A Concept for Hybrid Fault Injection in Distributed Systems

Christian Trödhandl and Bettina Weiss
TU Vienna, Embedded Computing Systems Group, Vienna, Austria
{troedhandl,bw}@ecs.tuwien.ac.at

1 Introduction

Fault tolerance is an essential part in the design of distributed computer systems [2]. Thus, testing algorithms for such systems often involves fault injection (FI) experiments to evaluate if the specified fault hypothesis holds for the implementation, i.e., faults can be seen as part of the valid input data of a fault-tolerant system. Basically, the different methods of FI can be grouped into three main categories (see [5] for a comparison): *Simulation-based FI* [1], *hardware FI* [6], and *software FI* [4].

Hardware FI tends to be less intrusive where timing is concerned than software FI and thus preferable. It is primarily used on the chip-level, however. Software FI, on the other hand, while allowing fault injection on the algorithmic level, considerably affects the timing behavior of the system and thus is not suitable for typical networked embedded systems with real-time constraints.

In this paper, we present a concept of a FI framework for distributed systems[1] that is based on a hybrid hardware-software based method of fault injection. Additional to the hybrid approach, we aim for a global fault model, where the conditions for the injection of a specific fault may depend on the internal states of all nodes of the tested distributed system, similar to the system proposed with the Loki [3] architecture. The improvement with regard to the Loki environment will be the implementation of a hardware-assisted FI mechanism that improves the timing properties of the FI system.

2 Fault Injection in Distributed Systems

To address the problem of deterministic fault injection with minimal or no influence on the timing behaviour of the nodes under test, we envision a hybrid hardware-software fault injection environment. In this framework we primarily use hardware fault injection for introducing high-level process and communication link faults: Special "FI-enabled" RAM, implemented within a field programmable gate array (FPGA), will be used to inject faults into defined data or code regions of the tested program. Those injected faults can be triggered either by internal (e.g., some variable set to a certain value), external (trigger message received over the FI controller network), or by a timer event. Since FI is done by hardware, temporally independent from the tested system, there should be minimal or no influence on the timing of the system under test. In addition, light-weight software fault injection is used for any fault that cannot be injected directly by hardware. Most importantly, the resulting fault injection tool will be able to handle global fault models, that is, models in which there exist some system-global constraints on the number and/or nature of faults. Moreover, it will be able to inject both deterministic and random faults and will allow to monitor the behavior of the affected algorithms.

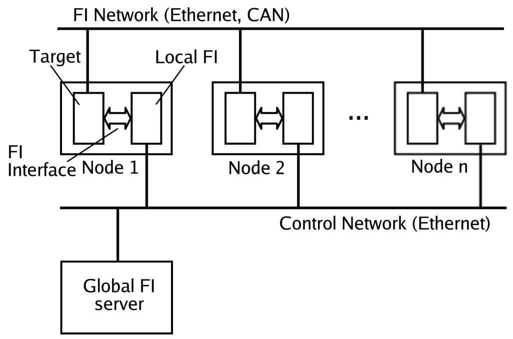

Figure 1. Fault injection cluster in wired setup

2.1 System Overview

The envisioned FI framework is a cluster of independent FI nodes, each consisting of a target system running the tested algorithm and a FI controller, controlling local fault injection and providing traces of the target (see Figure 1). This set of nodes is connected through two independent LANs, one communication network for the tested algorithm and the other to allow the system-wide coordination of the local FI controllers. In addition to the wired setup, the FI nodes also provide an external interface to wireless sensor

[1]This work is part of the FIDIS-project and received support from the Austrian BMVIT FIT-IT initiative under grant 813441 and the SPAWN-project with support from Austrian FWF under grant P18264.

nodes (e.g., ZigBee). On the controller network, a central FI server will serve as coordinator for FI experiments and store the resulting FI logs of the local FI nodes.

2.2 Hardware

The main means of fault injection is the transparent manipulation of memory contents of the target system. This is done by the use of a field programmable gate array (FPGA) that is connected to the external processor bus and emulates main memory to the CPU. This emulated RAM will feature a second interface to the controller node. An additional network controller can be integrated within the FPGA which may also be used for injecting faults (e.g., reordering, delaying, or dropping of network packets). The usage of dedicated hardware will minimize the probe effect compared to software-only solutions, which is of major concern when real-time algorithms are tested.

To establish a common timebase, the FI-nodes are equipped with a GPS-receiver that provides a periodic timing signal and can be configured to generate time-stamps for external events.

2.3 Software

The FI software can be grouped into the following parts: A modified development environment for the target nodes, minimal software changes to the OS of target nodes (e.g., RTLinux), the local FI software on the controller nodes, and the central server software for configuration, coordination, and logging of the FI experiments. The development environment for target boards will be modified to “understand” special annotations for variables and buffers that should be subject to FI. The linker will put those variables into special sections of the resulting executable. On the FI nodes, the OS will be adapted to map these sections to the “emulated” FI-RAM. The software on the local controller nodes injects predefined faults into the FI-RAM on the target nodes, either at a predefined time or in response to local triggers or trigger messages sent over the controller network. The software on the central FI server will setup and coordinate FI experiments and will log the results to a database.

2.4 Intended Applications

With the described FI environment, we aim to investigate two application scenarios: A setup for wired networks and one for wireless sensor nodes. In the wired setup, PowerPC single-board computers are used as target nodes, running either a special embedded RTLinux distribution or VxWorks. The wireless configuration will feature small wireless sensor nodes, based on 8-bit AVR microcontrollers, running for example MoteWorks or a ZigBee protocol stack. In both cases the FI controller node, based on the same PowerPC platform as the target node in the wired setup, will run RTLinux as operating system. Typical applications that will be examined by means of the described FI environment are distributed algorithms (e.g., clock synchronisation, consensus) with real-time constraints.

Possible types of faults, which will be produced by manipulating the memory of either the sender or receiver of messages, are symmetric faults (fault injected into the memory of the sender), byzantine messages (selectively modifying received messages originating from the “faulty” node), message omissions, reordering (delaying messages within the network controller). Additionally, the simulation of temporary or permanent memory corruption is possible (either deterministic or random), again resulting in symmetric or byzantine faults.

3 Conclusion

When it comes to testing distributed computer systems, fault injection is an important tool for verifying the conformance to the specified fault hypothesis. By using hardware FI the negative influence on the timing of the system can be kept to the minimum, but since in most cases this method is deployed on the chip or network level, the range of possible fault patterns is limited. Software FI is characterized by a broad range of possible fault scenarios, but lacks the good timing properties with regard to real-time applications of hardware FI. In this paper we proposed a hybrid solution, where special hardware (emulated RAM within a FPGA) is used together with a distributed software FI environment to provide for a wide area of FI models with low influence on the timing of the system.

References

[1] G. A. Alvarez and F. Cristian. Centralized failure injection for distributed, fault-tolerant protocol testing. *Proceedings of the 17th International Conference on Distributed Computing Systems*, pages 78–85, May 1997.

[2] J. Arlat, et al. Fault injection for dependability validation: A methodology and some applications. *IEEE Transactions on Software Engineering*, 16(2):166–182, Feb 1990.

[3] R. Chandra, R. M. Lefever, M. Cukier, and W. H. Sanders. Loki: A state-driven fault injector for distributed systems. *Proceedings International Conference on Dependable Systems and Networks, 2000. DSN 2000.*, pages 237–242, 2000.

[4] S. Dawson, F. Jahanian, T. Mitton, and T.-L. Tung. Testing of fault-tolerant and real-time distributed systems via protocol fault injection. *Proceedings of Annual Symposium on Fault Tolerant Computing*, pages 404–414, Jun 1996.

[5] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer. Fault injection techniques and tools. *IEEE Computer*, 30(4):75–82, 1997.

[6] J. Karlsson, et al. Integration and comparison of three physical fault injection techniques. In *Predictably Dependable Computing Systems*, pages 309–329. Springer Verlag, 1995.