Fault Tolerance SOAP using TMR Scheme

Wichan Phetmanee and Kajornsak Pongthana
Faculty of Engineering, Rajamangala University of Technology Srivijaya Songkhla, Thailand
1 Rajadamnoen-Nok Road, Muang Songkhla 90000, Thailand
Tel: +667-4317-196, Fax +667-4317-195
E-mail: wichan.rmutsv@gmail.com and pkajornsak@yahoo.com
Keywords: SOAP, TMR, Information Redundancy

Abstract: A Web services is a software system designed to support interoperable Machine to Machine interaction over a network. Specifically, web services use a common communication based on Simple Object Access Protocol (SOAP) that exchanging Extensible Markup Language (XML) based messages over the world wide with high degree of communication fault and less reliability such as internet. This paper presents the SOAP fault tolerance communication using Triple Modular Redundancy (TMR) and Information Redundancy technique. The results of simulation and the results of experimental show that high level of fault-tolerance is achievable while some of the experimental results show only costing a reasonable time overhead.

1. Introduction

Web applications run large-scale software applications for e-commerce, entertainment, and numerous other activities [1]. They run on distributed hardware platforms and heterogeneous computer systems. The addition of Web services provide a common communications infrastructure based on Simple Object Access Protocol (SOAP) [2] and transfers data in Extensible Markup Language (XML) [3] that is supported by many web application vendors [4]. They are based on a distributed environment that does not require a specific underlying technology platform for development or deployment.

The type of faults that can affect web services may be categorized [5] as follows: - 1) Physical faults effecting memories, motherboards or processors 2) Software faults such as programming errors, algorithm errors and design/specification errors, 3) Resource-management faults in forms of memory leakage and inadvertent exhaustion of resource, 4) Communication faults or Network level fault like message deletion, duplication, reordering or corruption (whilst in traditional distributed systems this class of errors is widely assumed to not have a large effect on middleware, since this is usually built on a reliable transport over a LAN, web services runs over WANs that may be more unreliable that LAN based systems, especially in message corruption and message delivery times [6]) and 5) Life-cycle faults where premature object destruction and delayed asynchronous responses. More problematic is an event that corrupts data leaving web services, which may be considered as a network communication fault. It was showed that corruption within network packets between 1 in 1,100 packets and 1 in 32,000 fails the TCP checksum, even on links where link-level CRCs should catch all but 1 in 4 billion errors [7]. It is insufficient to rely only on available mechanisms like CRCs and TCP checksum to ensure data integrity in mission-critical web services. Moreover, when corrupted data is passed from one web services to another this should be handled by present mechanisms (i.e., TCP). Unfortunately, no current technology is used to prevent corrupted data from passing all the way up to an application such as another web services. But for an extremely high level of reliability in mission critical applications, hence it is important for this class of systems to have a capability to tolerate the communication faults that may occur, though hardly the chance is. So this paper describes our novel fault-tolerant scheme for communication of web services based on an extension of a standard SOAP message via an algorithm based on information redundancy and TMR (Triple Modular Redundancy) voting scheme.

2. Fault injection design

System dependability (of which reliability is a part) can be assessed using either model or measurement techniques [5]. Each of these techniques has its own advantages. The *modeling* can be used during the design phase to predict probability faults and errors in used algorithms, but only predictions can be made of system dependability. *Measurement* can be applied to existing systems to provide metrics on dependability. Measurement technique is useful because they can be applied to existing systems without requiring access to source code or design documentation. And there are two main measurement techniques:

1) Observation. Observation of failures in a large set of deployed systems can be performed. Existing record logs are used and analysis of this data can obtain information on the frequency of failures and the activity that was in progress when they occurred. Unfortunately, failures may occur infrequently, data must be collected over a long period of time from many systems or even only one system, which poses a huge drawback.

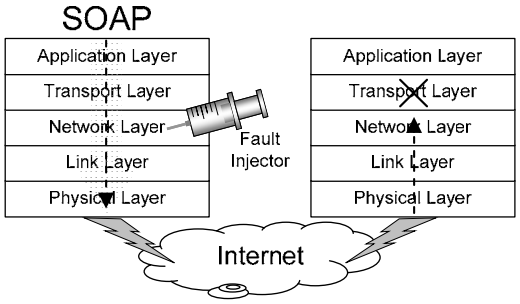

Fig.1. Network level fault injection.

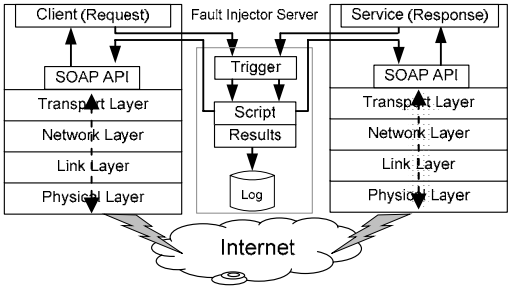

Fig.2. Fault injection framework.

2) Fault Injection. As it may take a very long time for some errors to occur, then fault injection attempts to more speed up this process by injecting faults into a running system. A failure or error condition will be generated and observed or the system fault tolerance mechanism will handle the error.

These methods can further be broken-down into Compile-Time injection (code mutation)which is an injection technique where source code is modified to inject simulated faults into a system. Although universal and accurate, the problem with this technique is the need to modify actual source code. Another technique, Runtime Injection, uses a software trigger to inject a fault into a running system.

For our purpose, the technique we employ to determine system dependability is in terms of a network level fault injection, which is a runtime technique concerned with the corruption, loss or reordering of network packets at the network interface. It is possible to directly inject fault in network packet but runs the risk of being detected and rejected by the receiving systems protocol stack (see Fig. 1). It is therefore preferable to inject the fault at the application level before this stage [5]. The faults injected are based on corrupting information and injecting random byte errors.

The existing idea on injecting network level faults allows us to create a framework to simulate faults injected at the API level via modifying SOAP message (see Fig. 2).

3. Fault tolerance SOAP

We modified a conventional SOAP message into a Fault-Tolerance SOAP message in the outgoing (both client and server/output-side) SOAP extension engine conforming to an information redundancy scheme (Fault Tolerance SOAP = (Original SOAP + Checksum) × 3) before sending it out over the Internet. When Fault Tolerance SOAP message arrives at the destination, the extension engine on the receiver (both client and server/input side) extracts Fault Tolerance SOAP to Original SOAP + Checksum according to an algorithm based on a TMR concept and finally rebuild to regular SOAP message.

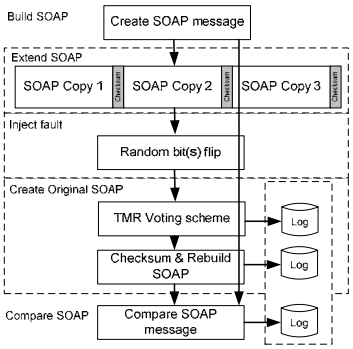

Fig.3. Simulation process.

Furthermore we modified TMR voting that extracts Fault Tolerance SOAP to Original SOAP plus checksum in order to faster and more accurate. Normally our TMR voting only focuses on character by character for speed performance reason. In case of character by character comparison fails, our TMR voting will focuses in bit by bit comparison for accuracy performance reason. And then the TMR voting return to character by character comparison and go on.

4. Simulation results

Network level fault injection is usually based upon more or less random corruption of bytes within network packets [8].

Our method according to network level fault injection and shown in 3. The simulation process is 1) the simulator begin with create standard SOAP massage. 2) the simulator attaches checksum and triplicate to fault tolerance SOAP. 3) the simulator injects random bit(s) fault to fault tolerance SOAP massage that we vary form 0 to 10 faults in each 100,000 rounds. 4) the simulator creates a SOAP massage and check sum for corruptions reason. 5) the simulator compare new SOAP with original SOAP.

The result of simulation is shown in table 1 with: 1) the percent of TMR fail (cannot compare and rebuild massage) 2) the percent of checksum fail. 3) the percent of SOAP massage fails (compare new SOAP with original SOAP).

According to the table that fault tolerance SOAP utilizing perfectly handles a 1-bit fault. The incomplete of SOAP are still 0% even we increase fault to 10 bits whereas our process can detected error of checksum and terminated that SOAP fail before SOAP with error get out form fault tolerance SOAP engine.

Table 1
Simulation results.

Bit(s) fault	1	2	3	4	5	6	7	8	9	10
TMR fail	0	0	0	0	0	0	0	0	0	0
Checksum fail	0	0.017	0.040	0.092	0.150	0.197	0.284	0.389	0.503	0.608
SOAP fail	0	0	0	0	0	0	0	0	0	0

5. Experimental results

We design SOAP engine in order to determine whether a standard SOAP message should or should not be extended. This is so that programmer, developer or user can be flexible in choosing to use fault tolerance SOAP for their web services or regular SOAP. So we insert code of SOAP engine in SOAP library for this reason.

To illustrate the level of fault-tolerance of our proposed framework, the experimental environment was created to observe the behavior of the system to the injection of network level fault. Namely, a simple service that receives a string and returns the same string back was created and deployed onto web service server. Web services (client side) was created and deployed onto the client machine (both client and server already with fault tolerance SOAP engine implemented into the SOAP library).

The fault injector was designed so that experiments can be easily distributed on multiple machines to reduce loading on a particular machine and also to allow truly distributed systems made up of a number of servers hosting Web services.

There was an issue that should be explained before going further. Specifically, it is improper to directly inject faults into network packets due to problems in altering encrypted/signed packets or problems regarding any mechanism to preserve information robustness after they have been constructed. The inject faults effectively would be discarded by the receiving transport layer and would consequently not relay the fault to the desired destination.

In other hand, we simulate the mentioned network level fault injection by injecting the faults at the API boundary between the application and the top of the protocol stack, this being the lowest, easily accessible point to inject faults before any encryption and signing or encoding has taken place. The injector was written as a separate process to the software under test. A small amount of hook software was written in the native language of the host system under test to allow communication between the two entities (see Fig. 2). And communication will be facilitated by a TCP/IP socket connection, with the fault injector acting as a server to allow multi-node experiments to be commissioned if desired.

Experiment 1: This experiment was intended to check the fault tolerance degree of a web services under faults situation. For each fault, 10,000 rounds of send-receive sequences of

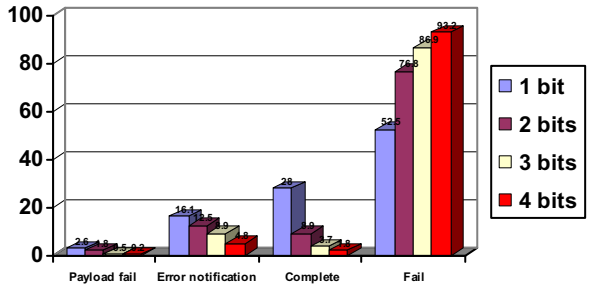

Fig.4. SOAP responses.

Fig.5. Fault tolerance SOAP responses.

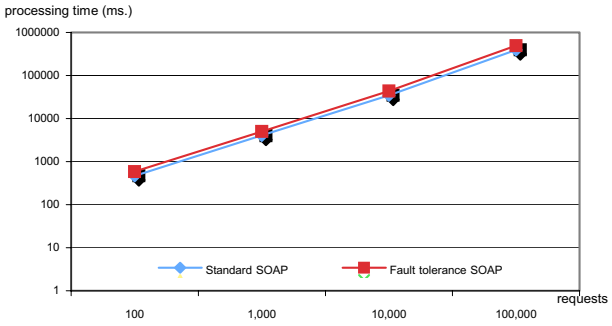

Fig.6. Processing time (ms).

SOAP message with injected random bit-flip fault (1- 4 bits) are executed before the statistics is collected.

Fig.4. shows the SOAP response and Fig.5. shows the fault tolerance SOAP response in the case of a message having bit(s) fault. The result is divided as

1) **Payload fails** (response but incorrect message)
2) **Error notification** (response but with error notification)
3) Complete (successful response with correct message)
4) **Fail** (no SOAP response)

According to the figure it is easy to see that web services utilizing the fault tolerance SOAP perfectly handles a 1-bit fault whereas regular web service reacts poorly, with under 30% of messages correctly returned. The same trend can be seen in the case of rising bits flip. The standard web services perform poorer to the point of non-existent complete

response. Our fault tolerant SOAP still provides a high degree of reliability, with only a error notification response whenever a complete response is not possible.

Experiment 2: This experiment was intended to investigate the time overhead due to a fault tolerance mechanism built into a web service. Fig.6 shows processing (in milliseconds) versus amount of requests for both a regular SOAP and a fault tolerant SOAP. The plot of processing time indicates that up to only 25% extra of processing time.

6. Conclusion

The main intend of this work to create fault tolerance SOAP by developing and proving a feasibility of the information redundancy and TMR scheme. Network-level fault injection is used as a measure to assess the dependability of the system. Simulation results and experimental results show that high level of fault-tolerance SOAP for web services is achievable while only costing a reasonable time overhead.

References

[1] Jeff Offutt, "Quality Attributes of Web Software Applications", IEEE Software, Vol.19, No.2, pp.25–32, 2002.

[2] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: An introduction to soap, wsdl, and uddi", IEEE Internet Computing, Vol.06(2), pp. 86–93, 2002.

[3] T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds), "Extensible Markup Language (XML) 1.0 (2nd Edition)," W3C Recommendation, 2000.

[4] J. Offutt and W. Xu, "Generating test cases for web services using data perturbation", SIGSOFT Softw. Eng. Notes, Vol.29(5), pp 1–10, 2004.

[5] E. Marsden, J. Fabre, and J. Arlat, "Dependability of corba systems: Service characterization by fault injection", presented at Symposium on Reliable Distributed Systems, Osaka, Japan 2002.

[6] N. Looker and J. Xu, "Assessing the dependability of soap rpc-based web services by fault injection", words, Vol.00, pp.163–170, 2003.

[7] J. Stone and C. Partridge, "When the CRC and TCP checksum disagree", SIGCOMM, pp.309–319, 2000.

[8] N. Looker, M. Munro, and J. Xu, "Assessing Web Services Quality of Service with Fault Injection", presented at Workshop on Quality of Service for Application Servers, Symposium on Reliable Distributed Systems, SRDS, Brazil, 2004.