A Runtime Dependability Evaluation 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

Service-oriented systems are usually built on top of Web service components, which are distributed across the Internet, making dependability a big challenge. In this paper, we propose a runtime dependability evaluation framework for fault tolerant Web services to attack this crucial problem. We first propose a user-collaborative framework for collecting Web service QoS information from both the service providers and service users. Then, Web service QoS models, fault tolerance strategies, and optimal Web service recommendation approaches are presented. Finally, the benefits of the runtime evaluation framework are demonstrated by real-world experiments. As illustrated by the experimental results, our proposed framework makes fault tolerance for distributed service-oriented systems feasible, reconfigurable, and optimized.

1. Introduction

Web services are loosely-coupled components provided by different organizations to support Machine to Machine interaction using standard protocols, such as WSDL and SOAP. In the service-oriented environment, complex distributed systems can be dynamically composed by invoking remote Web services via network. Dependability of the service-oriented systems become a big challenge, since the Web service components are usually distributed across the Internet and invoked via communication links,

Software fault tolerance by design diversity is an important approach for building reliable systems [5]. This used-to-be expensive approach now becomes a viable solution to the fast-growing service-oriented computing arena, since functionally equivalent Web services can be employed for building diversity-based fault tolerant SOA systems, which react to Web service component failures to prevent them to turn into system failures. When applying traditional fault tolerance technologies to the service-oriented systems, there are several challenges need to be addressed: (1) since the Web services are provided and hosted by different organizations and some may charge for invocations, it is really difficult to conduct traditionally exhaustive evaluation and testing on the target Web services before employing them. However, without sufficient evaluation, we have no idea on the dependability of remote Web services and thus are difficult to determine optimal fault tolerance strategy. (2) Traditionally, fault tolerance strategy was fixed statically at design-time. Due to the unpredictable nature of the Internet, the lack of control of the remote Web services, and the dynamic workload changes of the invoked Web services, static fault tolerance approaches are infeasible to apply to the service-oriented systems.

Therefore, in the service-oriented computing arena, we need runtime dependability evaluation of the remote Web service components and feasible proactive approaches for the reconfiguration of the fault tolerance strategy to build more reliable systems. In our previous work, a distributed mechanism, named *WS-DREAM*, is proposed for Web service reliability evaluation [10, 11], where the optimal fault tolerance strategy is fixed at design time. In this paper, we focus on runtime dependability evaluation of Web services and proactive fault tolerance strategy reconfiguration. The contribution of this paper includes:

- A runtime user-collaborative framework for dependability evaluation of Web services. To the best of our knowledge, this paper is the first work that propose the idea of collaboration between service users and service providers for achieving efficient and effective runtime Web service evaluation;
- A systematic introduction of various fault tolerance strategies, as well as how to dynamically determine optimal Web service and optimal fault tolerance strategy;
- Real-world experiments for verifying the feasibility of our framework.

The rest of this paper is organized as follows: Section 2

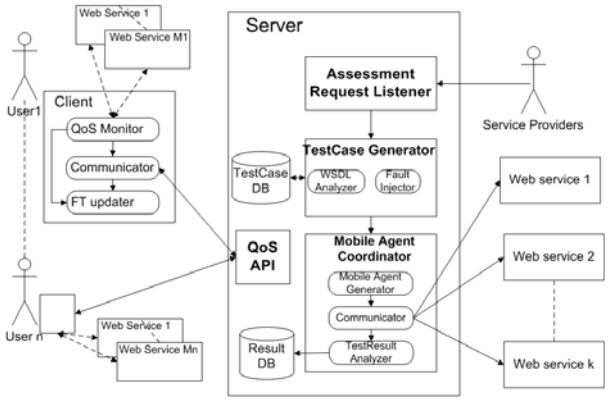

Figure 1. Runtime Evaluation Framework

introduces the system architecture. Section 3 presents the methodologies. Section 4 shows our implementation and experiments and Section 5 concludes the paper.

2. System Architecture

To make runtime evaluation of Web services efficient and effective, we propose a framework for reusing the evaluation results from other service users as well as the service providers. As shown in Fig.1, the proposed runtime evaluation framework includes a centralized evaluation server, a number of distributed service users, and a number of service providers. We assume that the centralized evaluation server is hosted by a trust-worthy third-party. This framework employs the concept of *user-collaboration*, which is the key concept of Web 2.0, for conducting runtime evaluation of Web services. In this framework, service users and service providers in different geographical locations collaborate with each other and share their evaluation results on the Web services.

As shown in Fig.1, from the perspective of service providers, the procedures of employing our framework are as follows:

(1). Evaluation registration. Service providers submit evaluation requests to the evaluation server.

(2). Test case generation. The *TestCase Generator* in the server automatically creates test cases based on the WSDL files of the target Web Services.

(3). Mobile-agent coordination. The generated test cases are deployed to mobile agents, and the mobile agents are sent to the target Web services for conducting evaluation. Since we assume that the centralized server is hosted by a trust-worthy third-party, so the service providers can trust the mobile agents and authorize them to conduct testing in their servers. After finish the evaluations, the mobile agents will carry back the evaluation results and the *TestResult Analyzer* will process and send back the evaluation results to the providers.

By employing the above approach, the service providers can save a lot of time, since they do not need to have good knowledge on the Web service testing and test case generation. By using mobile agents, the testing is conducted in the servers of the target Web services, thus save a lot of bandwidth and make the evaluation more efficient. The mobile agent testing mainly focuses on the functionalities, which are the same to all the service users and thus no need to be conducted in different locations. Evaluations on the non-functionalities (e.g., response-time, success-rate, etc.) are conducted by the service users in different locations under different network conditions. By employing our evaluation framework, the service providers can also obtain the evaluation results contributed by distributed service users.

From the perspective of service users, the procedures of employing our framework can be explained as follows.

(1). Runtime Web service evaluation. the *QoS Monitor* of the service user records the QoS performance of the invoked Web services.

(2). QoS information exchanging. The *Communicator* automatically contributes the individually obtained QoS information to the centralized evaluation server for exchanging QoS information from other service users and service providers. The exchanging mechanism can be designed as a Web service for enabling programmatic invocations.

(3). Fault tolerance strategy reconfiguration. Employing the updated QoS data, the *FT Updater* module automatically adjust the fault tolerance strategy to be the most properly configuration with optimal Web service candidates.

By employing our framework, the service users can obtain: (1) evaluation results observed by himself/herself; (2) evaluation results contributed by other service users; and (3) evaluation results provided by the service providers. By this way, the evaluation o target Web services become more efficient and less-expensive. Since the service users are under different network conditions, the evaluation result from others may not be useful for the current user. How to make properly use of the QoS information from other service users will be discussed in Section 3.3.

From the perspective of the centralized evaluation server, by providing evaluation services to the service providers, the evaluation server obtains the evaluation results from the service providers. By exchanging the QoS information with the active service users, the centralized server obtains the individually observed Web service QoS information from distributed service users. By this *user-collaboration* way, evaluation results can be reused and resource (e.g., network traffic, Web service invocations, etc.) can be saved. The test case generation, which has been discussed in work [9, 4], is out of the scope of this paper.

3. Methodologies

In this section, we first introduction an extendable QoS model for Web services (Section 3.1) and various fault tolerance strategies for Web services (Section 3.2). Then, the approaches for the optimal Web service selection are discussed (Section 3.3). Finally, the dynamic fault tolerance strategy reconfiguration is presented (Section 3.4).

3.1. QoS Model of Web services

QoS provides non-functional characteristics for the optimal Web service selection. The most commonly used QoS properties for Web services include: *availability, price, popularity, data-size, process-time, success-rate, response-time*, and so on [1]. The QoS properties can be divided into two types: QoS properties provided by the service providers and QoS properties observed by the service users. For example, *availability, price, popularity, data-size* are the same for all the service users and are provided by the service providers, while *success-rate, process-time* and *response-time* is measured by the service users and may different from user to user.

The detailed introduction on these QoS properties can be found in [12]. Apart from the above well-known QoS properties, we propose two more properties, *Overall-success-rate* and *overall-response-time*, where *Overall-success-rate* is the average value of the invocation *success-rate* provided by different service users, and *overall-response-time* is the average value of the *response-time* of different service users. These two values can be obtained by employing our *user-collaborative* framework in Section 2. These overall QoS properties provide useful information for the Web service selection, especially for the new service users, who have no knowledge on the performance of different Web service candidates.

Given the above quality properties, the QoS performance of a Web service can be presented as:

$$q = (q^1, ..., q^m), \tag{1}$$

where q^i is the value of the i^{th} QoS property and m is the number of QoS properties. The above QoS model is extensible, where the existing QoS properties can be replaced and new QoS properties [6] can be added in the future easily without fundamental changes.

3.2. Fault Tolerance Strategies

Due to the employment of remote service components, dependability becomes a major issue when applying service-oriented systems to critical domains. Software

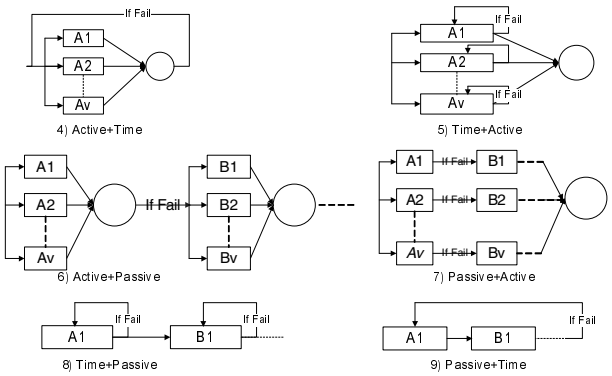

Figure 2. Fault Tolerance Strategies

fault tolerance by design diversity [5] is a feasible approach for building reliable service-oriented systems. *Time-redundancy*, and *space-redundancy* are two types of redundancy for tolerating faults [3, 7]. *Time-redundancy* uses extra computation/communication time to tolerate faults, while *space-redundancy* employs extra resources, such as hardware or software, to mask faults. *Active-replication* and *passive-replication* are two types of *space-redundancy*. *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 [2]. On the other hand, *passive-replication* invokes a primary candidate to process the request first. Backup candidates will be invoked only when the primary candidate fails. These basic fault tolerance strategies are named $Time$, $Passive$, and $Active$, respectively.

More feasible fault tolerance strategies can be obtained by combining the basic fault tolerance strategies. As shown in Fig. 2, there are six combined fault tolerance strategies, named *Active+Time, Time+Active, Active+Passive, Passive+Active, Time+Passive, and Passive+Time*, respectively. The systematic introduction of these fault tolerance strategies can be found in [10] and the formula for calculating the *success-rate* and *response-time* of different strategies are shown in Table 1. As discussed in the work [3], we assume the remote Web services are failed in a fixed rate, and the invocation of each Web service candidates is independent with each other.

3.3. Optimal Web Service Selection

Dynamic optimal Web service selection can be conducted by employing the Web service QoS information from other service users. The easiest way is to rank the target Web service candidates based on their overall QoS

Table 1. Formula for FT Strategies

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

performance (OP), which is defined as:

$$OP = \sum_{i=1}^{m} w_i \tilde{q}^i, \tag{2}$$

where w_i is the weight for the i^{th} QoS property ($\sum_{i=1}^{m} w_i = 1$), m is the number of QoS properties, and $\tilde{q}^i$ is the average value of the i^{th} QoS property of different users. $\tilde{q}^i$ can be calculated by:

$$\tilde{q}^i = \frac{1}{n} \sum_{j=1}^{n} q^{ij}, \tag{3}$$

where n is the number of service users, and q^{ij} is the QoS value provided by the j^{th} service user.

However, since different service users are under different network conditions, the current service user may not be able to obtain similar Web service performance (e.g., response-time, success-rate, etc.) as other service users. To make proper use of the QoS information provided by other service users, we propose a collaborative filtering method for optimal Web service recommendation. The main idea is that only the similar service users will be employed to predict the QoS values of the current service user. The procedures of Web service recommendation are as follows:

(1) Similarity computation.

Pearson Correlation Coefficient (PCC) [8] is employed to define the similarity between two service users a and u based on the Web service items they commonly employed using the following equation:

$$Sim(a, u) = \frac{\sum_{i \in I}(r_{a,i} - \overline{r}_a)(r_{u,i} - \overline{r}_u)}{\sqrt{\sum_{i \in I}(r_{a,i} - \overline{r}_a)^2} \sqrt{\sum_{i \in I}(r_{u,i} - \overline{r}_u)^2}}, \tag{4}$$

where $I = I_a \cap I_u$ is the subset of Web service items which user a and user u both invoked, $r_{a,i}$ is the QoS value of Web service item i observed by service user a, and $\overline{r}_a$ represents the average QoS value of the service user a. From this definition, the service user similarity, $Sim(a, u)$, is in the interval of [-1,1] with a larger value indicating that users a and u are more similar.

(2) Similar user selection. After calculating the similarities between the current user and different users, the top-k similar users can be employed for the QoS value prediction.

(3) QoS value prediction. Collaborative filtering methods use similar users to predict the missing values (QoS values) for the active users by employing the following equation:

$$P(r_{u,i}) = \overline{u} + \frac{\sum_{u_a \in S(u)} Sim'(u_a, u)(r_{u_a,i} - \overline{u}_a)}{\sum_{u_a \in S(u)} Sim'(u_a, u)}, \tag{5}$$

where $\overline{u}$ is the average QoS value of different Web services observed by the active user u, and $\overline{u}_a$ is the average QoS value of different Web services observed by the similar service user u_a.

(4) Web service recommendation. After predicting the QoS values for the current user, the top-k optimal Web services can be recommended to the current user. Due to space limitation, more details of our collaborative filtering algorithm can be found in [13].

3.4. Dynamic FT Strategy Reconfiguration

After obtaining the QoS value predictions of different Web service candidates, proactive approach can be employed for dynamically reconfiguring optimal fault tolerance strategy. Optimal fault tolerance strategies for service-oriented applications vary from case to case, which are influenced not only by Web service QoS performance, but

also influenced by subjective service user requirements. For example, latency-sensitive applications may prefer *Active* strategy for obtaining better response-time performance, while resource-constrained applications may prefer the *Passive* strategy for better resource conservation. By employing the formula in Table 1 and the QoS information of the Web service candidates, the QoS performance of different fault tolerance strategies can be calculated and the optimal one can be determined. The detailed fault tolerance reconfiguration algorithms can be found in our work [12].

4. Implementation and Experiments

4.1. Implementation and Experiment Setup

To illustrate our runtime dependability evaluation framework, a prototype (*www.wsdream.net*) is implemented. The mobile agents are realized as *Java Applets*, which can be loaded and executed automatically by Internet browsers of Web service providers. The centralized evaluation server is implemented in Java language. The client-side evaluation modules of service users are implemented as Java packages and deployed to different geography locations (CN, AU, US, SG, TW and HK) for conducting real-world experiments. The functionally equivalent *Amazon Web services*, which are located in US, Japan, Germany, Canada, France and UK, respectively, are employed as redundant Web service candidates. The detailed experimental results are reported in the following.

4.2. Evaluation of Individual Web Services

Figure 3 shows the *response-time* performance observed by different service users of invoking the US Amazon Web service (*a-us*). Figure 3 shows that RTT performance of target Web services change dramatically from place to place. For example, in the user-location of CN, the average RTT value is 4184 milli-seconds (ms), while in the user-location of US, the average RTT value is only 74 ms. Moreover, figure 3 also shows that even in the same location, the RTT values vary drastically from time to time. The unstable RTT performance degrades service quality and makes the latency-sensitive applications easy to fail.

Figure 4 shows the *process-time* performance of the six Amazon Web services. Figure 4 shows that the average *process-time* of all the six Amazon Web services are less than 30 milliseconds, which is very small compared with the *response-time* shown in Figure 3. It indicates that the *response-time* mainly consists of *network-latency* rather than server *processing-time* in our experiment. Among all the six Amazon Web services, *a-jp* provides the worst *process-time* performance, which may be related to the server workload.

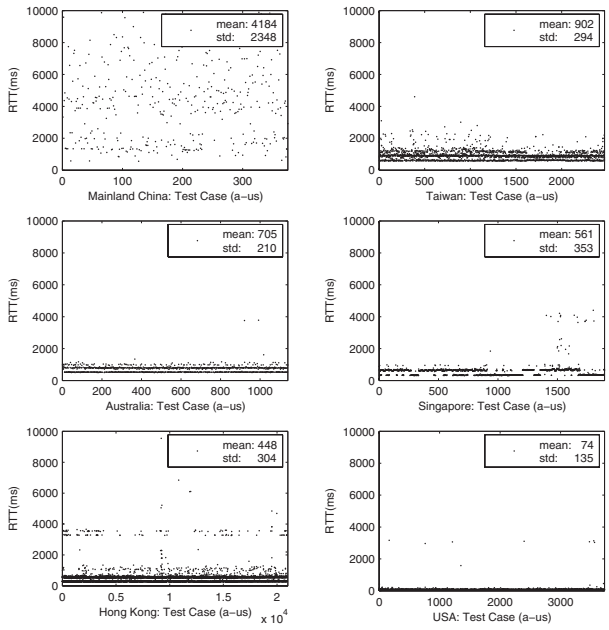

Figure 3. RTT Performance of Accessing a-us

Table 2. Failure-rate of the Web Services

WS	CN	TW	AU	SG	HK	US
a-us	22.52	0	0	0	0.38	0
a-jp	26.55	0.03	0	0	0.05	0
a-de	23.40	0	0	0	3.45	0
a-ca	24.23	0.19	0	0	0.58	0
a-fr	19.27	0	0	0	3.52	0
a-uk	20.28	0.03	0.25	0	3.87	0

Table 2 shows the failure-rate of the six Amazon Web services observed by the six distributed service users. Table 2 shows that users with worst RTT performance (CN) have the highest failure rate, while users with best RTT performance (US) have the lowest failure rate. This indicates that network performance can greatly influence the invocation success-rate, since unstable RTT performance will degrade service quality and can even lead to timeout-failures. The failures observed in our experiment includes *TimeoutException*, *Service Unavailable* (http code 503) and *Bad Gateway* (http code 502).

4.3. Evaluation of FT Strategies

Table 3 shows the experimental results of various fault tolerance strategies from the user-location of *HK*. Table 3 shows that strategy 1 ($Active$) has the best RTT performance. This is because $Active$ strategy invokes all the six candidates at the same time and employs the first response as the final result. Its failure-rate is high compared with

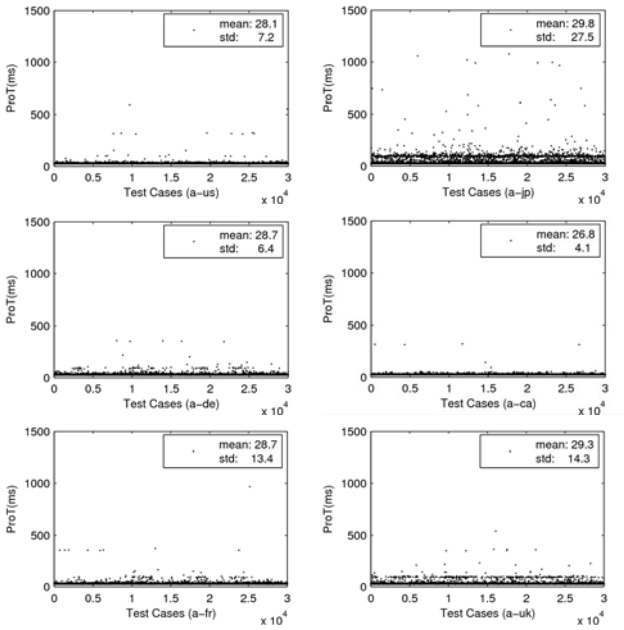

Figure 4. Process Time of the Web Services

Table 3. Evaluation of FT 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

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 2. RTT performance of sequential strategies (Strategies 2, 3, 8 and 9) is worse than other strategies, because sequential strategies invoke candidates one by one. Hybrid strategies also (Strategies 4, 5, 6 and 7) achieve good RTT performance, although not the best. Due to space limitation, more detailed experimental results can be found in [10].

5. Conclusion

In this paper, we propose a runtime user-collaborative evaluation framework for Web services. QoS-model, fault tolerance strategies, dynamic Web service selection, and dynamic fault tolerance strategy reconfiguration are presented. A prototype is implemented to illustrate the advantages of our framework.

Our future work will consider more advance features of the framework, such as investigate more QoS properties and study stateful Web services.

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. CUHK4158/08E), and a grant from the Research Committee of The Chinese University of Hong Kong (Project No. CUHK3/06C-SF).

References

[1] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Software Eng.*, pages 369–384, 2007.

[2] P. P.-W. Chan, M. R. Lyu, and M. Malek. Making services fault tolerant. In *ISAS*, pages 43–61, 2006.

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

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

[5] M. R. Lyu. *Software Fault Tolerance*. Trends in Software, Wiley, 1995.

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

[7] N. Salatge and J.-C. Fabre. Fault tolerance connectors for unreliable web services. In *DSN*, pages 51–60, 2007.

[8] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.

[9] M. Vieira, N. Laranjeiro, and H. Madeira. Assessing robustness of web-services infrastructures. In *DSN*, pages 131–136, 2007.

[10] Z. Zheng and M. R. Lyu. A distributed replication strategy evaluation and selection framework for fault tolerant web services. In *ICWS*, pages 145–152, Beijing, China, Sept. 2008.

[11] Z. Zheng and M. R. Lyu. Ws-dream: A distributed reliability assessment mechanism for web services. In *DSN*, pages 392–397, Anchorage, Alaska, USA, June 2008.

[12] Z. Zheng and M. R. Lyu. A qos-aware fault tolerant middleware for dependable service composition. In *DSN*, Lisbon, Portugal, June 2009.

[13] Z. Zheng, H. Ma, M. R. Lyu, and I. King. Wsrec: A collaborative filtering based web service recommender system. In *ICWS*, 2009.