[Proceeding] FAUST: fault-injection script-based tool

**Original Citation:**

**Availability:**
This version is available at: [http://porto.polito.it/1416278/](http://porto.polito.it/1416278/) since: October 2006

**Publisher:**
IEEE Computer Society

**Published version:**
DOI: [10.1109/OLT.2003.1214386](https://doi.org/10.1109/OLT.2003.1214386)

**Terms of use:**
This article is made available under terms and conditions applicable to Open Access Policy Article ("Public - All rights reserved"), as described at [http://porto.polito.it/terms_and_conditions.html](http://porto.polito.it/terms_and_conditions.html)

Porto, the institutional repository of the Politecnico di Torino, is provided by the University Library and the IT-Services. The aim is to enable open access to all the world. Please share with us how this access benefits you. Your story matters.

(Article begins on next page)
FAUST: FAUlt-injection Script-based Tool

A. BENSO, S. DI CARLO, G. DI NATALE, P. PRINETTO, I. SOLCIA, L. TAGLIAFERRI

Politecnico di Torino

Dipartimento di Automatica e Informatica Torino, Italy

{benso, dicarlo, dinatale, prinetto, solcia, tagliaferri}@polito.it

Abstract

The tool described in this paper aims at evaluating the effectiveness of software-implemented fault-tolerant techniques used in safety-critical systems. The target application is stressed with the injection of transient or permanent faults. The user can therefore observe the real behaviour of the application in presence of a fault, and, if necessary, take the appropriate countermeasures. The accent is put on the extreme easiness of use and the portability on all UNIX platforms.

1. Introduction

Dependable software used in human-critical applications (automotive, medical and spatial) should be able to detect whether an error appears, locate its position within the code, and recover from possible consequences [1]. To ensure that these skills are achieved by a fault-tolerant system, several techniques have been proposed during the last years and they are mainly based on analytical models, experimental techniques and fault-injection. The main advantage of fault-injection, with respect to the previous alternatives, is that a realistic test environment can be set up, putting the software under test in a condition that is very close to the normal context of execution [2]. FAUST project lies in this context: this tool has been designed in order to test fault tolerant software, without the need of external hardware or cost effective software. The injector is not related to any specific language or operating system so that a wide number of COTS applications (commercial off-the-shelf) are targeted and can be injected. The portability of our approach deal with the high portability of the tool which, thanks to its implementation, can be exported to all the UNIX based systems and to the reduced time overhead employed to perform the injections.

2. FAUST Architecture

The tool deals with applications written in every language supported by GDB and does not require any modification of the target application. From the user point of view, the injector appears as a command line tool whose parameters are:
- injection location (code segment, data segment, stack segment, CPU registers, variables);
- injection type (transient or permanent);
- injection time (random or user chosen);
- number of injections.

FAUST exploits the previous parameters to:
- generate the sequence of instructions to load and run the application within the debugger;
- inject the fault;
- collect the results.

For each injection experiments the tool categorizes the program results by comparing them with the correct one previously gathered. The expected result can be of three different types: no errors if the injection doesn’t affect program execution, crash if the program halts and wrong output when program provides results different from the expected ones. Then, for each injection, the results are collected in one log-file.

3. References

