Developing Aerospace Applications with a Reliable Web Services Paradigm

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

Abstract—One of the latest achievements of the Internet usage is the availability of Web services technology. Web services provide an efficient and convenient way for service provisioning, exchanging and aggregating, which facilitates a resourceful platform for the aerospace industry. The aerospace industry usually involves products of complex synthesis of various technologies and sciences. These different technical resources can be provided in the form of Web services to increase their availability, efficiency and performance. However, in aerospace area, reliability is an ultimately important issue. In this paper, we target on providing a reliable Web service paradigm for the industry. We describe the methods of reliability enhancement by redundancy in space and redundancy in time, identify parameters impacting Web service reliability, and present a Web service composition algorithm. The replication algorithm and the detailed system configuration are described. The Web services are coordinated by a replication manager, which schedules the workload of the Web services and keeps updating the availability of each Web service. We also perform a series of experiments employing several replication schemes and compare them with a non-redundant single service to evaluate the reliability of the proposed paradigm. Finally, we also model the Web services with Petri-Net and Markov chains to demonstrate the performance and reliability of the Web services.

TABLE OF CONTENTS

1-4244-1488-1/08/$25.00 ©2008 IEEE
IEEEAC Paper #1602, Version 1.0 Updated 20 October 2007.

1. INTRODUCTION

A Web service is based on Service-Oriented Architectures (SOA) [1]. This approach simplifies interoperability as only standard communication protocols and simple broker-request architectures are needed to facilitate exchanges of services. Web services are becoming more popular and are beginning to pervade all aspects of human life, including the aerospace industry. Web services provide an efficient and convenient way for service provisioning, exchanging and aggregating. They facilitate a resourceful platform for the aerospace industry. However, in the aerospace industry, the problems of service dependability, security and timeliness are critical.

One important element in the delivery of reliable Web services is that the software itself should be reliable. To achieve this, software needs to be fault-tolerant. Several fault-tolerant approaches have been proposed for Web services in the literature [2], [3], [4], [6], [7], [8], but the field still requires theoretical foundations, appropriate models, effective design paradigms, practical implementations, and in-depth experimentation. We attack these issues in a unified approach in our research, which is aimed at building reliable Web services with credible modeling techniques and critical analyzes.

The rest of the paper is organized as follows. Related work on dependable services is presented in Section 2, in which the problem statement about reliable Web services is identified. In Section 3, methodologies for reliable Web services are presented, and a roadmap to dependable Web services is offered. Web service composition is proposed in Section 4, experimental results are presented in Section 5, and reliability modeling is laid out in Section 6. Finally, conclusions are made in Section 7.

2. RELATED WORK

In aerospace systems, reliability, availability, and maintainability are extremely important. The solar cells in aerospace applications is an example [9], which is shown in Figure 1. A stable supply of electrical power is crucial for the success of space missions. Photovoltaic arrays are the most common means of in-orbit energy generation; however, mechanical solar cell defects have the potential to impact their reliability considerably.

Figure 1. A solar cell in space.

In [10], wireless chip-to-chip communications and interconnects are discussed. Communication is critical for the aerospace application. The ever-growing demand for onboard spacecraft processing combined with the exponential advance in chip development and high pin count devices is resulting in an increased complexity in high reliability interconnect technology. When the thermal management, control of ground bounce, and power distribution issues are considered, achieving the required reliability levels for space applications with traditional interconnect technologies is becoming more of a concern.

Some of the aerospace applications can be developed with Web services, such as the manual Failure Reporting and Corrective Action Process (FRACAS) [11] which is implemented for collecting reliability data from different parts of the organization for aircrafts. The goal of FRACAS is to generate reliability graphs for the entire system and the key components to determine system availability and ensure contractual compliance. Overall, the process is time-consuming and expensive, taking as much as two months of effort to determine key metrics. Additionally, it relies heavily on the knowledge and experience of one key individual personnel.

As more and more aerospace applications are using the Web service technologies, building reliable Web services with fault tolerance is critical for this industry. Fault tolerance can be achieved via spatial or temporal redundancy, including replication of hardware (with additional components), software (with special programs), and time (with diversified operations) [12], [13], [14]. Spatial redundancy can be static or dynamic, both of which use replication. In static redundancy, all replicas are active at the same time and voting takes place to obtain a correct result. The number of replicas is usually odd and the approach is known as n-modular redundancy (NMR). For example, under a single-fault assumption, if services are triplicated and one of them fails, the remaining two will still guarantee the correct result. Dynamic redundancy, on the other hand, engages one primary active replica at one time while others are kept in an active or in a standby state. If the primary replica fails, another active (secondary) replica can be employed immediately with little impact on response time. In the other case, if the active replica fails, a previously inactive replica must be initialized and take over the operations. This may also increase the recovery time because of its dependence on time-consuming error-handling stages such as fault diagnosis, system reconfiguration, and resumption of execution. It is noted that redundancy can be achieved by replicating hardware modules to provide backup capacity when a failure occurs, or redundancy can be obtained using software solutions to replicate key elements of a critical process.

In any redundant systems, common-mode failures result from failures that affect more than one module at the same time, generally due to a common cause. These include design mistakes and operational failures that may be caused externally or internally. Design diversity has been proposed in the past to protect redundant systems against common-mode failures [15], [17] and has been used in both firmware and software systems. The basic idea is that, with different design and implementations, common failure modes can be reduced. One of the design diversity techniques is N-version programming [16], and the other one is Recovery Blocks [18]. The key element of N-version programming or Recovery Block approaches is diversity. By attempting to make the development processes diverse, it is hoped that the independently designed versions will also contain faults that are either non-identical or dissimilar.

Based on the these fault-tolerant approaches, a number of reliable Web services techniques have appeared in the recent literature. WS-FTM (Web Service-Fault Tolerance Mechanism) is an implementation of the classic N-version model for Web services [3], which can easily be applied to existing systems with minimal change. The Web services are implemented in different versions, and the voting mechanism is conducted in the client program. FT-SOAP [4], on the other hand, is aimed at improving the reliability of the Simple Object Access Protocol (SOAP) when using Web services. The system includes different approaches to function replication management, fault management, logging/recovery mechanism and client fault tolerance transparency. FT-SOAP is based on the work of FT-CORBA [5], in which a fault-tolerant SOAP-based middleware platform is proposed.

FT-Grid [19] is another design, which is a deployment of design diversity for fault tolerance in Grid. It is not originally specified for Web services, but the techniques are applicable to Web Services. FT-Grid allows a user to manually search through any number of public or private Universal Description, Discovery and Integration (UDDI) repositories, to select a number of functionally-equivalent services, to choose the parameters for each service, and to invoke these services. The application can then perform voting on the results returned by the services, with the aim of filtering out any anomalous results.

Although a number of approaches have been proposed to increase Web service reliability, there is a need for systematic modeling and experiments to understand the tradeoffs and to verify the reliability of the proposed methods. We proposed a framework [20] for the deployment of reliable Web services, and enhanced the scheme with a Round-robin algorithm and N-version programming for Web services [21]. In this paper, we focus on the systematic analysis of the replication techniques when applied to Web services. A generic Web service system with spatial as well as temporal replication is proposed, and its prototype is implemented as an experimental testbed. To make more versions of Web services available for the proposed paradigm, a dynamic Web service composition is developed and its correctness is verified through different experiments.

3. Methodologies for Reliable Web Services

In this section, we propose a replication Web service system for reliable Web services. Its architecture is shown in Figure 2. In our system, the dynamic approach is adopted.

Scheme Details

In the proposed system, we apply two different approaches for managing spatial replication, including a Round-robin (RR) algorithm and N-version programming. We perform different experiments to evaluate the reliability of the system.

Round-robin approach—In the first approach, the Web servers work concurrently and a Round-robin algorithm [22] is employed for scheduling the work among the Web services. The Web service is replicated on different machines. When there is a Web service failure, other Web servers can immediately provide the required service. The replication mechanism shortens the recovery time and increases the reliability of the system.

The main component of this system is the replication manager (RM), which acts as a coordinator of the Web services. The replication manager is responsible for:

1. Choosing (with an anycasting algorithm) the best (fastest, most robust, etc.) Web service [23] to provide the service, which is called the primary Web service.
2. Keeping the availability list of the Web services.
3. Registering the Web Service Definition Language (WSDL) with the Universal Description, Discovery, and Integration (UDDI).
4. Continuously checking the availability of the Web services by using a watchdog.
5. Applying the Round-robin algorithm for scheduling the workload of the Web service.

The replication manager schedules the work of the Web service using the Round-robin algorithm; therefore, the resources of the system can be fully utilized. The replication manager distributes the work to different Web servers according to the availability of the servers. The requests are sent to different Web services accordingly. Whenever the server changes, the replication manager maps the new address of the Web service providing the service to the WSDL; thus, the clients can still access the Web service with the same URL. This failover process is transparent to the users.

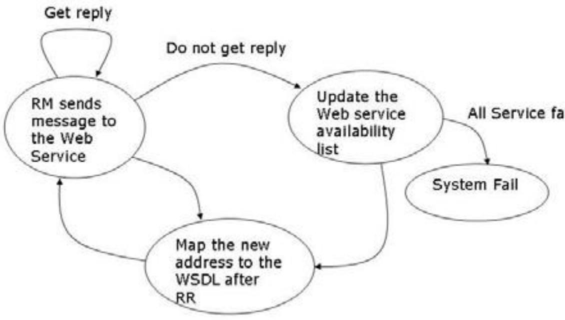

Get reply
Do not get reply
RM sends message to the Web Service
Update the Web service availability list
All Service failed
System Fail
Map the new address to the WSDL after RR

Figure 3. Workflow of the Replication Manager

The workflow of the replication manager is shown in Figure 3. The replication manager is running on a server, which keeps checking the availability of the Web services by a polling method: namely it sends messages to the Web services periodically. If it does not get a reply from the primary Web service, it will select another Web service to replace the primary one and map the new address to the WSDL. The system is considered failed if all the Web services have failed.

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

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

1. Selecting the primary Web service for executing the voting procedure. Once the selected Web service gets the request, it will forward the request to all the Web services.
2. Keeping the availability list of the Web services.
3. Registering the Web Service Definition Language (WSDL) with the Universal Description, Discovery, and Integration (UDDI).
4. Continuously checking the availability of the Web services by using a watchdog.

Roadmap for Experimental Research

We take a pragmatic approach by starting with a single service without any replication. The only approach to fault tolerance in this case is the use of redundancy in *time*. If a service is considered as an atomic action or a transaction in which

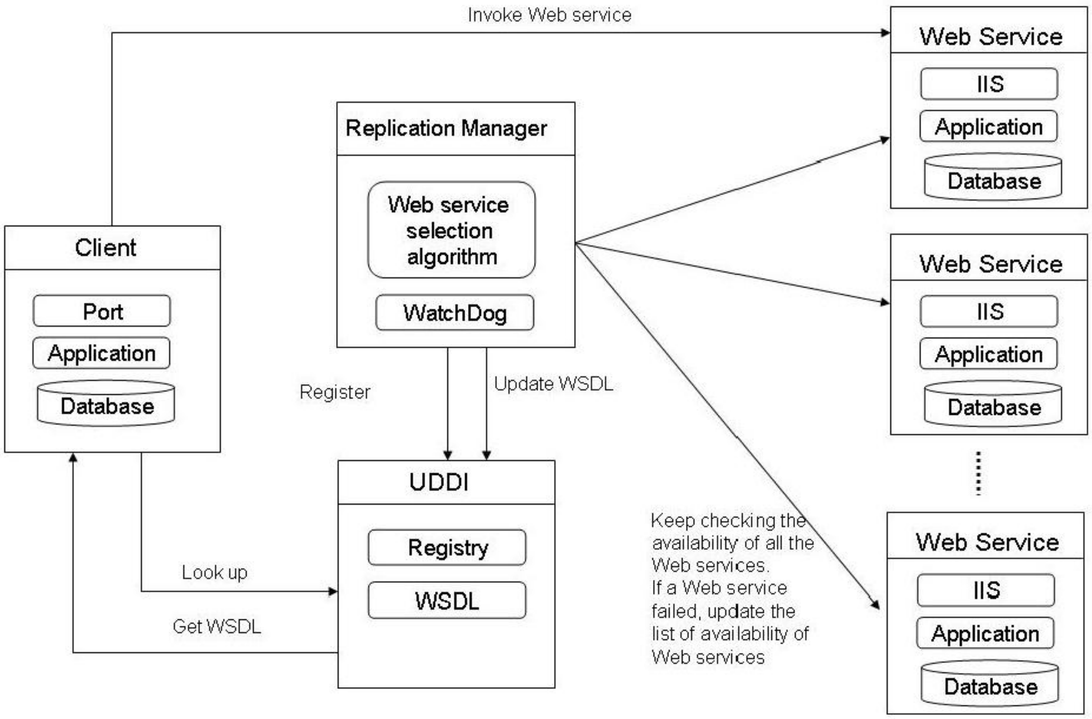


Figure 2. Architecture for dependable Web services.

the input is clearly defined, no interaction is allowed during its execution, and the outcome has two possible states: correct or incorrect. In this case, the only way to make such a service fault tolerant is to retry or reboot it. This approach allows tolerance of temporary faults, but it will not be sufficient for tolerating permanent faults within a server or a service. One issue is how much delay the user can tolerate, and another issue is the optimization of the retry or the reboot time.

If redundancy in time is not appropriate to meet dependability requirements or if the time overhead is unacceptable, the next step is redundancy in *space*. Redundancy in space for services means replication where multiple copies of a given service may be executed sequentially or in parallel. If the copies of the same services are executed on different servers, different modes of operations are possible:

1. Sequentially, meaning that we await a response from a primary service and in case of timeout or a service delivering incorrect results, we invoke a back-up service (multiple backup copies are possible). This is known as static redundancy.
2. In parallel, meaning that multiple services are executed simultaneously and if the primary service fails, the next one takes over. Another variant is that the service whose response arrives first is taken.
3. There is also a possibility of majority voting using n-modular redundancy, where results are compared and the final outcome is based on at least $\lfloor n/2 + 1 \rfloor$ services agreeing on the result. This is known as dynamic redundancy.

If diversified versions of different services are compared, the approach can be seen as either a Recovery Block (RB) system, where backup services are engaged sequentially until the results are accepted (by an Acceptance Test), or an N-version programming (NVP) system where voting takes place and majority results are taken as the final outcome. In case of failure, the failed service can be masked and the processing can continue.

NVP and RB have undergone various challenges and lively discussions. Critics state that the development of multiple versions is too expensive and dependability improvement is questionable in comparison with a single version, provided the development effort equals the development cost of the multiple versions. We argue that, in common with the maturity of service-oriented computing technologies, diversified Web services now predominate and the objections to NVP or RB can be mitigated. Based on market needs, service providers are competitively and independently developing their services and making them available on the market. With an abundance of services available for specific functional requirements, it is apparent that fault tolerance by design diversity will be a natural choice. Moreover, NVP can be applied to services not only for dependability but also for higher performance purposes, due to *locality* considerations.

Finally, a hybrid method may be employed where both space and time redundancy are applied, and depending on system parameters, a retry might be more effective before switching to the back-up service. This type of approach will require a further investigation.

4. Web Service Composition

Diversity is one of the key elements in the proposed paradigm. In the emergence of service-oriented computing, different versions of Web services or even different versions of their components are abundantly available in the Internet. The combination of different versions of the Web service or their components is thus becoming critical for enabling different versions in a Web service application using the N-version approach. In this section, we propose an approach for composing Web services with an N-version Programming Web for improving the reliability of the overall system.

Web Service Description

The description of the Web service is statically provided by WSDL, which includes Web service functional prototypes. However, its static nature limits the flexibility for composing Web services. Different Web services provide their service at different times, and so a dynamic composition approach is necessary for composing different versions of Web services available in the Internet.

In Web services, the communication mainly depends on the messages exchanged between different Web servers. The Web Service Choreography Interface (WSCI) [25] is an XML-based language for the description of the observable behavior of a Web service in the context of a collaborative business process or work-flow. WSCI describes the dynamic interface of the Web Service participating in a given message exchange by means of reusing the operations defined for a static interface. It defines the flow of messages exchanged by a stateful Web service, describing its observable behavior. By specifying the temporal and logical dependencies among the message exchange, WSCI is used to describe a service in such a way that other Web services can unambiguously interact with the described service in conformity with the intended collaboration. Though WSCI provides a message-oriented view of the process, it does not define the internal behavior of the Web service or the process.

Proposed Composition Method

Our proposed service composition method is based on two standard Web service languages: WSDL and WSCI. WSDL describes the entry points for each available service, and WSCI describes the interactions among WSDL operations. WSCI complements the static interface details provided by a WSDL file describing the way operations are choreographed and the associated properties. This is achieved with the dynamic interface provided by WSCI through which the relationship between different operations can be observed in the context of a particular operational scenario.

The flow of the composition procedure is as follows: First, get the WSDL of the Web service components from UDDI. Then, through the messages between the Web services, obtain the WSCI of the components. Afterwards, determine the input and output of the components through WSDL and determine the interactions between different components to provide the service through WSCI. Perform the composition of the Web service with the information obtained using the algorithm described below. The detailed composition algorithm is as follows.

Algorithm 1 Algorithm for Web service composition

```
Require: $I[n]$: required input, $O[n]$: required output
1:  $CP_n$: the $n^{th}$ Web services component
2:  for all $O[i]$ such that $0 \leq i \leq 10$ do
3:    Search the WSDL of the Web services, and find the
      $CP_n$ 's operation output = $O[i]$. Then, insert $CP_n$ into
      the tree.
4:    if the input of the operation = $I[j]$ then
5:      Insert the input to the tree as the child of $CP_n$.
6:    else
7:      Search the WSCI of $CP_n$, WSCI.process.action =
        operation.
8:      Find the previous action needing to be invoked.
9:      Search the operation in WSDL equal to the action.
10:     if input of the operation = $I[i]$ then
11:       Insert input to the tree as the child of $CP_n$
12:     else
13:       go to step (8)
14:     end if
15:   end if
16:   until reaching the root of WSCI and not finding the
      correct input, search other WSDL with their output =
      $I[j]$, insert $CP_m$ as the child of $CP_n$ and go to step (7)
      to do the searching in WSCI of $CP_m$.
17: end for
```

In the algorithm, we aim to build the tree for the Web service composition. We use a bottom-up approach to perform the composition, that is, we build the composition tree from output to input.

When we get the required output, we search the Web services in the WSDL. In the operation tag of the WSDL, the output information is stated. When the desired output is found, that Web service component (CP_n) is inserted as the root of the tree. Then, if the input of that operation matches the required input, the searching is completed and the input is inserted as a child of the CP_n. Otherwise, we will search the action in WSCI which matches the operation in CP_n. After the action is found, we can determine the previous action. Then, we can find the operation prototypes in the WSDL. If the input of this operation matches the required input, their composition is completed. Otherwise, we will iterate until the root of the

WSCI is reached.

If the desired input is still not found, we will search for the operations in the other WSDL whose output is equal to the input of CP_n. If the next Web service component found is CP_m, then CP_m is inserted as the child of CP_n. We perform the searching iteratively to continues to build the tree until all the inputs match the required input.

Case Study

For spacecrafts, entry, descent, and landing are important issues for the whole driving process. Figure 4 shows the configuration of this process. It is a new landing system developed by Mars Science Laboratory [24]. In the landing process, it includes cruise stage separation, de-spin, cruise balance mass jettison, turn to entry attitude, entry interface, peak heating, peak deceleration, heading alignment and deploy parachute. There are a lot of calculations and optimizations in the landing process, including route finding, timing, information searching, communication . . . etc. To simulate the similar example for experimentation, the Best Route Finding system (BRF) is chosen for case study. The architecture is shown in Figure 5.

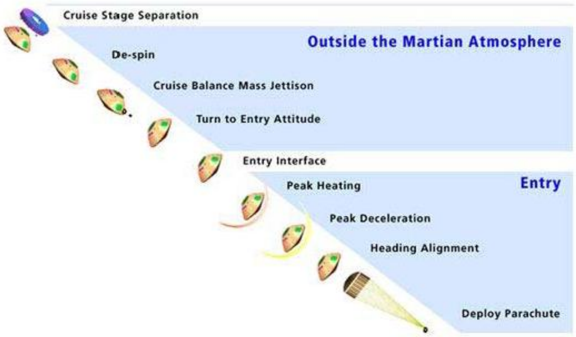


Figure 4. Procedures of landing system.

This system suggests the best route for a journey within Hong Kong by public transport, based on input consisting of the starting point and the destination. BRF consists of different components, including a search engine, agent servers, and the public transport companies. We acquired several versions of BRF, which are implemented by different teams using different components. Also, the Web service components may differ between parties; thus, to illustrate the Web service composition procedure, in this experiment, we try to composite the Web services from different versions with the provided WSDL and WSCI.

The following shows part of the WSDL specification of the search engine. The WSDL shows the input and output parameters of the services provided by the search engine.

```
<?xml version="1.0" encoding="UTF-8"?>
 ...
<portType name=BRF">
  <operation name=shortestpath">
    <input message=
           "tns:startpointDestination"/>
    <output message="tns:pathArray"/>
  </operation>

  <operation name=addCheckpoint">
    <input message="tns:pathArray"/>
    <output message=
           "tns:addAcknowledgement"/>
  </operation>
  ...
  </operation>
</portType> </definitions>
```

The following shows part of the WSCI specification of the search engine.

```
<correlation name=pathCorrelation
property=tns:pathID></correlation>

<interface name=busAgent>
  <process instantiation="message">
    <sequence>
       <action name="ReceiveStartpointDest
          role="tns:busAgent
          operation="tns:BRF/shortestpath">
        </action>
        <action name="Receivecheckpoint
          role=" tns:busAgent
          operation="tns:BRF/addCheckpoint">
          <correlate correlation=
             tns: pathCorrelation/>
          <call process=tns:SearchPath/>
        </action>
    </sequence>
  </process>
...
```

Based on the algorithm, the Composition tree is built, giving the result as shown in Figure 6.

Verification with Petri-Net

To verify the correctness of the composed Web service, Petri-Net [26] is employed. We first construct a Petri-Net for the Web service with the information provided in Business Process Execution Language for Web services (BPEL) [27].

BPEL—BPEL is a language used for the definition and execution of business processes using Web services. BPEL enables the top-down realization of Service Oriented Architecture (SOA) through composition, orchestration, and coordination of Web services. BPEL provides a relatively easy and straightforward way to compose several Web services into new composite services called business processes. After a Web service is composed with the proposed algorithm in the

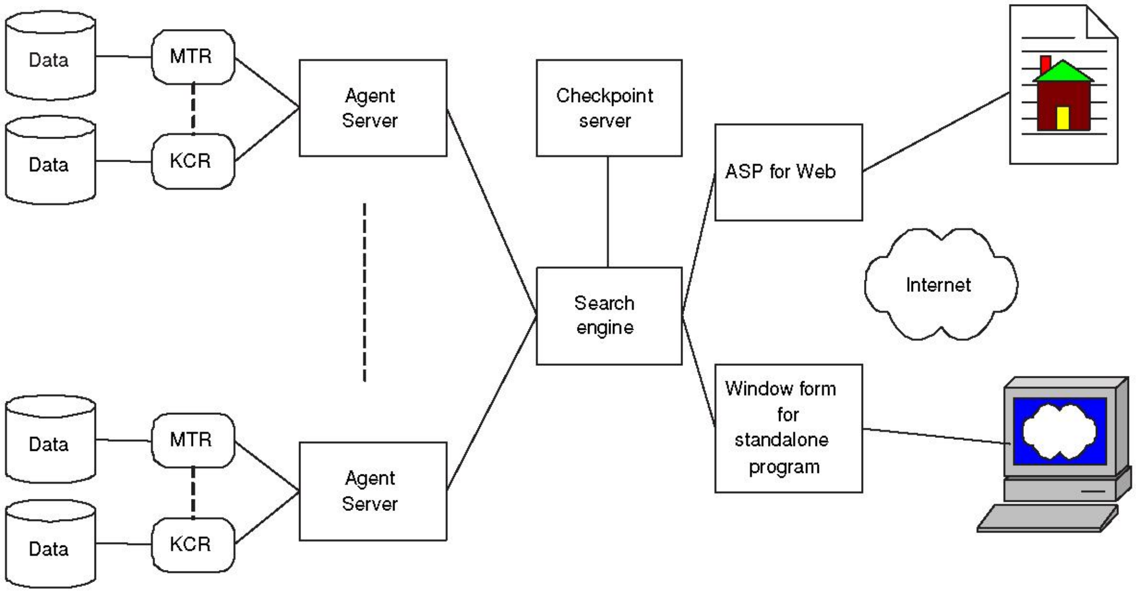


Figure 5. Best Route Finding system architecture.

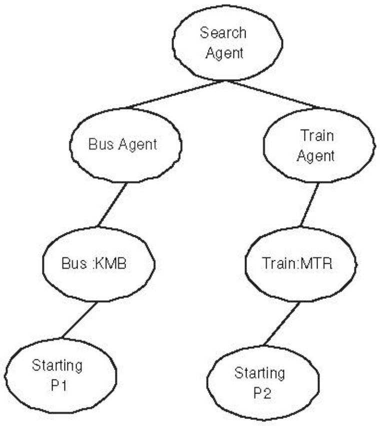


Figure 6. Composition tree of BRF.

pervious section, a BPEL is constructed. BPEL describes the composition properties of the Web service, such as communication and specific behaviors.

A BPEL process specifies the exact order in which participating Web services should be invoked, either sequentially or in parallel. With BPEL, conditional behaviors can be expressed. For example, an invocation of a Web service can depend on the value of a previous invocation. Also it can construct loops, declare variables, copy and assign values, define fault handlers, and so on. By combining all these constructs, the flow of the Web service can be defined.

Building Block of Petri-Net—In the verification process, we employ Petri-Net to build the model of the Web service to prevent deadlock and construct dynamical relations. Different building blocks of Petri-Net are defined according to the activities in BPEL schema, including the inner-service, intra-service, inter-activity, and intra-activity. With the defined blocks, we map the operations or activities specified in BPEL to the Petri-Net building blocks. Then, a Petri-Net for a specified Web service is generated. Some major building blocks are defined in Table 1 and Table 2, respectively.

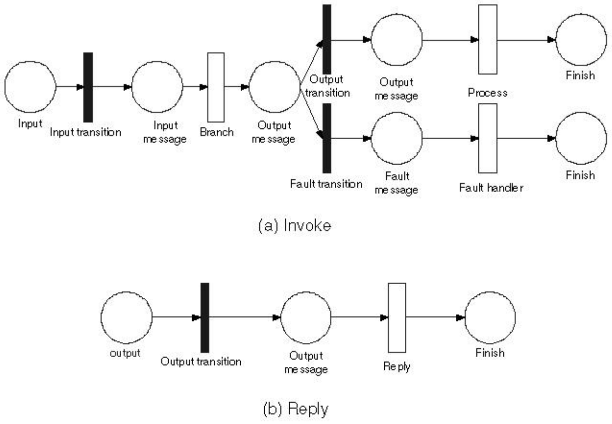
Table 1. Petri-Net building blocks of basic activities

Building Block type	Description
Invoke	The Invoke activity directs a Web service to perform an operation.
Reply	The Reply activity matches a Receive activity. It has the same partner link, port type, and operation as its matching Receive. Use a Reply to send a synchronous response to a Receive.
Empty	The Empty activity is a no operation instruction in the business process.
Assign	The Assign activity updates the content of variables.
Terminate	The Terminate activity stops a business process.
Throw	The Throw activity provides one way to handle errors in a BPEL process.
Wait	The Wait activity tells the business process to wait for a given time period or until a certain time has passed.

Table 2. Petri-Net building blocks of structure activities

Building Block type	Description
While	Repeat the same sequence of activities as long as some condition is satisfied.
Switch	Use "case-statement" to produce branches.
Sequence	Definition of a series of steps for the orderly sequence.
Link	Link different activities work together.
Flow	A series of steps should be specified in parallel implementation.

A Web service operation is composed by basic activity (receive, reply, assign, invoke, empty, terminate, throw and wait) and structures activity (while, switch, sequence, link and flow). Two sample basic activities translation is shown in Figure 7. Web services are described procedurally. A Place connected to a transition intuitively expresses the states before and after executing the corresponding action. Firing a transition means that the corresponding action is executed. Web service invocation is expressed by entering a token in a place which denotes the starting point of the operation.



(a) Invoke

(b) Reply

Figure 7. Basic Petri-Net building block.

Figure 8 illustrates a basic Web service operation composed with Petri-Net building blocks. A building block is presented by a place with a token whose type is specified by the block type. An arc is used to link the transition with another arc connecting to the input/output message consisting of those blocks based on the relationship defined in the schema [27].

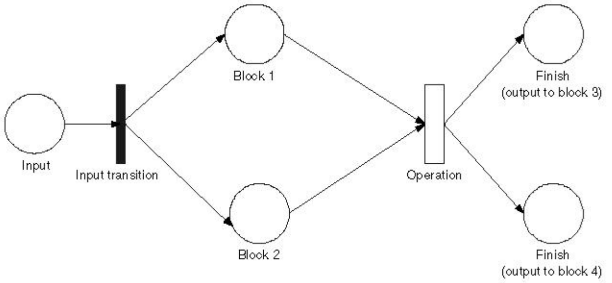


Figure 8. Composed Petri-Net building block graph.

With the Petri-Net building blocks and the BPEL of the BRF, Petri-Net of different version BRF can be generated. One of the composed BRF is shown in Figure 9. With the constructed Petri-Net, we perform the operation to check the correctness and verify that the developed Web service is deadlock free.

5. EXPERIMENT

In this section, we describe the various approaches and some experiments in more detail. We generate different version of BRF with the Web service composition algorithm and evaluate with program metric. Furthermore, we perform experiments to evaluate the reliability of the paradigm proposed in Section 3. We formulate several additional quality-of-service parameters from the viewpoint of service customers. We propose a number of fault injection experiments showing both dependability and performance with and without diversified Web services. The outlined roadmap to fault-tolerant services leads to ultra reliable services where hybrid techniques of spatial and time redundancy can be used for optimal.

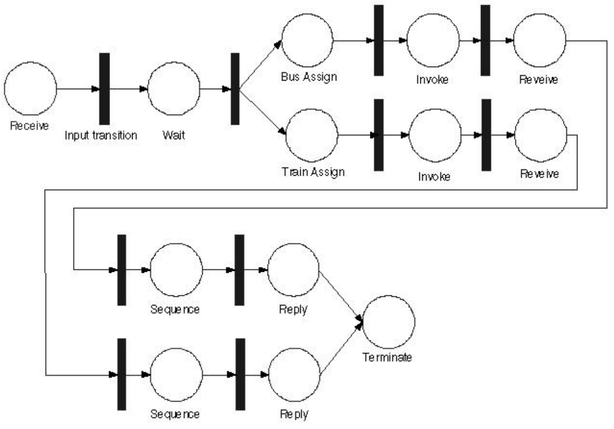


Figure 9. Petri-Net of BRF.

Table 3. Program metrics for 15 versions

ID	Lines	Line without comment	Number of function	Complexity
01	3452	3052	59	64
02	2372	1982	47	87
034	2582	2033	26	45
04	3223	3029	78	124
05	2358	2017	34	89
06	4478	3978	56	107
07	1452	1320	38	46
08	5874	5275	80	124
09	3581	3214	45	74
10	4578	4187	47	113
11	2364	2015	36	76
12	2987	2336	65	147
13	4512	3948	75	155
14	3698	3247	60	192
15	4185	3856	34	88

Different versions of BRF

According to the Web service composition algorithm described above section, different versions of BRF are composed. The program metrics for 15 versions of BRF are shown in Table 3 where the first 11 versions are implemented by different teams and the rest are composed by the proposed algorithm.

A series of experiments are designed and performed for evaluating the reliability of the Web service. In the system, we apply retry, reboot and spatial replication with Round-robin or N-version Web services. We perform the experiments with different combinations. Table 4 shows all the combinations of the experiments.

Table 4. Summary of the experiments

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

Table 5. Parameters of the experiments

	Parameters	Current setting/metric
1	Request frequency	1 req/min
2	Polling frequency	10 per min
3	Number of versions	15
4	Client timeout period for retry	10 mins
5	Max number of retries	5
6	Failure rate λ	number of failures/hour
7	Load (profile of the program)	78.5%
8	Reboot time	10 min
9	Failover time	1 s
10	Communication time to Computational time ratio	10:1
11	Round-robin rate	1 s
12	Temporary fault probability	0.01
13	Permanent fault probability	0.001

Experiment Setup

In our experiments, we run a variety of Web services in the system to evaluate the reliability of the proposed fault tolerant techniques under different situations. Some faults exist in the original version, some faults are injected in the code [28], and some faults are injected in the system using fault injection techniques similar, for example, to those in [6], [29]. A number of faults may occur in the Web service environment [30]. The types of fault injected include permanent fault (the server is down permanently once this fault occurs), temporary fault (the fault only occurs randomly), Byzantine fault [31], [32] and network fault [28]. A Byzantine fault is an arbitrary fault that occurs during the execution of an algorithm by a Web service. In our experiment, several teams implement various versions of the Web service using a number of algorithms, in which the injected faults are triggered. To generate a network fault, WS-FIT fault injection is applied. The fault injector decodes the SOAP message and can inject faults into individual RPC parameters, rather than randomly corrupting a message, for instance by bit-flipping.

Our experimental environment is defined by a set of parameters. Table 5 shows the parameters of the Web services in our experiments. For each of the approaches described in Section 3.1, several experiments are performed. In each experiment, we compare eight approaches, as shown in Table 4, for providing the Web services. They are single server without retry and reboot, single server with retry, single server with reboot, single server with retry and reboot, spatial replication with Round-robin / N-version, spatial replication with Round-robin / N-version and retry, spatial replication with Round-robin / N-version and reboot, and hybrid approach with spatial replication, retry and reboot.

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

Experimental Results

The Web services were executed for 7 days for each experiment, generating a total of 10000 requests from the client. A single failure is counted when the system cannot reply to the client. For the approach with retry, a single failure is counted when a client retries five times and still cannot get the result. A summary of the results with the Round-robin algorithm and N-version programming is shown in Table 6, which shows the improvement of the reliability of the system with our proposed paradigm. In the normal case, no failures are introduced into the system. For the other cases, we insert various kinds of faults into the systems.

When no redundancy techniques are applied on the Web service system (Exp 1), it is clearly seen that the average failure rate of the system is the highest. The results from the two different ways of improving reliability investigated here, i.e., spatial redundancy with replication and temporal redundancy with retry or reboot, are described below.

Single server with retry—When the system is under temporary faults and network fault, the experiment shows that the temporal redundancy helps to improve the reliability of the system. For the Web service with retry (Exp 2), the number of failures is reduced to 0.04% where these failures are due to the original faults in the program. This shows that the temporal redundancy with retry approach can significantly improve the reliability of the Web service. When a fault occurs in the Web service, on average, the clients need to retry twice to get the response from the Web service. However, the response time of the Web service is increased. Also, when there is a permanent fault, this scheme cannot reduce the number of failures in the system.

Single server with reboot—Another temporal redundancy approach is Web service with reboot (Exp 3). Our results show that the failure rate of the system is reduced when there is a permanent fault, in which case the server will try to reboot. Once the server finishes rebooting, it can provide the service again. The resulting failure rate is reduced from 88.5% to 10.6%. For temporary faults, the improvement is not as substantial as that of the temporal redundancy with retry. This is due to the fact that, when the Web service cannot be provided, the server will take time to reboot.

Single server with retry and reboot—With retry and reboot, the failure rate of both temporary and permanent cases are reduced. This approach enjoys the advantages of both algorithms. For temporary faults, the number of failures is reduced to zero. For permanent faults, the number of failures is significantly reduced from 88.5% to 1%; however, the response time is also greatly increased.

Table 6. Experimental results

Experiment ID (number of failures / response time(s))	1	2	3	4	5 (RR)	6 (RR)
Normal case	5/186	3/192	2/190	3/187	4/188	2/195
Temporary	1025/190	4/223	1106/231	4/238	1044/187	3/233
Permanent	8945/3000	8847/3000	1064/3000	5/1978	5637/3000	5532/3000
Byzantine fault	315/188	322/208	314/186	326/205	152/189	5/219
Network fault	223/187	2/227	239/193	3/231	237/193	3/213
Average	2102/730	1835/770	541/220	68/568	1415/751	1109/772
Experiment ID (number of failures / response time(s))	7 (RR)	8 (RR)	5 (N-version)	6 (N-version)	7 (N-version)	8 (N-version)
Normal case	3/193	2/190	0/189	0/190	0/188	0/188
Temporary	1057/188	2/231	0/190	0/190	0/189	0/187
Permanent	213/187	3/191	3125/191	3418/192	197/189	0/191
Byzantine fault	187/192	3/194	0/190	0/191	0/190	0/188
Network fault	206/197	2/192	0/190	0/192	0/188	0/187
Average	333/191	2/199	925/190	851/191	40/189	0/188

Spatial replication with Round-robin—With the spatial replication approach in Exp 5 (RR), the failure rate in the permanent fault is reduced from 88.5% to 56.4%. The failure rate is reduced because there are more servers in the system. When a server fails, the replication manager will update the availability list and forward the requests to other servers. However, when all the servers fail, the system will not be able to handle the requests from the clients. For the Byzantine failure, the failure rate is also reduced. This is because, with the Round-robin algorithm, different servers are employed for different requests; therefore, the failure rate is reduced.

Spatial replication with N-version programming—With the spatial replication approach in the N-version implementation of Exp 5, the failure rate of the Web service is greatly reduced. When a fault occurs in a Web service, other Web services are still operating, from which the majority result will be selected and returned to the client. Thus, the fault of a Web service will be tolerated in the system. When permanent faults occur, the failure rate is reduced from 88.5% to 31.4% with this scheme. For Byzantine and network faults, the N-version approach can even reduce the failure rate to zero in our experiment. It is noted that in the N-version approach, the failure rate is much lower than that of the Round-robin approach. This shows the majority results are normally more reliable than the results produced by an individual version.

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

Spatial replication, retry or reboot with N-version—In Exp 6 and 7, hybrid approaches with retry (Exp 6) or reboot (Exp 7) are conducted. We find that the failure rate is reduced to zero.

Spatial replication with Round-robin / N-version, retry and reboot—After performing the above experiments, we propose a hybrid approach for improving the reliability of Web services, including spatial redundancy, retry and reboot. The reliability of the system is improved most significantly by this approach: The failure rate of the system is reduced from 88.5% to 0 and the average response time is shortened. The replication manager keeps checking the availability of the Web services. When there is a server fault, other servers are responsible for handling the requests. At the same time, the failed server will reboot. Thus, the response time for handling the requests is greatly reduced. In Exp 8, it is demonstrated that this approach results in the lowest failure rate. This indicates that combining spatial and temporal redundancy in a hybrid approach achieves the highest gain in reliability improvement of the Web service.

Table 7. Model parameters

ID	Description	Value
λ_N	Network fault rate	0.045
λ^*	Web service fault rate	0.028
λ_1	Temporary fault rate	0.01
λ_2	Permanent fault rate	0.001
μ^*	Web service repair rate	0.523
μ_1	Temporary fault repair rate	0.954
μ_2	Permanent fault repair rate	0.954
C_1	Probability that the RM response is on time	0.978
C_2	Probability that the server reboots successfully	0.978

6. Reliability Modeling

We develop a reliability model of the proposed Web service paradigm using Markov chains [33]. The model is shown in Figure 10. The reliability model is analyzed and verified using the SHARPE tool [34]. The Markov chains model is developed to analyze the system reliability.

In Figure 10(a), the state s represents the normal execution state of the system with n Web service replicas. In the event of a fault causing the primary Web service to fail, the system will either go into the other states (i.e., $s - j$, which repre-

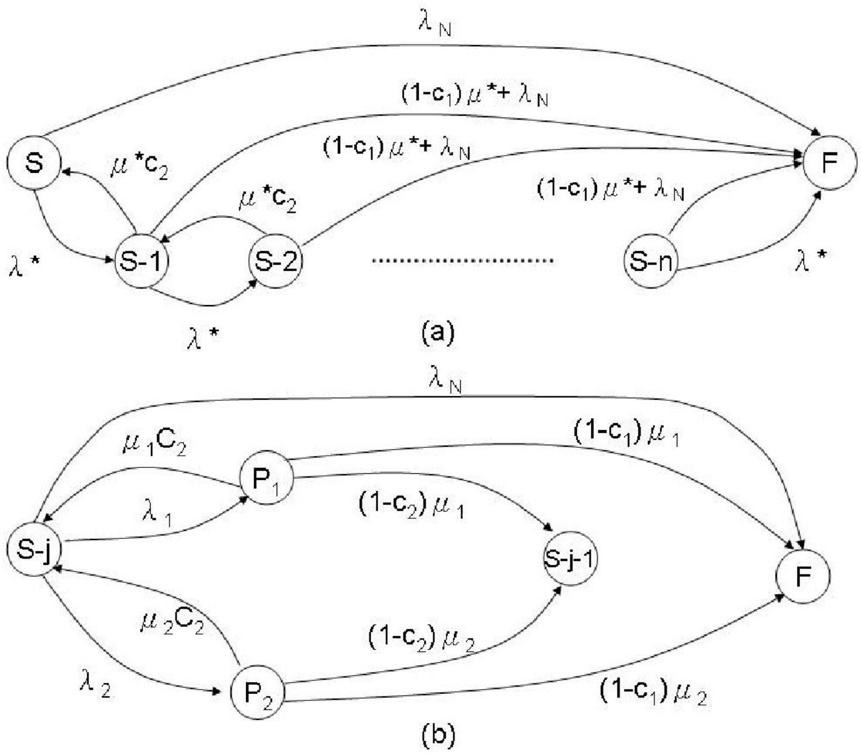


Figure 10. Markov chain based reliability model for the proposed system

sents the system with $n - j$ working replicas remaining, if the replication manager responds on time), or it will go to the failure state F with conditional probability $(1 - C_1)$. λ^* denotes the failure rate, i.e., the rate of occurrence of failures from which recovery cannot be completed, and C_1 represents the probability that the replication manager responds in time to switch to another Web service.

When the failed Web service is repaired, the system will go back to the previous state, $s - j + 1$. μ^* denotes the rate at which successful recovery is performed in this state, and C_2 represents the probability that the failed Web server reboots successfully. λ_n represents the network fault rate.

States $(s-1)$ to $(s-n)$ in Figure 10(a) represent the working states of the n Web service replicas and the reliability model of each Web service is shown in Figure 10(b). There are two types of faults simulated in our experiments: P_1 denotes a temporary fault and P_2 denotes a permanent fault. If a fault occurs in the Web service, either the Web service can be repaired with μ_1 (to enter P_1) or μ_2 (to enter P_2) repair rates with conditional probability C_1. If the fault cannot be recovered, the system goes to the next state $(s - j - 1)$ with one less Web service replica available. If the replication manager cannot respond in time, it will go to the failure state. From the graph, two formulae can be obtained:

$$\lambda^* = \lambda_1 \times (1 - C_1)\mu_1 + \lambda_2 \times (1 - C_2)\mu_2 \qquad (1)$$

$$\mu^* = \lambda_1 \times \mu_1 + \lambda_2 \times \mu_2 \qquad (2)$$

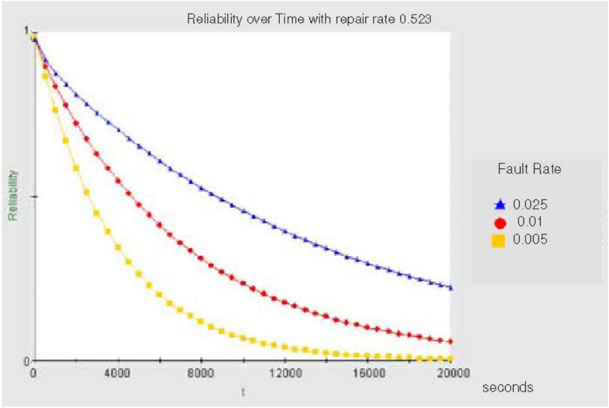


Figure 11. Reliability with different fault rates and repair rates

Based on the experiments described in Section 3.3, we obtain the fault rates and the repair rates of various components in the system; the results are shown in Table 7. The reliability of the system over time is further calculated with the tool SHARPE. Figure 11 shows the reliability over time at different fault rates λ^*, where the repair rate is (set at) 0.523 faults/s. Note that the fault rate obtained from the experiments is 0.03 failure/s. This failure rate is measured under an accelerated testing environment.

7. Conclusions

In the paper, we survey and address applicability of replication and design diversity techniques for reliable Web services and propose a hybrid approach to improve the availability of Web services. Furthermore, we carry out a series of experiments to evaluate the availability, performance and reliability of the proposed Web service system. From the experiments, we conclude that both temporal redundancy and spatial redundancy are important to the reliability improvement of the Web service. Modeling techniques by Petri-Net and Markov chain provide further insights of Web service system reliability with the proposed fault tolerant mechanisms.

Acknowledgments

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

References

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

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

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

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

[5] D. Liang, C. Fang, and S. Yuan. A fault-tolerant object service on corba. *Journal of Systems and Software*, 48:197–211, 1999.

[6] M. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan. Thema: Byzantine-fault-tolerant middleware for web-service application. In *Proc. of IEEE Symposium on Reliable Distributed Systems*, Orlando, FL, Oct 2005.

[7] A. Erradi and P. Maheshwari. A broker-based approach for improving web services reliability. In *Proc. of IEEE International Conference on Web Services*, volume 1, pages 355–362, 11-15 Jul 2005.

[8] W. Tsai, Z. Cao, Y. Chen, and R. Paul. Web services-based collaborative and cooperative computing. In *Proc. of Autonomous Decentralized Systems*, pages 552–556, 4-8 Apr 2005.

[9] C. Zimmermann. The Impact of Mechanical Defects on the Reliability of Solar Cells in Aerospace Applications . *IEEE Transactions on Device and Materials Reliability*, 6(3):486–494, Sept 2006.

[10] H. Celebi, M. Sahin, H. Arslan, J. Haque, E. Prado, and D. Markell. Ultrawideband design challenges for wireless chip-to-chip communications and interconnects. In *Proc. of IEEE Aerospace Conference 2006*, 4-11 Mar 2006.

[11] http://www.relex.com/products/fracas.asp

[12] B. Kim. Reliability analysis of real-time controllers with dual-modular temporal redundancy. In *Proc. of the Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pages 364–371, 13-15 Dec 1999.

[13] K. Shen and M. Xie. On the increase of system reliability by parallel redundancy. *IEEE Transactions on Reliability*, 39(5):607–611, Dec 1990.

[14] 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.

[15] M. R. Lyu, editor. *Software Fault Tolerance*. John Wiley and Sons Inc, Apr 1995.

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

[17] A. Avizienis and J. Kelly. Fault tolerance by design diversity: Concepts and experiments. *IEEE Transactions on Computer*, pages 67–80, Aug 1984.

[18] B. Randell. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, 1(2):220–232, 1975.

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

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

[21] P. Chan, M. Lyu, and M. Malek. Reliable web services: Methodology, experiment and modeling. In *Proc. of IEEE International Conference on Web Services*, Salt Lake City, Utah, USA, 9-13 Jul 2007.

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

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

[24] http://mars.jpl.nasa.gov/msl/mission/sc_edl.html

[25] A. Arkin, S. Askary, S. Fordin, W. Jekeli, and et. al. *Web Service Choreography Interface (WSCI) 1.0.* W3C, http://www.w3.org/TR/wsci/, 2002.

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

[27] A. Alves and et. al. Web services business process execution language version 2.0. In *http://www.oasis-open.org/committees/documents.php*, 2006.

[28] N. Looker, M. Munro, and J. Xu. A comparison of network level fault injection with code insertion. In *Proc. of the 29th Annual International Computer Software and Applications Conference 2005*, volume 1, pages 479–484, 26-28 Jul 2005.

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

[30] Y. Yan, Y. Liang, and X. Du. Controlling remote instruments using web services for online experiment systems. In *Proc. of IEEE International Conference on Web Services (ICWS) 2005*, 11-15 Jul 2005.

[31] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[32] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

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

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

BIOGRAPHY

Pat Pik Wah Chan received the B.E. degree in Computer Engineering, in 2002, and the M.Phil. degree in Computer Science and Engineering, in 2004, from The Chinese University of Hong Kong, Shatin, Hong Kong. She currently is the Ph.D student in the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong. Her research interests include reliability, Web services related research, multimedia security, digital watermarking, video processing and augmented reality.

Michael R. Lyu received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, China, 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 University of California, Los Angeles, in 1988.

He is currently a Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong. 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, The University of Iowa, Iowa City, as an Assistant Professor. From 1992 to 1995, he was a Member of the Technical Staff in the applied research area of Bell Communications Research (Bellcore), Morristown, New Jersey. From 1995 to 1997, he was a Research Member of the Technical Staff at Bell Laboratories, Murray Hill, New Jersey. His research interests include software reliability engineering, distributed systems, fault-tolerant computing, wireless communication networks, Web technologies, digital libraries, and E-commerce systems. He has published over 150 refereed journal and conference papers in these areas. He received Best Paper Awards in ISSRE98 and ISSRE 2001. 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 (Piscataway, NJ: IEEE and New York: McGraw-Hill, 1996).

Dr. Lyu initiated the First International Symposium on Software Reliability Engineering (ISSRE) in 1990. He was the General Chair for ISSRE2001, and the WWW10 Program Co- Chair. He has been frequently invited as a keynote or tutorial speaker to conferences and workshops in U.S., Europe, and Asia. He was an Associate Editor of IEEE Transactions on Reliability, IEEE Transactions on Knowledge and Data Engineering, and Journal of Information Science and Engineering. Dr. Lyu is a Fellow of IEEE and a Fellow of AAAS.