Optimal Fault Tolerance Strategy Selection for Web Services

Zibin Zheng, The Chinese University of Hong Kong, China

Michael R. Lyu, The Chinese University of Hong Kong, China

ABSTRACT

Service-oriented systems are usually composed by heterogeneous Web services, which are distributed across the Internet and provided by organizations. Building highly reliable service-oriented systems is a challenge due to the highly dynamic nature of Web services. In this paper, the authors apply software fault tolerance techniques for Web services, where the component failures are handled by fault tolerance strategies. In this paper, a distributed fault tolerance strategy evaluation and selection framework is proposed based on versatile fault tolerance techniques. The authors provide a systematic comparison of various fault tolerance strategies by theoretical formulas, as well as real-world experiments. This paper also presents the optimal fault tolerance strategy selection algorithm, which employs both the QoS performance of Web services and the requirements of service users for selecting optimal fault tolerance strategy. A prototype is implemented and real-world experiments are conducted to illustrate the advantages of the evaluation framework. In these experiments, users from six different locations perform evaluation of Web services distributed in six countries, where over 1,000,000 test cases are executed in a collaborative manner to demonstrate the effectiveness of this approach.

Keywords: *Fault Tolerance, QoS, Service-Oriented Systems, Service Selection, Web Services*

1. INTRODUCTION

Web services are self-contained, self-describing, and loosely-coupled computational components designed to support machine-to-machine interaction by programmatic Web method calls, which allow structured data to be exchanged with remote resource. In the environment of service-oriented computing (Zhang et al., 2007), complex service-oriented systems are usually dynamically and automatically composed by distributed Web service components. Since the Web service components are usually provided by different organizations and may easily become unavailable in the unpredictable Internet environment, it is difficult to build highly reliable service-oriented systems employing distributed Web services. However, reliability is a major issue when applying service-oriented systems to critical domains, such as e-commerce and e-government. There is thus an urgent need for practical reliability enhancement techniques for the service-oriented systems.

By tolerating component faults, software fault tolerance is an important approach for building reliable systems and reducing the expensive roll-back operations in the long-running business processes. One approach of

DOI: 10.4018/jwsr.2010100102

software fault tolerance, also known as design diversity, is to employ functionally equivalent yet independently designed program versions for tolerating faults (Lyu, 1995). This used-to-be expensive approach now becomes a viable solution to the fast-growing service-oriented computing arena, since the distributed Web services with overlapping or equivalent functionalities are usually independently developed by different organizations. These alternative Web services can be obtained from the Internet and employed for the construction of diversity-based fault tolerant service-oriented systems. By fault tolerance techniques, long-running business process roll-backs can be reduced since failures of the components can be tolerated by employing alternative candidates (other Web services). Although a number of fault tolerance strategies have been proposed for establishing reliable traditional systems (Lyu, 1995), in the fast-growing field of service computing, systematic and comprehensive studies on software fault tolerance techniques to transactional Web services are still missing.

When applying fault tolerance techniques to the service-oriented systems, several challenges need to be addressed:

- The commonly-used fault tolerance strategies should be identified and their performance needs to be investigated and compared extensively by theoretical analysis and real-world experiments.
- Quality-of-service (QoS) values of the Web services are needed for determining the optimal fault tolerance strategy. However, some nonfunctional performance of the Web services (e.g., response-time and failure-rate) is location-dependent and difficult to obtain.
- Feasible optimal fault tolerance strategy selection approaches are needed since the Internet is highly-dynamic and the performance of Web services are changing frequently. However, the optimal fault tolerance strategy is application dependent subject to the user preference.

In this paper, we present a distributed fault tolerance strategy evaluation and selection framework for Web services, which is designed and implemented as WS-DREAM (Distributed REliability Assessment Mechanism for Web Ser-vice) (Zheng & Lyu, 2008b, a). In WS-DREAM, the QoS performance of Web services can be obtained via user-collaboration and the optimal fault tolerance strategy is determined in such a way to optimize the performance of the service-oriented system with a given set of user requirements. The contributions of the paper are threefold:

- Identify various commonly-used fault tolerance strategies and design a distributed evaluation framework for Web services.
- Propose a dynamic optimal fault tolerance strategy selection algorithm, which can be automatically reconfigured at runtime.
- Implement a working prototype and conduct large-scale real-world experiments. More than 1,000,000 Web service invocations are executed by 6 distributed service users different locations on 8 Web services located in different countries.

Let's consider motivating example that user named Ben plans to build reliable service-oriented application using available fault tolerance strategies. He faces several challenges:

(1) What are the commonly-used fault tolerance strategies?
(2) How to know the performance of the remote Web services?
(3) How to select the optimal fault tolerance strategy based on the user preference?
(4) How to dynamically reconfigure the fault tolerance strategy when the performance of remote Web services is changed?

To address these challenges, this paper first identifies the commonly-used fault tolerance strategies with systematic mathematical formulas in Section 2. Then, a user-collaborated evaluation framework is proposed for obtaining QoS values of Web services efficiently in Section 3. A dynamic optimal fault tolerance selection algorithm with user-requirement models is subsequently proposed in Section 4. To illustrate the evaluation framework and to study the performance of various fault tolerance strategies, a prototype is designed and implemented in Section 5, and detailed experimental results are presented in Section 6. Finally, related-work is introduced in Section 7 and conclusion is provided in Section 8.

2. FAULT TOLERANCE STRATEGIES

Due to the compositional nature of Web services, reliability of the service-oriented systems becomes a formidable challenge. Software fault tolerance by design diversity (Lyu, 1995) is a feasible approach for building reliable service-oriented systems. The major fault tolerance strategies can be divided into *time-redundancy* and *space-redundancy* (Leu et al., 1990; Salatge & Fabre, 2007), where *time-redundancy* uses extra computation/communication time to tolerate faults, and *space-redundancy* employs extra resources, such as hardware or software, to mask faults.

Space-redundancy includes *active-replication* and *passive-replication*. *Active-replication* is performed by invoking all service candidates at the same time to process the same request, and employing the first returned response as the final outcome (Chan et al., 2007). *Passive-replication* invokes a primary candidate to process the request first. Backup candidates will be invoked only when the primary candidate fails. The *time-redundancy, active-replication*, and *passive-replication* are named *Time*, *Active*, and *Passive*, respectively, in this paper.

As shown in Table 1, combining the basic strategies (*Time*, *Active*, and *Passive*) can produce more feasible fault tolerance strategies. As shown in Figure 1, a strategy named *A(B)* means that Strategy *B* is employed at the lower level and Strategy *A* at the higher level. As discussed in the work (Leu et al., 1990), we assume the remote Web services are failed in a fixed rate, and the Web service candidates are independent with each other.

In the following, we provide detailed introduction and the mathematical formulas for calculating the *failure-rate* and *response-time* of these fault tolerance strategies. Failure-rate (*f*) is the probability that a service request is incorrectly responded within the maximum expected time, and response-time (*t*) is the time duration between sending a request and receiving a response of a service user.

1. **Active**: All the *n* Web service candidates are invoked in parallel and the first successfully returned response will be selected as final result. The formulas for calculating the *failure-rate* (*f*) and *response-time* (*t*) of this strategy are defined as:

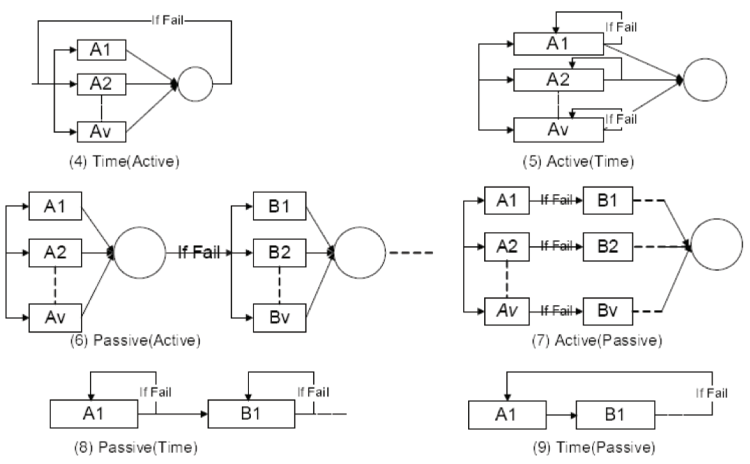
$$f = \prod_{i=1}^{n} f_i; \tag{1}$$

where n is the number of candidates, T_c is a set of Round-Trip Times (RTT) of the successful invocations, and T_f is a set of RTT of the unsuccessful invocations. When all the parallel

Table 1. Combination of the basic fault tolerance strategies

	Active	Time	Passive
Active	1. Active	5. Active(Time)	7. Active(Passive)
Time	4. Time(Active)	2. Time	9. Time(Passive)
Passive	6. Passive(Active)	8. Passive(Time)	3. Passive

Figure 1. Fault tolerance strategies



invocations are failed ($|T_c| = 0$), the maximal RTT value is employed as the response-time.

2. **Time**: The original Web service will be retried for a certain times if it fails. The formulas for calculating *failure-rate* and *response-time* are defined as:

$$f = (f_1)^m; \tag{2}$$

where m is the retried times, f_1 is the *failure-rate* of the remote Web service, and t_i is the *response-time* of the i^{th} Web service invocation.

3. **Passive:** Another backup Web service will be tried sequentially if the primary Web service fails. The formulas for calculating *failure-rate* and *response-time* are defined as:

$$f = \prod_{i=1}^{m} f_i; \tag{3}$$

where m is the recovery times, t_i is the invocation response-time of the i^{th} Web service, and f_i is the *failure-rate* of the i^{th} Web service.

4. **Time(Active)**: As shown in Figure 1 (4), the first *v* best performing candidates are invoked in parallel. The whole parallel block will be retried if all parallel invocations fail. The formulas for *failure-rate* and *response-time* are defined as:

$$f = (\prod_{i=1}^{v} f_i)^m; \tag{4}$$

where *v* is the parallel invocation number, m is the retry times, t_i' is the response-time of the i^{th} time of invoking the whole parallel block. t_i' can be calculated by $t_i' = \begin{cases} \min\{T_c^i\} : |T_c^i| > 0 \\ \max\{T_f^i\} : |T_c^i| = 0 \end{cases}$, where $T_c^i \cup T_f^i = \{t_i\}_{i=1}^{v}$.

5. **Active(Time)**: As shown in Figure 1 (5), the *v* best performing candidates are invoked in parallel. The candidates will be retried individually if they fail. The formulas are defined as:

$$f = \prod_{i=1}^{v} (f_i)^m; \tag{5}$$

where *m* is the retried times, $T_c \cup T_f = \{t'_i\}_{i=1}^{v}$, and $t'_i = \sum_{j=1}^{m} t_{ij}(f_i)^{j-1}$.

6. **Passive(Active)**: As shown in Figure 1 (6), another set of backup candidates will be tried if all of the primary *v* candidates fail. The formulas are defined as:

$$f = \prod_{i=1}^{m}\prod_{j=1}^{v} f_{ij}; \tag{6}$$

where *m* is the recovery times and $t'_i = \begin{cases} \min\{T_c^i\} : |T_c^i| > 0 \\ \max\{T_f^i\} : |T_c^i| = 0 \end{cases}$.

7. **Active(Passive)**: As shown in Figure 1 (7), the best performing *v* candidates are invoked in parallel. Each individual candidate in the primary v candidates will try another backup candidate sequentially if it fails. The formulas are defined as:

$$f = \prod_{j=1}^{v}\prod_{i=1}^{m} f_{ij}; \tag{7}$$

where *m* is the recovery times, $T_c \cup T_f = \{t'_i\}_{i=1}^{v}$, and $t'_i = \sum_{j=1}^{m}(t_{ij}\prod_{k=1}^{j-1} f_{ik})$.

8. **Passive(Time)**: As shown in Figure 1 (8), the primary candidate will retry itself for *m* times before trying other backup candidates. Only a set of *u* best performing candidates are employed as backup candidates among all the *n* replicas. The formulas are defined as:

$$f = \prod_{i=1}^{u}(f_i)^m; \tag{8}$$

where $t'_i = \sum_{j=1}^{m} t_i f_i^{j-1}$.

9. **Time(Passive)**: As shown in Figure 1 (9), a replica will try another backup candidate first if it fails. After trying *u* candidate without success, all the *u* candidates will be retried sequentially. The formulas are as:

$$f = (\prod_{i=1}^{u} f_i)^m; \tag{9}$$

where *m* is the retried times and $t'_i = \sum_{j=1}^{u}(t_j\prod_{k=1}^{j-1} f_k)$.

These fault tolerance strategies can be divided into three types:

- **Parallel (Strategy 1)**: All Web service candidates are invoked at the same time. Parallel type strategies can be employed to obtain good response-time performance, although it consumes a considerable amount of computing and networking resources.
- **Sequential (Strategies 2, 3, 8 and 9)**: The Web service candidates 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)**: A subset of the Web service candidates are invoked in parallel. Hybrid strategies consume fewer resources than parallel strategies and have better response time performance than the sequential strategy.

3. DISTRIBUTED EVALUATION FRAMEWORK

For calculating the response-time and failure-rate of various fault tolerance strategies, the QoS performance (response-time and failure-rate) of target Web services are needed. Without accurate QoS values of the Web services, it is really difficult to calculate the performance of different fault tolerance strategies and make optimal fault tolerance strategy selection.

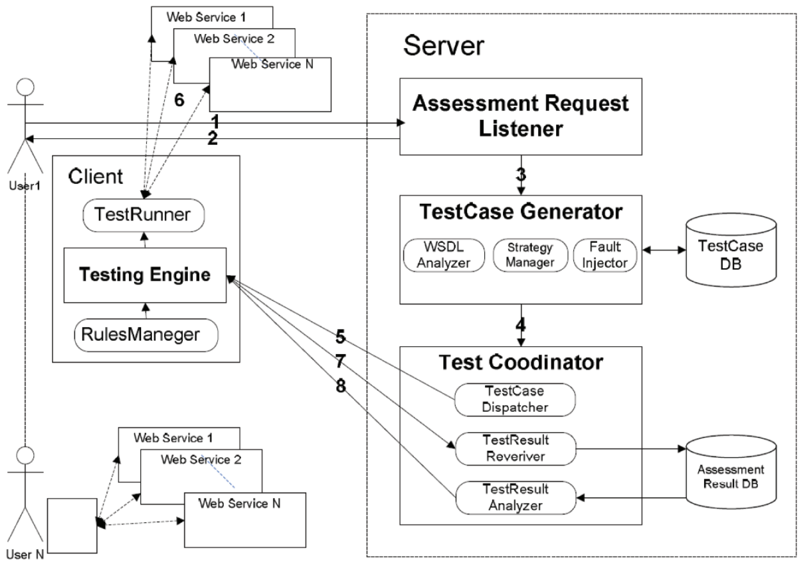
Since the service providers may not deliver the QoS they declared and some QoS properties (e.g., response-time and failure-rate) are highly related to the locations and network conditions of service users, Web service evaluation can be performed at the client-side to obtain more accurate QoS performance (Wu et al., 2007; Zeng et al., 2004). However, several challenges have to be solved when conducting Web service evaluation at the client-side: (1) It is difficult for the service users to make professional evaluation on the Web services themselves, since the service users are usually not experts on the Web service evaluation, which includes WSDL file analysis, test case generation, evaluation mechanism implementation, test result interpretation and so on; (2) It is time-consuming and resource-consuming for the service users to conduct a long-duration evaluation on many Web service candidates themselves; and (3) The common time-to-market constraints limit an in-depth and accurate evaluation of the target Web services.

To address these challenges, we propose a distributed evaluation framework for Web services, together with its prototyping system WS-DREAM (Zheng & Lyu, 2008b, a), as shown in Figure 2. This framework employs the concept of *user-collaboration*, which has contributed to the recent success of BitTorrent (Bram, 2003) and Wikipedia (www.wikipedia.org). In this framework, users in different geographic locations share their observed QoS performance of Web services by contributing them to a centralized server. Historical evaluation results saved in a data center are available for other service users. In this way, QoS performance of Web services becomes easy to be obtained for the service users.

As shown in Figure 2, the proposed distributed evaluation framework includes a centralized server with a number of distributed clients. The overall procedures can be explained as follows.

1. **Registration**: Service users submit evaluation requests with related information, such as the target Web service addresses, to the WS-DREAM server.
2. **Client-side application loading**: A client-side evaluation application is loaded to the service user's computer.

Figure 2. 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).
4. **Test coordination**: Test tasks are scheduled based on the number of current users and test cases.
5. **Test cases retrieval**: The distributed client-side evaluation applications get test cases from the centralized server.
6. **Test cases execution**: The distributed client-side applications execute the test cases to conduct testing on the target Web services.
7. **Test result collection**: The distributed client-side applications send back the test results to the server, and repeat the steps 5, 6 and 7 to retrieval and execute more test cases.
8. **Test result analysis**: The *TestResult Analyzer* in the server-side is engaged to process the collected data and send back the detailed evaluation results to the service user.

The advantages of this user-collaborated evaluation framework include:

1. This framework can be implemented and launched by a trust-worthy third-party to help service users conduct accurate and efficient Web service evaluation in an easy way, without requiring service users to have professional knowledge on evaluation design, test case generation, test result interpretation, and so on.
2. The historical evaluation results on the same Web services can be reused, making the evaluation more efficient and save resource for both the service users and service providers.
3. The overall evaluation results from different service users can be used as useful information for optimal Web service selection. The assumption is that the Web service, which has good historical performance observed by most of the service users, has higher probability to provide good service to the new service users.

By this framework, evaluation on Web services becomes accurate, efficient and effective.

4. FAULT TOLERANCE SELECTION

In this section, we propose an algorithm for dynamic optimal fault tolerance strategy selection.

4.1 Notations and Utility Function

The notations used in the remainder of this paper are defined in Table 2, where *t* is an abstract task and $\{ws\}_{i=1}^{n}$ is a set of Web service candidates for *t*; q^1, q^2, and q^3 are three QoS properties which present *response-time*, *failure-rate*, and *parallel-invocation-number*, respectively. All of these three QoS properties are negative, where smaller value stands for better quality. Q^1, Q^2, and Q^3 are the user requirements on these three QoS properties, respectively. The values of Q^1, Q^2, and Q^3 are set by the service users. For example, $Q^1 = 1000$ms means that the task *t* must be finished within one second. As a result, the Web service candidates with response-time (q^1) larger than 1 second will not be selected. Q^3 presents the user-requirement on the *parallel-invocations-number*. For example, the parallel Web service invocation can be disabled by setting Q^3 to be 1 when the Web service invocations are payment-oriented.

To quantize performance of different Web service candidates, a utility function is defined as:

$$u = \sum_{j=1}^{m} w_j \times \frac{q^j}{Q^j} \tag{10}$$

where w_j is the user-defined weights for setting the priorities of QoS properties, *m* is the number of QoS properties, and a smaller value of utility value *u* means better performance. The value of *m* is three in this paper, since we consider three quality properties in our selection algorithm. More QoS properties can be added to our algorithm easily in the future without fundamental changes.

The design consideration of the utility function is that response-time performance (q^1) of a particular Web service is related to the corresponding user requirement (Q^1). For example, 100 ms is a large latency for the latency-sensitive applications, while it is neglectable for the non-latency-sensitive applications. By using $\frac{q^1}{Q^1}$, we have a more personalized representation of the response-time performance of Web services. Failure-rate (q^2) and parallel invocation number (q^3) are similarly considered.

4.2 Selection Algorithm

The target of the selection algorithm is to find out the optimal fault tolerance strategy for an abstract task *t* based on the objective QoS performance of Web service candidates as well as subjective requirements of service users. To determine the optimal fault tolerance strategy, we first rank the Web service candidates based on their QoS performance using the utility function. Then, the optimal parallel invocation number is determined by solving an optimization problem. Finally, the optimal fault tolerance strategy is determined.

4.2.1 Web Service Candidate Ranking

The Web service candidates $\{ws\}_{i=1}^{n}$ for the task *t* are ranked by their utility values, which can be calculated by $u_i = \sum_{j=1}^{3} w_j \times \frac{q_i^j}{Q^j}$, where u_i is the utility value of the i^{th} candidate, q_i^j is the j^{th} quality property of the candidate, and $q_i^3 = 1$ since there are no parallel invocations when ranking the candidates. After the ranking, $\{\tilde{ws}\}_{i=1}^{n}$ is a set of ranked Web service candidates, where $\tilde{ws}_1$ is the best performing Web service with the smallest utility value.

4.2.2 Parallel Invocation Number Determination

By finding out the optimal parallel invocation number *v*, the optimal fault tolerance 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
Minimize:

$$\sum_{i=1}^{n} \tilde{u}_i x_i \tag{11}$$

Subject to:

$$\sum_{i=1}^{n} \tilde{q}_i^k x_i \le Q^k (k = 1,2,3) \tag{12}$$

$$\sum_{i=1}^{n} x_i = 1 \tag{13}$$

$$x_i \in \{0,1\} \tag{14}$$

In Problem 1, Equation 11 is the objective function, where $\tilde{u}_i$ is the utility value of invoking the first *i* best performing Web service candidates in parallel ($\{\tilde{ws}\}_{j=1}^{i}$). There are totally *n* solutions to this problem, which are *i* = *1*, ..., *i* = *n*. Equation 12 is the constraint function which makes sure the QoS performance of the solution meets the requirements of service users. In Equation 12, $\tilde{q}_i^1$ and $\tilde{q}_i^2$ is the overall response-time performance and overall failure-rate performance of invoking the first *i* Web service candidates in parallel, which can be calculated by employing the Equation 1 in Section 2, $\tilde{q}_i^3$ is the parallel invocation number ($\tilde{q}_i^3 = i$) and Q^k is the user requirements. Equation 13 and Equation 14 are to make sure that only one solution will be selected for the task, where x_i is set to 1 if the first *i* service candidates are invoked in parallel and 0 otherwise. Algorithm 1 is designed to solve Problem 1. For each potential solution, we first use Equation 1 in Section 2 to calculate the overall QoS values. Then the solutions which cannot meet the user-requirements are excluded. After that, the utility values of the remanding solutions are calculated by using the utility function in Equation 10. Finally, the solution with smallest utility value u_x will be selected as the final solution for Problem 1 by setting *v = x*.

4.2.3 Sequential Fault Tolerance Strategy Determination

If *v = 1*, sequential strategies (Strategies 2, 3, 8 and 9) will be selected. To determine the optimal sequential strategy, the poor performing candidates, which will greatly influence the response-time performance of sequential strategies, will be excluded. A set of good performing candidates *W* will be selected out by using:

$$W = \{\tilde{ws}_i \mid u_i \le a, 1 \le i \le n\} \tag{15}$$

where *a* is the threshold on candidate performance and u_i is the utility value of the candidate $\tilde{ws}_i$. If there is no candidate meet the performance threshold ($|W| = 0$), the service user needs to provide more candidates or devalue the performance threshold *a*. When $|W| = 1$, strategy 2 (*Time*) is employed, since all other strategies need redundant candidates. When $|W| = n$, strategy 3 (*Passive*) is employed. Otherwise, strategy 8 (*Passive(Time)*) or strategy 9 (*Time(Passive)*) will be employed.

$p_1 = u_2 - u_1$, which is the performance degradation between $\tilde{ws}_1$ and $\tilde{ws}_2$, is employed to find out the optimal strategy between strategy 8 and strategy 9. When the performance degradation is large ($p_1 \ge b$, where *b* is the threshold of performance degradation), retrying the original Web service ($\tilde{ws}_1$) first is more likely to obtain better performance (strategy 8) than invoking the backup candidate (strategy 9).

4.2.4 Hybrid Fault Tolerance Strategy Determination

If *1 < v < n*, hybrid fault tolerance strategies will be selected. p_2 represents the performance

Figure 3. Algorithm 1- parallel invocation number calculation

```
Data: Ranked service candidates $\{\tilde{ws}\}_{i=1}^{n}$, user-requirements
      $Q^k (1 \le k \le 3)$
Result: Optimal parallel invocation number $v$.
for $(i = 1; i \le n; i{+}{+})$ do
    if $\forall k (q_i^k \le Q^k)$ then
        $u_i = utility(q_i)$;
    end
end
if no solution available then
    Throw exception;
end
$u_x = \min\{u_i\}$;
$v = x$ ;
```

difference between the primary *v* candidates and the secondary *v* candidates. p_2 can be calculated by

$$p_2 = \frac{1}{v}\sum_{i=1}^{v}(u_{i+v} - u_i) \tag{16}$$

where *v* is the parallel invocation number. If the performance difference is large ($p_2 \geq b$), retrying the original parallel candidates first is more likely to obtain better performance (Strategies 4 and 5) than invoking the secondary *v* backup candidates (Strategies 6 and 7).

p_3 is the failure frequency of the first *v* candidates, which can be calculated by

$$p_3 = \frac{1}{v}\sum_{i=1}^{v} q_i^2 \tag{17}$$

where q^2 is the failure-rate of the i^{th} candidate. In the erroneous environment ($p_3 \geq c$), strategy 5 and strategy 7 will be selected, since strategy 4 and strategy 6 need to wait for all responses of the parallel candidates before retrying/recovering, which will greatly degrade the response-time performance.

4.2.5 Parallel Fault Tolerance Strategy Determination

If *v* = *n*, strategy 1 (Active) will be selected. Strategy Active invokes all the candidates in parallel. Figure 4 shows the whole fault tolerance strategy selection procedures discussed above. First, the sequential, hybrid, and parallel types are determined based on the parallel invocation number *v*. Then, the detailed strategy will be determined based on the values of W , p_1 , p_2 , and p_3 .

4.3 Dynamic Fault Tolerance Strategy Reconfiguration

The performance of Web services may change dramatically or the services may even become unavailable in the unpredictable Internet environment. Moreover, the user-requirements of the optimal fault tolerance strategy may change from time to time. To enable dynamic reconfiguration of the optimal fault tolerance strategy, we propose a dynamic reconfiguration approach as shown in Figure 5. The reconfiguration procedures are as follows: (1) the initial optimal fault tolerance strategy is calculated by employing the selection algorithm in Section 4.2. (2) The service-oriented application invokes the remote Web services with the selected fault tolerance strategy, and records the QoS performance (e.g., response-time, failure-rate) of the invoked Web services. (3) If the performance of the fault tolerance strategy is unacceptable or the renewal time is come, the service-oriented application will update the user requirements and employed the updated information for recalculating the new optimal fault tolerance strategy.

By the above reconfiguration approach, service users can handle the frequently context information changes by recalculating optimal fault tolerance strategy using updated QoS performance of the target Web services as well as updated user-requirements. The recalculation frequency is application-dependent and controlled by the service users based on their preference, which is out of the scope of this paper.

5. IMPLEMENTATION

To illustrate the distributed evaluation framework and the fault tolerance strategy selection algorithm, a prototype (www.wsdream.net) is implemented. The client-side of WS-DREAM is realized as a Java Applet, which can be loaded and run automatically by the Internet browsers of the service users. The server-side of WS-DREAM is implemented as several components, including a *HTML Web site* (www.wsdream.net), a *TestCaseGenerator* (Java application), a *TestCoodinator* (Java Servlet), and a data center for recording evaluation results and test-cases (MySQL).

To study the performance of the nine fault tolerance strategies presented in Section 2 and

Figure 4. Fault tolerance strategy selection procedure

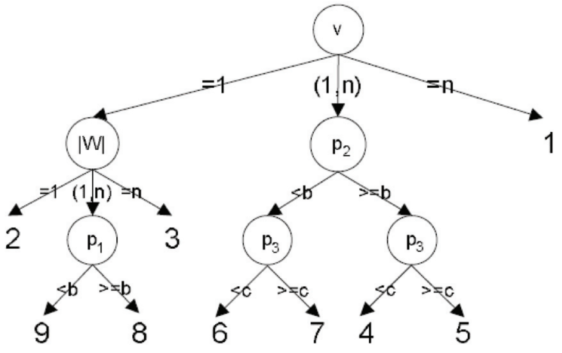
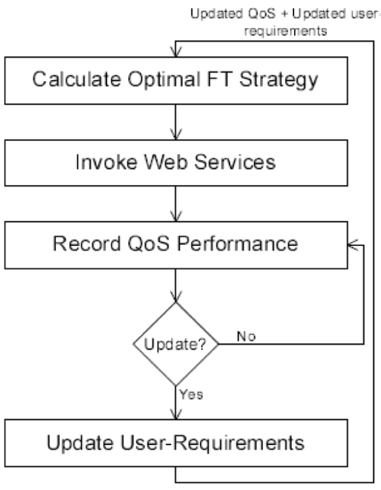


Figure 5. Optimal fault tolerance strategy reconfiguration procedures



the strategy selection algorithm proposed in Section 4, a large-scale real-world experiment is conducted. In the experiment, a set of 8 real-world Web services are employed. As shown in Table 3, these Web services include 6 functionally equivalent Amazon Web services located in 6 countries, a *GlobalWeather* Web service located in US and a *IPService* located in US. More than 1,000,000 Web service invocations are executed by these service users and the detailed experimental results will be reported in Section 6.

6. EXPERIMENTS

6.1 Individual Web Services

Figure 6 and Figure 7 show the experiment results from the six distributed service users (*US, HK, SG, CN, TW and AU*) on the six Amazon Web services (*a1–a6*). In Figure 6, under the Location column, *U* stands for user-locations and WS presents the Web services. *cn, tw, au, sg, hk, us* present the six user-locations conducting the evaluation. As shown in Table 3, *a1, a2, a3,*

Table 2. Locations of the web services

WS Names	Providers	Locations
ECommerceService	Amazon	US
ECommerceService	Amazon	Japan
ECommerceService	Amazon	Germany
ECommerceService	Amazon	Canada
ECommerceService	Amazon	France
ECommerceService	Amazon	UK
GlobalWeather	WebserviceX.net	US
GeoIP	WebserviceX.net	US

a4, a5 and a6 stand for the six Amazon Web Services, which are located in US, Japan, Germany, Canada, France, and UK, respectively. The Cases column shows the failure-rate (*F%*), which is the number of failed invocations (*Fail*) divided by the number of all invocations (*All*). The RTT column shows the average (*Avg*) and standard deviation (*Std*) of the response-time/Round-Trip-Time (RTT) performance. The *ProT* column shows the average (*Avg*) and standard deviation (*Std*) of the process-time (*ProT*), which is the time consumed by the Web service server for processing the request (time duration between the Web service sever receives and request and sends out the corresponding response).

The experimental results in Figure 6 and Figure 7 show:

- As shown in Figure 7 (a), the response-time (RTT) performance of the target Web services change dramatically from user to user. For example, invoking a-us only needs 74 milliseconds on average from the user location of us, while it requires 4184 milliseconds on average from the user-location of *cn*.
- As indicated by the *Std* values in Figure 6, even in the same location, the RTT performance vary drastically from time to time. For example, in the user-location of *cn*, the RTT values of invoking a1 vary from 562 milliseconds to 9906 milliseconds in our experiment. The unstable RTT performance degrades service quality and makes the latency-sensitive applications easy to fail.
- The ProT values in Figure 6 indicate that the response-times of the Amazon Web services are mainly consist of network-latency rather than server processing-time. Since the average process-times of all the six Amazon Web services are all less than 50 milliseconds, which is very small compared with the RTT values shown in Figure 6.
- Users under poor network conditions are more likely to suffer from unreliable service, since unstable RT T performance degrades service quality and even leads to timeout failures. Figure 7 (b), which illustrates the failure-rates of the Web services, shows that the service user with the worst RTT performance (*cn*) has the highest failure rate, while the service user with the best RTT performance (*us*) has the lowest failure-rate.

Figure 8 and Figure 9 show the experimental results of the *GlobalWeather* and *GeoIP* Web services. The same as Figure 6, Figure 8 and Figure 9 shows that performance of the Web services is quite different from location to location. Comparing with the *GlobalWeahter* and *GeoIP* Web services, the *ECommerceService* Web services provide better failure-rate performance. This may related to the fact that the ECommerceService Web services are pro-

Figure 6. Experimental results of the Amazon web services

Location		Cases			RTT (ms)		ProT(ms)	
U	**WS**	**All**	**Fail**	**F%**	**Avg**	**Std**	**Avg**	**Std**
cn	a1	484	109	22.52	4184	2348	42	19
	a2	482	128	26.55	3892	2515	46	27
	a3	487	114	23.40	3666	2604	42	17
	a4	458	111	24.23	4074	2539	45	21
	a5	498	96	19.27	3654	2514	43	18
	a6	493	100	20.28	3985	2586	45	20
au	a1	1140	0	0	705	210	42	16
	a2	1143	0	0	577	161	44	29
	a3	1068	0	0	933	272	45	115
	a4	1113	0	0	697	177	42	17
	a5	1090	0	0	924	214	44	23
	a6	1172	3	0.25	921	235	44	24
hk	a1	21002	81	0.38	448	304	42	21
	a2	20944	11	0.05	388	321	44	33
	a3	21130	729	3.45	573	346	43	18
	a4	21255	125	0.58	440	286	43	20
	a5	21091	743	3.52	575	349	44	20
	a6	20830	807	3.87	570	348	43	20
tw	a1	2470	0	0	902	294	44	22
	a2	2877	1	0.03	791	315	44	40
	a3	2218	0	0	1155	355	44	17
	a4	2612	5	0.19	899	300	43	20
	a5	2339	0	0	1144	370	44	21
	a6	2647	1	0.03	1150	363	45	23
sg	a1	1895	0	0	561	353	44	19
	a2	1120	0	0	503	322	43	33
	a3	1511	0	0	638	409	43	20
	a4	1643	0	0	509	240	44	15
	a5	1635	0	0	638	310	44	24
	a6	1615	0	0	650	308	43	16
us	a1	3725	0	0	74	135	42	18
	a2	3578	0	0	317	224	43	33
	a3	3766	0	0	298	271	43	16
	a4	3591	0	0	239	260	43	19
	a5	3933	0	0	433	222	44	30
	a6	3614	0	0	293	260	44	19

vided by big company (*Amazon*) and are built for e-business purpose.

6.2 Fault Tolerance Strategies

Real-world experiments are conducted to study the performance of different fault tolerance strategies. Figure 10 shows the experimental results of various fault tolerance strategies employing the six functionally equivalent Amazon Web services (*a1,...,a6*) as redundant service candidates. Figure 10 shows that Strategy 1 (Active) has the best RTT performance. This is reasonable, since Strategy 1 invokes all the six candidates at the same time and employs the first returned 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 fault tolerance strategies, as shown in Figure 6. RTT performance of sequential type strategies (Strategies 2, 3, 8 and 9) is worse than other strategies, because they invoke candidates one by one.

Figure 7. Response-time and failure-rate performance

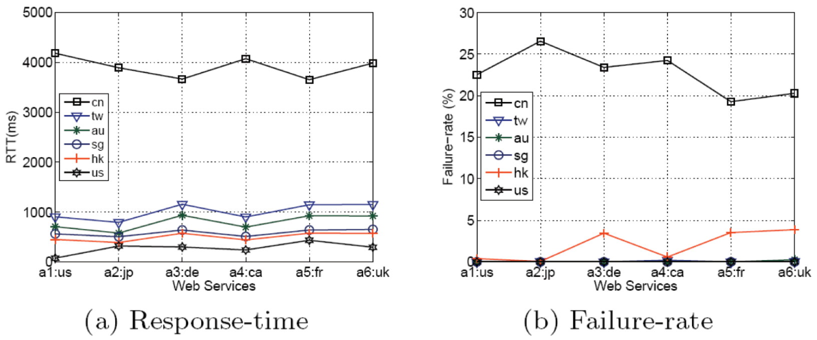


Figure 8. Experimental results of the GlobalWeather web services

Location		Cases			RTT (ms)			
U	WS	All	Fail	F%	Avg	Std	Min	Max
cn	GW	409	337	82.39	6643	2003	2094	9969
tw	GW	1981	35	1.76	1105	1401	343	9844
au	GW	1104	5	0.45	503	544	234	9375
sg	GW	1363	0	0	1403	1544	265	9937
hk	GW	21148	1426	6.74	1563	1560	406	9999
us	GW	3837	0	0	1290	1346	125	9828

Figure 9. Experimental results of the GeoIP web services

Location		Cases			RTT (ms)			
U	WS	All	Fail	F%	Avg	Std	Min	Max
cn	GIP	540	32	5.92	2125	1927	531	9781
tw	GIP	2822	60	2.12	732	1270	265	9875
au	GIP	1125	0	0	355	609	234	9360
sg	GIP	1312	0	0	571	878	265	9594
hk	GIP	21007	1263	6.01	849	1582	203	9999
us	GIP	3621	0	0	675	1348	125	9938

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 but worse than sequential type strategies. All the 15 failures shown in Figure 10 are due to timeout of all the six Web service candidates, which may be caused by client-side network problems.

6.3 Optimal FT Strategy Selection Scenarios

6.3.1 Scenario 1: Commercial Application

In Section 1, we present a scenario that a service user named Ben plans to build a reliable service-oriented application and faces several challenges. More specifically, we assume that Ben's service-oriented application will be

Figure 10. QoS performance of the fault tolerance strategies

Type	Cases			RTT(ms)			
	All	Fail	F%	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

deployed in Hong Kong and the Amazon Web services will be invoked for book displaying and selling in this commercial Web application. The followings are the performance requirements of Ben:

1) Response-time. Ben sets the requirement on response-time Q^1 as 500 ms, since too large response latency will lead to loss of business. .
2) Failure-rate. Since the application is commercial, the failure-rate requirement Q^2 is set to be 0.5% by Ben.
3) Parallel invocation number. Invoking too many parallel candidates will consume significant computing and networking resources. Therefore, Ben sets Q^3 to be 3.

Employing the optimal fault tolerance strategy selection algorithm proposed in Section 4.2 and the QoS performance of the Amazon Web services shown in Figure 6, the selection procedures are shown in Figure 11. In Figure 11, for ease of discussion, we set $w_1 = w_2 = w_3 = 1/3$, which are the weights for different quality properties. The values of the thresholds *a*, *b*, and *c* are set as *a = 2; b = 1; c = 1%* empirically. Q^1, Q^2 and Q^3 are set based on the user-requirements. $\{ws_i\}_{i=1}^6$ is a set of Web service candidates. The response-time (q^1) and failure-rate (q^2) of these candidates are shown in Figure 6. After calculating the utility values of these candidates using Equation 10, the candidates are ranked as $\{\tilde{ws}_i\}_{i=1}^6$. For determining the optimal parallel invocation number (*v*), Algorithm 1 is employed. The utility values of invoking different number of parallel candidates are shown in $\{\tilde{u}_i\}_{i=1}^6$. The smallest utility value is selected out and *v* is set to be 1 accordingly.

Since *v = 1*, sequential type strategy will be selected (Strategy 2, 3, 8, and 9). Because $|W| = 3$ (only the top three best performing candidates are selected), and $p_1 < 1$ (the difference between the primary candidate and secondary candidate is not significant), Strategy 9 (*Time(Passive)*) is selected (backup candidates will be invoked if the primary candidate fails, and all the three candidate will be retried if all of them fail).

As shown in Figure 6, from the location of Hong Kong, network condition is good and the failure-rate is low. The response-time improvement of invoking candidates in parallel is limited. Therefore, sequential strategies for this scenario are reasonable.

6.3.1 Scenario 2: Noncommercial Web Page

In scenario 2, we assume another service user named Tom in the location of *cn* also plans to employ the Amazon Web services to provide

Figure 11. Selection procedures of scenario 1

$$
\begin{array}{l}
a = 2; b = 1; c = 1\%; w_1 = w_2 = w_3 = 1/3; \\
n = 6; Q^1 = 500ms; Q^2 = 0.5\%; Q^3 = 3; \\
\{ws_i\}_{i=1}^{6} = \{\text{a-us, a-jp, a-de, a-ca, a-fr, a-uk}\}; \\
\{q_i^1\}_{i=1}^{6} = \{448, 388, 573, 440, 575, 570\}; \\
\{q_i^2\}_{i=1}^{6} = \{0.38\%, 0.05\%, 3.45\%, 0.58\%, 3.52\%, 3.87\%\}; \\
\{q_i^3\}_{i=1}^{6} = \{1, 1, 1, 1, 1, 1\}; \\
\{u_i\}_{i=1}^{6} = \{0.66, 0.4, 2.79, 0.79, 2.84, 3.07\}; \\
\{\tilde{ws}_i\}_{i=1}^{6} = \{\text{a-jp, a-us, a-ca, a-de, a-fr, a-uk}\}; \\
\{\tilde{u}_i\}_{i=1}^{6} = \{0.4, 0.44, 0.53, 0.63, 0.74, 0.85\}; \\
v = 1; \\
W = \{ws_i | u_i =\leq 2\} = \{\text{a-jp, a-us, a-ca}\}; \\
|W| = 3; \\
p_1 = u_2 - u_1 = 0.66 - 0.4 = 0.26; \\
v = 1 \& 1 < |W| < 6 \& p_1 < b \Rightarrow \text{Strategy 9};
\end{array}
$$

book information query service in his personal home page. The performance requirements of Tom are as follows:

1) Response-time. Since the Web page of Tom is noncommercial, the response-time requirement Q^1 is set to be 3000 milliseconds.
2) Failure-rate. Since the Web page of Tom is not for critical purposes, the failure-rate Q^2 is set to be 5%.
3) Parallel invocation number. Q^3 is set to be 3.

After conducting the selection procedures shown in Figure 12, Strategy 7 (*Active(Passive)*) with three parallel invocations is selected as the optimal strategy for Tom. In this scenario, the failure-rates of individual Web services are high. Hybrid strategy with suitable number of parallel invocations can be employed to improve the failure-rate performance as well as response-time performance. Our algorithm provides suitable fault tolerance strategy selection result for this scenario.

7. RELATED WORK AND DISCUSSION

A number of fault tolerance strategies for Web services have been proposed in the recent literature (Salatge & Fabre, 2007; Chan et al., 2007; Foster et al., 2003; Moritsu et al., 2006; Vieira et al., 2007). The major approaches can be divided into sequential strategies and parallel strategies. Sequential strategies invoke a primary service to process the request. The backup services are invoked only when the primary service fails. Sequential strategies have been employed in FT-SOAP (Fang et al., 2007) and FT-CORBA (Sheu et al., 1997). Parallel strategies invoke all the candidates at the same time, which have been employed in FTWeb (Santos et al., 2005), Thema (Merideth et al., 2005) and WS-Replication (Salas et al., 2006). In this paper, we provide systematic introduction on the commonly-used fault tolerance strategies. Moreover, we present the hybrid fault tolerance strategies, which are the combination of the basic strategies.

A great deal of research effects have been performed in the area of Web service evalua-

Figure 12. Selection procedures of scenario 2

$$
\begin{array}{l}
a = 2; b = 1; c = 1\%; w_1 = w_2 = w_3 = 1/3; \\
n = 6; Q^1 = 3000ms; Q^2 = 5\%; Q^3 = 3; \\
\{ws_i\}_{i=1}^{6} = \{\text{a-us, a-jp, a-de, a-ca, a-fr, a-uk}\}; \\
\{q_i^1\}_{i=1}^{6} = \{4184, 3892, 3666, 4074, 3654, 3985\}; \\
\{q_i^2\}_{i=1}^{6} = \{22.52\%, 26.55\%, 23.4\%, 24.23\%, 19.27\%, 20.28\%\}; \\
\{q_i^3\}_{i=1}^{6} = \{1, 1, 1, 1, 1, 1\}; \\
\{u_i\}_{i=1}^{6} = \{2.08, 2.31, 2.08, 2.18, 1.8, 1.91\}; \\
\{\tilde{ws}_i\}_{i=1}^{6} = \{\text{a-fr, a-uk, a-us, a-de, a-ca, a-jp}\}; \\
\{\tilde{u}_i\}_{i=1}^{6} = \{1.8, 0.82, 0.65, 0.67, 0.74, 0.83\}; \\
v = 3; \\
p_2 = \frac{1}{3} \times \sum_{i=1}^{3} u_{i+v} - u_i = 0.26; \\
p_3 = \frac{1}{3} \times \sum_{i=1}^{3} q_i^2 = 20.69\%; \\
1 < v < 6 \& p_2 < b \& p_3 \geq c \Rightarrow \text{Strategy 7.}
\end{array}
$$

tion. Various approaches, such as Qos-aware middleware (Zeng et al., 2004), reputation conceptual model (Maximilien & Singh, 2002), and Bayesian network based assessment model (Wu et al, 2007), have been proposed. Some recent work (Rosario et al., 2008; Zeng et al., 2004; Wu et al., 2007; Deora et al., 2003) also take subjective information, such as provider reputation, user rating and user requirement, into consideration to make evaluation more accurate. For presenting the non-functional characteristics of the Web services, QoS models of Web services have been discussed in a number of recent literature (Ardagna & Pernici, 2007; Jaeger et al., 2004; O'Sullivan et al., 2002; Ouzzani & Bouguettaya, 2004; Thio & Karunasekera, 2005). The QoS data of Web services can be measured from either the service user's perspective (e.g., response-time, success-rate, etc.) or the service provider's perspective (e.g., price, availability, etc.). In this paper, we consider the most representative QoS properties (response-time, failure-rate, and parallel-invocation-number). QoS measurement of Web services has been used in the Service Level Agreement (SLA) (Ludwig et al., 2003), such as IBMs WSLA framework (Keller & Ludwig, 2002) and the work from HP (Sahai et al., 2002). In SLA, the QoS data are mainly for the service providers to maintain a certain level of service to their clients and the QoS data are not available to others. In this paper, we introduce the concept of user-collaboration and provide a framework to enable the service users to share their individually-obtained Web service QoS values for the best fault tolerance strategy. The experimental prototype is shown to make Web service evaluation and selection effcient, effective and optimal.

Recently, dynamic Web service composition has attracted great interests, where complex applications are specified as service plans and the optimal service candidates are dynamically determined at runtime by solving optimization problems. Although the problem of dynamic Web service selection has been studied by a number of research tasks (Ardagna & Pernici, 2007; Bonatti & Festa, 2005; Yu et al., 2007; Zeng et al., 2004; Sheng et al., 2009), very few previous work focuses on the problem of dynamic optimal fault tolerance strategy selection. In this paper, we address this problem by proposing an optimal fault tolerance strategy selection algorithm, which can dynamically update the optimal fault tolerance strategy to deal with the frequent context information changes.

The WS-Reliability (OASIS, 2005) can be employed in our overall Web services framework for enabling reliable communication. The proposed WSDREAM framework can be integrated into the SOA runtime governance framework (Kavianpour, 2007) and applied to industry projects.

9. CONCLUSION

This paper proposes a distributed fault tolerance strategy evaluation and selection framework for Web services. Based on this framework, we study and compare various fault tolerance strategies by theoretical formulas as well as experimental results. Based on both objective QoS performance of Web services as well as subjective user requirements, an optimal strategy selection algorithm is designed. Motivated by the lack of large-scale real-world experiments in the field of service-oriented computing, a prototype (WS-DREAM) is implemented and comprehensive real-world experiments are conducted by distributed service users all over the world. The experimental results are employed for performance study of the individual Web services, the fault tolerance strategies, and the proposed strategy selection algorithm. With the facility of the proposed framework, accurate evaluation of Web services and fault tolerance strategies can be acquired effectively through user-collaboration, and optimal fault tolerance strategy can be obtained dynamically at runtime.

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 (e.g., the values of the thresholds a, b and c), the investigation of more QoS properties, and better use of historical Web service evaluation results.

ACKNOWLEDGMENT

The work described in this paper was fully supported by Microsoft Research Asia Research Grant FY09-RES-OPP-103, and a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4158/08E).

REFERENCES

Ardagna, D., & Pernici, B. (2007). Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 369–384. doi:10.1109/TSE.2007.1011

Bonatti, P. A., & Festa, P. (2005). On optimal service selection. In Proceedings of the WWW (pp. 530-538).

Bram, C. (2003). Incentives build robustness in bittorrent. In *Proceedings of the First Workshop on the Economics of Peer-to-Peer Systems.*

Chan, P. P., Lyu, M. R., & Malek, M. (2007). Reliable web services: Methodology, experiment and modeling. In *Proceedings of the ICWS* (pp. 679-686).

Deora, V., Shao, J., Gray, W. A., & Fiddian, N. J. (2003). A quality of service management framework based on user expectations. In *Proceedings of the ICSOC*.

Fang, C. L., Liang, D., Lin, F., & Lin, C. C. (2007). Fault tolerant web services. *Journal of Systems Architecture*, *53*(1), 21–38. doi:10.1016/j.sysarc.2006.06.001

Foster, H., Uchitel, S., Magee, J., & Kramer, J. (2003). Model-based verification of web service compositions. In *Proceedings of the ASE*.

Jaeger, M. C., Rojec-Goldmann, G., & Muhl, G. (2004). Qos aggregation for web service composition using workflow patterns. In *Proceedings of the EDOC* (pp. 149-159).

Kavianpour, M. (2007). Soa and large scale and complex enterprise transformation. In *Proceedings of the ICSOC* (pp. 530-545).

Keller, A., & Ludwig, H. (2002). The wsla framework: Specifying and monitoring service level agreements for web services. In *Proceedings of the IBM Research Division.*

Leu, D., Bastani, F., & Leiss, E. (1990). The effect of statically and dynamically replicated components on system reliability. *IEEE Transactions on Reliability*, *39*(2), 209–216. doi:10.1109/24.55884

Ludwig, H., Keller, A., Dan, A., King, R., & Franck, R. (2003). A service level agreement language for dynamic electronic services. *Electronic Commerce Research*, *3*(1-2), 43–59. doi:10.1023/A:1021525310424

Lyu, M. R. (1995). *Software Fault Tolerance. Trends in Software*. New York: Wiley.

Maximilien, E., & Singh, M. (2002). Conceptual model of web service reputation. *SIGMOD Record*, *31*(4), 36–41. doi:10.1145/637411.637417

Merideth, M. G., Iyengar, A., Mikalsen, T., Tai, S., Rouvellou, I., & Narasimhan, P. (2005). Thema: Byzantine-fault-tolerant middleware forweb-service applications. In *Proceedings of the SRDS* (pp. 131-142).

Moritsu, T., Hiltunen, M. A., Schlichting, R. D., Toyouchi, J., & Namba, Y. (2006). Using web service transformations to implement cooperative fault tolerance. In *Proceedings of the ISAS* (pp. 76-91).

O'Sullivan, J., Edmond, D., & ter Hofstede, A. H. M. (2002). What's in a service? *Distributed and Parallel Databases*, *12*(2/3), 117–133. doi:10.1023/A:1016547000822

OASIS. (2005). *Web services reliable messaging*. Retrieved from http://specs.xmlsoap.org/ws/2005/02/rm/ws-reliablemessaging.pdf

Ouzzani, M., & Bouguettaya, A. (2004). Efficient access to web services. *IEEE Internet Computing*, *8*(2), 34–44. doi:10.1109/MIC.2004.1273484

Rosario, S., Benveniste, A., Haar, S., & Jard, C. (2008). Probabilistic qos and soft contracts for transaction-based web services orchestrations. *IEEE Trans on Services Computing*, *1*(4), 187–200. doi:10.1109/TSC.2008.17

Sahai, A., Durante, A., & Machiraju, V. (2002). Towards automated sla management for web services. In *Proceedings of the HP Laboratory*.

Salas, J., Perez-Sorrosal, F., Marta Pati, M., & Jim'enez-Peris, R. (2006). Wsreplication: a framework for highly available web services. In *Proceedings of the WWW* (pp. 357-366).

Salatge, N., & Fabre, J. C. (2007). Fault tolerance connectors for unreliable web services. In *Proceedings of the DSN* (pp. 51-60). DOI http://dx.doi.org/10.1109/DSN.2007.48

Santos, G. T., Lung, L. C., & Montez, C. (2005). Ftweb: A fault tolerant infrastructure for web services. In *Proceedings of the EDOC* (pp. 95-105).

Sheng, Q. Z., Benatallah, B., Maamar, Z., & Ngu, A. H. (2009). Configurable composition and adaptive provisioning of web services. *IEEE Trans on Services Computing*, *2*(1), 34–49. doi:10.1109/TSC.2009.1

Sheu, G. W., Chang, Y. S., Liang, D., Yuan, S. M., & Lo, W. (1997). A fault-tolerant object service on corba. In *Proceedings of the ICDCS* (p. 393).

Thio, N., & Karunasekera, S. (2005). Automatic measurement of a qos metric for web service recommendation. In *Proceedings of the ASWEC* (pp. 202-211).

Vieira, M., Laranjeiro, N., & Madeira, H. (2007). Assessing robustness of web-services infrastructures. In *Proceedings of the DSN* (pp. 131-136). doi: http://dx.doi.org/10.1109/DSN.2007.16

Wu, G., Wei, J., Qiao, X., & Li, L. (2007). A bayesian network based qos assessment model for web services. In *Proceedings of the SCC*.

Yu, T., Zhang, Y., & Lin, K. J. (2007). Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans Web, 1*(1), 6. doi: http://doi.acm.org/10.1145/1232722.1232728

Zeng, L., Benatallah, B., Ngu, A. H., Dumas, M., Kalagnanam, J., & Chang, H. (2004). Qos-aware middleware for web services composition. *IEEE Trans Softw Eng, 30*(5), 311-327. doi: http://dx.doi.org/10.1109/TSE.2004.11

Zhang, L. J., Zhang, J., & Cai, H. (2007). *Services Computing*. New York: Springer.

Zheng, Z., & Lyu, M. R. (2008a). A distributed replication strategy evaluation and selection framework for fault tolerant web services. In *Proceedings of the ICWS* (pp 145-152).

Zheng, Z., & Lyu, M. R. (2008b). Ws-dream: A distributed reliability assessment mechanism for web services. In *Proceedings of the DSN* (pp. 392-397).

Zibin Zheng received his B.Eng. degree and M.Phil. degree in Computer Science from the Sun Yat-sen University, Guangzhou, China, in 2005 and 2007, respectively. He is currently a Ph.D. candidate in the department of Computer Science and Engineering, The Chinese University of Hong Kong. He received SIGSOFT Distringuish Paper Award at ICSE'2010, Best Student Paper Award at ICWS'2010, and IBM Ph.D. Fellowship Award 2010-2011. He served as program committee member of IEEE CLOUD'2009 and CLOUDCOMPUTING'2010. He also served as reviewer for international journal and conferences, including TSE, TPDS, TSC, JSS, DSN, ICEBE, ISSRE, KDD, SCC, WSDM, WWW, etc. His research interests include service computing, software reliability engineering, and cloud computing.

Michael R. Lyu received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 1981; the M.S. degree in computer engineering from University of California, Santa Barbara, in 1985; and the Ph.D. degree in computer science from the University of California, Los Angeles, in 1988. He is currently a Professor in the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong, China. He is also Director of the Video over Internet andWireless (VIEW) Technologies Laboratory. He was with the Jet Propulsion Laboratory as a Technical Staff Member from 1988 to 1990. From 1990 to 1992, he was with the Department of Electrical and Computer Engineering, University of Iowa, Iowa City, as an Assistant Professor. From 1992 to 1995, he was a Member of Technical Staff in the applied research area of Bell Communications Research (Bellcore), Morristown, NJ. From 1995 to 1997, he was a Research Member of Technical Staff at Bell Laboratories, Murray Hill, NJ. His research interests include software reliability engineering, distributed systems, fault-tolerant computing, mobile networks, Web technologies, multimedia information processing, and E-commerce systems. He has published over 270 refereed journal and conference papers in these areas. He has participated in more than 30 industrial projects and helped to develop many commercial systems and software tools. He was the editor of two book volumes: Software Fault Tolerance *(New York: Wiley, 1995) and* The Handbook of Software Reliability Engineering *(New York: IEEE and New McGraw-Hill, 1996). Dr. Lyu received Best Paper Awards at ISSRE'98 and ISSRE'2003. Dr. Lyu initiated the First International Symposium on Software Reliability Engineering (ISSRE) in 1990. He was the Program Chair for ISSRE'96 and General Chair for ISSRE'2001. He was also PRDC'99 Program Co-Chair, WWW10 Program Co-Chair, SRDS'2005 Program Co-Chair, PRDC'2005 General Co-Chair, and ICEBE'2007 Program Co-Chair, and served in program committees for many other conferences including HASE, ICECCS, ISIT, FTCS, DSN, ICDSN, EUROMICRO, APSEC, PRDC, PSAM, ICCCN, ISESE, and WI. He has been frequently invited as a keynote or tutorial speaker to conferences and workshops in the U.S., Europe, and Asia. He has been on the Editorial Board of the* IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, *the* IEEE TRANSACTIONS ON RELIABILITY, *the* Journal of Information Science and Engineering, *and* Software Testing, Verification & Reliability Journal. *Dr. Lyu is an IEEE Fellow and AAAS Fellow for his contributions to software reliability engineering and software fault tolerance. He was also named Croucher Senior Research Fellow in 2008.*