A Distributed Replication Strategy Evaluation and Selection Framework for Fault Tolerant Web Services

Zibin Zheng and Michael R. Lyu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
{zbzheng, lyu}@cse.cuhk.edu.hk

Abstract

Redundancy-based fault tolerance strategies are proposed for building reliable Service-Oriented Architectures/Applications (SOA), which are usually developed on the unpredictable remote Web services. This paper proposes and implements a distributed replication strategy evaluation and selection framework for fault tolerant Web services. Based on this framework, we provide a systematic comparison of various replication strategies by theoretical formula and real-world experiments. Moreover, a user-participated strategy selection algorithm is designed and verified. Experiments are conducted to illustrate the advantage of this framework. 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.

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 time, 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 allow 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, network disconnect, and so on), and poor performance (long latency, high failure-rate, and so on), are remain unsolve. A number of application/Web service level fault tolerance strategies have been proposed in the recent literature for establishing reliable service-oriented applications [2, 3, 4]. 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 [5]. Passive replication, which employs a primary replica to process the service request first and invokes backup replicas only when the primary replica fails, has been employed in FT-SOAP [6] and FT-CORBA [7]. Active replication, which invokes all replicas at the same time and employs the first properly returned response as the final outcome, has been employed in FTWeb [8], Thema [9], WS-Replication [10] and in work [11].

It is a challenge for service-oriented application developers to determine the optimal replication strategy, which requires not only performance information of the target Web services, but also good knowledge on various available fault tolerance replication strategies. Comparing with the numerous investigations on individual Web service evaluation [12, 13, 14, 15], the investigations on faut tolerance replication strategies evaluation and selection is still limited. Assuming that a user named Ben plans to build a service-oriented Web site. By using the approaches proposed in [5, 16], he has obtained several appropriate Web services with identical interface to serve as alternative replicas. However, determining the optimal fault tolerance replication strategy becomes a challenge for Ben, since he has no idea on the performance of target replicas as well as the features of various available replication strategies. To address this challenge, this paper proposes a distributed replication strategy evaluation and selection framework for service users. The contribution of this paper includes:

- Design and implement a distributed evaluation and selection framework 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.

This paper is organized as follows: Section 2 proposes the distributed evaluation framework. Section 3 introduces various replication strategies. Section 4 provides an optimal strategy selection algorithm. Section 5 implements a prototype of our evaluation framework. Section 6 presents experimental results, and Section 7 concludes the paper.

2. A Distributed Evaluation Framework

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

- **Evaluation location:** The service users are usually from different locations with different network conditions. Therefore, conducting evaluation on the target Web services from various locations is necessary.
- **Evaluation accuracy:** Few service users have good knowledge on replication strategies, test case generation [17], test result analysis and so on, making accurate replication strategies evaluation difficult.
- **Evaluation efficiency:** It is time-consuming for service users to conduct evaluation themselves. More efficient approaches are needed.

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 strategy evaluation framework to address the above challenges. As introduced in [18], this framework employs the concept of user-collaboration, which has contributed to the recent success of BitTorrent [19] and Wikipedia [20]. In this framework, 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. As shown in Fig.1, the proposed distributed evaluation framework 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, strategies selection parameters, and so on, to the server.
2. **Client-side application loading:** A client-side evaluation application is loaded to the user's computer.

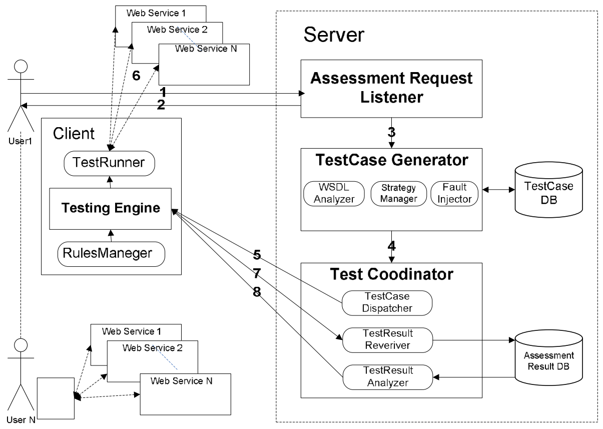

Figure 1. Distributed Evaluation Framework

3. **Test case generation:** The *TestCase Generator* in the server automatically creates test cases based on the interface of the target Web Services (WSDL files). Two types of test cases are created: single test cases for individual Web service evaluation, and multiple test cases for replication strategy evaluation.
4. **Test coordination:** Test tasks are scheduled based on the number of current users and test cases.
5. **Test cases retrieval:** Distributed client-side applications get test cases from the server.
6. **Test cases execution:** Distributed client-side applications execute testing on the target Web services.
7. **Test result collection:** Test results are sent back to the server. Then steps 5, 6 and 7 are repeated to retrieval and execute more test cases.
8. **Test cases analysis:** After the test is completed, a *TestResult Analyzer* is engaged to process the collected data and send back the detailed results to the user.

By this framework, the challenge of conducting evaluation from various locations can be addressed by collaborating with other users. The challenges of evaluation accuracy and efficiency also can be addressed, since this framework can be implemented and launched by a third-party to help service users conduct accurate and efficient Web service evaluation in an easier way, without requiring service users to have professional knowledge on evaluation design, test case generation, test result interpretation, and so on.

3. Replication Strategies

Dependability is a major issue when applying service-oriented applications to critical domains [21]. Web service level redundancy-based fault tolerance strategy [22] is a feasible approach for building reliable service-oriented applications. There are two types of redundancy: time redundancy, and space redundancy [23]. Time redundancy is based on using extra computation or communication time

Table 1. Replication Strategy Combinations

	Active	Time	Passive
Active	1.Actie	4.Active+Time	6.Active+Passive
Time	5.Time+Active	2.Time	8.Time+Passive
Passive	7.Passive+Active	9.Passive+Time	3.Passive

to tolerate faults, while space redundancy is based on using extra resources, such as hardware or software, to mask faults. Space redundancy includes Active replication and passive replication.

As shown in Table 1, combining $Time$ redundancy, $Active$ replication and $Passive$ replication can produce nine more sophisticated replication strategies. Strategies named *A+B* means that Strategy *A* is employed at the lower level and Strategy *B* at the higher level. The formula of calculating failure rate and response time of different strategies are shown in Table 2. The p, s and h in Table 2 represent $parallel$, $sequential$ and $hybrid$ strategy type respectively, which will be introduced later. As discussed in the work [23], we assume the remote Web services are failed in a fixed rate, and the execution of each test cases is independent (stateless Web services). The introduction of these replication strategies are in the following:

1. **Active:** All the n replicas are invoked in parallel. The system reliability (r) and mission time (t) of this strategy are shown in Table 2, where n is the number of replicas, T_c is a set of Round-Trip Times (RTT) of the properly returned test cases, and T_f is a set of RTT of the failed test cases. When all the test cases are failed ($|T_c| = 0$), the max RTT value is employed as the mission time, since $active$ strategy does not know itself about the failure until all test cases return.
2. **Time:** The original Web service will be tried for a certain times if it fails. m is the retried times.
3. **Passive:** Another backup Web service will be tried sequentially if the primary Web service fails. m is the backup recovery times.
4. **Active+Time:** The v best performing replicas among all the n replicas are invoked in parallel. All the selected v replicas will be re-executed if all of them fail. m is the retried times.
5. **Time+Active:** The v best performing replicas are invoked in parallel. Replicas will be retried individually if they fail.
6. **Active+Passive:** Another backup v replicas will be tried if all of the primary v replicas fail. m is the recovery times.
7. **Passive+Active:** An individual replica in the primary v replicas will try another backup replica sequentially if it fails. m is the recovery times.
8. **Time+Passive:** The primary replica will retry itself first for m times. Then another backup replica will be executed. Only u best performing replicas are employed among all the n replicas.

Table 2. Replication strategy formula

	Formula
$\overset{p}{1}$	$r = 1 - \prod_{i=1}^{n}(1-r_i);$ $t = \begin{cases} \min\{T_c\} : \lvert T_c\rvert > 0 \\ \max\{T_f\} : \lvert T_c\rvert = 0 \end{cases} ; T = \{t_1, ..., t_n\} = T_c \cup T_f$
$\overset{s}{2}$	$r = 1 - (1-r_1)^m; t = \sum_{i=1}^{m} t_i(1-r_1)^{i-1};$
$\overset{s}{3}$	$r = 1 - \prod_{i=1}^{m}(1-r_i); t = \sum_{i=1}^{m} t_i \prod_{k=1}^{i-1}(1-r_k)$
$\overset{h}{4}$	$r = 1 - (\prod_{i=1}^{v}(1-r_i))^m;$ $t = \sum_{i=1}^{m} t_i(\prod_{j=1}^{v}(1-r_j))^{i-1}; t_i = \begin{cases} \min\{T_c^i\} : \left\lvert T_c^i\right\rvert > 0 \\ \max\{T_f^i\} : \left\lvert T_c^i\right\rvert = 0 \end{cases}$
$\overset{h}{5}$	$r = 1 - \prod_{i=1}^{v}(1-r_i)^m;$ $t = \begin{cases} \min\{T_c\} : \lvert T_c\rvert > 0 \\ \max\{T_f\} : \lvert T_c\rvert = 0 \end{cases} ; t_i \in T = \sum_{j=1}^{m} t_{ij}(1-r_i)^{j-1}$
$\overset{h}{6}$	$r = 1 - \prod_{i=1}^{m}\prod_{j=1}^{v}(1-r_{ij});$ $t = \sum_{i=1}^{m} t_i \prod_{k=1}^{i-1}\prod_{j=1}^{v}(1-r_{kj}); t_i = \begin{cases} \min\{T_c^i\} : \left\lvert T_c^i\right\rvert > 0 \\ \max\{T_f^i\} : \left\lvert T_c^i\right\rvert = 0 \end{cases}$
$\overset{h}{7}$	$r = 1 - \prod_{j=1}^{v}\prod_{i=1}^{m}(1-r_{ij});$ $t = \begin{cases} \min\{T_c\} : \lvert T_c\rvert > 0 \\ \max\{T_f\} : \lvert T_c\rvert = 0 \end{cases} ; t_i = \sum_{j=1}^{m} t_{ij} \prod_{k=1}^{j-1}((1-r_{ik})$
$\overset{s}{8}$	$r = 1 - \prod_{i=1}^{u}(1-r_i)^m;$ $t = \sum_{i=1}^{u}((\sum_{j=1}^{m} t_i(1-r_i)^{j-1}) \prod_{k=1}^{i-1}(1-r_k)^m);$
$\overset{s}{9}$	$r = 1 - (\prod_{i=1}^{u}(1-r_i))^m;$ $t = \sum_{i=1}^{m}((\sum_{j=1}^{u} t_j \prod_{k=1}^{j-1}(1-r_k)(\prod_{j=1}^{u}(1-r_j))^{i-1});$

9. **Passive+Time:** A replica will try another backup replica first if it fails. After trying u replicas without success, all the u replicas will be retried sequentially. m is the retried times.

These strategies can be divided into three types:

- **Parallel** (Strategy 1): All replicas are invoked at the same time. Parallel type strategies can be employed to obtain good response time performance, however, consume more resources.
- **Sequential** (Strategies 2, 3, 8 and 9): Replicas are invoked sequentially. Sequential strategies consume fewer resources, but suffer from bad response time performance in erroneous environments.
- **Hybrid** (Strategies 4, 5, 6 and 7): Some replicas are invoked in parallel. Hybrid strategies consume fewer resources than parallel strategies and have better response time performance than sequential strategies.

4. A Replication Strategy Selection Algorithm

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 requirements 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 of service users.

f_{user} : the failure-rate requirement of service 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 consume more resources (invoke more replicas in parallel) to obtain better response time performance. f_{user} represents the user requirement on the service-oriented application failure-rate.

All the target Web service replicas $\{ws_i\}_{i=1}^{n}$ are ranked by their performance s_i, where ws_1 is the best performing replica (smallest s_i value). The performance of a particular target Web service s_i can be obtained by $s_i = \frac{t_i}{t_{user}} + \frac{f_i}{f_{user}}$. The underlying consideration is that response time performance of a particular Web service 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 $\frac{t_i}{t_{user}}$, we can have a better representation of the response time performance for service users. 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 ($Time$) is employed, since all other strategies need space redundant replicas. When $|W| = n$, Strategy 3 ($Passive$) is employed. Otherwise, Strategy 8 ($Time + Passive$) and Strategy 9 ($Passive + Time$) 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 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 ($Active$), which invokes all the target replicas in parallel, will be selected to obtain better response time performance. Figure 2 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.

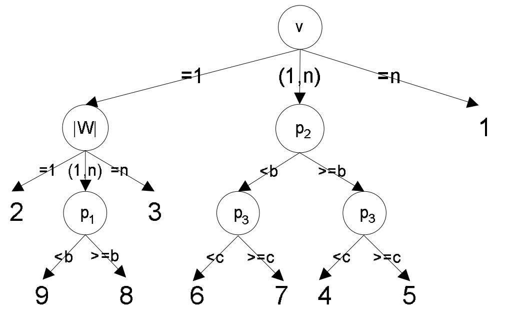

Figure 2. Replication Strategy Selection Tree

5. Implementation

To illustrate the distributed evaluation framework and the replication strategy selection algorithm, a prototype [26] is implemented. 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 *HTML Web site*, a *TestCaseGenerator* (Java application), a *TestCoodinator* (Java Servlet), and a data center for recording results and test cases ($MySQL$).

To provide meaningful illustration of our framework, 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) 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 [24] for book information displaying, a Global Weather Web service [25] for weather information displaying, and a GeoIP Web service [25] 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. 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 [26].

6. Experiments

6.1. Evaluation of Individual Web Services

Table 3 shows the detailed evaluation results of individual target Web services provided by our distributed evaluation framework. *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, which are located in the USA. *Cases* column 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 3, RTT values of the target Web services change dramatically from place to place. For example, in our experiment, accessing *a-us* only needs 74 milliseconds on average from USA, while it requires 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. For example, in Mainland China, the RTT values of accessing *a-us* vary from 562 milliseconds to 9906 milliseconds. This RTT variance degrades service quality and affects user experiments.

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 Table 3, users with the worst RTT performance (in Mainland China) have the highest failure rate, while users with the best RTT performance (in USA) have the lowest failure rate. This indicates that failures are related to network conditions. Among all 6322 failure cases observed in our experiment, 3865 are *Timeout*, 2456 are *Service Unavailable* (http code 503) and 1 is due to *Bad Gateway* (http code 502).

6.2. Evaluation of Replication Strategies

In this experiment, the six identical Amazon Web services are used as redundant replicas for fault tolerance purpose. Table 4 shows the performance of various replication strategies from the location of Hong Kong. We can see that Strategy 1 ($Active$) has the best RTT performance. This is reasonable, because 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, which may be caused by opening too many connections simultaneously. Nevertheless, the failure rate of 0.027% is relatively small compared with the failure rate incurred without employing any replication strategies, as shown in Table 3.

Table 3. 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

Table 4. Evaluation of Replication Strategies

Type	Cases			RTT(ms)			
	All	Fail	R%	Avg	Std	Min	Max
1	21556	6	0.027	279	153	203	3296
2	22719	0	0	389	333	203	17922
3	23040	0	0	374	299	203	8312
4	21926	4	0.018	311	278	203	10327
5	21926	1	0.004	312	209	203	10828
6	21737	2	0.009	311	225	203	10282
7	21737	2	0.009	310	240	203	13953
8	21735	0	0	411	1130	203	51687
9	21808	0	0	388	304	203	9360

RTT performance of sequential type strategies (Strategies 2, 3, 8 and 9) is worse than other strategies, because they invoke replicas one by one. The reliability performance of these strategies is the best (without any failure). Hybrid type strategies (Strategies 4, 5, 6 and 7) achieve good RTT performance, although not the best. The reliability performance is also in the middle, better than parallel type strategy and worse than sequential type strategies.

6.3. 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 (more experience is needed for better tuning of these a, b and c values).

Scenario 1: Commercial Web site in Hong Kong

We assume a user named Ben in Hong Kong 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 failure-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 workload of the Web site server. After making a tradeoff, Ben sets the t_{user} to be 100 milliseconds.

Based on the strategy selection algorithm proposed in Section 4, the selection procedure is shown in Table 5, where $\{ws\}_{i=1}^{6}$ is a set of ranked target Web services. Values of t_i, f_i are provided by the distributed evaluation framework (see Table 3 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 the distributed evaluation framework. 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

Table 5. 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;

primary replica and secondary replica is not significant), Strategy 9 ($Passive + Time$) is selected.

As shown in Table 3, 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.

Scenario 2: Personal 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 failure-rate f_{user} is set to be 5%.
2) **Response time & resource conservation.** Since Tom's home page is running on a server with restricted resource and narrow network bandwidth, the t_{user} is set to be 500 milliseconds.

After conducting the selection procedure as shown in Table 6, Strategy 7 ($Passive + Active$) 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.

The detailed RTT and failure-rate performance with different replica number of these two scenarios are shown in Fig.3, where Fig. 3(a) shows the RTT performance, and Fig. 3(b) shows the failure-rate performance. Fig.3 (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.3 (b) shows that failure-rate is greatly reduced by invoking parallel replicas in Mainland China, indicating that

Table 6. 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.

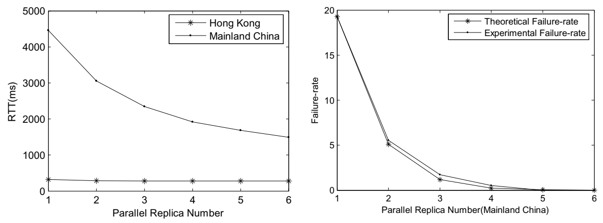

Figure 3. (a)RTT and (b)Failure-rate Performance with Different Replica Number

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.3 (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 by our distributed evaluation framework, 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 (Table 2) 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.

7. Conclusion

This paper proposes a distributed replication strategy evaluation and selection framework for fault tolerant Web services. Based on this framework, we compare various replication strategies by using theoretical formula and experimental results, and propose a strategy selection algorithm based on both objective performance information as well as subjective requirements of users. Motivated by the lack of real-world data for studying performance of Web service as well as various replication strategies, comprehensive real-world experiments on individual Web services and replication strategies are conducted to illustrate the evaluation framework and the selection algorithm proposed in this paper. With the facility of the proposed framework, an accurate evaluation of target Web services and various replication strategies can be acquired through user collaboration, and optimal replication strategies engaging fault tolerance can be effectively obtained.

Currently, this distributed evaluation framework can only work on stateless Web services. More investigations are needed to apply it to stateful Web services. Our future work will also include the tuning of the selection algorithm (i.e., the values of a, b and c, and involvement of more QoS properties) for better performance, the improvement of system feature of our distributed framework for facilitating user test case sharing and contribution, and better use of historical evaluation results for performance prediction purpose.

8. Acknowledgement

The work described in this paper was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4150/07E), and a grant from the Research Committee of The Chinese University of Hong Kong (Project No. CUHK3/06C-SF).

References

[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 (ICWS'07), pp. 679-686, 2007.

[5] 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.

[6] D. Liang, C. Fang, and C. Chen, "FT-SOAP: A Fault Tolerant Web Service," in Tenth Asia-Pacific Software Engineering Conference, Chiang Mai, Thailand, 2003.

[7] 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.

[8] 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.

[9] M. G. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan, "Thema: Byzantine-Fault Tolerant Middleware for Web-Service Applications", in IEEE Symposium on Reliable Distributed Systems, pp. 131 - 140, 2005.

[10] 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.

[11] 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 (ARES'06), pp. 400C407, 2006.

[12] 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.

[13] 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.

[14] 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'07), Utah, USA, pp.498-505, 2007.

[15] 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'03), Trento, Italy, 2003.

[16] 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.

[17] M. Vieira, N. Laranjeiro, and H. 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.

[18] 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.

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

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

[21] M.R. Lyu (ed.), "Handbook of Software Reliability Engineering," McGraw-Hill, New York, 1996.

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

[23] 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.

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

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

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