

Testing Web Services

Nik Looker, Malcolm Munro, Jie Xu

Department of Computer Science,
University of Durham, UK

n.e.looker@durham.ac.uk, malcolm.munro@durham.ac.uk, jie.xu@durham.ac.uk

This paper describes the current state of the art in testing techniques that can be applied to web services. It reviews the challenges encountered when testing web services in a distributed environment. It then goes on to detail test methods and tools that can be applied to the problem. Finally it outlines a promising line of research that can be applied to this problem.

Introduction

Web Services form the basis, not only of standard web based eCommerce applications but also the new Globus implementation of Open GRID Systems Architecture (OGSA). Given the prominence of this technology, test methods and tools are required to ensure that robust, fault tolerant software services are deployed. Testing is required not only to uncover existing problems with the system but to also provide potential users with metrics to compare similar serviced based solutions.

Fault injection is a well-proven method of assessing the dependability of a system. Although much work has been done in the area of fault injection and distributed systems in general, there appears to have been little research carried out on applying this to web service based systems [1, 2].

Previous research in the field of service testing via fault injection has concentrated on tightly coupled, RPC based distributed systems [1]. In defining a testing method for web services new sets of challenges are faced which require different solutions. Key differences that are encountered when testing web services are: (1) greater chance for network failure, (2) higher levels of security and encryption, (3) more generic nature of the platform and the need to support multiple programming languages, (4) timing constrains and the asynchronous nature of web service operations. This is due to the loosely coupled nature of typical SOAP based systems that implement services.

Dependability Assessment

System dependability can be assessed using either model or measurement techniques [1]. Both techniques have their merits. Modeling can be used in the design stage to predict potential errors and faults in algorithms. Measurement can be applied

to existing systems to provide metrics on dependability. Modeling can only make predictions of the dependability of a system since it is derived from system design, specification. Once a system has been implemented, actual measurement techniques can be applied to obtain specific metrics and allow data on dependability to be derived from them. Measurement techniques are useful because they can be applied to existing systems without requiring access to source code or design documentation. There are two main measurement techniques:

Observation: measurements can be performed by the observation of errors and failures in a large set of deployed systems. This technique uses existing logs. Analysis of the data can obtain information on the frequency of faults and the activity that was in progress when they occurred. Since failures and errors may occur infrequently data must be collected over a long period of time and from a large number of systems. It is unlikely that this technique will catch rarely seen errors [3] since they may take a very long time to occur (in the order of years).

Fault Injection: since it may take a very long time for some errors to occur, fault injection attempts to speed up this process by injecting faults into a running system. There are a number of different methods of fault injection but all methods attempt to cause the execution of seldom used control pathways within a system. By doing this either an error condition will be generated or the systems fault tolerance mechanism will handle the fault.

Software Fault Injection Methods

There has been more research in developing Software Implemented Fault Injection (SWIFI) based tools. This is partly because a SWIFI tool does not require any expensive hardware. SWIFI also allow specific systems running on target hardware to be effectively targeted without injecting faults into other parts of the system.

SWIFI does have some drawbacks: (1) Faults cannot be injected into memory that is protected by the operating system, (2) Instrumentation of the code to be tested may perturb the operation of the system, (3) Timing of events may be inaccurate because the timers available to a software system on some hardware platforms may not have a high enough resolution to capture short latency faults.

SWIFI techniques can be categorized into two types. *Compile-Time Injection* and *Runtime Injection*.

Compile-Time Injection

Compile-Time Injection (sometimes know as code mutation) is an injection technique where source code (or assembler code) is modified to inject simulated faults into a system. A simple example of this technique could be changing $a = a + 1$ to $a = a - 1$.

This technique has the advantage that it can be used to simulate both hardware and software faults. It has been shown to induce faults into a system that are very close in nature to those produced by programming faults [4]. Since the faults are coded into

the executables it is possible to emulate permanent faults as well as transient faults. Finally this system is very simple to implement.

The main drawbacks of this technique are that it requires the modification of the actual source code and therefore the source code must be available to the test team (which may not be the case for COTS systems for example). Also since the code is being modified there is the chance that unintended faults will be introduced. Lastly since the source code is being altered this technique cannot be used as part of a certification processes since the system under test will be a different system to that which is shipped.

Runtime Injection

Runtime Injection techniques use a software trigger to inject a fault into a running system. Faults can be injected via a number of techniques. Triggers can be implemented in a number of ways:

Time based triggers: these types of triggers use either hardware or software timers. When the timer reaches a specified time an interrupt is generated and the interrupt handler associated with the timer can inject the fault. Since this trigger method cannot be tied with any accuracy to specific operations it produces unpredictable effects in a system. Its main use is to simulate transient and intermittent faults within a system.

Interrupt based triggers: these types of triggers use hardware exceptions and the software trap mechanism to generate an interrupt at a specific place in the system code or on a particular event within the system, e.g. access to a specific memory location. This method of trigger implementation is capable of injecting a fault on a specific event and has the advantages that it requires no modification to the system code. Its main disadvantage is that on systems that support a process protection model the fault injector may be required to run as a system level process.

Code Insertion: this technique involves inserting code into the target system source just before an event is to occur. This code performs the fault injection and then the original statement can execute with the fault present in the system. This method differs from compile-time injection in that it injects its faults at runtime rather than at compile time. Rather than corrupt existing code, it adds code to perform the fault injection. Its main disadvantage is that it requires the system source code to be modified but it has the advantage that the fault injector can be compiled into the system as a library and run as part of the system in user mode on systems that support this process protection model.

A promising technique that has been applied to RPC based systems is network level fault infection [2, 8]. This injects faults by manipulating network packets.

Fault Injection Tools

A number of SWIFI Tools have been developed and a brief review of a selection of these tools is given here.

Ferrari (Fault and Error Automatic Real-Time Injection) uses a system based around software traps to inject errors into a system. The traps are activated by either a call to a specific memory location or a timeout. When a trap is called the handler injects a fault into the system. The fault can either be transient or permanent. Research conducted with Ferrari shows that error detection is dependant on the fault type and where the fault is inserted [5].

FTAPE (Fault Tolerance and Performance Evaluator) can inject faults not only into memory and registers but into disk accesses as well. This is achieved by inserting a special disk driver into the system that can inject faults into data sent and received from the disk unit. FTAPE also has a synthetic load unit that can simulate specific amounts of load for robustness testing purposes.

Doctor (Integrated Software Fault Injection Environment) allows injection of memory and register faults, as well as network communication faults. It uses a combination of time-out, trap and code modification. Time-out triggers are used to inject transient memory faults. Traps are used to inject transient emulated hardware failures, such as register corruption. Code modification is used to inject permanent faults.

Xception [6] is designed to take advantage of the advanced debugging features available on many modern processors. It is written to require no modification of system source and no insertion of software traps. Xception uses the processors exception handling capabilities to trigger fault injection. These are based around accesses to specific memory locations. Such accesses could be either for data or fetching instructions. It is therefore possible to accurately reproduce test runs because triggers can be tied to specific events, instead to timeouts. Research comparing it to static analysis methods has shown that it fails to detect some classes of faults (namely those contained in infrequently executed pieces of code) [7] but it did prove to be effective at detecting faults in frequently executed code.

Future Direction for Service Testing

There appears to have been little research with regard to the application of fault injection techniques to services products [1, 2, 8]. Most dependability assessment of services appears to have been conducted using observational measurement techniques [9]. Some research has been conducted using fault injection to test the dependability of CORBA implementations using network level fault injection [1] with successful results.

Research shows that Compile-Time Injection SWIFI tools prove to be effective at emulating fault conditions but have the disadvantage of requiring that source code be modified. Most runtime-implemented SWIFI tools are implemented using either exceptions or traps, which allow specific events to be triggered on with a good level of repeatability although research has shown that fault injection tools, which rely on the execution of certain pieces of code, can prove ineffective at detecting some classes of failure. Most SWIFI tools inject simulated hardware faults, such as corrupted memory, into a running process, but some tools can inject network level faults.

Network Level Fault Injection has been shown to give good results with testing network protocol stacks. Injected faults are traditionally concerned with corrupting packet header information or random corruption within the packet payload.

Promising work has been carried out on applying network level fault injection to SOAP RPC based systems [2, 8]. WS-FIT [2] (Web Service - Fault Injection Technology) is designed to integrate into SOAP based web services. It injects meaningful faults into specific SOAP messages via a script based trigger system. By injecting meaningful faults into RPC SOAP messages WS-FIT can simulate API level faults without the need for modifying code.

WS-FIT uses WSDL to obtain a detailed definition of a SOAP-RPC so that parameters can be manipulated via network-level fault injection. It then uses a combination of techniques commonly applied to the domain of compiler grammar parsing to enable the web service source code to be parsed into an intermediate meta-data. This meta-data can then be used to construct test campaigns by combining abstract syntax trees, with reference to internal state, to determine potential error conditions. These potential error conditions are assessed by injection into the finite state machine contained in the service under test. This is achieved by using WS-FIT to manipulate input parameters into the service and thus provoke error conditions.

References

- [1] 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.
- [2] N. Looker and J. Xu, "Assessing the Dependability of SOAP-RPC-Based Web Services by Fault Injection," *9th IEEE International Workshop on Object-oriented Real-time Dependable Systems*, pp. 163-170, 2003.
- [3] H. Hecht and M. Hecht, "Qualitative Interpretation of Software Test Data," presented at Computer-aided design, test, and evaluation for dependability, Beijing, 1996.
- [4] M. Daran and P. Thevenod-Fosse, "Software Error Analysis: A Real Case Study Involving Real Faults and Mutations," *Software Engineering Notes*, vol. 21, pp. 158-171, 1996.
- [5] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, "FERRARI: A tool for validation of system dependability properties," presented at International Symposium on Fault-Tolerant Computing (FTCS'92), Boston, MA, 1992.
- [6] J. Carreira, H. Madeira, and J. G. Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers," *Ieee Transactions on Software Engineering Se*, vol. 24, pp. 125-136, 1998.
- [7] H. Madeira, D. Costa, and M. Vieira, "On the Emulation of Software Faults by Software Fault Injection," presented at Dependable systems and networks, New York, NY, 2000.
- [8] N. Looker and J. Xu, "Assessing the Dependability of OGSA Middleware by Fault Injection," *Proceedings of the Symposium on Reliable Distributed Systems*, pp. 293-302, 2003.
- [9] M. Kalyanakrishnam, Z. Kalbarczyk, and R. Iyer, "Failure Data Analysis of a LAN of Windows NT based Computers," presented at Reliable distributed systems, Lausanne, Switzerland, 1999.