The Chinese University of Hong Kong
Department of Computer Science and Engineering

Ph.D. – Term Paper

Title:	WS-DREAM: A Distributed Reliability Assessment Mechanism for Web Services
Name:	ZHENG, Zibin
Student I.D.:	07024860
Contact Tel. No.:	9064 9145
Email A/C:	zbzheng@cse.cuhk.edu.hk
Supervisor:	Prof. Michael R. Lyu
Markers:	TBD
Mode of Study:	Full-time
Submission Date:	April 24, 2008
Term:	2
Fields:	
Presentation Date:	TBD
Time:	TBD
Venue:	TBD

WS-DREAM: A Distributed Reliability Assessment Mechanism for Web Services

Abstract

Redundancy-based fault tolerance strategies are proposed for building reliable Service-Oriented Applications (SOA), which can be developed on the unpredictable remote Web services. This paper proposes and implements a distributed reliability assessment mechanism for Web services, named WS-DREAM. Based on this mechanism, we provide a systematic comparison of various replication strategies by theoretical formula and real-world experiments. Moreover, a user-participated strategy selection algorithm is proposed and verified. Experiments are conducted to illustrate the advantage of this mechanism. In these experiments, users from six different locations all over the world perform evaluation of Web services distributed in six countries. Over 1,000,000 test cases are executed in a collaborative manner and detailed results are also provided.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Web services are self-contained, self-describing, and loosely-coupled computational components designed to support Machine to Machine interaction via networks, including the Internet. They are widely employed to implement the increasingly popular Service-Oriented Architectures/Applications (SOA). Because the reliability and effectiveness of remote Web services are unclear, and the performance of Internet is also unpredictable, it is difficult to guarantee the performance (e.g., response latency, failure-rate, stability and so on) of these service-oriented applications, which are developed on Web services.

WS-ReliableMessaging [1] is a Web service specification designed to allows messages to be delivered reliably between distributed applications. However, it can only guarantee communication reliability. Problems, such as unavailability of remote Web service (server crash down, overload, network disconnect, and so on), and poor performance (long latency, unstable, high failure-rate, and so on), are remain unsolve. A number of application/Web service level redundancy-based fault tolerance strategies have been proposed in the recent literature for establishing trustworthy and reliable service-oriented applications [2, 3, 4, 5]. These strategies use Web services with similar or identical interfaces as redundant replicas for fault tolerance and performance improvement purpose. There are two commonly used replication strategies: passive replication and active replication [6]. Passive replication, which employs a primary replica to process the request first and invokes backup replicas only when the primary replica fails, has been employed in FT-SOAP [7] and FT-CORBA [8]. Active replication, which invokes all replicas in parallel and employs the first properly returned response as the final outcome, has been employed in FTWeb [9], Thema [10], WS-Replication [11] and in work [12]. Also, an N-version model is involved in WS-FTM [13] for Web services. Design diversity and voting are employed in FT-Grid [14] and in work [15] to tolerant faults.

It is a challenge for application developers to select out the optimal replication strategy, which requires performance information of the target replicas and good knowledge on various available replication strategies. A

number of investigations are focus on individual Web service evaluation [16, 17, 18, 19, 20, 21], however, the investigations on replication strategies evaluation and selection is still limited.

1.2 Contributions

Assuming that the developers have obtained several appropriate replicas manually or using approaches proposed in [6, 31], this paper proposes a distributed assessment mechanism for suggesting an optimal replication strategy for service users. The contribution of this paper includes:

- Design and implement a distributed reliability assessment mechanism for Web services and replication strategies.
- Provide a systematic introduction of various replication strategies, and propose a replication strategy selection algorithm.
- Comprehensive real-world experiments are conducted, where more than 1,000,000 test cases are executed by users in six locations all over the world on target Web services located in six countries.

1.3 Organization

This paper is organized as follows: Chapter 2 proposes a distributed reliability assessment mechanism for Web services. Chapter 3 introduces various replication strategies. Chapter 4 provides an optimal strategy selection algorithm. Chapter 5 implements a prototype of our assessment mechanism. Chapter 6 presents experimental results, and Chapter 7 concludes the report.

Chapter 2

A Distributed Assessment Mechanism

When conducting replication strategies evaluation and selection, there are several challenges to be solved:

- **Evaluation location:** The service-oriented applications are usually distributed around different locations after being deployed, and the network conditions of these locations are different from each other. Therefore, conducting evaluation on the target Web services from various locations is necessary.
- **Evaluation accuracy:** To conduct accurate Web services and replication strategies evaluation, the service users need to have professional knowledge on various replication strategies, test case generation [22], test result analysis and so on. Unfortunately, few service users meet these requirements.
- **Evaluation efficiency:** It is time-consuming for service users to conduct evaluation themselves, which will entail studying various available strategies, designing test cases, implementing evaluation mechanisms and conducting evaluation in various locations. More efficient approaches are required.

Taking the viewpoint of service users where the remote Web service is treated as a black box without any internal design or implementation information, this section proposes a distributed Web service/replication reliability assessment mechanism to address the above challenges. This mechanism, which have also been introduced in [23], employs the concept of user-collaboration, which has contributed to the recent success of BitTorrent [24] and Wikipedia [25]. In this mechanism, users in different geographical locations help each other to conduct evaluation of individual Web services or replication strategies under the coordination of a centralized server. Historical test cases and evaluation results are saved in a data center for further use (e.g. performance prediction, evaluation efficiency and accuracy improvement). As shown in Fig.2.1, WS-DREAM includes a centralized server with a number of distributed clients. The overall process can be explained as follows.

1) **Evaluation registration:** Users submit evaluation requests with related information, such as the target Web service addresses, particular test cases, preferred replication strategies, and so on, to the server.

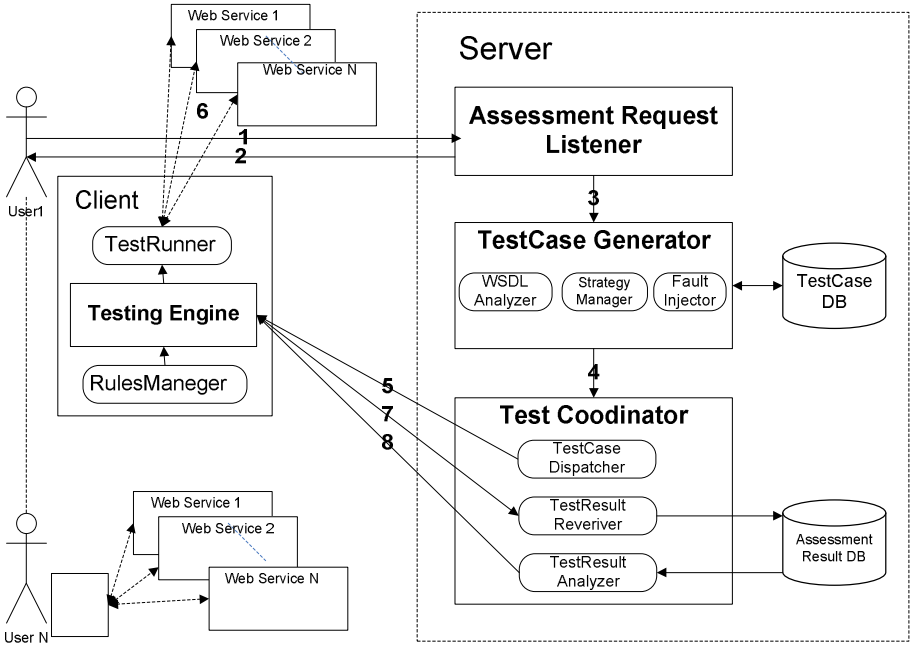

Figure 2.1. Architecture of WS-DREAM

2) **Client-side application loading:** A client-side evaluation application is loaded and executed in the user's computer.

3) **Test case generation:** The *TestCase Generator* in the server automatically creates test cases based on the interface of the target Web Service (WSDL file). User-contributed test cases as well as accumulated historical test cases are retrieved from the database of the server. Fault injection techniques [22, 26, 27] are employed to create various fault-trigger test cases in addition to normal test cases. Two types of test cases are created: single test cases for individual Web service performance evaluation, and multiple test cases, which contain several test cases and a running rule, for fault tolerance replication strategy evaluation.

4) **Test coordination:** The *Test Coodinator* schedules testing tasks based on the number of current users and test cases.

5) **Test cases dispatch:** Distributed client-side evaluation applications gets test cases from the server.

6) **Test cases execution:** Distributed client-side evaluation applications execute testing on the target Web services.

7) **Test result collection:** The test result is sent back to the server. Then steps 5, 6 and 7 are repeated to execute more test cases.

8) **Test cases analysis:** After the test is completed, the server engages a *TestResult Analyzer* to process the collected data. Moreover, historical evaluation data relating to the target Web services are obtained from the database for efficiency and accuracy improvement. The detailed evaluation result is sent to the user.

The challenges shown above can be addressed by WS-DREAM. The evaluation duration is greatly shortened by employing historical evaluation result of the same target Web service. The challenge of conducting evaluation from various locations under different network condition can be addressed by conducting evaluation in a collaborative manner. The quality of test cases is greatly improved, since users can contribute individually-designed test cases to supplement the test cases generated automatically by the evaluation server. Therefore, those users who plan to assess the same Web service benefit from the intelligence of each other (due to their individual design of test cases). They also benefit from the cumulated intelligence of prior users who have performed the Web evaluation before. Finally, the evaluation efficiency is also greatly improved, because in contrast to design and implement the evaluation mechanism all by themselves, it is much more efficient to employ the WS-DREAM as proposed in this paper, which can be implemented and launched by a third-party. With WS-DREAM, it is much easier for users to conduct accurate and efficient evaluation of individual Web service or replication strategies.

Chapter 3

Replication Strategies

Dependability is a major issue when applying Web services to critical domains, such as government management systems, commercial trading systems and so on. Nowadays, there are a number of identical or similar Web services available in the Internet, making redundancy-based fault tolerance strategies [28] a natural choice for building reliable service-oriented applications out of unreliable remote Web services. There are two types of redundancy: time redundancy (repetition of computation or communication), and space redundancy (replication of hardware or software) [29]. Only time redundancy and software redundancy will be discussed in this paper; hardware redundancy is outside our scope.

Time redundancy is based on using extra execution time to tolerate faults. $Retry$, which employs time redundancy, is a commonly used strategy for reliability improvement. Although $Retry$ can be employed to mask temporal faults, it will increase response time and cannot tolerate permanent faults. Therefore, space redundancy is also needed to provide better reliability performance.

Space redundancy is based on using extra resources, such as hardware or software, to mask faults. There are two types of software redundancy: active replication and passive replication [30]. Active replication, such as N-Version Programming (NVP), is performed by invoking all replicas at the same time to process the same request. There are two ways to determine the final result: $voting$ and *without voting* [6]. Majority voting is usually employed to mask logical faults by design diversity in traditional fault tolerance techniques. The *without voting* approach involves the first properly returned response as the final outcome. It is usually employed to mask network communication faults and remote *Service Unavailable* faults. Active replication is computing and networking resource consuming.

On the other hand, passive replication, through means such as a recovery block (RB), invokes a primary replica to process the request first. Standby replicas will be invoked only when the primary replica fails. In an erroneous environment, the response time of the passive replication strategy is large, since different replicas are invoked sequentially. Also, performance overheads will be increased by the failure-detection, failure-handling and system

Table 3.1. Combination of Replication Strategies

	NVP	**Retry**	**RB**
NVP	1.NVP	4.NVP+Retry	6.NVP+RB
Retry	5.Retry+NVP	2.Retry	8.Retry+RB
RB	7.RB+NVP	9.RB+Retry	3.RB

reconfiguration actions employed by passive replication.

As shown in Table 3.1, combining the three basic replication strategies: time redundancy ($Retry$), active replication (NVP) and passive replication (RB), will generate more complex replication strategies. The theory and explanation of these replication strategies are as follows:

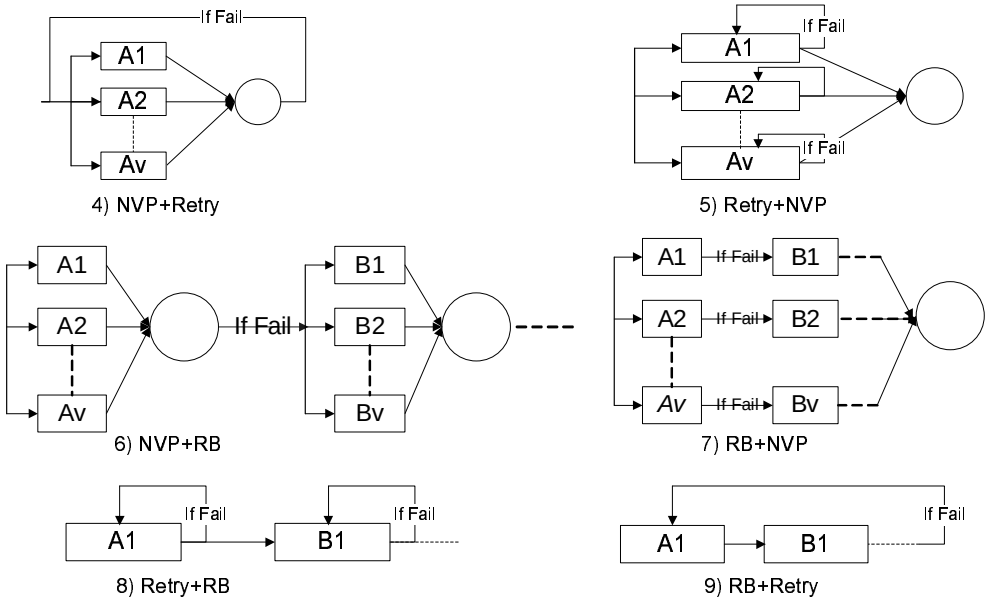

Figure 3.1. Replication Strategies

1) **NVP:** NVP invokes all the n replicas in parallel. Because most failures of Web services are caused by network communication faults or server unavailability, *Without voting* is employed to determine the final result. The system reliability (r) and mission time (t) of this strategy are shown in Eq. (3.1) below, where n is the number of all replicas, T_c is a set of Round-Trip Times (RTT) of the properly returned test cases, and T_f is a set of Round-Trip Times of the failed test cases. When all test cases are failed ($|T_c| = 0$), the max RTT value of the failed test cases is employed as the overall mission time, since NVP does not know itself

fail until all test cases return.

$$r = 1 - \prod_{i=1}^{n}(1 - r_i); t = \begin{cases} \min\{T_c\} : |T_c| > 0 \\ \max\{T_f\} : |T_c| = 0 \end{cases}; T = \{t_1, ..., t_n\} = T_c \cap T_f \tag{3.1}$$

2) **Retry:** Time redundancy is employed in retry. The same Web service will be tried for a certain number of times if it fails. m is the retried times.

$$r = 1 - (1 - r_1)^m; \qquad t = \sum_{i=1}^{m} t_i (1 - r_1)^{i-1}; \tag{3.2}$$

3) **RB:** In this strategy, another standby Web service will be tried sequentially if the original Web service fails. m is the recovered times.

$$r = 1 - \prod_{i=1}^{m}(1 - r_i); \qquad t = \sum_{i=1}^{m} t_i \prod_{k=1}^{i-1}(1 - r_k) \tag{3.3}$$

4) **NVP+Retry:** As shown in Fig. 3.1, only v best performing replicas among all the n replicas are employed. The whole NVP will be re-executed if fails. m is the retried times.

$$r = 1 - (\prod_{i=1}^{v}(1 - r_i))^m; t = \sum_{i=1}^{m} t_i (\prod_{j=1}^{v}(1 - r_j))^{i-1}; t_i = \begin{cases} \min\{T_c^i\} : |T_c^i| > 0 \\ \max\{T_f^i\} : |T_c^i| = 0 \end{cases}; \tag{3.4}$$

5) **Retry+NVP:** As shown in Fig. 3.1, only v best performing replicas among all the n replicas are employed. The replicas in the NVP will be retried individually if they fail.

$$r = 1 - \prod_{i=1}^{v}(1 - r_i)^m; t = \begin{cases} \min\{T_c\} : |T_c| > 0 \\ \max\{T_f\} : |T_c| = 0 \end{cases}; t_i \in T = \sum_{j=1}^{m} t_{ij}(1 - r_i)^{j-1}; \tag{3.5}$$

6) **NVP+RB:** As shown in Fig. 3.1, another NVP employing the secondary v replicas will be tried if the first NVP fails. m is the retried times.

$$r = 1 - \prod_{i=1}^{m}\prod_{j=1}^{v}(1 - r_{ij}); t = \sum_{i=1}^{m} t_i \prod_{k=1}^{i-1}\prod_{j=1}^{v}(1 - r_{kj}); t_i = \begin{cases} \min\{T_c^i\} : |T_c^i| > 0 \\ \max\{T_f^i\} : |T_c^i| = 0 \end{cases}; \tag{3.6}$$

7) **RB+NVP:** As shown in Fig. 3.1, a replica in the NVP will try another standby replica sequentially if it fails. m is the retried times, and v is the number of replicas in the NVP

$$r = 1 - \prod_{j=1}^{v}\prod_{i=1}^{m}(1 - r_{ij}); t = \begin{cases} \min\{T_c\} : |T_c| > 0 \\ \max\{T_f\} : |T_c| = 0 \end{cases} ; t_i \in T = \sum_{j=1}^{m} t_{ij} \prod_{k=1}^{j-1}((1 - r_{ik}); \quad (3.7)$$

8) **Retry+RB:** A replica will retry itself first for m times. Then another standby replica will be executed. Only u best performing replicas are employed among all the n replicas.

$$r = 1 - \prod_{i=1}^{u}(1 - r_i)^m; t = \sum_{i=1}^{u}((\sum_{j=1}^{m} t_i(1 - r_i)^{j-1}) \prod_{k=1}^{i-1}(1 - r_k)^m); \quad (3.8)$$

9) **RB+Retry:** A replica will try another standby replica first if it fails. After trying u replicas without success, the whole RB process will be retried. m is the retried times.

$$r = 1 - (\prod_{i=1}^{u}(1 - r_i))^m; t = \sum_{i=1}^{m}((\sum_{j=1}^{u} t_j \prod_{k=1}^{j-1}(1 - r_k)(\prod_{j=1}^{u}(1 - r_j))^{i-1}); \quad (3.9)$$

These replication strategies can be divided into three types as follows:

1) **Parallel (Strategy 1):** Parallel strategies invoke all standby replicas at the same time and employ the first properly returned response as the final result. They can be employed to obtain good response time performance. However, they consume a considerable amount of computing and networking resources. Also, opening too many network connections at the same time may lead to network jam-up at the client-side, especially when the communication data size is large.

2) **Sequential (Strategies 2, 3, 8 and 9):** Sequential strategies employ $Retry$ or RB to mask faults. The standby replicas are invoked one by one. Sequential strategies consume fewer resources, but suffer from bad response time performance in erroneous environments.

3) **Hybrid (Strategies 4, 5, 6 and 7):** Hybrid strategies combine both sequential and parallel approaches. They only invoke a handful of best performing replicas at the same time, and if these primary replicas fail, $Retry$ or RB will be employed to mask faults. This approach consumes fewer resources than parallel strategies and has better response time performance than sequential strategies.

Chapter 4

Replication Strategy Selection

Optimal replication strategies for service-oriented applications vary from case to case, which are influenced not only by objective replica performance, but also by subjective performance requirement of service users (application developers). For example, developers of latency-sensitive applications may prefer parallel strategies to obtain better response time performance, while developers of resource-constrained applications may prefer sequential strategies for better resource conservation. In this section, based on both objective replica performance and subjective user requirements, we propose an algorithm for replication strategy selection.

The following defines some notations.

$\{ws\}_{i=1}^{n}$: a set of ranked Web service replicas.

t_i: the average Round-Trip Time (RTT) of ws_i.

f_i: the failure-rate of ws_i.

s_i: the overall performance of ws_i.

t_{user} : the response time requirement provided by users.

f_{user} : the failure-rate requirement provided by users.

a: the performance threshold for replicas.

b: the performance degrade threshold for replicas.

c: the failure threshold for replicas.

The subjective user requirements are obtained by requiring the user to provide two values: t_{user} and f_{user}. t_{user} represent the user requirement on response time improvement of increasing one parallel replica. It is designed to facilitate the user to make a tradeoff between the response time performance and resource consuming. t_{user} with small value means response time performance is regarded as more desirable than resource conservation. Such kind of users are more likely to call more replicas in parallel to obtain better response time performance, although more resource will be consumed. f_{user} represents the user requirement on failure-rate of the application.

All the target Web service replicas $\{ws_i\}_{i=1}^{n}$ are ranked by their performance s_i, where ws_1 is the best per-

forming replica (smallest s_i value). The performance of a particular target Web service s_i can be obtained by calculating $s_i = \frac{t_i}{t_{user}} + \frac{f_i}{f_{user}}$. The underlying consideration is that performance of a particular response time is related to user requirement. For example, 100 ms is a large latency for the latency-sensitive applications, while it is neglectable for non-latency-sensitive applications. By using $\frac{t_i}{t_{user}}$, where t_{user} represents the user requirement on response time, we can have a better representation of the response time performance for various users with different requirements. Failure rate is similarly considered.

By finding out the optimal parallel replica number v, the optimal strategy type can be determined as: $Sequential$ ($v = 1$), $Hybrid$ ($1 < v < n$) and $Parallel$ ($v = n$). The value of v can be obtained by solving the following optimization problem:

Problem 1: Given:

- *A set of target Web service replicas* $\{ws_i\}_{i=1}^{n}$*, which are ranked by the performance.*
- *The overall response time performance of employing the first* x *(*$1 \leq x \leq n$*) replicas in parallel* $\mathcal{T}(x)$*, which is obtained by* $\mathcal{T}(x) = \frac{1}{g} \times \sum_{i=1}^{g} t(i,x)$*, where* $t(i,x)$ *is the response time of the* i^{th} *test case by employing* x *parallel replicas, and* g *is the number of test cases.*
- *User's subjective expectation on response time improvement by increasing one parallel replica* t_{user}*.*

Maximize: x*, the number of parallel replicas.*

Subject to:

- $|\mathcal{T}(x) - \mathcal{T}(x-1)| \geq t_{user}$.

If $v = 1$, sequential strategies (Strategies 2, 3, 8 and 9) will be selected. To determine the optimal sequential strategy, the poor performing replicas, which may greatly influence the response time performance of sequential strategies, will be excluded. A set of good performance replicas W will be selected out by using $W = \{ws_i | s_i \leq a \&\& 1 \leq i \leq n\}$, where a is the replica performance threshold. When $|W| = 0$ (no replica meet the performance requirement), the user needs to include other good performing replicas or reduce the performance threshold a. When $|W| = 1$, Strategy 2 ($Retry$) is employed, since all other strategies need space redundant replicas. When $|W| = n$, Strategy 3 (RB) is employed. Otherwise, Strategy 8 ($Retry + RB$) and Strategy 9 ($RB + Retry$) are optimal. p_1, which is the performance degradation between ws_1 and ws_2 obtained by $p_1 = s_2 - s_1$, is employed to find out the optimal strategy between Strategies 8 and 9. When the performance degradation is large ($p_1 \geq b$), retrying the ws_1 first is more likely to obtain better performance (Strategy 8) than invoking the secondary backup replica (Strategy 9).

If $1 < v < n$, hybrid strategies will be selected. p_2, obtained by $p_2 = \frac{1}{v} \sum_{i=1}^{v} (s_{i+v} - s_i)$, represents the performance difference between the primary v replicas and the secondary v replicas. If the performance difference is large ($p_2 \geq b$), retrying the original parallel block first is more likely to obtain better performance (Strategies 4

and 5) than invoking the secondary v backup replicas (Strategies 6 and 7). p_3 is the failure frequency of the first v replicas, which can be calculated by $p_3 = \frac{1}{v}\sum_{i=1}^{v} f_i$. In erroneous environment ($p_3 \geq c$), performance of Strategy 4 and Strategy 6 is not good, because they need to wait for all replicas to fail before retrying/recovering, making the response time longer. Therefore, Strategies 5 and 7 are optimal.

If $v = n$, Strategy 1 (NVP), which invokes all the target replicas in parallel, will be selected to obtain better response time performance. Figure 4.1 shows the strategy selection procedure. First, the sequential, hybrid, and parallel types are selected based on the values of v. Then, the detailed strategy will be determined based on the value of $|W|$, p_1, p_2 and p_3. Values of a, b, c and verifications of this algorithm will be presented in Section 6.3.

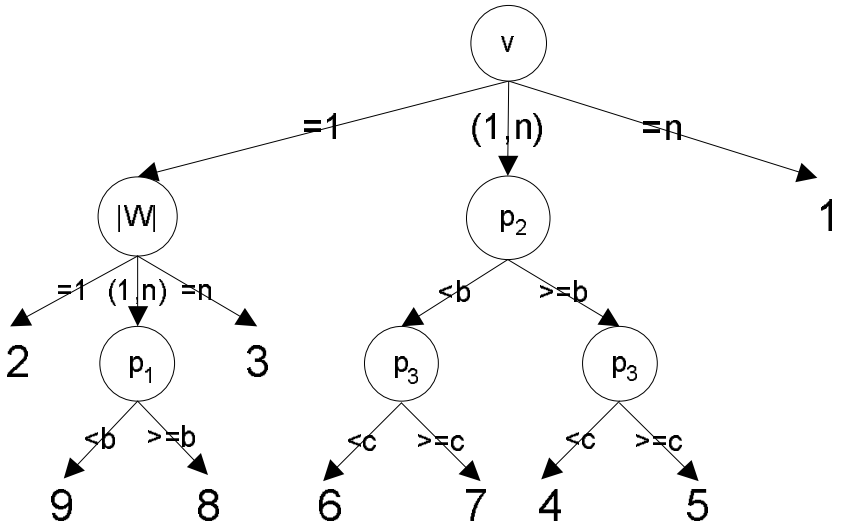

Figure 4.1. Replication Strategy Selection Tree

($R1$): if $v = 1 \&\& |W| = 1 \Rightarrow$ Strategy 2.

($R2$): if $v = 1 \&\& 1 < |W| < n \&\& p_1 < b \Rightarrow$ Strategy 9.

($R3$): if $v = 1 \&\& 1 < |W| < n \&\& p_1 \geq b \Rightarrow$ Strategy 8.

($R4$): if $v = 1 \&\& |W| = n \Rightarrow$ Strategy 3.

($R5$): if $1 < v < n \&\& p_2 < b \&\& P_3 < c \Rightarrow$ Strategy 6.

($R6$): if $1 < v < n \&\& p_2 < b \&\& P_3 \geq c \Rightarrow$ Strategy 7.

($R7$): if $1 < v < n \&\& p_2 \geq b \&\& P_3 < c \Rightarrow$ Strategy 4.

($R8$): if $1 < v < n \&\& p_2 \geq b \&\& P_3 \geq c \Rightarrow$ Strategy 5.

($R9$): if $v = n \Rightarrow$ Strategy 1.

Chapter 5

Implementation

To illustrate the distributed assessment mechanism designed in Section 2, as well as the strategy selection algorithm proposed in Section 4, a prototype [34] is implemented using $Java$. To provide a convenient way for users to conduct testing seamlessly, the client-side evaluation application is realized as a signed *Java Applet*, which can be run and updated automatically by users' Internet browsers. The server-side includes an *HTTP Web site* (running on an Apache HTTP Server), a *TestCaseGenerator* (Java project), a *TestCoodinator* (Java Servlet running on Tomcat 6.0), and a data center for recording results and test cases ($MySQL$).

To provide meaningful illustration of WS-DREAM, more than 1,000,000 test cases are executed by users in six locations (CMU@US, CUHK@HK, NTU@SG, SYSU@CN, NTHU@TW and SUT@AU) all over the world under various network conditions to eight target Web services located in six countries (US, JP, DE, CA, FR and UK). The nine replication strategies discussed in Section 3 are evaluated and compared, and the strategy selection algorithm proposed in Section 4 is verified.

The eight target Web services involved in the experiment include six identical commercial Amazon Web services in different geographical locations [32] for book information displaying (will be used as replicas), a Global Weather Web service [33] for weather information displaying, and a GeoIP Web service [33] for geographical location information querying by IP addresses. The non-commercial Global Weather Web service and GeoIP Web service are involved for making comparison with the commercial Amazon Web services. In this experiment, the timeout threshold is set to be 10 seconds. If a Web service does not respond within the 10-second period, the request will be terminated and a timeout failure is recorded. In practice, the value of timeout threshold is application-dependent and can be set by users based on the need of their applications. Detailed information of test cases, test plans, and test results of the experiment is available in [34].

Three types of experiments are conducted:

1) Evaluation of individual Web services to illustrate the WS-DREAM, as well as to get typical information

for the strategy selection algorithm.

2) Evaluation of various replication strategies for performance comparison.

3) Verification of the strategy selection algorithm by two scenarios.

Chapter 6

Experiments

6.1 Evaluation of Individual Web Services

Table 6.1 shows the detailed experimental results of individual target Web services employing WS-DREAM. *cn, tw, au, sg, hk, us* stand for the six user locations that conducted the evaluation. *a-us, a-jp, a-de, a-ca, a-fr* and *a-uk* stand for the six Amazon Web Services located in US, Japan, Germany, Canada, France, and UK, respectively. *GW* and*GIP* stand for the corresponding Global Weather Web Service and GeoIP Web Service located in the USA. *Cases* shows the failure rate (*R%*), which is the number of failed test cases (*Fail*) divided by the number of all executed test cases (*All*). *RTT* shows the average (*Avg*), standard deviation (*Std*), minimum (*Min*) and maximum (*Max*) values of test case communication Round-Trip-Times (RTT). Only values of correct cases are calculated in the RTT, because most of the failed cases have large RTT values, which distort the accuracy of the result. All time units are in milliseconds (ms).

As shown in Table 6.1, RTT values of target Web services change dramatically from place to place. For example, in our experiment, accessing *a-us* only needs 74 milliseconds on average from the USA, while it needs 4184 milliseconds on average from Mainland China. Moreover, even in the same location, the RTT values vary drastically from case to case, especially in user locations under poor network conditions. As shown in Fig. 6.2, in Mainland China, the RTT values vary from 562 milliseconds to 9926 milliseconds. This RTT variance degrades service quality and affects user experiments.

The experimental result indicates that RTT is mainly made up of network latency. As described in

$$RTT = NetworkLatency + ProT, \tag{6.1}$$

RTT is composed of two parts. *NetworkLatency* is the time occupied by network package transmission, and *ProT* is the time consumed by a Web service server for processing the request. Figure 6.3 shows ProT values of the six Amazon Web services. The ProT of GW and GIP are unavailable, as these two Web services server do not

Table 6.1. Evaluation Results of the Eight Target Web Services

Location		Cases			RTT (ms)				Location		Cases			RTT (ms)			
L	WS	All	Fail	R%	Avg	Std	Min	Max	L	WS	All	Fail	R%	Avg	Std	Min	Max
cn	a-us	484	109	22.52	4184	2348	562	9906	tw	a-us	2470	0	0	902	294	578	4609
	a-jp	482	128	26.55	3892	2515	547	9937		a-jp	2877	1	0.03	791	315	407	5016
	a-de	487	114	23.40	3666	2604	687	9844		a-de	2218	0	0	1155	355	765	4547
	a-ca	458	111	24.23	4074	2539	610	9953		a-ca	2612	5	0.19	899	300	562	4032
	a-fr	498	96	19.27	3654	2514	687	9999		a-fr	2339	0	0	1144	370	734	4813
	a-uk	493	100	20.28	3985	2586	719	9875		a-uk	2647	1	0.03	1150	363	750	5093
	GW	409	337	82.39	6643	2003	2094	9969		GW	1981	35	1.76	1105	1401	343	9844
	GIP	540	32	5.92	2125	1927	531	9781		GIP	2822	60	2.12	732	1270	265	9875
au	a-us	1140	0	0	705	210	500	3782	sg	a-us	1895	0	0	561	353	297	4406
	a-jp	1143	0	0	577	161	406	2594		a-jp	1120	0	0	503	322	250	3687
	a-de	1068	0	0	933	272	672	6094		a-de	1511	0	0	638	409	375	4735
	a-ca	1113	0	0	697	177	500	2672		a-ca	1643	0	0	509	240	297	4125
	a-fr	1090	0	0	924	214	672	2906		a-fr	1635	0	0	638	310	390	5468
	a-uk	1172	3	0.25	921	235	672	3859		a-uk	1615	0	0	650	308	375	4297
	GW	1104	5	0.45	503	544	234	9375		GW	1363	0	0	1403	1544	265	9937
	GIP	1125	0	0	355	609	234	9360		GIP	1312	0	0	571	878	265	9594
hk	a-us	21002	81	0.38	448	304	250	9547	us	a-us	3725	0	0	74	135	31	3171
	a-jp	20944	11	0.05	388	321	203	9937		a-jp	3578	0	0	317	224	109	9219
	a-de	21130	729	3.45	573	346	343	9360		a-de	3766	0	0	298	271	109	9390
	a-ca	21255	125	0.58	440	286	250	9515		a-ca	3591	0	0	239	260	31	9515
	a-fr	21091	743	3.52	575	349	343	9703		a-fr	3933	0	0	433	222	187	3906
	a-uk	20830	807	3.87	570	348	328	9734		a-uk	3614	0	0	293	260	124	9157
	GW	21148	1426	6.74	1563	1560	406	9999		GW	3837	0	0	1290	1346	125	9828
	GIP	21007	1263	6.01	849	1582	203	9999		GIP	3621	0	0	675	1348	125	9938

provide the ProT information. As shown in Fig. 6.3, the average ProT values of all the six Amazon Web services are less than 30 milliseconds, which is very small compared with the RTT, indicating that RTT consist mainly of network latency rather than server processing time. Among all the six Amazon Web services, *a-jp* provides the worst performance, which may be related to the server workload.

Users under poor network conditions are more likely to suffer from unreliable service, since unstable RTT performance will degrade service quality and can even lead to timeout failure. As shown in Fig. 6.1 and Table 6.1, users with worst RTT performance (in Mainland China) have the highest failure rate, while users with best RTT performance (in USA) have the lowest failure rate. This indicates that failures are substantially caused by excessive RTT. Moreover, failure rate is related to request frequency. For example, in our experiment, user in Hong Kong suffer from high failure rate, although its RTT performance is good. The failures arose because the high request frequency make the user blocked by the server from time to time, , triggering a $ServiceUnavailable$ failure (httpcode 503). Among all 6322 failure cases observe in our experiment, 3865 are *Timeout* (longer than

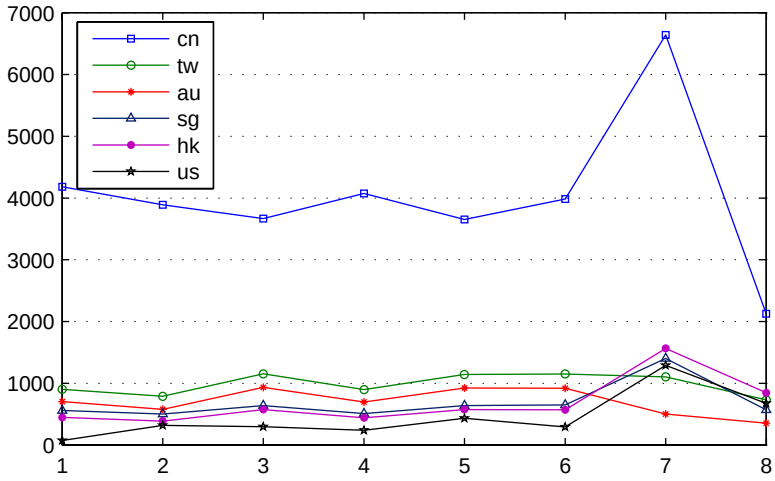

Figure 6.1. RTT of the Eight Web Services from Different Locations

10 seconds), 2456 are *Service Unavailable* (http code 503) and 1 is due to *Bad Gateway* (http code 502). All the *Service Unavailable* occur in Hong Kong, where the request frequency is designed to be much higher than other locations.

The response time and failure-rate performance of the target Web services will be employed for replication strategy selection in Section 6.3.

6.2 Evaluation of Replication Strategies

WS-DREAM also can be employed to conduct performance evaluation of various replication strategies. The six identical Amazon Web services are used as redundant replicas in this experiment to show the performance of various replication strategies. In this experiment, strategy 1 invokes all the six replicas at the same time; in strategies 4, 5, 6 and 7, only the first three best performing replicas are invoked at the same time, while the remaining replicas are employed as standby replicas; in strategies 2, only the best performing replica is employed; in strategy 3, all the six replicas are used as standby replicas; in strategies 8 and 9, only the top three performing replicas are employed.

To clearly show the performance of these strategies in erroneous networking conditions, fault injection techniques [8,9] are applied to generate faulty test cases. Table 6.2 shows the performance of various replication strategies under correct cases and faulty cases.

As shown in Table 6.2, the parallel type strategy (strategy 1:NVP) has the best RTT performance under both the $correctcases$ and $faultycases$, since it invokes all the six replicas at the same time and employs the first properly response as the final result. However, its failure-rate is high compared with other strategies. All the eight failures of this strategy are due to timeout of all the six replicas. This may be caused by a client-side network problem,

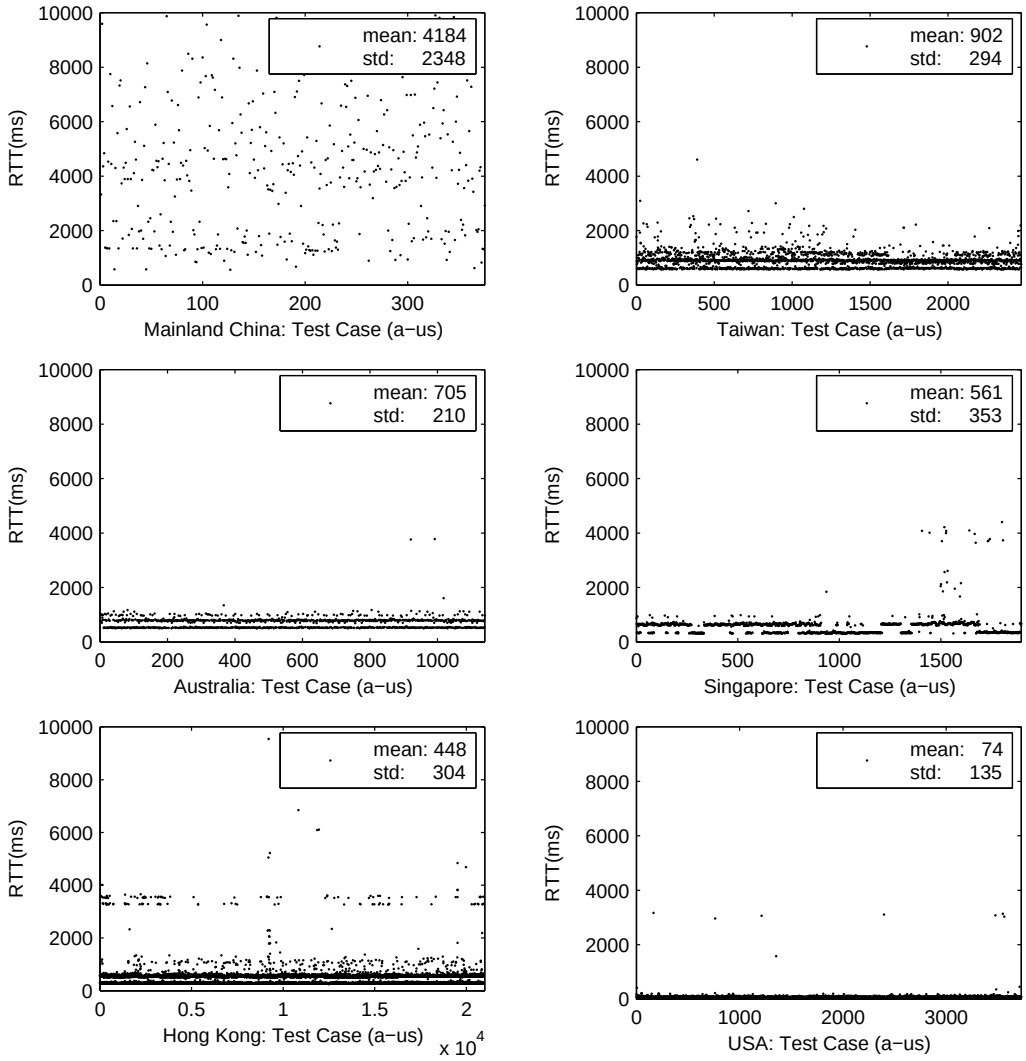

Figure 6.2. RTT of Accessing*a-us* from the Six User Locations

since several connections are opened simultaneously. Nevertheless, the failure rate of 0.027% and 0.097% in the $correctcases$ and $faultycases$ respectively is relatively small compared with the failure rate incurred without employing any replication strategies, as shown in Table 6.1.

RTT performance of sequential type strategies (strategies 2, 3, 8 and 9) is worse than other strategies, especially in the $faultycases$, because they invoke replicas one by one. The reliability performance of these strategies are the best, without any failure. This may be the result of employing time redundancy for fault tolerance, which is not used in the parallel type strategy.

Hybrid type strategies (strategies 4, 5, 6 and 7) have good RTT performance in both the $correctcases$ and $faultycases$, although not the best. The reliability performance is also in the middle, better than parallel type strategy and worse than sequential type strategies. From Fig. 6.4, we can clearly see that the the RTT performance

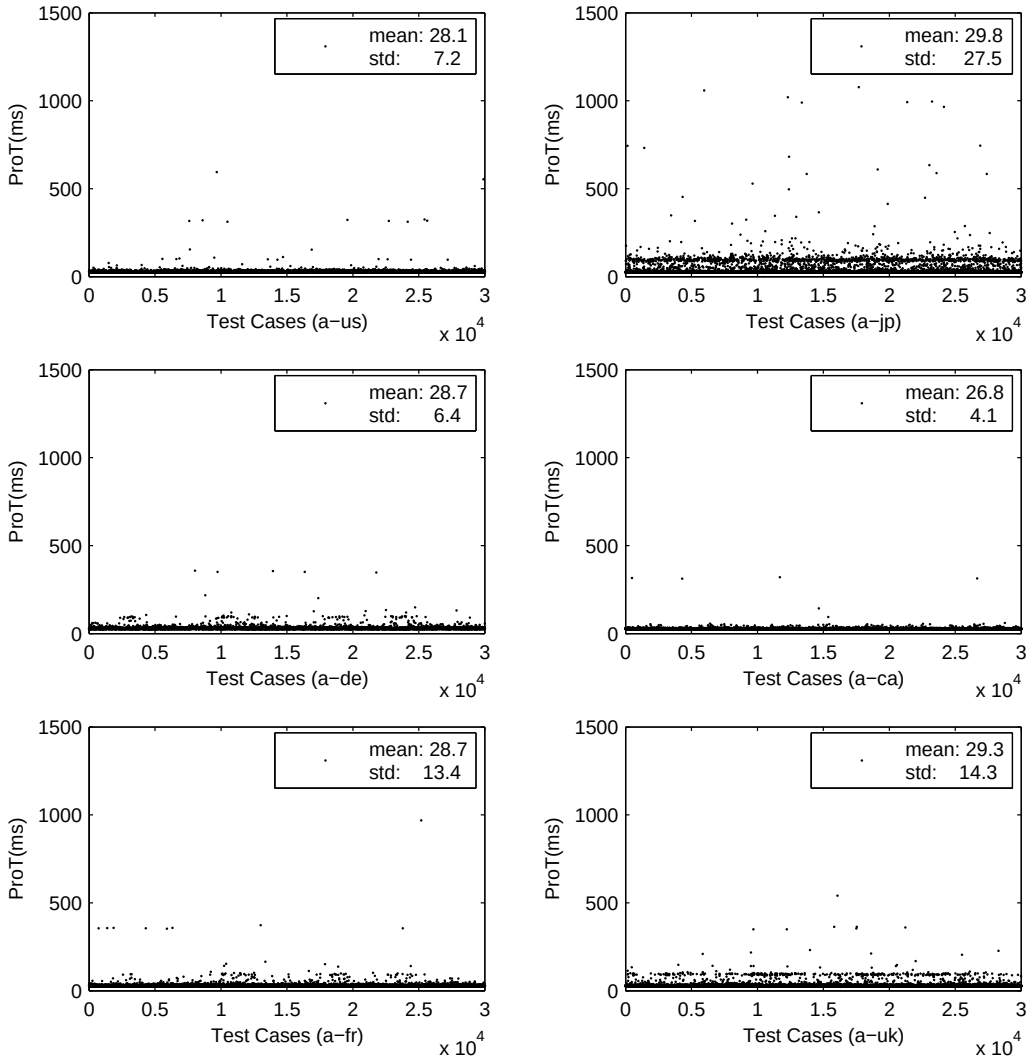

Figure 6.3. Process Time of the Six Amazon Web Services

of strategy 1: NVP (parallel type) is the best, the RTT performance of strategy 2: $Rety$ (Sequential type) is in the worst, and the RTT performance of strategy 4: $NVP + Rety$ (Hybrid type) is in the middle.

6.3 Replication Strategy Selection Scenarios

We provide two scenarios in this section to illustrate and verify the strategy selection algorithm. The values of a, b and c in the algorithm are set to be $20, 5, 5\%$, respectively.

Table 6.2. Evaluation Results of Replication Strategies

Type	Correct Cases							Faulty Cases						
	All	Fail	R%	Avg	Std	Min	Max	All	Fail	R%	Avg	Std	Min	Max
1	21556	6	0.027	279	153	203	3296	2043	2	0.097	321	163	203	3375
2	22719	0	0	389	333	203	17922	2460	0	0	751	721	203	11312
3	23040	0	0	374	299	203	8312	2495	0	0	833	681	203	6031
4	21926	4	0.018	311	278	203	10327	2412	0	0	397	376	203	10375
5	21926	1	0.004	312	209	203	10828	2393	0	0	401	259	203	3781
6	21737	2	0.009	311	225	203	10282	2318	0	0	425	384	203	8000
7	21737	2	0.009	310	240	203	13953	2350	0	0	420	326	203	3781
8	21735	0	0	411	1130	203	51687	2400	0	0	761	961	203	35031
9	21808	0	0	388	304	203	9360	2335	0	0	765	694	203	9953

6.3.1 Scenario 1: Commercial Web site in Hong Kong

We assume a user named Ben in Kong Hong plans to employ the Amazon Web services for book displaying and selling in his commercial Web site. The followings are performance requirements provided by Ben:

1) **Reliability.** Since the Web site is commercial, Ben aims to make it as reliable as possible to maximize business benefit and reputation. Therefore, the fail-rate (f_{user}) is set to be 0.1%.

2) **Response time & resource conservation.** Too large response latency will lead to loss of business; however, invoking too many parallel replicas for response time improvement will increase computing and networking overhead to the Web site server. After making a tradeoff, Ben sets the t_{user} to be 100 milliseconds.

The strategy selection algorithm proposed in Section 4 is employed to help Ben select the optimal strategy. The selection procedure is shown in Table 6.3.

Based on the strategy selection algorithm proposed in Section 4, the selection procedure is shown in Table 6.3, where $\{ws\}_{i=1}^{6}$ is a set of ranked target Web services. Values of t_i, f_i are provided by WS-DREAM (see Table 6.1 for detailed results). $\mathcal{T}(i)$ is the overall RTT values of invoking i number of parallel replicas, the values of which are also provided by WS-DREAM. Based on the values of $\mathcal{T}(i)$ and t_{user}, the value of v is calculated by solving the *Problem 1* in Section 4. Since $v = 1$, sequential type strategy will be selected. Because $|W| = 3$ (only the top three performing replicas are selected), and $p_1 < 5$ the difference between primary replica and secondary replica is not significant), Strategy 9 ($RB + Retry$) is selected.

As shown in Table 6.1, from the location of Hong Kong, network condition is good and the failure-rate is low. The improvement of invoking replicas in parallel is quite limited; therefore, sequential strategies are reasonable. Our algorithm can provide suitable selection in this scenario.

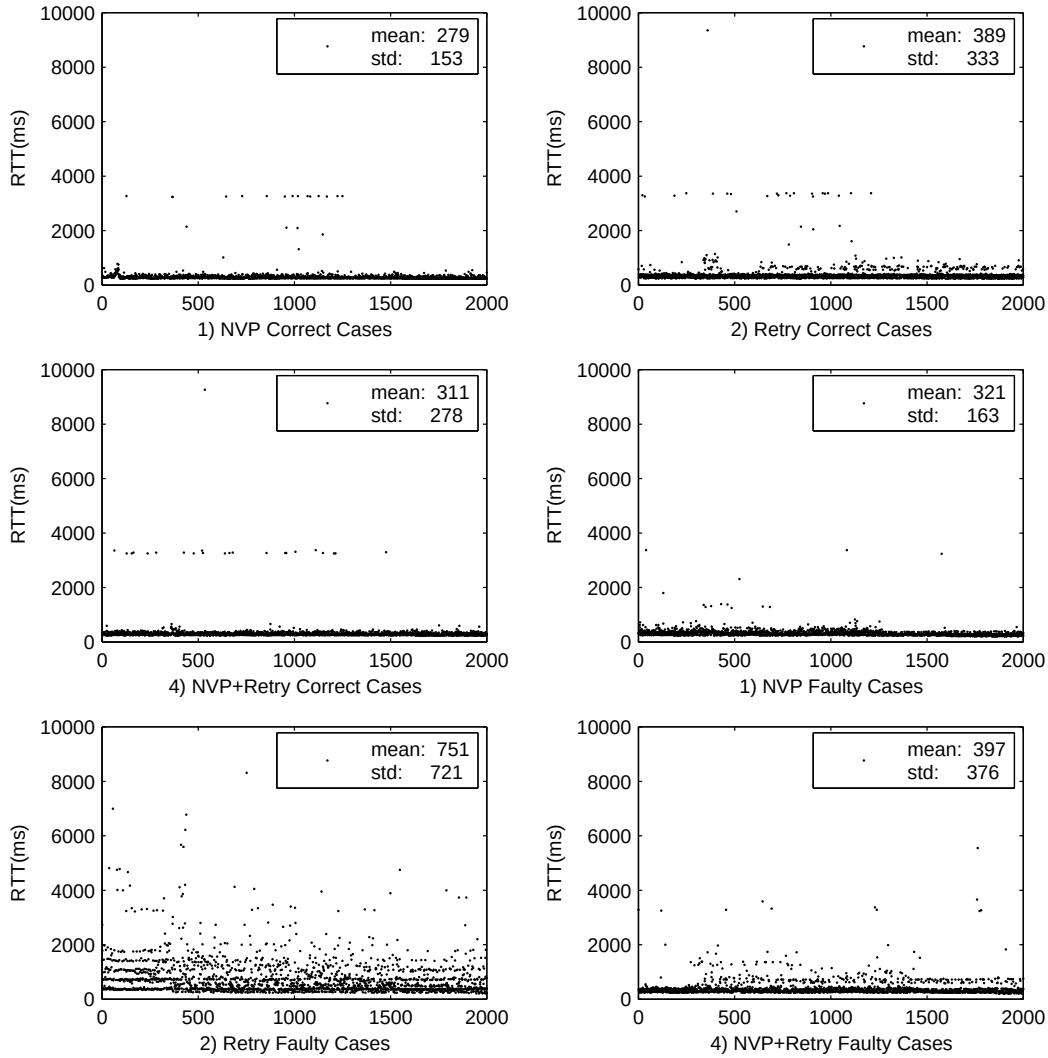

Figure 6.4. RTT Performance of Replication Strategies

6.3.2 Scenario 2: Noncommercial Web page in Mainland China

Another user named Tom in Mainland China also plans to employ the Amazon Web services to provide book information query service in his personal Home page. The performance requirements of Tom are as follows:

1) **Reliability.** Since the Home page of Tom is noncommercial and the Web service is not used for critical purposes, the fail-rate (f_{user}) is set to be 5%.

2) **Response time & resource conservation.** Since Tom's Home page are running on a server with narrow network bandwidth, network resource conservation is important. Therefore, the t_{user} is set to be 500 milliseconds.

Table 6.3. Scenario 1: Selection Procedure

$a = 20; b = 5; c = 5\%$

$n = 6; g = 21587; t_{user} = 100; f_{user} = 0.1\%$

$\{ws_i\}_{i=1}^{6}$ =a-jp, a-us, a-ca, a-de, a-fr, a-uk};

$\{t_i\}_{i=1}^{6} = \{388, 448, 440, 573, 575, 570\};$

$\{f_i\}_{i=1}^{6} = \{0.05\%, 0.38\%, 0.58\%, 3.45\%, 3.52\%, 3.87\%\};$

$\{s_i\}_{i=1}^{6} = \{4.38, 8.28, 10.2, 40.23, 40.95, 44.4\};$

$\{\mathcal{T}(i)\}_{i=1}^{6} = (321, 285, 282, 281, 280, 279);$

$v = 1;$

$W = \{ws_i | s_i =\leq 20 \&\& 1 \leq i \leq 6\} = \{4.38, 8.28, 10.2\};$

$|W| = 3;$

$p_1 = s_2 - s_1 = 3.9;$

$v = 1 \&\& 1 < |W| < 6 \&\& p_1 < 5 \Rightarrow$ Strategy 9;

Table 6.4. Scenario 2: Selection Procedure

$a = 20; b = 5; c = 5\%$

$n = 6; g = 576; t_{user} = 500; f_{user} = 5\%$

$\{ws_i\}_{i=1}^{6}$ =(a-fr, a-jp, a-de, a-uk, a-us, a-ca);

$\{t_i\}_{i=1}^{6} = (3654, 3192, 3666, 3985, 4184, 4074);$

$\{f_i\}_{i=1}^{6} = (19.27\%, 26.55\%, 23.4\%, 20.28\%, 22.52\%, 24.23\%);$

$\{s_i\}_{i=1}^{6} == (11.16, 11.69, 12.01, 12.02, 12.87, 12.99);$

$\{\mathcal{T}(i)\}_{i=1}^{6} = (4462, 3052, 2344, 1920, 1686, 1491);$

$v = 3;$

$p_2 = \frac{1}{3} \times \sum_{i=1}^{3} s_{i+v} - s_i = 1.01;$

$p_3 = \frac{1}{3} \times \sum_{i=1}^{3} f_i = 23.07\%;$

$1 < v < 6 \&\& p_2 < 5 \&\& p_3 \geq 5\% \Rightarrow$ Strategy 7.

After conducting the selection procedure as shown in Table 6.4, Strategy 7 ($RB + NVP$) with three replicas is selected as the optimal strategy for Tom. In this scenario, the network condition is poor and failure-rate is high, hybrid strategy with suitable number of parallel replicas can employed to improve the performance. Our algorithm can provide suitable selection for this scenario.

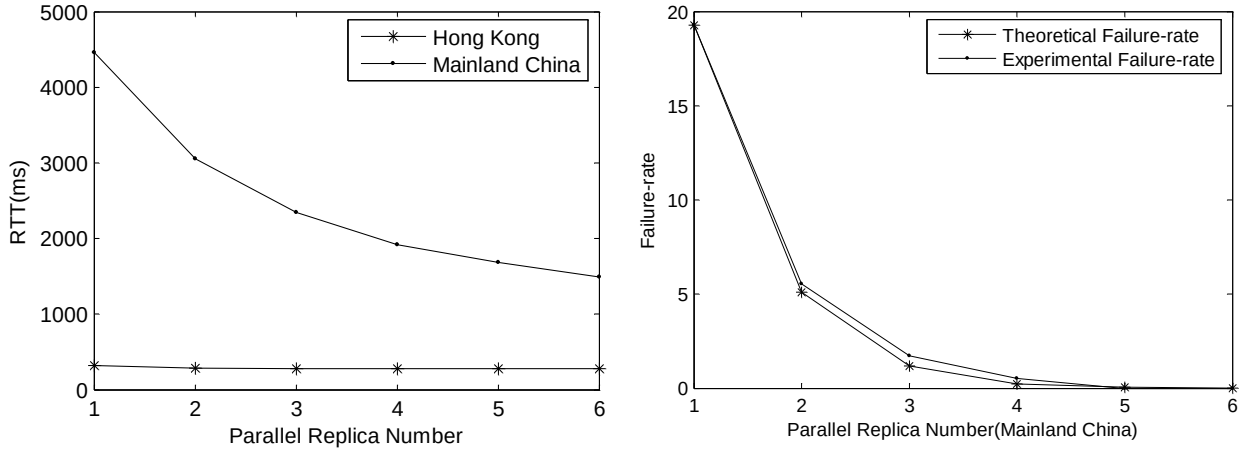

Figure 6.5. (a)RTT and (b)Failure-rate Performance with Different Replica Number

The detailed RTT and failure-rate performance with different replica number of these two scenarios are shown in Fig.6.5, where Fig. 6.5(a) shows the RTT performance, and Fig. 6.5(b) shows the failure-rate performance. Fig.6.5 (a) indicates that response time improvement by invoking parallel replicas is significant under poor network condition (Scenario 2: Mainland China), while under good network condition (Scenario 1: Hong Kong), the improvement is limited. Fig.6.5 (b) shows that failure-rate is greatly reduced by invoking parallel replicas in Mainland China, indicating that the failure-rate improvement by invoking replicas in parallel is significant under erroneous environment, while in Hong Kong it is not obvious and unnecessary (failure-rate of Hong Kong is not shown in the figure since all values are 0 with parallel replicas). Also, Fig.6.5 (b) shows that the experimental failure-rate observed in Mainland China is quite close to the theoretical failure-rate, which can be calculated by $\prod_{i=1}^{v} f_i$, indicating the accuracy of our experiment.

In summary, by employing the evaluation results provided WS-DREAM, the replication strategy selection algorithm can provide suitable selections for users in these two scenarios. When the general property of the Web service execution scenarios can be obtained and analyzed, a systematic selection procedure under various replication strategies can be quantitatively formulated, and more inclusive mathematical models can be constructed for a comprehensive system assessment. This will be pursued in our future work.

Chapter 7

Conclusion

This paper proposes a distributed reliability assessment mechanism for Web services. Based on this mechanism, we compare various replication strategies by using theoretical formulas and experimental results. A strategy selection algorithm is also proposed. Real world experiments on Web services and replication strategies are conducted to illustrate the mechanism as well as verify the selection algorithm. With the facility of WS-DREAM, accurate evaluation of target Web services can be acquired though user collaboration, and optimal replication strategies engaging fault tolerance and design diversity schemes can be effectively obtained to improve the reliability of service-oriented applications.

Our future work will include an automatic mechanism for users to search for similar or identical Web Services as replicas for design diversity purpose, the tuning of selection algorithm for better performance, and the improvement of the system feature for facilitating user test case contributions.

Acknowledgements

I would like to thank Prof. Michael R. Lyu for his kind guidance. I also thank Dr. Jianke Zhu for providing the template of this report.

Bibliography

[1] WS-ReliableMessaging, http://docs.oasis-open.org

[2] W.T. Tsai, R Paul, L Yu, A Saimi, Z Cao,"Scenario-Based Web Service Testing with Distributed Agents," IEICE Transaction on Information and System, Vol. E86-D, No.10, pp. 2130-2144, 2003.

[3] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based Verification of Web Service Compositions", in 18th IEEE International Conference on Automated Software Engineering, pp.152-161, 2003.

[4] P. W. Chan, M.R. Lyu and M. Malek, "Reliable Web Services: Methodology, Experiment and Modeling," in IEEE International Conference on Web Services, pp. 679-686, 2007.

[5] P.W. Chan, M.R. Lyu, and M. Malek, "Making Services Fault Tolerant," in the 3rd International Service Availability Symposium (ISAS 2006), Helsinki, Finland, May 15-16, pp. 43-61, 2006.

[6] N. Salatge and J.C. Fabre, "Fault Tolerance Connectors for Unreliable Web Services," in 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), Edinburgh, UK, pp. 51-60, 2007.

[7] D. Liang, C. Fang, and C. Chen, "FT-SOAP: A Faulttolerant web service," in Tenth Asia-Pacific Software Engineering Conference, Chiang Mai, Thailand, 2003.

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

[9] G. T. Santos, L. C. Lung, and C. Montez, "FTWeb: A Fault Tolerant Infrastructure for Web Services," in Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05), pp. 95-105, September 2005.

[10] M. G. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan, "Thema: Byzantine-faulttolerant middleware for Web-service applications", in IEEE Symposium on Reliable Distributed Systems, pp. 131 - 140, 2005.

[11] J. Salas, F. Perez-Sorrosal, M. Patino-Martinez, and R. Jimenez-Peris, ”WS-Replication: a framework for highly available web services”, In 15th International Conference on the World Wide Web, pp. 357-366, 2006.

[12] J. Osrael, L. Froihofer, K. M. Goeschka, S. Beyer, P. Galdamez, and F. Munoz, ”A system architecture for enhanced availability of tightly coupled distributed systems”, In 1st International Conference on Availability, Reliability and Security (ARES06), pp. 400C407, 2006.

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

[14] 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.

[15] N. Looker, M. Munro, and J. Xu, ”Increasing Web Service Dependability Through Consensus Voting”, 2nd Int. Workshop on Quality Assurance and Testing of Web-Based Applications, COMPSAC, Edinburgh, Scotland, July, 2005.

[16] L.Z. Zeng, B. Benatallah, A.H.H.Ngu, M.Dumas, J.Kalagnanam, and H.Chang, ”Qos-Aware Middleware for web services composition”, IEEE transaction on software engineering, Vol.30, No.5, May 2004.

[17] E.M. Maximilien, and M.P.Singh, ”Conceptual Model of Web Service Reputation”, ACM SIGMOD Record (Special section on semantic web and data management), vol.31, issue 4, pp.36-41, 2002.

[18] G. Wu, J. Wei, X. Qiao and L. Li, ”A Bayesian network based Qos assessment model for web services”, in IEEE International Conference on Services Computing(SCC 2007), Utah, USA, pp.498-505, 2007.

[19] V. Deora, J.H. Shao, W.A.Gray and N.J.Fiddian, ”A Quality of service management framework based on user expectations”, in First International Conference on Service Oriented Computing (ICSOC 2003), Trento, Italy, 2003.

[20] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy, ”Dependability in the Web Services Architecture,” In Architecting Dependable Systems. LNCS 2677, June 2003.

[21] L.H. Vu, M. Hauswirth, and K.Aberer. Qos-Based Service Selection and Ranking with Trust and Reputation Management. Coopis 2005.

[22] M. Vieira, N. Laranjeiro, and He. Madeira, ”Assessing Robustness of Web-Services Infrastructures,” in 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07), Edinburgh, UK, pp. 131-136, 2007.

[23] Z. Zheng, M.R. Lyu, ”WS-DREAM: A Distributed Reliability Assessment Mechanism for Web Services”, in 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’08), Anchorage, Alaska, USA, June 24-27, 2008.

[24] BitTorrent, http://www.bittorrent.com

[25] Wikipedia, http://www.wikipedia.org

[26] X. Bai, W. Dong, W.T. Tsai and Y. Chen (2005),”WSDL-Based Automatic Test Case Generation for Web Services Testing”, Proceedings of IEEE Workshop on Service-Oriented System Engineering, pp. 207 - 212, 2005.

[27] N. Looker and J. Xu, ”Assessing the Dependability of Soaprpc-based Web Services by Fault Injection,” In Proc. of the 9th IEEE International Workshop on Object-oriented Real-time Dependable Systems, pp. 163-170, 2003.

[28] M.R. Lyu, ”Software Fault Tolerance”, Wiley Trends in Software book series, John Wiley & Sons, Chichester, February 1995.

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

[30] D. Pradhan, Editor, Fault-Tolerant Computing: Theory and Techniques Vol. II, Prentice Hall, Englewood Cliffs, NJ (1986).

[31] J. Wu and Z. Wu, ”Similarity-based Web Service Matchmaking,” in IEEE International Conference on Services Computing (SCC’05), Vol-1, pp. 287-294, 2005.

[32] Amazon Web Service Homepage, http://aws.amazon.com

[33] WebServiceX.NET, http://www.webservicex.net

[34] WS-DREAM, http://www.wsdream.net