A Methodology for Testing Simulated Environments

Pablo C. CAÑIZARES, Mercedes G. MERAYO and Alberto NÚÑEZ
Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Madrid, Spain

*Corresponding author

Keywords: Formal methods, Mutation testing.

Abstract. Currently, testing is the most widely used technique to check the validity of complex systems. Its application to software is a critical aspect in the development of systems. However, there are several difficulties for applying these techniques, such as high economic and computational cost. One of the main difficulties when applying testing techniques is to obtain an appropriate test suite. This is especially difficult when the size of the system under test is large, like HPC clusters or cloud systems. The main goal of this paper is to provide mechanisms that allow to evaluate the suitability of test suites to check distributed systems, in an inexpensive and efficient way. In this work, we propose a methodology to integrate mutation testing techniques with simulation techniques. For modelling and simulating distributed systems, the SIMCAN [1] simulation platform has been used. It is intended to unify, in one framework, functionalities of different tools, such as simulators, mutation frameworks and tools for generating tests. This work also presents some experiments for analysing the suitability of test suites over different distributed applications. Each of these applications has been modelled with SIMCAN, and different experiments have been carried out by applying mutation testing techniques on these models.

Introduction

In today’s society, there is a high percentage of technology corporations that invest a relevant percentage of their capital in testing software projects. An error during the execution of a system could result in considerable economic loss. As an example, in 1999 an error in the Earth’s computer’s metric system of the Mars Climate Orbiter caused the destruction of the satellite valued in $327 million [2]. The use of testing techniques can reduce the percentage of catastrophes occurring in these systems.

Over the last decades, there has been a change in business models in consequence of the high-speed Internet era. This new model carries a growth of the infrastructure and services, increasing both data generation and processing. This revolution requires the development and deployment of new applications that provide high performance to process large volumes of data. However, processing large volumes of data increases the complexity of the applications that process it, which also complicates the process of testing these applications.

Currently, testing is the most widely used technique to check the validity of complex systems. Testing techniques usually require the generation and application of a test suite that can determine whether the observed behaviour of the system corresponds to the expected one. One of the main difficulties in applying testing techniques is to have an appropriate test suite. This is especially difficult when the size of the system under test is large, such as distributed clusters or cloud systems. Fortunately, there exist several techniques to deal with these challenges. One of them is known as mutation testing. One of the most relevant features of mutation testing is the simplicity of its operation. Basically, mutation testing tries to replicate the mistakes often made by competent programmers, introducing syntactic changes in the original source. The sets of tests are evaluated, through the execution of the modified code, to determine its effectiveness in finding errors. However, the application of mutation testing techniques in distributed environments can be very expensive.
Moreover, in many cases, there is no access to the physical system in which the tests must be executed. In these cases, the scientific community has opted for the use of simulation tools.

This paper integrates modelling tools and simulation of highly distributed systems with the aim of evaluating the effectiveness of test suites for this type of systems in a scalable, efficient and accessible way. The application of mutation testing techniques on applications running in a simulated environment, will allow to deploy and configure different infrastructures faster, simpler and cheaper than performing the same process in a physical environment.

State of the Art

Currently, to the best of the authors’ knowledge, it does not exist in the literature any simulation framework for modelling distributed systems which integrates mutation testing techniques. On the one hand, there exist several simulation frameworks for modelling distributed systems [3, 4, 5], but none of these approaches includes testing techniques. Usually, in order to check the validity of an application, the test cases are generated by hand and in a low exhaustive way. Therefore, the number of test cases generated is insufficient and the checking process becomes an error-prone task. On the other hand, although there exist a wide variety of mutation testing frameworks [6, 7, 8], none of these tools is adequate for the simulation of highly distributed systems. This is due to the lack of specific mechanisms for simulation engines. Thus, an elevated number of classical mutation operators are applied, and in consequence, an excessive number of mutants are generated. This fact increases the computational cost associated with this technique.

In this way, Rutherford et al. proposed some studies that consider both aspects, simulation and mutation testing [9, 10]. However, these techniques are designed using tools that are not adequate to high scale systems. In order to perform simulation techniques the authors used SimJava, a simulation framework that does not propose real models of basic computational resources such as disks, processors or communication technologies. In consequence, the proposed environments are composed of a reduced amount of nodes; this fact restricts the scalability of distributed systems.

Methodology

This paper describes a methodology to integrate mutation testing techniques in a simulation platform. One of the main goals of this methodology is to take advantage of both simulation tools and mutation testing techniques. However, the integration of these techniques can be a complex task. This section describes the proposed schema to alleviate this integration.

In this work, we have selected the SIMCAN platform to integrate testing mechanisms. SIMCAN is a simulation platform which is oriented towards analyzing and studying parallel applications on distributed systems. SIMCAN has been designed to provide flexibility, accuracy, performance, and scalability. The main characteristics of SIMCAN are the flexibility to model different architectures easily, and the ability to scale those models keeping a good level of performance and accuracy. In SIMCAN a node is basically a building block for creating distributed environments (See Figure 1). A distributed environment consists of nodes, communication devices like switches or routers, and communication networks.

Initially, the user must build a model using the GUI (Graphical User Interface) provided by SIMCAN (see Figure 2). This model is composed of the configuration of the modeled distributed system, including the physical machines, the communication network and the application under test. In order to model the system, SIMCAN provides the user with a predefined model repository. These models simulate the behaviour of the devices that compound the distributed systems, such as CPUs, hard disks, memory and communication networks.

Once the model is designed, the mutation testing engine must generate the SIMCAN configuration files necessary to simulate the environment modeled by the user. In addition, the mutation testing
engine must be also provided with a test suite. Next step consists in configuring the mutation engine module.

Once the mutation engine is configured, this module receives the selected application to start the mutant generation process. Then, SIMCAN simulator receives the distributed system model and builds both the architecture and the network topology designed by the user.

Next, the test suite is used to test the original program and the set of mutants, following the classical mutation testing scheme. Finally, the results obtained in the testing process are reported to the user.

Experiments

In this section, we present the experiments that have been performed in order to evaluate the usefulness of MuTomVo. The experiments have been carried out using two applications. The first one, known as appCPU, performs computationally intense operations using a dataset. Initially, the dataset is stored at a remote server. Then, the application loads a subset of data in memory to process it and stores the results in the remote server.
The second application, known as *appMR*, is a simplified version of the Map-Reduce model proposed by Google. This application, whose main goal is to process in parallel a massive dataset, has two types of processes involved: coordinators and workers. These processes are grouped into frames; each of these frames includes a coordinator process and a set of worker processes.

<table>
<thead>
<tr>
<th>Tests</th>
<th>Application</th>
<th>#Mut</th>
<th>Mutation</th>
<th>Compilation</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsAppCPU</td>
<td>appCPU</td>
<td>87</td>
<td>475ms</td>
<td>2m 33s</td>
<td>2h 41m</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>254ms 1m</td>
<td>1 m 12s</td>
<td>4m 45s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>123ms</td>
<td>1m 15s</td>
<td>7m 15s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>tsAppMR</td>
<td>appMR</td>
<td>121</td>
<td>793ms</td>
<td>2m 43s</td>
<td>26h 11m</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24</td>
<td>390ms</td>
<td>1m 20s</td>
<td>4h 08m</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23</td>
<td>438ms</td>
<td>1m 12s</td>
<td>5h 35m</td>
</tr>
</tbody>
</table>

Figure 3 shows the performance results of the mutation testing process obtained using a random test suite. This graph uses the logarithmic scale at x-axis to show the computational cost, in minutes, associated with the different phases of the testing process. The results are grouped by different operator types. The suffix *Mut, Comp* and *Exec* represents the times associated with the mutation, compilation and execution of mutation testing, respectively.

Let us remark that the overall time of the mutation testing process is variable (See Table 1) and depends on the mutation operators applied to generate the mutants set. Thus, through the application of the proposed mutation operators, the number of mutants generated is significantly lower than the classical ones. Therefore, the overall performance of the system is increased.

**Conclusions and Future Work**

We have introduced a methodology to integrate simulation and mutation testing techniques. In addition, this paper reports the results of the experiments performed. These results show the time costs reduction obtained using specific simulation mechanisms. The application of mutation testing
techniques requires a high computational cost. Thus, as future work, we would use parallel techniques
to improve the performance of the mutation testing scheme, and consequently, reduce the time costs
associated to this computationally expensive technique.

Acknowledgements
Research partially supported by the Spanish projects ESTuDio (TIN2012-36812-C02-01) and
DArDOS (TIN2015-65845-C3-1).

References
and expandable simulation platform for modelling and simulating distributed architectures and
and simulating Data Grids: An extension to GridSim. Concurrency and Computation: Practice and
mesquite.com/.
simulation-based models of distributed systems. IEEE TRANSACTIONS ON SOFTWARE