The reliability model is analyzed and verified through using the tool SHARPE tool [12]. Petri-Net is built for evaluating the performance of the system and the Markov chains model is developed for analyzing the system availability.

In Figure 2(a), we model a system which is composed

of two Web services. When the messages arrive at the first Web service, a message queue is formed and the Web service will handle the request in the message queue. In the model, the messages are queued up in the *queue* state. Next, the token will be passed to the service and arrive at *finished job* state. Then, the token is passed to the *quene2* which is for the second Web service.

The availability of the service is directly affected by the availability of the Web server. When the server is down, the two Web services will not be available. In our system, when the server fails, the backup server will be invoked.

In Figure 2(b), we model the system with N versions of Web service. The messages are queued up in the *queue* state. Then, the token will be passed to the service $servicev1$ to $servicev5$. When the job is finished, it arrives at the *finished job* state. Finally, the token is passed to the *voting* state.

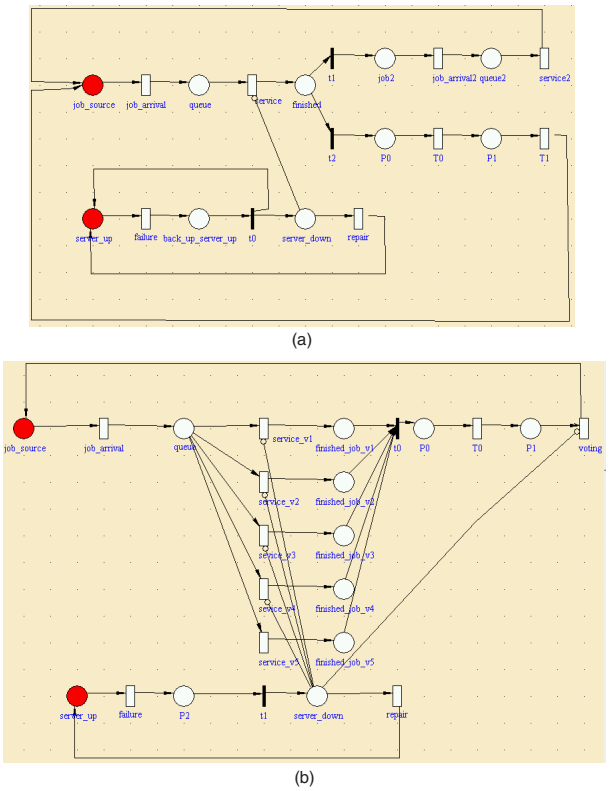

Figure 2. (a) Petri-Net based reliability model for the proposed system with Round-robin algorithm (b) Petri-Net based reliability model for the proposed system with N-version programming

In Figure 3(a), the state s represents the normal execution state of the system with n Web service replicas. In the event of a fault causing, the primary Web service to fail,

Table 8. Model parameters

ID	Description	Value
λ_N	Network fault rate	0.01
λ^*	Web service fault rate	0.025
λ_1	Temporary fault rate	0.01
λ_2	Permanent fault rate	0.001
μ^*	Web service repair rate	0.286
μ_1	Temporary fault rate	0.979
μ_2	Permanent fault 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

the system will either go into the other states (i.e., $s-j$ which represents the system with $n-j$ working replicas remaining, if the replication manager responds on time), or it will go to the failure state F with conditional probability $(1-C_1)$. λ^* denotes the fault 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.

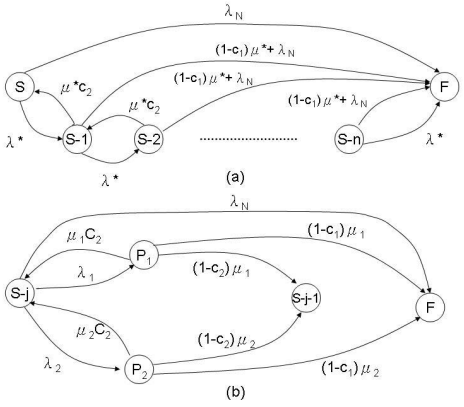

Figure 3. Markov chain based reliability model for the proposed system

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, and C_2 represents the probability that the failed Web server reboots successfully. λ_n represents the network fault rate.

States $(s-1)$ to $(s-n)$ in Figure 3(a) represent the working states of the n Web service replicas and the reliability model of each Web service is shown in Figure 3(b). There are two types of faults simulated in our experiments: P_1 denotes a temporary fault and P_2 denotes a permanent fault. If a fault occurs in the Web service, either the Web service can be repaired with μ_1 (to enter P_1) or μ_2 (to en-

ter P_2) repair rates with conditional probability C_1. If the fault cannot be recovered, the system goes to the next state $(s - j - 1)$ with one less Web service replica available. If the replication manager cannot respond in time, it will go to the failure state. From the graph, two formulae can be obtained:

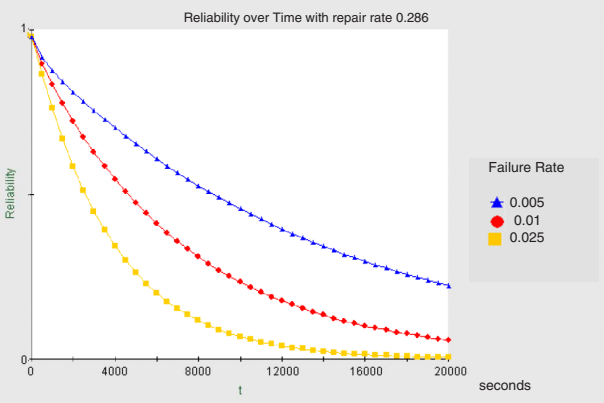

Figure 4. Reliability with different fault rates and repair rates

Based on the experiments described in Section 3.3, we obtain the fault rates and the repair rates of various components in the system; the results are shown in Table 8. The reliability of the system over time is further calculated with the tool SHARPE. Figure 4 shows the reliability over time at different fault rates λ^*; the repair rate is (set at) 0.286 faults/s. Note that the fault rate obtained from the experiments is 0.025 failure/s. This failure rate is measured under an accelerated testing environment.

5. Conclusions

In the paper, we surveyed and addressed applicability of replication and design diversity techniques for reliable Web 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, performance and reliability of the proposed Web service system. From the experiments, we conclude that both temporal redundancy and spatial redundancy are important to the reliability improvement of the Web service. Modeling techniques by Petri-Net and Markov chain provide further insights of Web service system reliability with the proposed fault tolerant mechanisms.

Acknowledgment

The work described in this paper was fully supported by a grant from an internal block grant project from the Research Committee of the Chinese University of Hong Kong, under Project No. 3/06C-SF.

References

[1] A. Avizienis and L. Chen. On the implementation of n-version programming for software fault-tolerance during program execution. In *Proc. of First International Computer Software and Applications Conference*, pages 149–155, 1977.

[2] R. Bilorusets and A. Bosworth. Web services reliable messaging protocol ws-reliablemessaging. Technical report, EA, Microsoft, IBM and TIBCO Software, Mar 2004.

[3] P. Chan, M. Lyu, and M. Malek. Making services fault tolerant. In *Proc. of the 3rd International Service Availability Symposium*, volume 4328, pages 43–61, Helsinki, Finland, 15-16 May 2006. Springer.

[4] K. Goseva-Popstojanova and K. Trivedi. Failure correlation in software reliability models. *IEEE Transactions on Reliability*, 49(1):37–48, Mar 2000.

[5] S. Jones. Toward an acceptable definition of service [service-oriented architecture]. *IEEE Transactions on Software*, 22(3):87–93, May-June 2005.

[6] J. Lala and R. Harper. Architectural principles for safety-critical real-time applications. In *Proc of IEEE*, volume 82, pages 25–40, Jan 1994.

[7] D. Leu, F. Bastani, and E. Leiss. The effect of statically and dynamically replicated components on system reliability. *IEEE Transactions on Reliability*, 39(2):209–216, 1990.

[8] D. Liang, C. Fang, and C. Chen. Ft-soap: A fault-tolerant web service. Technical report, Institute of Information Science, Academia Sinica, 2003.

[9] N. Looker and M. Munro. Ws-ftm: A fault tolerance mechanism for web services. Technical report, University of Durham, 19 Mar 2005.

[10] N. Looker and J. Xu. Assessing the dependability of soap-rpc-based web services by fault injection. In *Proc. of the 9th IEEE International Workshop on Object-oriented Real-time Dependable Systems*, number 163-170, 2003.

[11] J. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.

[12] R. Sahner, K. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems. An Example-BasedApproach Using the SHARPE Software Package*. Kluwer Academic Publishers, Boston/London/Dordrecht, 1996.

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

[14] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round-robin. *IEEE/AMC Transactions on Networking*, 4(3):375–385, June 1996.

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