The Journal of Systems and Software 103 (2015) 182–197

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Practical and representative faultloads for large-scale software systems

Pedro Costa [a,*], João Gabriel Silva [b], Henrique Madeira [b]

[a] CISUC/ISCAC – Polytechnic Institute of Coimbra, 3040 Coimbra, Portugal
[b] DEI/CISUC – University of Coimbra, 3030 Coimbra, Portugal

ARTICLE INFO

Article history:
Received 8 December 2013
Revised 31 October 2014
Accepted 2 February 2015
Available online 7 February 2015

Keywords:
Experimental dependability evaluation
Dependability benchmarking
Injection of software faults

ABSTRACT

The faultload is one of the most critical elements of experimental dependability evaluation. It should embody a repeatable, portable, representative and generally accepted fault set. Concerning software faults, the definition of that kind of faultloads is particularly difficult, as it requires a much more complex emulation method than the traditional stuck-at or bit-flip used for hardware faults. Although faultloads based on software faults have already been proposed, the choice of adequate fault injection targets (i.e., actual software components where the faults are injected) is still an open and crucial issue. Furthermore, knowing that the number of possible software faults that can be injected in a given system is potentially very large, the problem of defining a faultload made of a small number of representative faults is of utmost importance. This paper presents a comprehensive fault injection study and proposes a strategy to guide the fault injection target selection to reduce the number of faults required for the faultload and exemplifies the proposed approach with a real web-server dependability benchmark and a large-scale integer vector sort application.

1. Introduction

It is nowadays generally accepted that most of the software components have residual faults, also known as software defects or bugs, which escape the traditional testing phases of software development process. Several research studies also show not only a clear predominance of software faults (Kalyanakrishnam et al., 1999; Lee and Iyer, 1995; Sullivan and Chillarege, 1992; Gray, 1990) when compared to other types of system faults, but also that its weight on the overall system dependability will tend to increase. Among the main causes for those circumstances, besides the well-known technical difficulties intrinsic to the software development and testing process (Lyu, 1996), one can mention the huge complexity of today's software and the increasing pressure to reduce time to market. This scenario emphasizes the importance of system dependability assessment as a measure of confidence that can be relied on a given system. This includes the evaluation of attributes like availability, reliability, safety, integrity, among others. More than ever, practical approaches for the evaluation of the dependability of computer systems are needed, especially standardized dependability benchmarks that allow comparing dependability attributes of analogous and alternative software products or components. A fundamental characteristic that distinguishes dependability benchmarking from existing experimental dependability evaluation and validation techniques is that benchmarks should represent an agreement that is accepted by the computer industry and/or by the user community. However, the experimental evaluation of the dependability of computer systems is very difficult (Carreira et al., 1995) as it depends on fault activation probability, which in turn depends on internal and external system factors like the different layers of the software, the actual hardware where the software is running, environment issues, and human interaction.

After the success of the performance benchmarking initiatives that caught the attention of the industry in the last decades and have driven the creation of organizations like TPC (Transaction Processing Performance Council) (TPC, 2015) and SPEC (Standard Performance Evaluation Corporation) (SPEC, 2015), dependability benchmarking has been the focus of attention of researchers and practitioners in recent years (Kanoun and Spainhower, 2008; Brown and Patterson, 2000; Vieira and Madeira, 2003; Zhu et al., 2003; Lightstone et al., 2003; Kanoun et al., 2001; Christmanson and Chillarege, 1996; Durães and Madeira, 2002). To many business critical systems and applications, dependability attributes like availability, integrity and reliability, among others, are as important as performance. The goal of dependability benchmarking is thus to provide a standard procedure specification to characterize a computer system or component, providing the assessment of dependability related measures. The main components of a dependability benchmark suite are (Kanoun and Spainhower, 2008; Koopman and Madeira, 1999):

- **Workload** – Representing the work to be done by the system during the benchmark run and used to create a realistic operating scenario.

* Corresponding author. Tel.: +351 239790000.
E-mail addresses: pncosta@dei.uc.pt (P. Costa), jgabriel@dei.uc.pt (J.G. Silva), henrique@dei.uc.pt (H. Madeira).

http://dx.doi.org/10.1016/j.jss.2015.02.001
0164-1212/

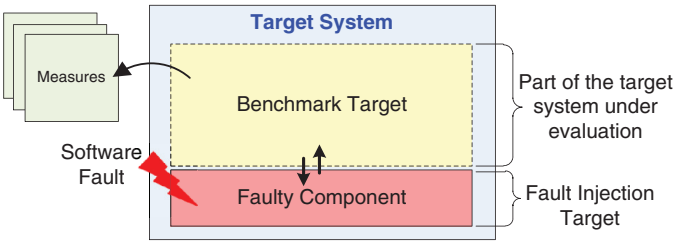

Fig. 1. Approach for software fault injection.

- **Faultload** – Representing a repeatable, portable, representative and generally accepted set of faults and stressful conditions that could lead to undependability, if not properly handled by the system.
- **Measures** – Characterizing the performance and dependability of a system executing the workload in the presence of the faultload.
- **Experimental setup and benchmark procedures** – Describing the setup required to run the benchmark and the set of procedures and rules that must be followed during the benchmark execution in order to ensure uniform conditions for measurement.

Among these components, one of the most critical and difficult to define is, undoubtedly, the faultload (Durães and Madeira, 2004), since it should represent a repeatable, portable, representative and generally accepted fault set. That difficulty is even higher in what concerns software faults, since they require a much more complex emulation method than the usual bit-flip fault injection approach used to emulate hardware faults. Furthermore, a faultload based on software faults requires a clear separation between the software components that are selected as fault injection target and the benchmark target (i.e., system under evaluation), as the injection of software faults actually changes the code of the target component. This way, the faults should be injected in one component (the target) in order to evaluate their impact on the other components or on the overall system (see Fig. 1). In fact, the software faults injected in the target component actually allow answering the question of what would happen to the system if a residual fault in such component became activated.

A representative faultload must contain faults that represent the common programming bugs that escape the traditional software testing phases and still persist in existent software products (Durães et al., 2004). Although the faultload definition of that kind of faults has already been proposed (Durães and Madeira, 2006), a problem still persists when that model is applied to very large and complex systems. Commonly, there are a large number of possible target components for fault injection and, consequently, that represents a huge number of possible software faults to be injected. Additionally, considering the time of each experiment (typically, the system should be restarted before injecting a new fault), one can easily observe that, in practice, it is impossible to run and test all the fault injection possibilities (i.e., the exhaustive set of software faults). For these reasons, the use of dependability benchmarks driven by software faultloads (e.g., such as the ones proposed in Kanoun and Spainhower (2008) has a major problem: it could take years to inject the complete faultload, which means that it is not possible to run such dependability benchmarks in practice. It is worth noting that the complete faultload encompasses the exhaustive set of software fault types and locations, representing the most common software bugs found in field, in all possible target locations. This limitation is especially significant in large and complex systems, where, in order to assure the necessary representativeness, the execution time of those benchmarks can take months or years due the mentioned faultload size. This is the case when the target system is a large piece of software, such as an operating system (OS).

Reducing the size of the faultload (but keeping it representative enough to obtain valid results) is therefore essential to show industry and the research community that it is possible to use dependability benchmarks in large-scale systems.

This paper presents the results of more than two years of continuous fault injection experiments in real systems and proposes a strategy to answer a still open and crucial question: how to choose adequate fault injection targets, and thus reduce the total software fault injection experiments, without restricting the benchmark results accuracy.

It should be noticed that among the mentioned faultload properties (repeatability, portability and representativeness), the representativeness is the one that needs special attention when reducing the faultload. In fact, properties such as repeatability and portability of the faultload are either not affected by the reduction of the number of faults or it is even easier to satisfy those properties with a reduced faultload.

This paper is organized as follows. Related research is discussed in Section 2. Section 3 presents the experimental strategy followed in our study as well as a detailed description of the proposed methodology for the definition of compact and representative faultloads. Section 4 describes the test-bed used to demonstrate the effectiveness of the proposed approach with two real and different applications: a web-server dependability benchmark and a large-scale integer vector sort application. Section 5 proposes an approach that can be used to reduce the size of a software faultload and proposes two ready-to-use calibrated faultloads specifically generated for the target system used in this research work. Finally, Section 6 presents the conclusions.

2. Background

Experimental dependability evaluation and dependability benchmarking has caught researchers' attention in the last few years and many experimental approaches for the evaluation of computer systems dependability have been proposed for several different application domains. This section briefly summarizes previous dependability evaluation systems proposals and surveys the different options used for the definition of faultloads, especially for the cases where faultloads are based on software faults.

A general methodology for benchmarking the availability of computer systems was introduced in Brown and Patterson (2000). This work uses fault injection to cause situations where software RAID (Redundant Array of Inexpensive Disks) systems availability may be compromised. It adopted the workload and performance measures from existing performance benchmarks.

A dependability benchmark for OLTP (On-line Transaction Processing) application environments is proposed in Vieira and Madeira (2003). This benchmark uses the workload of the TPC-C performance benchmark (TPC, 2012), an already well-established and agreed benchmark, and specifies the measures and all the steps required to evaluate both the performance and dependability features of OLTP systems, with emphasis on availability. This study uses as faultload a set of operator faults that emulates real faults experienced by OLTP systems in the field. Another dependability benchmark for transactional systems is proposed in (Buchacker and Tschaeche, 2003). Although this study also adopted the workload from the TPC-C performance benchmark, it considers a faultload based on hardware faults.

Research work at Sun Microsystems proposed a high-level framework specifically dedicated to availability benchmarking of computer systems (Zhu et al., 2003). The proposed approach decomposes availability in three key components: fault/maintenance rate, robustness and recovery. Within the scope of that framework, two dependability benchmarks were developed: one that measures specific aspects of a system robustness on handling maintenance events, such as the replacement of a failed hardware component or the installation of a software patch (Zhu et al., 2003); and a second benchmark for measuring system recovery on a non-clustered standalone system (Mauro et al., 2004).