Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information and Software Technology 47 (2005) 657–662
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Parallel testing of distributed software

Alexey Lastovetsky

Department of Computer Science, University College Dublin, Belfield, Dublin 4, Ireland

Received 18 March 2003
Available online 25 December 2004

Abstract

The paper presents the experience of the use of parallel computing technologies to accelerate the testing of a complex distributed programming system such as Orbix 3, which is IONA's implementation of the CORBA 2.1 standard. The design and implementation of the parallel testing system are described in detail. Experimental results proving the high efficiency of the system are given.

Keywords: Parallel computing; Software testing; Distributed programming systems; Software maintenance; CORBA; Orbix

1. Introduction

Software testing is a very labour-intensive and hence very expensive process. It can account for 50% of the total cost of software development [1–3]. Software testing is also a very costly part of software maintenance in terms of contribution in the total time of the process of maintenance. If the process of testing could be accelerated, significant reductions in the cost of software development and software maintenance could be achieved.

In this paper, we present a case study demonstrating the use of parallel computing for acceleration of the testing of a complex distributed programming system such as Orbix 3 [4], which is IONA's implementation of the CORBA 2.1 standard [5].

Orbix 3 is a distributed programming system used by thousands corporate users around the world. It is an axiom that any software system has bugs. The wider and more intensive is the usage of the software system, the more bugs are exposed during its exploitation. Orbix 3 is not an exception to the rule. Every day its users report new bugs affecting the functionality or performance of the Orbix 3 software. A dedicated team of software engineers is constantly working on the bugs and making appropriate changes in the Orbix 3 code.

The maintenance process includes running an Orbix 3 test suite before and after any changes made in the Orbix 3 source code in order to:

- See if the bug has been fixed;
- Check that the changes themselves do not introduce new bugs into the software.

The test suite consists of many hundreds of test cases. Each fixed bug results in one more test case added to the test suite. This test case should test the problem associated with the bug and demonstrate that the problem has been solved. Thus the number of test cases in the test suite is constantly growing.

The serial execution of a test suite on a single machine might take from 9 to 21 h depending on the particular machine and its workload. The test suite must be run against at least three major platforms. For each platform the test suite must be run at least twice, namely before and after the corresponding changes are made in the Orbix 3 source code. Thus, on average, the best time for running a test suite is 90 h per bug. Often, however, it takes longer. For example, if a bug is reported in some minor platform, the test suite should be run against both all major platforms and the minor platform. If the bug has enough complexity, the very first solution of the problem may introduce new bugs, and hence more than one solution will have to be tested during the work on the bug.

E-mail address: alexey.lastovetsky@ucd.ie.

0950-5849/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2004.11.006

In terms of time, serial running of the test suite is the most expensive stage of the maintenance process. So its acceleration could significantly improve the overall performance of the maintenance team. Since the local network of computers available to the maintenance team includes more than one machine for each major platform and most of the machines are multiprocessor workstations, parallel execution of the test suite seems to be a natural way to speed up its running.

2. Parallel execution of the Orbix test suite on a cluster of multiprocessor workstations

As all major platforms, against which Orbix 3 should be tested, are Unix clones, an immediate idea is to use the GNU `make` utility for parallel execution of different test cases of the test suite. On Unix platforms, the '`-j`' option tells `make` to execute many jobs simultaneously. If the '`-j`' option is followed by an integer, this is the number of jobs to execute at once. If this number is equal to the number of available processors, there will be as many parallel streams of jobs as processors. As the utility assigns jobs to parallel streams dynamically, the load of the processors will be naturally balanced.

This simple approach has several restrictions. One is that it can only parallelize the execution of a set of jobs on a single multiprocessor machine. Another restriction is that if some jobs in the set are not fully independent, the straightforward parallelization may not guarantee their proper execution. For example, a number of jobs may share the same resources (processes, data bases, etc.), whose state they both change and depend on in their behaviour. Such jobs should not be executed simultaneously, but the GNU `make` utility provides no direct way to specify that constraint.

A typical test case from the Orbix 3 test suite builds and executes a distributed application. It normally includes the following steps:

- Building executables of the server(s) and clients of the distributed application.
- Running the application.
- Analysing the results and generating a report. The report says whether the test case passed or failed, and includes the start time and end time of its execution.

On completion of the execution of the test suite, all individual reports produced by the test cases are summarised into a final report.

During the serial running of the test suite on a single computer, its test cases share the following resources:

- Basic system software such as compilers, interpreters, loaders, utilities, libraries, etc.
- An Orbix daemon, through which servers and clients of Orbix distributed applications interact with one other. The daemon is started up once, before the test suite starts running.
- An interface repository, which stores all necessary information about server interfaces. This information can be retrieved by clients to construct and issue requests for invoking operations on servers at run time.

What happens if multiple test cases are executed simultaneously on the same computer? Can the sharing of the above resources cause unwanted changes in their behaviour?

The basic system software should cause no problem. Each test case just uses its own copy of any compiler, interpreter, loader, utility, or static (archive) library. As for dynamic shared libraries, their simultaneous use by multiple test cases should also cause no problem. This is simply because a dynamic shared library is by definition a library whose code can be safely shared by multiple, concurrently running programs so that the programs share exactly one physical copy of the library code, and do not require their own copies of that code.

There further should be no problem with sharing one physical copy of the Orbix daemon by multiple concurrently running distributed applications. This is because that sharing is just one of the core intrinsic features of the Orbix daemon. Moreover, in terms of testing, simultaneous execution of multiple distributed applications is even more desirable than their serial execution as it provides more realistic environment for functioning Orbix software.

Problems may occur if multiple concurrently running test cases share the same interface repository. In order to specify the problems, let us briefly outline how interfaces and interface repositories may be used in the Orbix 3 test suite.

In order to stress object orientation of the CORBA distributed programming technology, server components of CORBA-based distributed applications are called server objects or simply *objects*. The CORBA Interface Definition Language (IDL) permits interfaces to objects to be defined independent of an objects implementation. After defining an interface in IDL, the interface definition is used as input to an IDL compiler, which produces output that can be compiled and linked with an object implementation and its clients.

CORBA supports clients making requests to objects. The requests consist of an operation, a target object, zero or more parameters and an optional request context. A request causes a service to be performed on behalf of a client, and results of executing the request returned to the client. If an abnormal condition occurs during execution of the request, the exception is returned.

Interfaces can be used either statically or dynamically. An interface is statically bound to an object when the name of the object it is accessing is known at compile time. In this case, the IDL compiler generates the necessary output to compile and link to the object at compile time. In addition, clients that need to discover an object at run time and