CASE STUDY: RUNNING TEST CASES FOR SEQUENTIAL PROGRAMS IN PARALLEL IN A CLUSTER ENVIRONMENT

P. Kroha and V. Vychegzhanin
Department of Information Systems and Software Engineering
TU Chemnitz, Strasse der Nationen 62, 09111 Chemnitz, Germany

Keywords: Testing, running test cases, cluster, parallel processing.

Abstract: Testing a software system consists of a test case construction and of running and evaluating them, i.e. comparing expected results with obtained results. Because the set of test cases is usually very large, some methods of automation may be used to run them. Usually, test cases run on the same machine where the tested software system should run later. In our case, we run the test cases of a software system for a sequential computer on a cluster in parallel. The goal is to accelerate the process of running and evaluating tests. In this paper we describe the architecture of our test execution tool, experiments, and obtained results concerning performance and efficiency of test automation of sequential programs in a cluster environment.

1 INTRODUCTION

Systematic testing is necessary to improve the software system quality. However, testing is often perceived as a bottleneck operation in the software development process, since the other phases are much more automated (CASE tools, automated GUI design tools). For each configuration of the software product and for each version of each component there must be a specific set of test cases available and after any change in any component we have to adapt and execute the corresponding test cases again. The test cases are implemented by test scripts that contain test preconditions, test code, test data, and expected results. Test scripts are grouped to test plans that are part of the product documentation and are used by test execution automation tools. These tools support a systematic approach to testing but they need a configurable test infrastructure. Test plans for sophisticated software products may contain thousands of test scripts containing test cases. Sequential play-back of all test scripts on one sequential computer takes a significant amount of time.

Our idea is to use a cluster where as many of its nodes as possible can be used to load the test scripts and execute them in parallel. We examined the possibility to execute test cases (unit tests) generated as scripts by TestGen4J on our cluster of computers running with Linux. The goal was to write a test execution tool that loads all tests from the front-end computer (sequential PC) to a back-end computer (a cluster with 528 nodes) where a test manager controls test case running on cluster nodes and sends the evaluation back to the front-end computer.

Our system CLUSTEST is an experimental prototype that supports an accelerated testing process by delivering new levels of infrastructure which are given by parallel processing of test cases in a cluster.

The rest of the paper is organized as follows. In Section 2 we discuss the related work. In Section 3 we describe how we constructed our experimental test scripts. The architecture of our system, the communication of its components, and the processing of test cases are introduced in Section 4. In Section 5 we introduce our practical experiments and parameters measured. Finally, in the last section we conclude our experience.

2 RELATED WORK

A number of work has already been carried out on issues concerning parallel testing in the sense that either a parallel (or a distributed) program will be

Kroha P. and Vychegzhanin V. (2008).
CASE STUDY: RUNNING TEST CASES FOR SEQUENTIAL PROGRAMS IN PARALLEL IN A CLUSTER ENVIRONMENT.
In *Proceedings of the Third International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 177-180
DOI: 10.5220/0001760301770180

tested or some test scripts run as threads on a sequential machine. These projects include project TAOS, project AGEDIS, and product TestSmith. Project TAOS (Testing with Analysis and Oracle Support) developed a toolkit and environment supporting analysis and testing processes (Richardson, 1994). A unique aspect of TAOS is its support for test oracles and their use to verify behavioral correctness of test executions. The project AGEDIS developed a methodology and tools for the automation of software testing in general, with emphasis on distributed component-based software systems (Hartman and Nagin, 2004). The commercial product TestSmith from Quality Forge enables parallel test script playback on one sequential workstation. Each playback runs in its own thread. This allows multiple scripts to be played at the same time, greatly speeding up the testing process.

The basic feature that distinguishes our approach from the above listed is the use of clusters for parallel testing.

3 CONSTRUCTION AND PREPARATION OF TEST CASES IN PARALLEL

To automate test running, test cases are to be prepared and organized in structures that are suitable for separation of code and data of the test case, test cases via test scripts, changing test cases, and changing test scripts.

From the view of our paper it is not important whether test cases are generated by a tool or constructed manually. Our focus is on running test cases in parallel not on their construction. Our goal was not to develop new methods for test case generation but we needed a set of test cases as experimental data.

To illustrate advantages of running test cases in parallel in a cluster we needed a large number of test cases that cannot be written manually. To obtain enough test cases for our experiments we used the public domain tools TestGen4J. The process of test case preparation starts the open-source tool Emma that instruments code before testing. Instrumented code serves to save coverage information. After testing, Emma collects coverage information and generates reports in HTML, text, or XML.

Emma does not construct test cases. For this purpose we needed the tool TestGen4J. Differently from JUnit Test Generator that generates only empty bodies of methods, TestGen4J uses generation based on boundary values of method parameters. The user has to define them by the help of rules. TestGen4J works together with two others tools - JTestCase and JavaDoc. To get the information about instrumented classes for TestGen4J, e.g. description of classes, description of methods, constructors, variables, etc., we used JavaDoc which is part of the J2SDk from Sun. JUnit that was chosen for test case processing holds test cases (code) and their test case data together. This brings disadvantages when test data should be changed. Because of that we used JTestCase that separates test case code from test case data. For one test case code TestGen4J generates test cases data for all combinations of parameter boundary values. Using JTestCase the load, run, and evaluate procedures can run in a loop for one test case code using more test case data.

Summarized we prepared test cases as follows instrumentation of application classes by the tool Emma, separating information about instrumented classes by tool Java Doc, test cases generation by tool TestGen4J, and separation of test case code from test case data by tool JTest Case. After preparation of test cases they can be executed and evaluated using JUnit as described below.

4 THE CLUSTEST SYSTEM FOR RUNNING TEST CASES

The system CLUSTEST consists of a front-end (running partially as a local host and partially as a remote AFS host on a sequential PC) and a back-end (running on the cluster). Both modules communicate via messages and share files stored in AFS (Andrew File System). The architecture and communication schema is shown in Fig. 1.

The front-end module represents the interface to the user (to select test scripts and to start them) and controls the processing. It runs on a sequential PC with a Java Virtual Machine.

First, the tool Emma will be started which produces instrumented classes from classes of the application under test. Instrumented classes contain some additionally inserted code necessary for test coverage. After all classes of the application have an instrumented counterpart, the front-end starts the tool TestGen4J that generates test cases. After all test cases have been generated, the test manager from the back-end obtains the instrumented classes and generated test cases and uses the available number of cluster modes to run and evaluate them by the help of the Java test framework JUnit. Results will be stored in a log file and later used to produce a report by the front-end.

When test cases are prepared as described above,

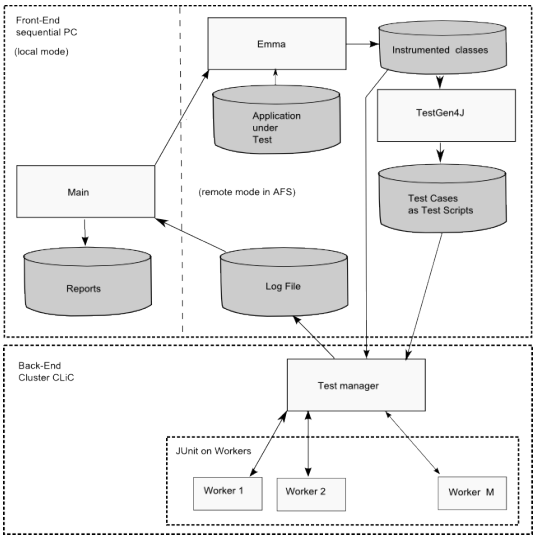

Figure 1: Architecture of CLUSTEST.

the front-end starts the back-end and passes to it the list of test scripts for parallel processing. Then the front-end waits until the back-end finishes test scripts playback, collects the logs and transfers them back to the front-end. At the end of test script playback execution, the user receives information of log files from the front-end in form of reports.

The back-end runs on the cluster CLiC (Chemnitz Linux Cluster). It is a cluster that consists of standard PC components. Each of its 528 nodes is a Pentium III processor working with clock frequency 800 MHz, 512 MB of memory and 20 GB disk space. All files of our CLUSTEST system are stored in distributed AFS file system so that they are accessible from the test manager and workers without any transfer. We used some concepts described already in (Kroha and Lindner, 1999).

5 EXPERIMENTS AND RESULTS

The tool CLUSTEST was implemented and described in (Vychegzhanin, 2006). As our experimental data we used a source program of a Java application Azureus. It is a file sharing client for BitTorrent-Protocol under the GNU General Public Licence. For our experiments we used 1024 classes of Azureus. Not all of its classes are suitable for automatic generation, some of classes have methods that cause infinite loops. As already said our goal was not to investigate test generation so we did not investigate the reasons deeply because it was not our focus. As we stated we generated the test cases (about 200,000) as a separate job on a sequential PC because of some technical problems (caching mechanism of AFS) and organization problems (too long queues) in using cluster.

It took 50 minutes to generate test cases for 1024 classes. We measured the elapsed time in two modes, with using a coverage test and without using a coverage test. The alternative without using a coverage test runs about 11 % more quickly. The elapsed time is the time between the start of the first test case and the finish of the last test case. It does not include data transfer of source data or scripts, time between the source request (Start of Emma) and the start of test manager, waiting because of other cluster users, reading test cases, and transfer of results from the log file to the front-end.

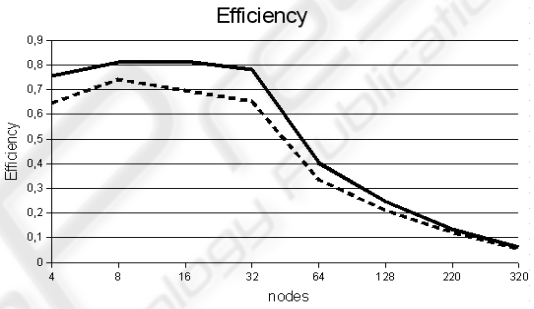

Figure 2: Efficiency.

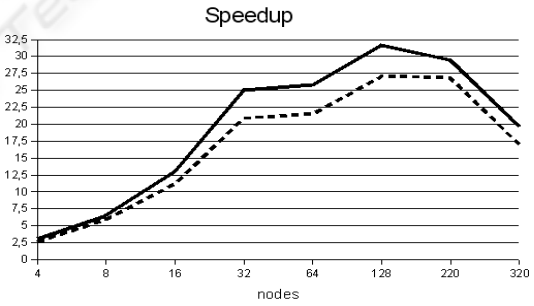

Figure 3: Speedup.

The obtained results can be seen in the following figures. The efficiency in Fig. 2 starts to fall for more then 32 nodes. In Fig. 3 we can see how the speedup depends on the number of nodes. It reaches its maximum for 128 nodes. As we can see in Fig. 4 it took 825 seconds to run and evaluate the used set of test cases on a sequential PC (1005 seconds including coverage test) but only 320 seconds using 4 nodes and only about 30 seconds using 128 nodes.

In Fig. 5 we can see the same in more details. With increasing number of nodes over 128 the overhead grows. The bottleneck is the single test manager.

In a future version we will partition the test cases and use more test managers for large numbers of nodes.

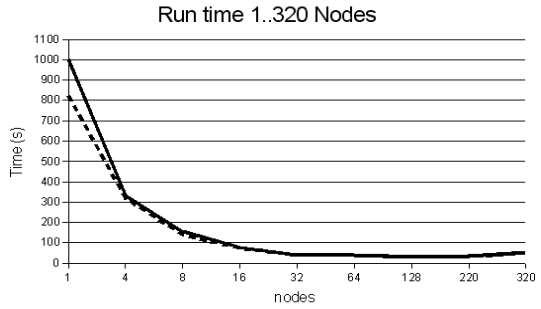

Figure 4: Run Time 1.

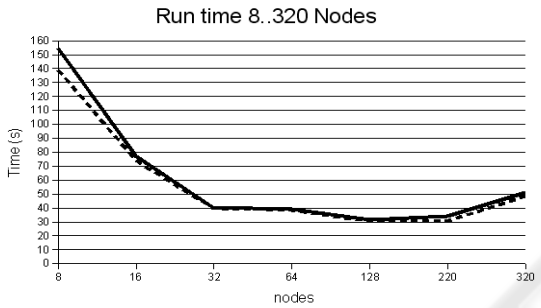

Figure 5: Run Time 2.

6 CONCLUSIONS

The application of clusters in the problem of sequential program test automation distinguishes the presented approach from other related works and may cause a significant progress in using clusters for purposes of software engineering. We have shown that the software testing phase can be shortened by the deployment of test automation into a cluster environment.

We presented practical results of our project CLUSTEST obtained by successful design and implementation of a tool for automated software test running on a cluster. The obtained experimental results were expected in their quality, i.e. we expected that test evaluation will accelerate, but it is interesting (and in practice it is necessary) to know also quantitative parameters (e.g. speedup and efficiency for a given number of nodes) of the test evaluation performed with a real software tools.

Our experiments revealed that parallel playback of test scripts gives a significant gain in time (see Fig. 2, Fig. 3) comparing to the sequential test scripts playback. A speedup of 25 for 32 used nodes and speedup 32 for 128 used nodes has been achieved which is an excellent result. In the future work we will investigate how general are our observations, i.e. we will test more systems and compare the results.

There is an interesting problem how faithfully the test system represents the environment of the system where it is expected to run. Even though we used a cluster of PC for running tests of PC programs we expect differences in run-time support libraries etc. In principle this is a problem of portability. Theoretically we have to accept the possibility that there could be some side-effects introduced by the cluster. In our case we did not found any problems of this kind but theoretically we cannot rule them out.

Cluster computing is generally applied to simulations in physics and chemistry, machine learning, aerospace technology, seismology, meteorology, etc. As for software engineering, clusters are almost never used until now for two reasons. First, the most tasks of the software development process do not require vast computing resources. Second, a cluster is not a cheap equipment. However, the complexity of software systems is steadily growing and the hardware prices are steadily falling. We argue that clusters as resources are preferable in the implementation of a test execution tool for execution of large number of test scripts as studied in our project.

Often, it happens that a project is behind schedule when testing should be started. We believe that a 32-machine-cluster will be low-priced soon and can bring much benefit (speedup 25) in such a way that it considerably contribute to the acceleration of testing.

REFERENCES

Hartman, A. and Nagin, K. (2004). The agedis tools for model based testing. In *Proceedings of ISSTA 2004, Boston*.

Kroha, P. and Lindner, J. (1999). Parallel object server as a data repository for case tools. In *Croll, P. / El-Rewini, H. (Eds.): Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems PDSE99,IEEE Computer Society, ICSE99, Los Angeles*, pages 148–156.

Richardson, D. (1994). Taos: Testing with analysis and oracle support. In *Proceedings of the International Symposium on Software Testing and Analysis*.

Vychegzhanin, V. (2006). Clustest Erweiterungen. Master's thesis, TU Chemnitz, Germany. M.Sc. Thesis (In German).