http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (1): 138-143, 2006
ISSN 1812-5638

Web Service Based Software Implemented Fault Injection

Mohammod Shamim Hossain
School of Science and Technology, Bangladesh Open University, Gazipur 1705, Bangladesh

Abstract: Toady's information society, offering fully automated dependable software for critical online and offline system is a challenging task. There is a great need for automated Software Implemented Fault Injection (SWIFI) tools to assist programmers and system designers with performance and dependability evaluation. This study presents a review of related research on Software Implemented Fault Injection. Based on some of the review, a web service based Software Implemented Fault Injection (SWIFI) framework is proposed. The proposed framework will support portability and leverage extended facilities of existing SWIFI tools.

Key words: Software implemented fault injection, web service, dependability, SOAP, UDDI

INTRODUCTION

The increasing demand for critical applications in aerospace, industry, business, defense, education and real time telecommunication, introduce a new challenge to the software industry in terms of dependability. Software Implemented Fault Injection (SWIFI) is a well proven technique for assessing dependability validation process: fault removal and fault forecasting. Other use of SWIFI includes assessing goodness of a design; finding defects in software; COTS validation or determining failure modes; safety verification; security assessment and software testability analysis. Recently, many emerging SWIFI tools such as FTAPE, xception, Ferrari etc. have been developed. Now attentions are paid on how to design and develop portable fully automated SWIFI tools for heterogeneous model, environment and fault types. If web-services technologies are considered in developing SWIFI systems, it is possible to fulfill the ambitious attention of portability. A lot of research have been done in the area of fault injection in general. Very little research has been done regarding testing web services or middleware[1,2]. There have been no research yet conducted on how to design and develop a web services based framework for SWIFI tools. This study will focus on a basic architecture of a web SWIFI.

A web service is a software application, accessible on the web through an URL that is accessed by clients using XML-based protocols, such as Simple Object Access Protocol (SOAP) sent over accepted Internet protocols, such as HTTP. Clients access a web service application through its interfaces and bindings, which are defined using XML artifacts, such as a Web Service Definition Language (WSDL) file[3]. Web services are built with a wide variety of platforms like NET, Sun J2EE and Apache. They are compatible with the protocol SOAP. How SWIFI task can be managed with this vast range of existing available web services is a great challenging task now. Therefore, a generic framework for Software Implemented Fault Injection using current web services technology is important.

SURVEY OF SOFTWARE IMPLEMENTED FAULT INJECTION

In Software Implemented Fault Injection (SWIFI), software is used to mimic hardware or software faults in a prototype. It basically consists of injecting faults into a system using a specific piece of software. In this approach, all locations in hardware and software that are accessible to machine instructions can be chosen by the user, as the point at which to inject errors/faults: errors corresponding to hardware faults are emulated through the implementation of incorrect instructions and access of incorrect data and errors corresponding to software faults (such as incorrect initialization of a variable, failure to check a boundary condition) are emulated by an appropriate code change.

SWIFI emulate transient, intermittent and permanent faults: transient faults are simple to emulate, since incorrect data/instructions can be corrected, as for example, faulty bits in memory or CPU registers can be overwritten by subsequent instructions; permanent and intermittent faults are emulated by repeatedly injecting the same fault for its duration, as for example, to emulate a permanent stuck-at-0 fault at a particular bit in a memory word, the bit is changed to 0 after every write operation to the word.

The target of fault injection can be the user application, the operating system, or both. In case of user

application, the fault injector is inserted into the user application or can be an extra layer of software between the user application and the operating system. On the other hand, for the operating system, the fault injector must be embedded in the operating system, because it is difficult to add a layer of software between the system hardware and the operating system.

SWIFI is complementary to hardware implemented injection. Because:

- Hardware-implemented fault injection provides better control of time accuracy during injection, but is more time-consuming than the SWIFI.
- Hardware-implemented fault injection is more precise in the specified faults that can be injected (especially for fault injection with contact), but the flexibility of SWIFI is greater.

Among the several techniques available to inject faults, SWIFI is commonly recognized as the best one. SWIFI tools can emulate a much larger set of hardware fault types such as faults in the address bus, arithmetic unit, memory and other functional units. Because, software has full control of the processor and memory functions, access to most of the hardware components and a capable of manipulating the information that it processes.

SWIFI approaches have several advantages compared to other approaches:

- It has lower complexity because no dedicated hardware or very detailed models are required
- It has increased portability
- It allows faults or errors to be injected at location and time under software control with no additional hardware support.
- It is less expensive in terms of time and effort, than hardware implemented fault injection.
- It can be used to target applications and operating systems, which is difficult to do with hardware one.

Though SWIFI is cost-effective, flexible and attractive, however, it has some shortcomings[4] which include:

- Faults cannot be injected to locations that are not accessible to software. Approximately one-third of the errors produced by logic-level fault injection cannot be emulated through SWIFI.
- The software instrumentation may perturb the system being studied, biasing fault injection results.
- The lower time resolution of SWIFI relative to

Hardware one may perturb the results obtained. A hybrid technique, using a hardware monitor, can be used to overcome time resolution problem. The hybrid technique combines the versatility of SWIFI with the accuracy obtained through hardware monitoring. It is well-suited for measuring extremely short latencies.

Categorization of software implemented fault injection: SWIFI can be categorized mainly in two ways based on the types of systems being studied: stand-alone systems and network or distributed systems. Stand-alone systems can further be divided in two ways: Compile-time and run-time[4].

Stand-alone systems: Stand-alone systems can be categorized on basis of when faults are injected: During Compile-time injection and run-time Injection: Compile-Time Injection is an injection technique where source code is modified to inject simulated faults into a system. Run-time injection techniques use a software trigger mechanism to inject a fault into a running system.

Network or distributed systems: Most concepts introduced for stand-alone systems are thus still valid for networks or distributed systems. Networks/distributed systems inject the same faults as those in stand-alone systems. It inject faults specifically related to communication in the network/distributed system. It coordinates the injection based on certain information about shared state of different hosts in the distributed system. To inject realistic faults in networks or distributed systems, one must have the ability to inject faults based on the state of the system. This knowledge about state can come from: the local portion of the application; information transmitted between portions of the application on different hosts and explicit information passed between nodes of the fault injector itself, to obtain system state information. However, there are two main problems: a) the injection of the fault in the right state and b) the verification that the fault was correctly injected. Because of the added complexity of injecting faults in distributed systems, fewer tools have been developed for networks/distributed systems than for stand-alone systems.

SOFTWARE IMPLEMENTED FAULT INJECTION (SWIFI) TOOLS

A large number SWIFI tools have been developed and going to develop by different organizations. Let's see some of these:

Tools for stand-alone systems: A review of some of theses tools for stand-alone systems are as follows:

JIFI[5] (JPL's Implementation of a Fault Injector) developed at Jet Propulsion Laboratory (JPL) of California Institute of Technology. JIFI is used with appropriate fault model in order to evaluate both software and hardware fault tolerance; to measure the effectiveness of fault detection, isolation and recovery strategies. JIFI is used to inject fault into user specified CPU registers and memory location. Memory fault locations are of two types: user area and operating systems kernel area. JIFI is an application level software implemented fault injection tools which allows fault injection campaigns and statistical data analysis along with verifiers, classifiers and run scripts. It can inject time triggered random faults and location triggered targeted faults.

FERRARI[6] (Fault and Error Automatic Real-Time Injection) provides software-implemented fault injection of standalone systems. This system was developed at the University of Texas at Austin. FERRARI consists of four modules: the initializer and activator, the user information specifier, the fault and error injector and the collector and analyzer. The four modules are controlled by the manager module, which coordinates the operation of the four modules.

The fault Tolerance and Performance Evaluator (FTAPE)[7] is a fault injector for stand-alone systems that integrates the injection of faults and the activity necessary to propagate the errors generated by the faults. This system was developed at the University of Illinois. A version of FTAPE intended for network systems, called NFTAPE which supports multiple platforms. The system activity is specified as a mixture of CPU, memory and/or I/O activity. Faults can be injected into the CPU, the memory and the I/O devices. The CPU fault models are: single/multiple bit-flip faults in CPU registers and value faults in CPU registers. The memory fault models are: single/multiple bit-flip faults in memory and value faults in memory. The I/O fault models are: Change the value in a register of the disk controller.

Xception[8] is a software fault injection and monitoring environment that uses counters and timers that exist in most modern processors to inject faults and to monitor the activation of the faults and their impact on the system. The system was first developed at the University of Coimbra in Portugal, which had the advantage of the advanced debugging and performance monitoring features present in many modern processors to inject more realistic faults. This approach allows injection of faults without modification of the target application; no software traps are inserted; injection of faults with minimum interference; definition of many fault triggers such as triggers related to the manipulation of data; monitoring of the activation of latent faults (such as faults introduced in a specific memory cell) by programming the hardware to cause an exception when the corrupted memory cell is addressed. Fault injection can be done in any process running on the target system, including the operating system; applications for which the source code is not available and the processor, memory and data/address buses. In order to trigger fault injection, it uses a processor's built-in hardware exception triggers. The fault injector is implemented as an exception handler and modification of the interrupt handler vector is required. Based on access to specific addresses, its faults are triggered, so the experiments are reproducible.

Tools for network or distributed systems: The followings are some of the review of the tools for network or distributed systems:

The DOCTOR[9] (integrated software fault injection environment) first developed at the University Michgan, can inject three types of fault: memory faults, CPU faults and communication faults. The user can select any combination of these three types to induce appropriate abnormal conditions. Memory faults can be injected as a single bit, two bit for compensating, whole byte or burst of multiple bytes error. The content of the memory at the selected address are partially or totally set, reset or toggled. CPU faults may occur in data registers, address registers, control registers, ALU and so on. The communication faults in DOCTOR can cause messages to be lost, changed, duplicated or delayed. Each three types of fault may be permanent, transient and intermittent. Fault inject plan can be a probabilistic formulation or based on the past event history. DOCTOR supports: generating synthetic workloads under which system dependability is evaluated; injecting various types of faults with different options and collecting performance and dependability data. It uses three triggering methods: time-out, trap and code insertion for triggering fault injection.

Orchestra[10] developed at the University of Michigan, is a fault injection tool for testing dependability and timing properties of distributed system protocol implementations. The objectives of orchestra are the portability to different platforms and the ability to insert

the fault injection probe into a protocol stack. This tool is based on a simple but powerful framework called script-driven probing and fault injections for the evaluation and validation of the fault tolerance and timing characteristics of distributed protocols. Another feature of this tool is to address the intrusiveness of fault injection on a target distributed systems. This system can employed in studying the three aspects of a target protocol by detecting design or implementations error; identifying violations of protocol specifications and obtaining insights into the design decisions of the implementers. The tool was initially developed on the Real-Time Mach operating system and later ported to other platforms such as Solaris and SunOS. This tool have used to conduct experiments on commercial and research systems including TCP communication protocol, a primary backup replication protocol, distributed group membership service, real time audio conferencing application and so on.

DEFINE[11] (Distributed Fault Injection and Monitoring Environment) developed at the University of Illinois to evaluate system dependability, investigate fault propagation and validate fault-tolerant mechanism. This tool can inject both hardware-induced software errors and software faults in any process running in distributed systems either in user mode or supervisor mode. These faults can be correlated or independent. DEFINE is extended from its antecedent FINE[12], with additional distributed capability and injection mechanism. DEFINE uses two fault injection techniques: (1) using hardware clock interrupts to control the time of fault injection and activation which allows injecting intermittent CPU/bus faults in order to ensure their activation (2) using software traps to inject faults and monitor fault activation in order to assist monitor whether the faults are activated and when they are activated. Experiments are conducted on SUN NFS-distributed file systems.

Loki[13] is a state driven fault Injector for Distributed Systems. Fault injection is performed based on the global state of a distributed system. It has the ability to check whether the faults are correctly injected. The concept of state is elementary to loki. The execution of a component of the distributed system under study can be regarded as a state machine. The global state of the system denotes the vector of the local states. It is enough to keep track partial view of the global state and its selection for the injection of the required faults. The global state may be either online or offline. An on-line partial global state is based on a) the partial global state is tracked at local components by the loki run-time, b) state changes can be caused by local events or remote events c) the on-line view of the global state is approximate and its accuracy must be determined after the fault injection campaign. An off-line determination of the partial global state correctness is based on a) a bound analysis to determine time uncertainty of each event and b) the selection of injections that occurred correctly with respect to global state. In loki, the distributed system under study is separated into basic units from which state information is collected and into which faults are injected. The basic unit along with the loki run time attached to it is called a node. The loki run-time only uses the necessary state change notifications between nodes in order to keep track of the partial view of the global state. It records state changes and fault injection occurrences. There could be an incorrect fault injections and incorrect measures. To overcome the problems, loki performs a post-runtime check on every fault injection.

The evaluation of any system using loki consists of the following steps[13]:

- An initial synchronization-message-passing phase,
- A fault injection and observation collection phase,
- A second synchronization-message-passing phase,
- Determination of experiments with properly injected faults and
- Computation of measures using these experiments.

PROPOSED FRAMEWORK FOR SWIFI

Inspite of all the research done for Software implemented fault injection; there are several limitations with the current existing tools. Some of these are follows:

- There is no SWIFI tool which allows more than two platforms except Orchestra. It supports two platforms mach and solaris,
- There is no SWIFI tool which allows more than one type of fault injection except FERRARI,
- There is a few SWIFI tools which allows more than two types of fault models,
- There is no SWIFI tool which allows more than one trigger method except FTAPE.

Because of these limitations, no tool is applicable if the target system has the following properties: multi-platform (heterogeneous) support; several fault model or injection methods; fault injection under different trigger conditions; using the same control and configuration process for each type of analysis. Only NFTAPE[14] solve some of the problems mentioned above. There is no SWIFI tool or system that uses web service. There appears very little research[1] in the field of testing web services by applying network level fault injection to

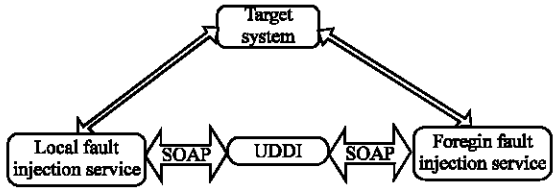

Fig. 1: Proposed web service based SWIFI system

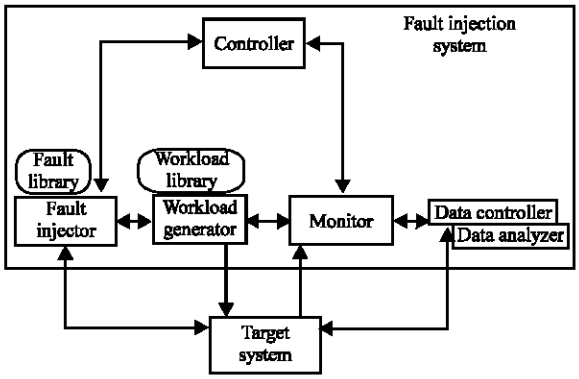

Fig. 2: Basic components of a fault injection environment[4]

SOAP RPC based system. However, there is no SWIFI tool using web service technology.

The proposed SWIFI system consists of local fault injection service, target systems and a foreign fault injection service. This framework may help to use most of the existing SWIFI tools such as xception, NFTAPE etc. using web services i.e. foreign service call them when necessary. Once the system receives a specific request, it compiles the response according to the user request. If a specific fault injection service is not available in the local system, a web services request will be made to a specific UDDI server requesting an appropriate injection service. If a service is found we make use of its capabilities, otherwise we try to find a bridge to solve the request. This request is done through SOAP message. Let's see its block diagram of the proposed system shown in Fig. 1.

Information about a different foreign fault injection service is given by the UDDI registry whenever any fault injection requested by the user is not found in the local service. Then SOAP message is sent to UDDI server and utilizes its registry to find the appropriate host such as SWIFI tools provider which could deliver the requested service for the desired platform, fault type etc. to the target system. The local Fault Injection service does the basic fault injection function by leveraging the architecture given by researcher[4].

Figure 2 shows a local fault injection service environment, which consists of the target system, a fault injector, fault library, workload generator, workload library, controller, monitor, data collector and data analyzer.

The fault injector injects faults into the target system and executes commands from the workload. Here, fault injector refers to the SWIFI which allows different fault types, fault locations, fault times and software or hardware properties from fault library. The monitor tracks the execution of the commands and initiates data collection. The data collector performs online data collection. The data analyzer[4] performs offline data processing and analysis. The controller controls the overall experiment.

CONCLUSIONS

Emerging Software Implementation Fault Injection (SWIFI) tools has been developing now a day. But there is no such SWIFI tool which introduced web service technologies. Web service based SWIFI system will support automated fault injection facilities; accommodate a variety of fault injection techniques; emulate different classes of faults; portability of different platforms; both online and offline target systems; language independency and leveraging existing facilities of most the SWIFI tools. The proposed framework for implementing web service based SWIFI systems is a result of brainstorming. More study and research is required to implement this web service based system.

REFERENCES

1. Looker, N. and J. Xu, 2003. Assessing the dependability of SOAP-PC-based web services by fault injection. IEEE Workshop on Object-oriented Real-time Dependable System, pp: 163-170.
2. Looker, N., M. Munro and J. Xu, 2004. Testing web services. In Proceeding 16th IFIP Intl. Conf. On Testing of Communicating Systems, Oxford, pp: 1-5. http://www.dur.ac.uk/ n.e.looker/papers/ testcom2004.pdf
3. Web Services Architecture, 2004. W3C Working Group Note. http://www.w3c.org/TR/ws-arch/#whatis
4. Hsueh, M.C., T.K. Tsai and R.K. Iyer, 1997. Fault injection techniques and tools. IEEE Computer, 3: 75-82.
5. Some, R.R., W.S. Kim, G. Khanoyan, L. Callum, A. Agrawal and J.J. Beahan, 2001. A softwaren implemented fault injection methodology for design and validation of system fault tolerance. Proc. IEEE DSN'01, pp: 501-506.

6. Kanawati, G.A., N.A. Kanawati and J.A. Abraham, 1995. FERRARI: A flexible software-based fault and error injection system. IEEE Trans. Computers, 44: 248-260.
7. Tsai, T.K., R.K. Iyer and D. Jewitt, 1996. An approach towards benchmarking of fault-tolerant commercial systems. Proc. FTC-96, pp: 314-323.
8. Carreira, J., H. Madeira and J.G. Silva, 1998. Xception: A technique for the evaluation of dependability in modern Computers. IEEE Trans. Software Eng., 24: 1-25. http://www.xception.org/files/IEEETSE98.pdf
9. Han S., K.G. Shin and H.A. Rosenberg, 1995. DOCTOR: An integrated software fault injection environment for distributed real-time systems. Proc. CPD'95, pp: 204-213.
10. Dawson, S., F. Jahanian, T. Mitton, T.L. Tung, 1996. Testing of fault-tolerant and real-time distributed systems via protocol fault injection. Proc. FTCS '96, pp: 404-414.
11. Kao, W.L. and R.K Iyer, 1994. DEFINE: A distributed fault injection and monitoring environment. Proc. IEEE-FTPDS'94, pp: 252-259.
12. Kao, W.L., R.K. Iyer and D. Tang, 1993. FINE: A fault injection and monitoring environment. IEEE Tarns. Software Eng., 19: 1105-1118.
13. Chandra, R., R.M Lefever, M. Cukier and W.H. Sanders, 2000. Loki: A state-driven fault injector for distributed systems. Proc.DSN-2000, pp: 237-242.
14. Tsai, T.K., R.K. Iyer and D. Jewitt, 2002. NFTAPE: Networked fault tolerance and performance evaluator. Proc. DSN'02, pp: 542-543.