Current Problems and
Tools Support for Testing Distributed Systems

Anh Thu Le

A dissertation submitted for the degree of
Bachelor of Applied Science with Honours in the
Department of Information Science at the University of Otago,
Dunedin, New Zealand

October 2006

Acknowledgments

First of all, I would like to thank my supervisor, Dr. Maryam Purvis for her valuable supervision, suggestions, and encouragements during the course of my thesis. A special thank goes to Tony Bastin Roy Savarimuthu for being a thoughtful and accessible supporter.

I thank the University of Otago Language Centre in Dunedin for making a number of systems available for testing. I also thank Professor Paul Strooper at the University of Queensland in Australia for his willingness in helping me to implement the ConAn testing tool.

Finally, I would like to thank my parents for their precious support during my four years undergraduate education in Dunedin, New Zealand.

Abstract

Testing is a critical activity in designing and implementing software and computer systems, especially distributed systems (Eickelmann & Richardson, 1996). In addition, validating distributed systems is a complex process, and it requires the knowledge of "the communication taking place and the order of communication messages exchanged between the individual components to avoid deadlocks or livelocks during runtime" (Ulrich, Zimmerer, Chrobok-Diening, 1999, p.2). This dissertation attempts to briefly emphasise the role of testing in distributed systems. In addition, it addresses few common issues in building distributed systems together with a number of solutions to these issues. The main focus of this dissertation is on three main features of distributed systems that often challenge the programmers and testers, which are concurrency, scalability, and fault tolerance. For the first two features, a number of open source testing tools are chosen to compare and contrast the strengths and weaknesses of each tool. A number of testing techniques are applied to show the importance of the third feature, namely, the fault tolerance. Besides, the importance of each feature is highlighted by testing a number of sample distributed systems.

Table of Contents

Lists of Figures

List of Tables

1. Introduction

Distributed systems have become common in modern society despite their complexity. These systems are used in almost every day-to-day operation. Like any other system, programmers and testers desire to enhance the system by achieving better performing and secured systems. This dissertation studies a set of issues that are faced by most programmers and testers when handling distributed systems. In particular, it aims to address the three main aspects of distributed systems, which are concurrency, scalability, and fault tolerance. The importance of each aspect is described and is highlighted in the context of testing a number of sample distributed systems. Additionally, open source testing tools are selected to compare and contrast to assist testers in making decisions on which tools should be used when performing these testing tasks.

The dissertation is structured into eight sections. Section 1 is the introduction which outlines the structure of the dissertation. The details of each section are described below.

Section 2 introduces the background information to provide an overview of distributed systems. In particular, the definition of a distributed system is given and a list of characteristics of distributed systems is described.

Section 3 lists the three main objectives of this dissertation.

Section 4 outlines testing in the context of distributed systems. It is followed by section 5 which describes a number of major problems that should be taken into consideration when working with distributed software systems. Solutions that help to overcome these problems are also included in this section.

Section 6 focuses on concurrency issue and the concurrency testing is described in detail. Three sample distributed systems (i.e., Cathy's Online Car Store, Calendar Micro Environment (CalME), and E-Shopping) are tested in order to show concurrency issues

involved in testing these distributed systems. GroboTestingJUnit and Concurrency Analyser (ConAn) are two open source testing tools which are selected to compare and contrast from this point of view.

Section 7 covers the scalability testing in distributed systems. Three sample web based systems, namely, Otago Language Centre Assets (OLC Assets), Dunedin City Restaurant Review (DCRR), and Employee Directory are tested by applying TestMaker, one of the open source testing tools. The purpose of these tests is to demonstrate how scalability testing can assist testers in making decision about system performance. The capability of this tool is then compared with Open Systems Testing Architecture (OpenSTA) and httperf tools in the context of testing system performance under heavy load.

Section 8 demonstrates a number of failures that often occur in distributed systems and how fault tolerance can be achieved in distributed systems. An outline of some possible failures that might happen to the OLC Assets system and CalME system are described.

Finally, the outcome of the dissertation, the conclusion, and future work are summarised in section 9.

2. Background

This section provides a description for distributed systems in terms of definition, and their characteristics such as resource sharing, openness, concurrency, transparency, scalability, fault tolerance, heterogeneity, and security.

2.1 Definition of a Distributed System

There are a number of definitions about distributed systems. The following four definitions are taken as examples for defining a distributed system.

A distributed system is defined by Katzan as a collection of computers which are remote from a central computing. These computers have the facility to facilitate the process of communication between themselves and the central computer (1979).

Hillston (2002) described a distributed system as “a collection or network of loosely coupled, autonomous, computers with the following characteristics: The nodes are relatively loosely coupled. Each node is a self-contained autonomous computer with its own peripherals. The system can survive various categories of node and network failures. The nodes may execute logically separate computations, though these may be related to concurrent computations on other nodes. The system is asynchronous.” (p.2).

Goel (2004) outlines a distributed system as “a computing facility built with many computers that operate concurrently, are physically distributed, have their own failure modes, have independent clocks and are linked by a network” (p.4).

These definitions characterise a distributed system in a very similar way, but the most suitable definition is defined by Tanenbaum and Steen (2002). A distributed system is “a collection of independent computers that appears to its users as a single coherent system” (2002). This definition covers the two important aspects of distributed system which are

(1) hardware aspect - the machines are autonomous; and (2) software aspect - the users think they are dealing with a single system.

Figure 1 shows a simple structure of a distributed system that is constructed of heterogeneous computers and networks. A layer of software called middleware is placed between a higher-level layer which consists of users and applications, and a lower-level layer which consists of operating systems (Tanenbaum & Steen, 2002). Middleware is designed to manage the complexity and heterogeneity in distributed systems.

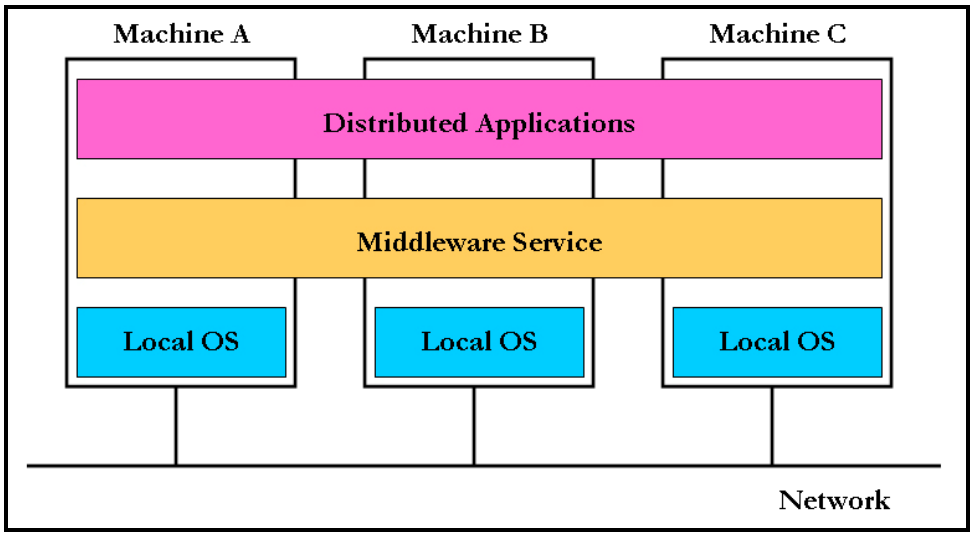


Figure 1: A Distributed System Organized as Middleware
(Source: Tanenbaum & Steen, 2002)

2.2 Characteristics of a Distributed System

These characteristics help to differentiate distributed system with other systems. According to Emmerich (1997), Goel (2004), Tanenbaum and Steen (2002), distributed systems consist of eight characteristics and are briefly described below.

- ***Resource sharing*** is the main goal of distributed systems, it provides users the ability to access resources anywhere in the system. Resources include hardware, software, data, etc.
- ***Openness*** is the second characteristic of distributed systems. A distributed system can be improved and extended from the original system without the need of

restructuring the entire system. New components will follow the standard rules, which are often described in an Interface Definition Language (IDL) in order to fit in with the existing components.

- ***Concurrency*** allows servers to handle different requests from multiple clients simultaneously. Because the clients can access and update shared resources from the server at any time, data integrity constraint should also be taken into account in order to maintain data consistency.
- ***Transparency*** is a very unique characteristic of distributed systems. The users cannot distinguish whether all tasks are handled amongst different computers across the network or between different servers in different networks. To the user of the system, it appears that all tasks are handled by a single computer. There are various types of transparency that exist in distributed systems, these include access transparency, location transparency, concurrency transparency, replication transparency, failure transparency, migration transparency, performance transparency, scaling transparency, relocation transparency, and persistence transparency. The following table explains the concepts associated with each type of transparency.

Transparency	Description
Access	Enables different data representations to be communicated between different processors. Apart from this, users are able to access the local and remote resources without the users' acknowledgment. For example, navigation in the web.
Location	Users are unable to differentiate the physical location of the resources that they access. For instance, pages in the web.
Concurrency	Users are able to access the same shared information concurrently without acknowledging that other users are making use of the same shared information. A table in a shared database is an example for this type of transparency.
Replication	Users are not able to recognize that several copies of one resource exist in the system. For example, a distributed database management system.

Failure	Users are not aware that in an event of failure, although a resource fails to work in the system, users are still able to complete their tasks. For instance, when one of the replicated databases fails, the other database can be used as a replacement.
Migration	Resources in the system can be moved without the need of changing the way of accessing that resource, such as a web page.
Performance	In the event of heavy load such as there is a large number of users using the system at the same time, the system is reconfigured to improve its performance.
Scaling	The system is expanded without the need of restructuring its structure or its algorithms. For example, the world-wide-web.
Relocation	Resources in the system can be relocated while they are being accessed without notifying and affecting the process. For instance, wireless laptop can continue to operate while moving without being disconnected.
Persistence	Resources are stored in volatile and non-volatile memory, and the users cannot identify when the server is moving from one state to another. For example, resources are first extracted from primary storage and copied into secondary storage, after being changed, the resource will be written back to the primary storage.

Table 1: Different Forms of Transparency in a Distributed System
Tabulated based on information provided in
Tanenbaum & Steen (2002) and Emmerich (1997)

- ***Scalability*** deals with the size of a system. Distributed systems tend to get larger, for instance, an increase in the number of computers in the network, and the extension of hardware and software. Scalability ensures that the system will still be able to perform without any performance degradation.
- ***Fault tolerance*** covers hardware, software and networks failures. In the event of failure, fault tolerance enables the system to continue to operate regardless of the failure and the users are unable to notice these faults.

- ***Heterogeneity*** is one of the important features of distributed systems. A distributed system tends to be expanded and the technology changes over time, as a result, distributed systems cannot always maintain the original homogeneity. Particularly, the structure of distributed systems relies on the underlying hardware. Each computer in the network may have different memory sizes, network protocols, and I/O bandwidth. Software heterogeneity is also another subordinate factor of heterogeneity since workstations in the network are running a variety of software and different operating systems.

- ***Security*** of distributed components and the security of transmitted data are important aspects of a distributed system. Security provides a secure communication channel for the users and ensures access to authenticated users only. There are a number of threats that can affect the system, these are known as interception, interruption, modification, and fabrication.

3. Research Objectives

Fagerström (1988) stated that “it is well known that programming and testing distributed systems can be extremely difficult” (p.88). Out of the vast range of research issues, this dissertation focuses on three main objectives:

- Highlighting the role of testing and identifying a set of issues when building and testing distributed systems. Studying the solutions to these issues that are addressed by the researchers.
- Presenting the three main aspects that often challenge testers in the context of testing distributed systems, which are concurrency, fault tolerance, and scalability. A number of sample applications are tested to illustrate the important role of each of the aspects.
- Examining a number of available techniques and tools that can be applied to test these three features, for example, GroboTestingJUnit and ConAn for concurrency testing, TestMaker, OpenSTA, and httperf for scalability testing, and a number of techniques which can be used to achieve fault tolerance in distributed systems. These tools and techniques are then compared and contrasted to identify their strengths and weaknesses.

4. Testing Distributed Systems

In this section, a brief introduction about testing and its essential role in constructing distributed systems are discussed in conjunction with different techniques that can be applied to each level of testing. These include unit testing, integration testing, system testing, acceptance testing, and regression testing. In addition, a list of important features that needs to be taken into consideration when designing distributed system is categorised and different solutions are discussed.

Testing is an important process in software development that helps to assure system quality. According to Myers, testing is "the process of executing a program with the intent of finding errors" (1978, p.4). The position of testing has gradually changed from being a final activity after the system has been completely built, to become a parallel activity which happens during the designing and building phases. In the early days (1950's), testing was done after the program had been completed (Ireland, 2006). This process is shown in Figure 2.

Figure 2: Traditional Testing Process

Today, testing is a crucial activity in software development process and should be designed before the actual code is written. Wake (2000) outlines the test and code cycle in programming, as shown in Figure 3. Additionally, the role of testing is depicted throughout a number of System Development Life Cycle Models (SDLC) that can be applied during the software development process. The most common SDLC models are Waterfall model, V software life-cycle model, and Spiral model. These models are shown in Figures 4, 5, and 6, respectively. As can be seen from these models that testing is not an isolated part of the system development cycle, instead, testing is integrated with other software development activities.

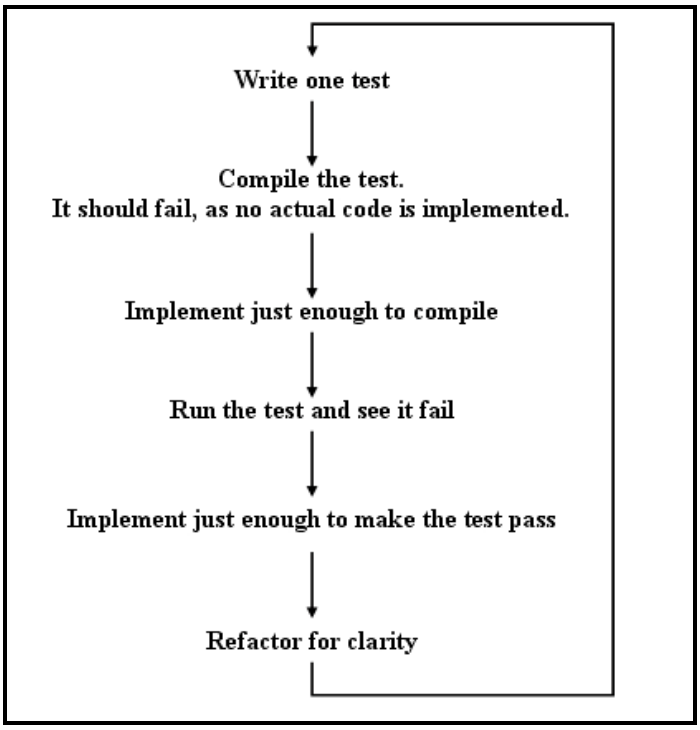


Figure 3: Test and Code Cycle in XP

Based on the description provided by Wake (2000)

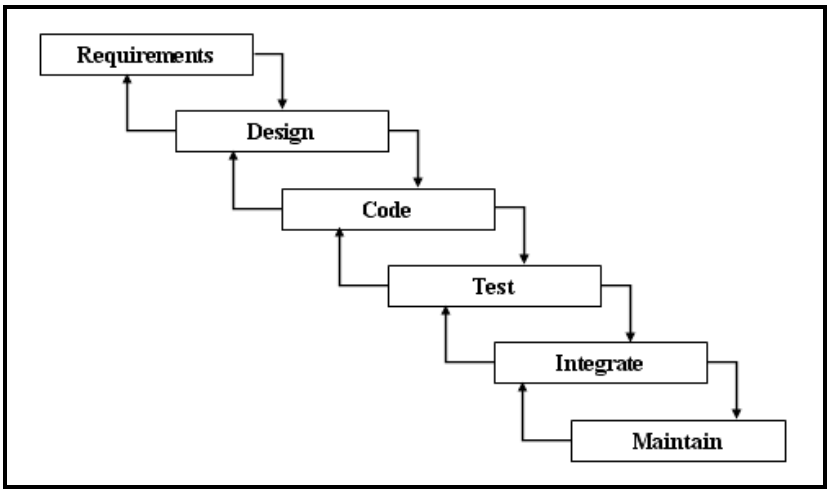


Figure 4: Waterfall Model

(Source: Dorfman, 1997)

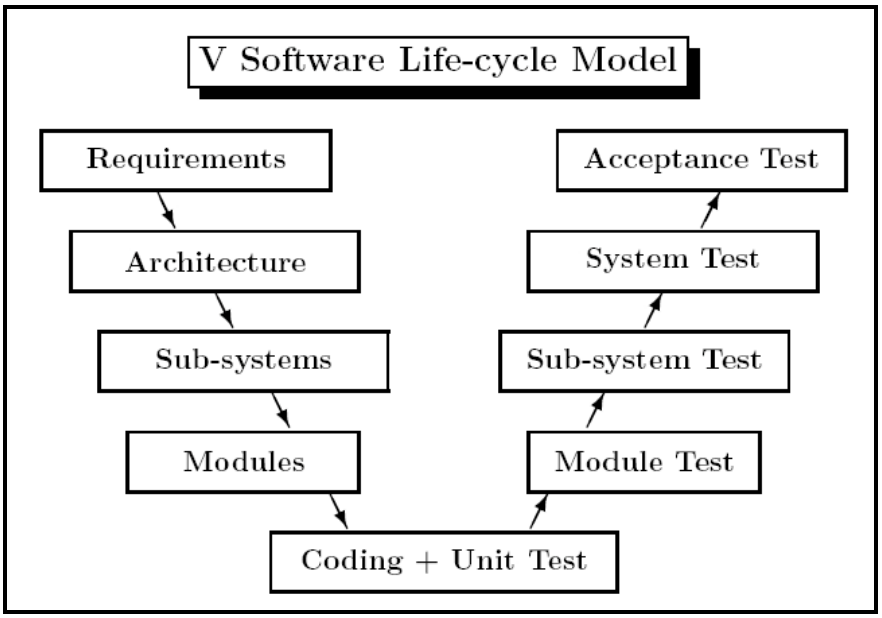


Figure 5: V Software Life-cycle Model

(Source: Ireland, 2006)

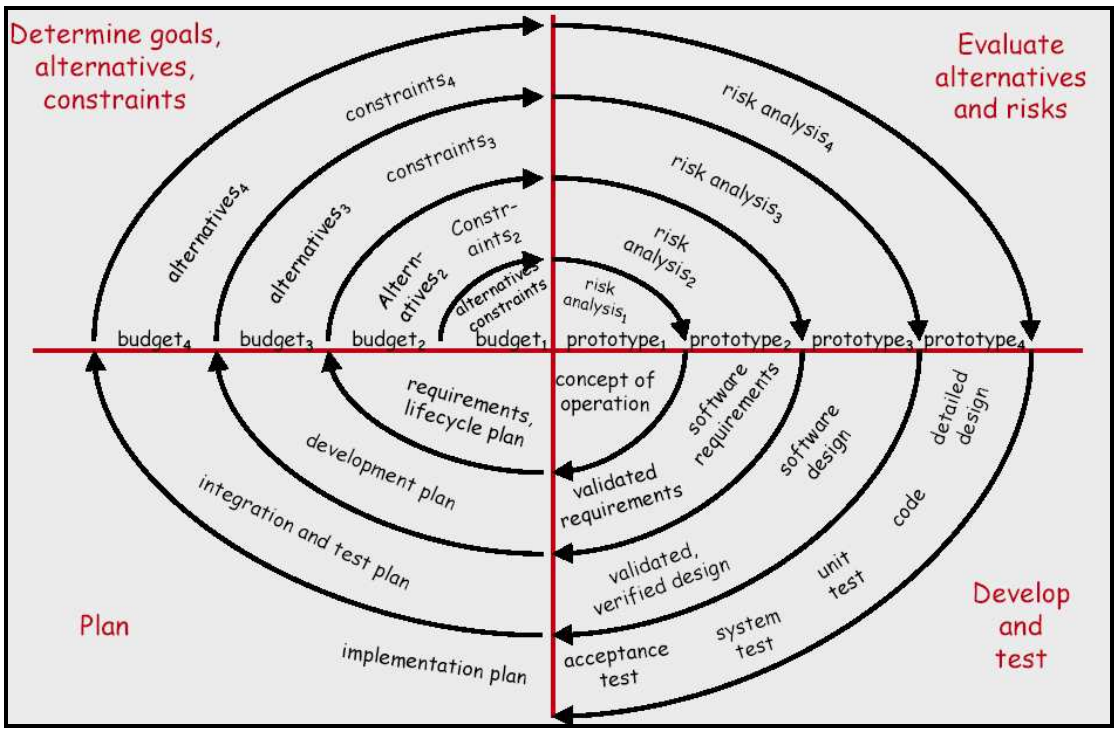


Figure 6: Spiral Model

(Source: Pfleeger, 1997)

In general, there are three main types of tests which are considered when designing test cases from a tester's point of view. They are known as black-box testing, white-box testing, and gray-box testing.

- Black-box testing is the technique in which the testers do not have access to the source code. Black-box testing is conducted by testers who should only know the condition of the inputs and the expected outputs without the need to enter a source code (Raishe, 1999). Tester and programmer are independent of one another, this helps to prevent the programmer's bias towards his/her own work. Tests are often done from the user's point of view and can be designed when the specifications are completed. On the other hand, black-box testing is not the most effective technique to test without the help of other techniques. Black-box testing also has some drawbacks, for instance, tests often do not cover all the possible inputs and some paths of the program may not be tested since it is impossible to test all the inputs.

- The term "white-box" is known as "glass box" and is defined as "a system or component whose internal contents or implementation are known" (IEEE, 1990, p.36). White-box testing is the technique which requires testers to have access to the logic and structure of the source code when performing testing tasks. White-box testing is used when designing unit test, integration test, and regression test (Williams, 2004). A white-box tester can design test cases that exercise independent paths within a module, exercise logical decisions, execute loops, and exercise internal data structures to ensure the validity (Pressman, 2001).

- Gray-box testing is the combination of black-box testing and white-box testing. Dustin (2002) explains gray-box testing is used to test a system by directly targeting the various components and levels of the system. Hence, gray-box testing requires the knowledge of the system's internal structure as well as the requirements from the user's point of view. Dustin (2002) also emphasises that unlike black-box testers, gray-box testers are able to identify the problem and the area of the source code where the error has occurred.

Apart from black-box testing, white-box testing, and gray-box testing, tests are also categorised into different levels. In addition to the three main levels of testing, namely, unit testing, integration testing, and system testing, there are other levels of testing such as acceptance testing and regression testing. As shown in Figure 5, project managers should choose suitable levels of testing depending upon the status of the project.

4.1 Unit testing

Unit testing or functional testing is the lowest testing level and often is written before the actual codes are derived. Unit testing ensures that each individual function/method in the program works correctly before integrating them. Unit testing helps to reduce the complexity when testing a complicated module. It is performed during the project's development cycle and as soon as the project begins. The most common unit testing framework is JUnit.

4.2 Integration testing

The next level of unit testing is integration testing. After each individual unit has been tested successfully, integration testing is performed by combining two or more units into one component and this component is tested. Although each unit proves to work well independently after unit testing is performed, it is found that problems still occur when units are combined to work together. According to the MSDN (2006), integration testing is done in the three common strategies which are the top-down approach, the bottom-up approach, and the umbrella approach.

The top-down approach tends to test the high level logic and data flow in the early stage of the process. However, since the approach starts too early, it cannot provide a good support for early release of limited functionality (MSDN, 2006); this also happens to the bottom-up approach. Additionally, the MSDN (2006) acknowledges that due to the nature of the top-down approach, low level utilities are tested quite late in the development cycle. In contrast, the bottom-up approach requires testing the lowest level of the system early in the development process. Unfortunately, the high level logic and data flow are tested relatively late. Another approach is the umbrella approach, testing can be done by merging the bottom-up approach and top-down approach in order to

support the early release of limited functionality. That is to say, the inputs for functions are integrated in the bottom-up manner and the outputs for each function are integrated in the top-down manner (MSDN, 2006).

4.3 Regression testing

Regression testing is done whenever the program is modified. It is necessary to perform regression testing because when a bug is fixed, the fix might create other bugs. This matter is solved by rerunning the existing tests against the change along with new tests. The MSDN (2006) points out that the most effective way to develop regression tests is to implement a library of tests that consisted of a number of test cases. These test cases can be run anytime there is a new version of the program. The number of test cases also needs to be written carefully to avoid having unnecessary test cases and causing time consuming when rerunning these tests. This happens when there is more than one person writing test code.

4.4 System testing

IEEE (1990) explains system testing is a test that "conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements" (p.74). Similarly, the goal of system testing is to measure the final system to its original objectives, this is emphasised by Myers (1978). System testing is accomplished to guarantee that the complete system behaves and functions in a correct manner, for instance, the system compiles, builds, and generates the accurate outputs. Myers (1978) identifies two main implications of system testing. First, system testing tends to demonstrate that some parts of the program do not meet the original objectives. Second, there should be a set of measurable objectives in order to compare the test outputs.

4.5 Acceptance testing

Acceptance testing generates tests from the end user's perspective. Acceptance testing is constructed in order to determine that the system performs correctly and meet all the required features. The tests are written by the business side of the project, for instance, the XP customers and business analysts. This factor helps to differentiate acceptance testing from system testing. Generally, acceptance tests should be written before the features are fully implemented. Miller and Collins (2001) proposed that when all the acceptance tests pass, the project is done. In addition, it is also stated that "acceptance tests are a 'contract' between the developers and the customer" (Miller & Collin, 2001, p.1). JAccept™ Editor is one example of an acceptance testing framework which was developed by Miller and Collins in 2001.

5. Issues and Solutions

This section covers a number of difficulties that often challenge the programmers in building distributed systems. These difficulties are identified as openness, concurrency, scalability, fault tolerance, latency, global clock, security, and heterogeneity. In this section, each issue is presented and is accompanied by the solutions proposed by the researchers. As mentioned in the introduction, concurrency, scalability, and fault tolerance aspects are explored in details in section 5, 6, and 7.

5.1 Openness

Openness is not only one of the important attributes that distinguishes distributed systems from other systems, but also the main objective that most developers would like to achieve. It is important to note that distributed systems tend to be extended and improved all the time, this can become a problem if the original system is not designed for this purpose and new components do not integrate with the existing components. For instance, when a new component is implemented, the system either fails to function as it used to or has to be reconstructed entirely to merge with the new component. Emmerich (1997) proposed that this problem is caused by the differences in data representation of interface types on different processors. To overcome this problem, Tanenbaum and Steen (2002) imply that standard rules that describe the syntax and semantics of the services provided through interfaces have to be published. These standard rules are formalized in an IDL, which includes the details of a particular function, that is, the syntax of services, and the features which these services provide, in other words, the semantics of interfaces. The aim of having detailed interfaces of components is to make sure that new components follow the standard rules of the original system (Tanenbaum and Steen, 2002).

Openness also includes the flexibility of a system. Distributed system should be flexible enough to be extended in a straightforward manner. In addition, its architecture should also be capable so that additional features can be easily added and the redundant features

can be easily removed. Standard and well-defined interfaces are essential in order to attain the flexibility.

5.2 Latency

In distributed systems, the time that takes to set up the communication is known as latency (Hillston, 2002). Apart from latency delay, there is transmission delay which is calculated by the length of the message and the data transfer rate, which is the time it takes to transfer the data of the message (Hillston, 2002). In short, message transmission time is the total of latency delay and the transmission delay. Figure 7 shows the delay in sending a message.

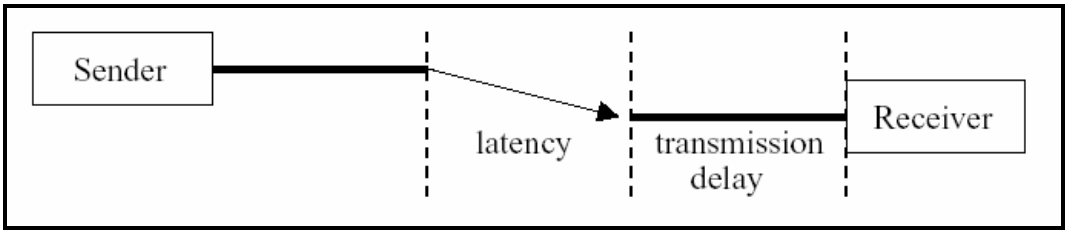


Figure 7: Message Transmission Time

(Source: Hillston, 2002)

There are issues that relate to latency. First, latency indicates the physical limitations of communication between sender and receiver. They can be propagation delay, insufficient bandwidth, network timeout, machine crashed, and connection refused. Second, Taylor, Redmiles, and Hoek point out that latency introduces uncertainty since the system fails to update the change to the clients (2006). Another issue is identified when latency becomes high, this means a request has to wait for a long time before being executed. Distributed system developers wish to minimize latency. According to Taylor et al. (2006), this can be achieved by using asynchronous notifications of resource changes and routing directly to relevant agencies.

5.3 Global clock

Unlike centralized systems which have a central server to control the system time, distributed systems do not have a global time. In a distributed system, although clocks are

synchronised, each individual clock does not have the exact time because of being run at different rate. This causes difficulties for cooperation between processes, for example, a request that is sent after another request may be assigned an earlier time, this leads to an incorrect and non-desirable effect. Burback (1998) explains that time in distributed systems is only known within a given precision, a clock may be synchronised with a more trusted clock but it is impossible to run an absolute value for clocks due to transmission and execution. This difficulty is addressed by using the Universal Time Coordinated (UTC) servers or the Internet Network Time Protocol (NTP). According to the NTP's documentation (2006), NTP is a fault tolerant protocol which uses the UTC to allow developers to synchronise computer clocks to national standard time via the Internet. NTP automatically selects the best of several available time sources to synchronise to.

5.4 Security

Like any other systems, distributed systems require a high level of security in order to prevent a number of threats. Pfleeger (1997) categorises four types of security threats that can affect the system which are interception, interruption, modification, and fabrication. Interception threat happens when a service or data is accessed by an unauthorised party. Interruption threat relates to a service or data that is made unavailable and inaccessible to other parties. Modification threat is to illegally modify the data so it appears to be different from the original data. Finally, fabrication includes generating additional data or activity that would normally not exist. As can be seen from these threats, security plays an important role in building and maintaining distributed systems.

Tanenbaum and Steen (2002) identify that security in distributed system can be divided in two main parts which are the secure communication channel and the authenticated access. These are achieved by a number of security methods such as encryption, authentication, authorization, and auditing. Firstly, encryption is the most common method that is used in computing security. Data is encrypted and is not easily understood by unauthorised users. Secondly, authentication is a method to verify the identity of a user. Normally, each user is assigned to a username or a password, or both. There are also

other ways to identify a user. Only an authenticated user is able to access the system. Thirdly, authorisation ensures that an authenticated user can only perform certain tasks depending on the permission that is given to a particular user. For instance, an administrator of a forum can delete a post, but not a standard member. Finally, the auditing method is used to map out the tasks that each user performs while using the system. This information is stored in audit logs which can help to identify intruders.

5.5 Heterogeneity

When a distributed service is designed for one type of computer and one type of network, its portability is restricted to that computer and network. This becomes one of the major difficulties and eliminates "the efficient and effective utilisation of resources" (Turnbull, 1987, p.11-22). Thus, maintaining heterogeneity in distributed systems is necessary. However, each computer in the network may have different memory sizes, network protocols, and I/O bandwidth. These are categorised into three main areas which are computer hardware heterogeneity, network heterogeneity, and software heterogeneity.

Nesterenko and Jin (2002) describe hardware heterogeneity as the difference in computer architectures of components in distributed system such as instruction sets and data representations. Network heterogeneity is the diversity of transmission media, signalling, protocols, and network interfaces. Finally, software heterogeneity relates to the difference in operating systems and application programs. These factors affect the distributed systems in terms of scalability, resource sharing, and openness. Nevertheless, support for heterogeneity "remains a mostly unsolved problem" (Nesterenko & Jin, 2002, p.2).

6. Concurrency Testing

Concurrency was previously mentioned in section 2.2 as one of the key characteristics of distributed systems, and the details about concurrency are described in this section. This section is structured into three parts. First, it addresses a number of problems caused by concurrency in the context of distributed systems, together with various mechanisms that help to resolve these problems. Second, three sample distributed systems are tested in order to demonstrate and emphasise the vital role of concurrency testing. Finally, two open source concurrency testing tools, GroboTestingJUnit and ConAn, are compared and contrasted to identify the strengths and weaknesses.

6.1 Concurrency in Distributed System

The core of a distributed system is resource sharing which includes software, hardware, and data. During the time the resource sharing takes place, there could be a number of clients who access the same resource simultaneously, or a number of processes are run at the same time. This results in a number of issues. Anderson (2001) lists a number of issues that are caused by concurrency. For instance, a process may use old data, lost and inconsistent updates, the order of updates may matter, the system may go into a deadlock state, and the data in different systems might never converge to consistent values.

The first common problem is data inconsistency. This happens when different users access and update the same data source. According to Anderson (2001), data inconsistency is prevented by applying the *callback* and *locking* mechanisms. *Callback* is one approach to maintain data consistency, the server maintains a list of all the clients, the clients are notified after the service is performed. Another approach to maintain data consistency is *locking*. It allows the first transaction to lock an entity, and other cannot access it until the first transaction has committed or aborted. Warfield, Coady, and Hutchinson (2001) present two types of locking which are pessimistic locking and optimistic locking.

Pessimistic locking is an approach when a user accesses an entity in the database, this prevents other users from working with this entity until the first user completes the task. For instance, a write lock allows the holder of the lock to update the entity and disallows others to read, update, or delete the entity (Ambler, 2006). Pessimistic locking is easy to implement and it ensures that the database is consistent. The drawback of pessimistic locking is when the system has many users or transactions are long lived, thus, the time to wait for a lock to be released increases and limits the number of simultaneous users that the system can support (Ambler, 2006). Other examples of locking to manage multiple revisions of files are Revision Control System (RCS) and Concurrent Versions System (CVS). RCS and CVS ensure that there is only one person who has write access at any one time to any given part of the file.

Ambler (2006) explains optimistic locking is to accept the fact that collisions occur infrequently, and the collision is only resolved when it does occur. For example, a check is performed anytime there is an update to the data in order to determine whether there is a collision. If there is no collision, the data is updated. If there is a conflict, that needs to be resolved. Optimistic locking has some disadvantages such as slowing down the update process, and users are not notified that a record is modified until they try to update it (Haddad & Donoghue, 1998).

Data consistency is also maintained by the timestamp approach. Each transaction has a timestamp and this helps to identify the order of transactions. Timestamps are compared and transaction may proceed, delay, or abort. Timestamp also helps to overcome the deadlock problem when systems are waiting for each other to perform a specific task (Amiri, 2005).

6.2 Systems: Cathy's Online Car Store, CalME, and E-shopping

Three distributed systems which are Cathys's Online Car Store, Calme, and E-shopping, are tested for concurrency. The purpose of these tests is to assert the importance of concurrency testing when handling distributed systems. It was observed that the three systems failed in handling requests simultaneously. There are five scenarios which are chosen for testing concurrency. Each scenario is explained in detail below.

6.2.1 Cathy's Online Car Store

The application is designed for an online car store. A user should be able to add a new car, order/delete/search for an existing car, and view the showroom. Figures 8 and 9 show the simple user interfaces of the system.

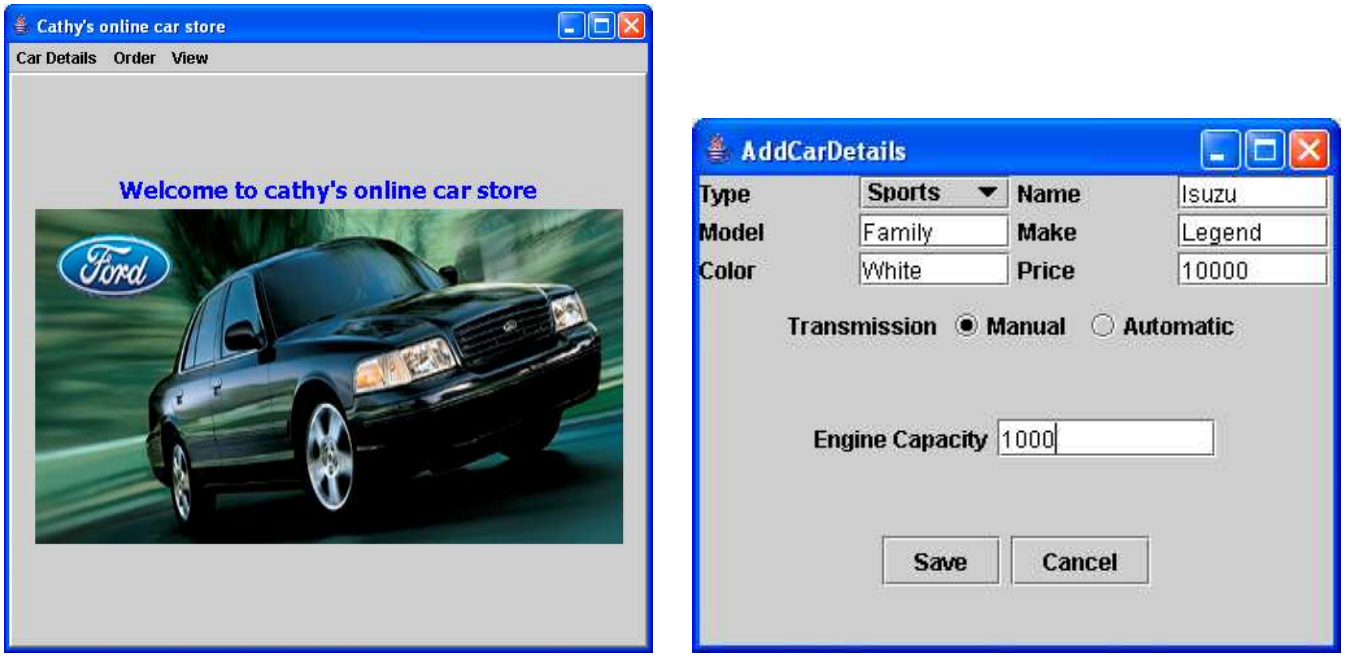


Figure 8: User adds a Car - Cathy's Online Car Store

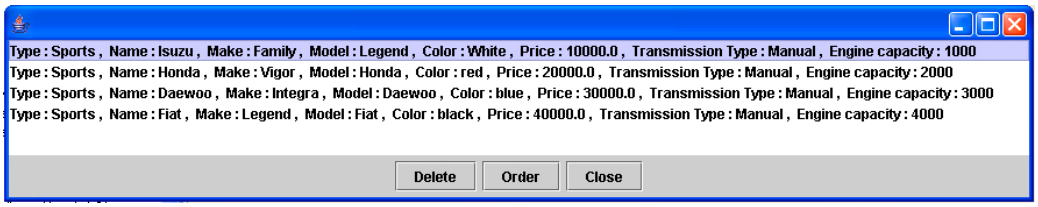


Figure 9: User Views/Deletes/Orders a Car - Cathy's Online Car Store

<u>Scenario 1:</u> ***While user A is in the process of deleting a car, user B selects the same car to be purchased.***

Two users are viewing the Cathy's Online Car Store's GUI application simultaneously. User A deletes car 3 from his application, at the same time, user B decides to purchase car 3. Although the two users are using the system concurrently, there is a chance that one task is completed before another task within a millisecond. In this case, user A completes his task before user B's task is performed. In other words, car 3 is deleted from the database but this change has not been updated on user B's GUI. As a consequence, when user B orders car 3, which no longer exists in the database, user B would accidentally order the next car (which is car 4) as a replacement of car 3. Figure 10 shows user A and user B's GUI, it also shows the response of the system in this circumstance. Source code for this test is listed in Appendix 1.

User A **User B**

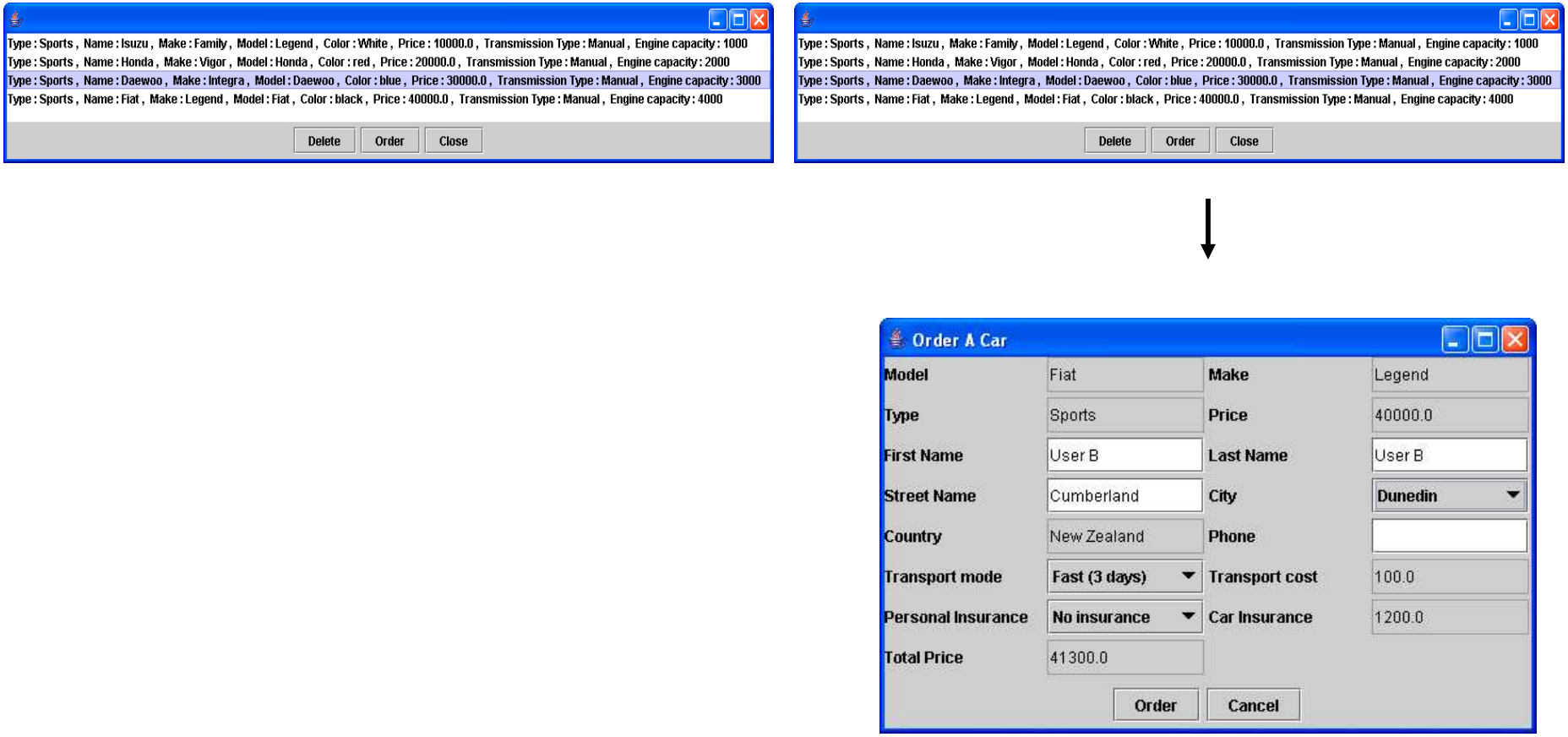
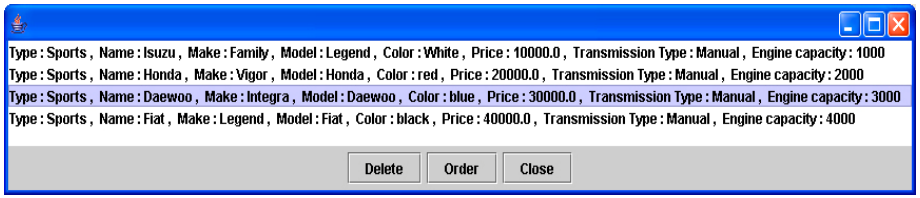


Figure 10: Scenario 1 – Cathy's Online Car Store

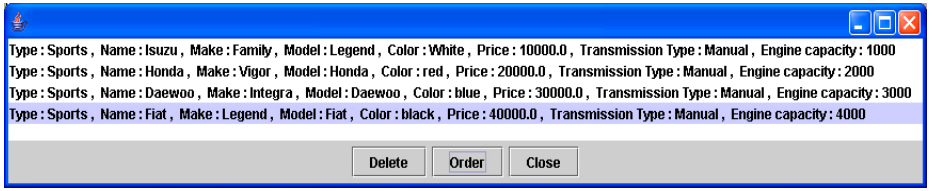
<u>Scenario 2:</u> ***While user A is in the process of deleting a car, user B selects another car to be purchased.***

This scenario is similar to scenario 1, but the test produces a different result. User A deletes car 3 from his machine, and at the same time, user B orders car 4. Similar to scenario 1, although the two users are using the system at the same time, there is a chance that one task is performed faster than the other within a millisecond. In this situation, user A deletes car 3 before user B orders car 4. As a result, car 4 replaces the position of car 3 in the database, and the original position of car 4 is now empty. This change has not been updated on user B's GUI, hence, user B cannot order car 4. Figure 11 demonstrates the error user B gets after ordering car 4.

User A



User B




```
java.lang.IndexOutOfBoundsException: Index: 3, Size: 3
      at java.util.ArrayList.RangeCheck(Unknown Source)
      at java.util.ArrayList.get(Unknown Source)
      at
GUI.OrderACar.populateDetails(OrderACar.java:603)
      at GUI.OrderACar.<init>(OrderACar.java:75)
      at
testCases.DeleteAndOrderCarTest.OrdACar(DeleteAndOrderC
arTest.java:55)
      at
testCases.DeleteAndOrderCarTest.main(DeleteAndOrderCarT
est.java:20)
               Exception in thread "main"
```

Figure 11: Scenario 2 – Cathy's Online Car Store

6.2.2 CalME

CalME is an application that is designed to run on Personal Digital Assistants (PDA) and is built using Peer-to-Peer architecture to operate in a wireless network. The CalME application is similar to Microsoft Outlook, which is designed for creating new groups and making appointments between members of a group. Users can create/edit/delete a group and an invitation will be send to all peers who are online at the time. Users can also create/edit/delete a personal appointment and create an appointment with other members of the group.

Scenario 3: ***Two concurrent users create two new groups that have the same name.***

This scenario aims to test one important aspect of the system, whether two groups with the same name can be created at the same time. Typically, the user should not be allowed to create a group that has the same name as the existing groups. The test reveals that although the rule is not violated when the two users use the system consecutively, it is violated when the two users use the system simultaneously. The output of this test is summarized in Table 2 and Figure 12. The source code of this test can be found in Appendix 2. It shows that two concurrent users A (Adam) and B (Jian) are still able to create two identical groups, which is INFO444. As can be seen, the system does not pass this test.

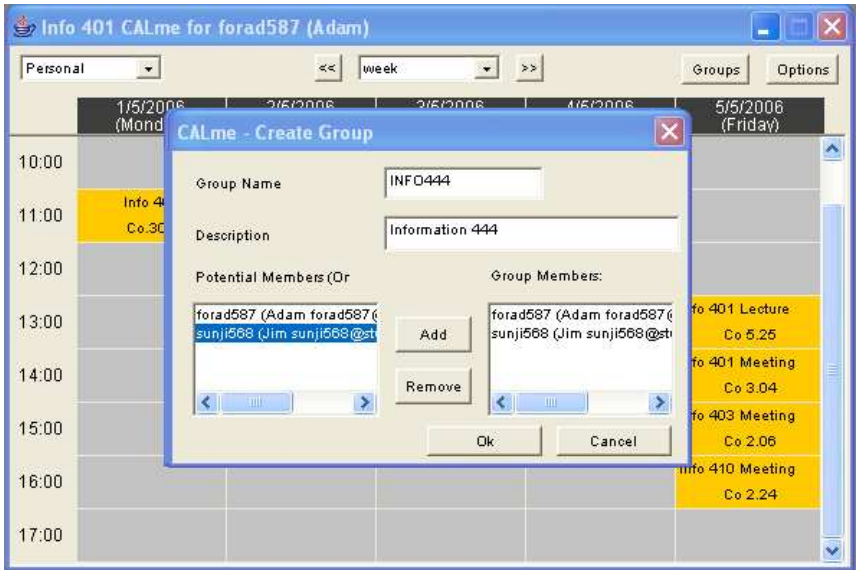
Test output:

User A - Adam	User B - Jian
Hours: 18 Minutes: 18 Seconds: 8 Milliseconds: 40 Executing... Adam creates an INFO444 group!	Hours: 18 Minutes: 18 Seconds: 8 Milliseconds: 40 Executing... Jian creates another INFO444 group concurrently!

Table 2: Scenario 3 – CalME output

Adam

Adam creates a new INFO444 group.



Jian

Jian creates a new INFO444 group.

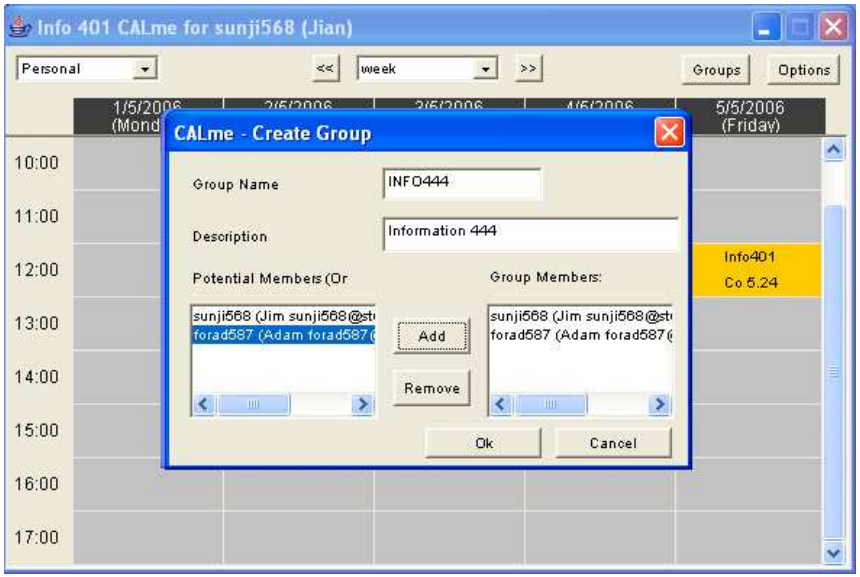


Figure 12: Scenario 3 – CalME

<u>Scenario 4:</u> ***Two concurrent users create two identical appointments for a group.***

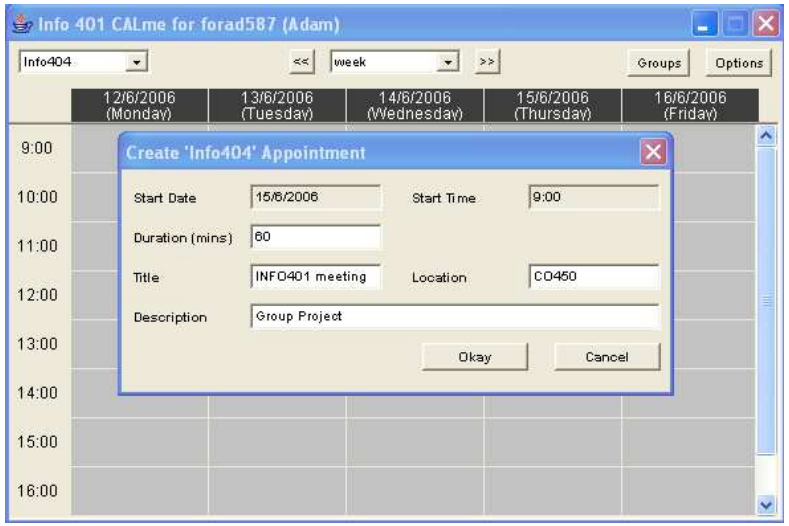
Similar to scenario 3, another aspect of CalME is to ensure that identical appointments of the same group should not be created. The test shows that this condition is not met when two users create similar appointments concurrently. The output of this test is summarised in Table 3. It proves that two concurrent users A (Adam) and B (Jian) are still able to create two identical appointments for the INFO401 group. Figure 13 shows the incorrect results that are displayed on both users' calendars. Source code for this test can be found in Appendix 3. The number of created appointments is 4 which is incorrect. This can be explained that when Adam creates a group appointment, this appointment is added in Jian's calendar together with Jian's appointment. This results in two appointments are displayed on each Jian and Adam's calendar (one appointment is from Adam and one appointment is from Jian). Unfortunately, at the same time Jian creates the same group appointment, this appointment is also added in Adam's calendar together with Adam's appointment. Hence, another two appointments are displayed on each Adam and Jian's calendar. As a result, the total number of appointments is four, in that, two appointments are from Adam's timetable and two appointments are from Jian's timetable. The system should display 2 since the group has only two members, user A - Adam and user B - Jian. Clearly, the system does not pass this test.

Test output:

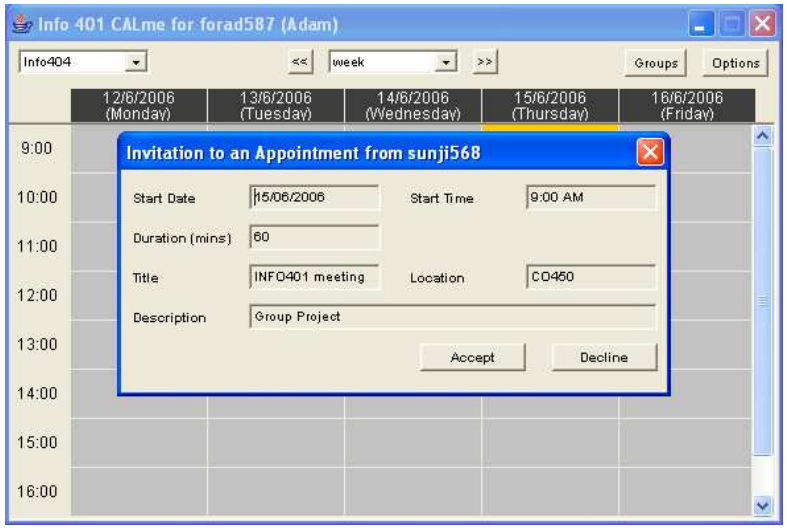
User A - Adam	User B - Jian
Hours: 18 Minutes: 34 Seconds: 25 Milliseconds: 375 Executing... Appointment 15/06/2006 09:00|60|CO450|INFO401 meeting|Group Project| has been created by Adam.	Hours: 18 Minutes: 34 Seconds: 25 Milliseconds: 375 Executing... Appointment 15/06/2006 09:00|60|CO450|INFO401 meeting|Group Project| has been created by Jian.

Table 3: Scenario 4 – CalME output

Adam

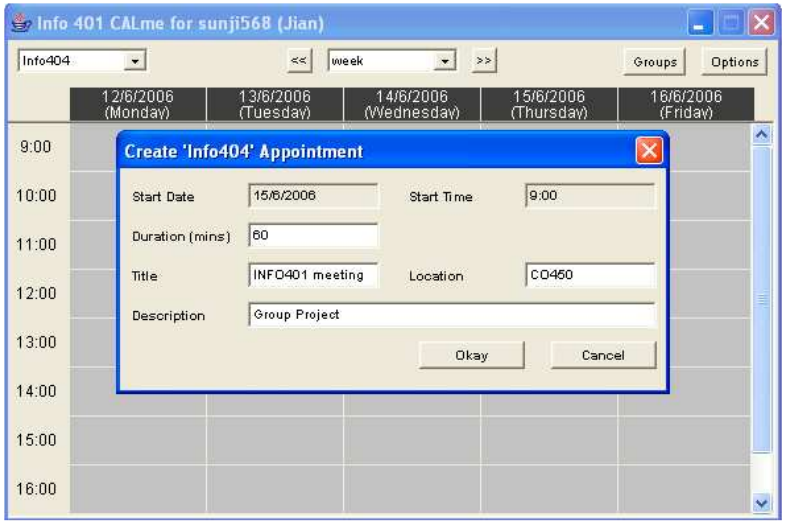


Adam creates a new Info404 appointment.

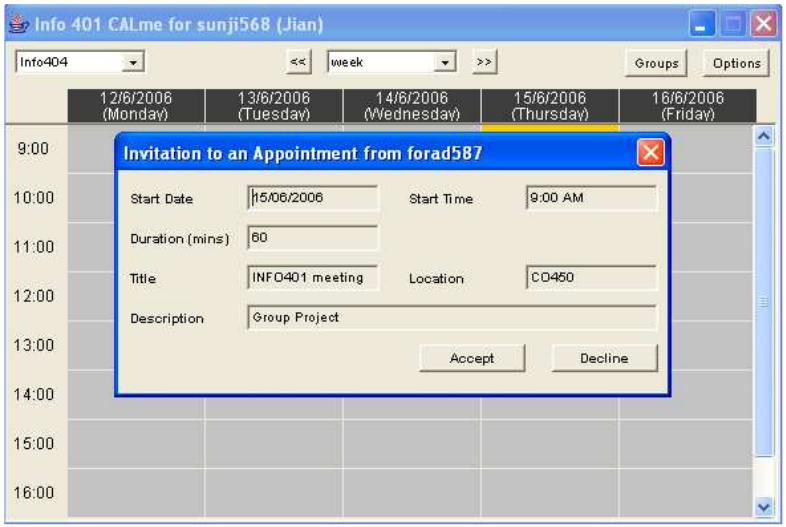


Adam receives an invitation from Jian.

Jian



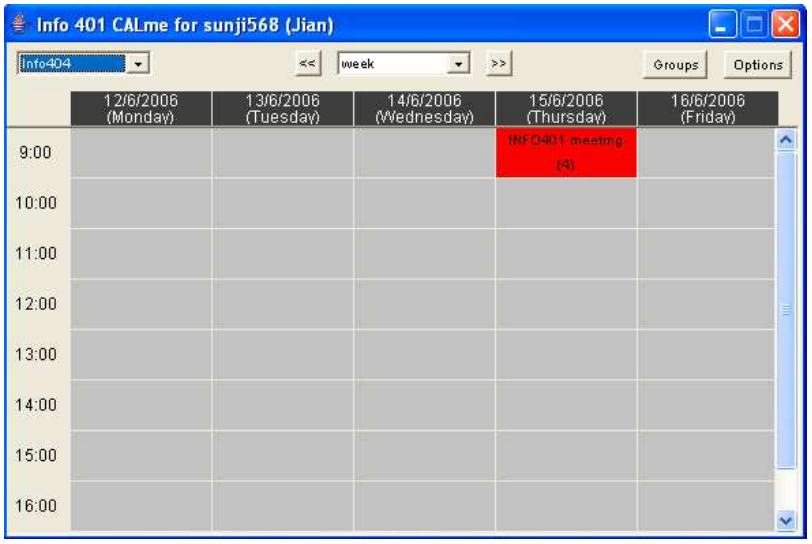
Jian creates a new Info404 appointment.



Jian receives an invitation from Adam.

Figure 13: Scenario 4 – CalME (also continued on next page)

There are 4 Info404 appointments are displayed on Adam's calendar.



There are 4 Info404 appointments are displayed on Jian's calendar.

Figure 13: Scenario 4 – CalME

6.2.3 E-Shopping

E-Shopping is an online application that permits users to add/view/edit/remove a category and to add/view/edit/remove a product that belongs to a category. Figures 14 and 15 show three main GUI of the system.

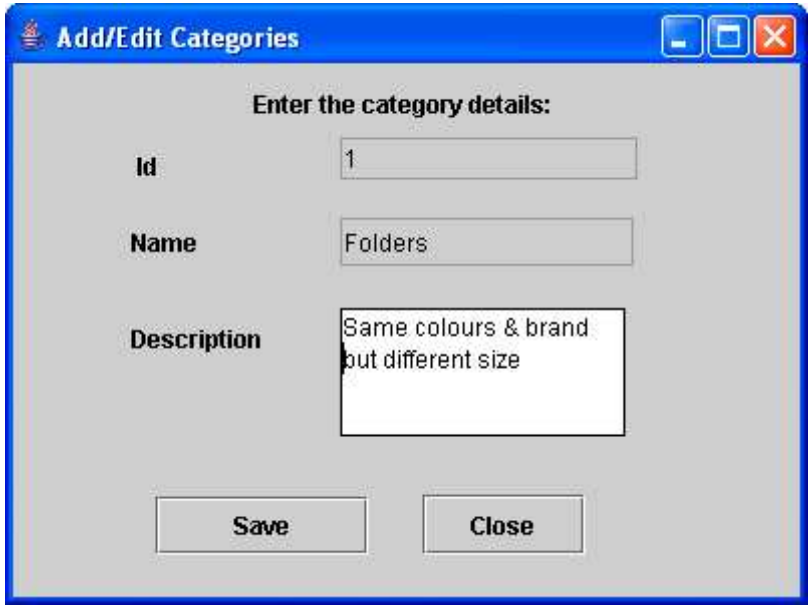
Figure 14: User Enters a New Category - E-Shopping

Figure 15: User Views/Edits/Removes a Category - E-Shopping

<u>Scenario 5:</u> ***User A and user B edit the same category simultaneously***.

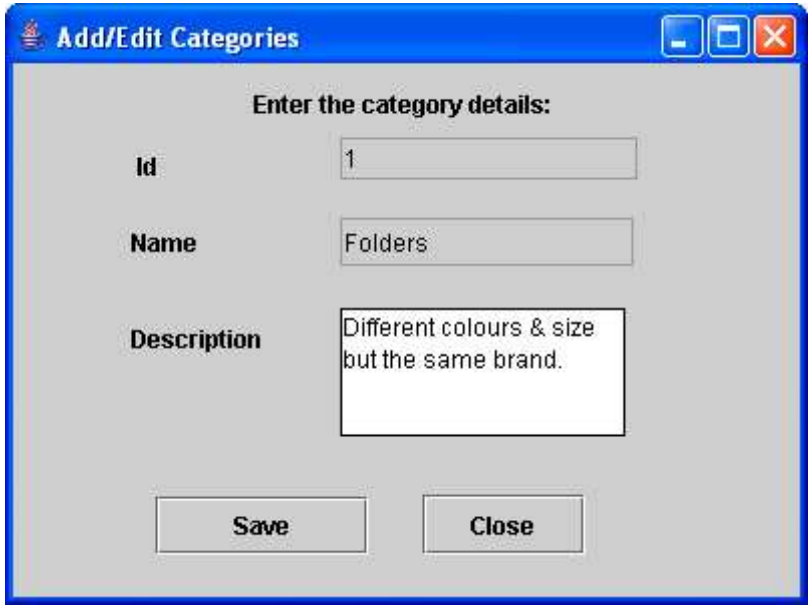
This scenario tests how the system behaves when two concurrent users access the same database. According to Figure 16, user A changes the description of item 1 from "Different colours & size" to "Same colours & brand but different size", simultaneously, user B changes the description of item 1 from "Different colours & size" to "Different colours & size but the same brand". When the event happens, user B does not know that the description of item 1 has already been changed by user A. The test verifies that the GUI has failed in updating the change to the users. Source code of this test can be found in Appendix 4.

User A



User A edits the description of category 1

User B



User B edits the description of category 1

Figure 16: Scenario 5 – E-Shopping

6.3 Tools: GroboTestingJUnit and ConAn

Cohen has emphasised that "measuring concurrency is a great way to measure how efficiently the system shares its resources (2004, p.385). Section 5.2 demonstrated a number of scenarios that were tested, and also found that these systems have failed to meet the concurrency requirement. It becomes clear that concurrency testing plays a significant role in testing distributed systems. During the process of testing these sample systems, test cases were designed in Java without the aid of external tools due to various difficulties faced when using these tools. These difficulties are identified by comparing and contrasting two open source testing tools that support concurrency testing. GroboTestingJUnit and ConAn are chosen amongst other open source tools because GroboTestingJUnit is a common tool when performing multithreaded testing and easy to integrate its API with JUnit. On the other hand, ConAn is a new concurrency testing framework but promises to be a good testing tool in the near future. Each of these tools is explained in detail in the following three sections.

6.3.1 GroboTestingJUnit (version 1.1.0)

GroboTestingJUnit is a subproject of GroboUtils, an open source project written by Albrecht (2003) and is distributed under the MIT license. GroboUtils is an extension of JUnit, which is very common to most testers when performing unit testing. According to GroboUtils online documents, GroboUtils covers multi-threaded tests, hierarchical unit tests, and a code coverage tool. The scope of this section is to look at GroboTestingJUnit from the multi-threaded perspective.

Rupp (2003) points out that JUnit has some problems in testing multithreaded programming, in other words, JUnit does not support multithreading. It is because JUnit finishes execution while the threads are still alive, this is caused by the JUnit's TestRunner. This problem is solved by the GroboTestingJUnit. Rupp (2003) explains that GroboTestingJUnit contains the MultiThreadedTestRunner and TestRunnable classes. Unlike JUnit's TestRunner, MultiThreadedTestRunner waits until all threads have

terminated and this forces JUnit to wait while the threads are performing the task. Apart from this, it has a very clear and simple API library.

6.3.2 ConAn (version 5.0)

ConAn is developed by Strooper, Duke, Wildman, Goldson, and Long (2004) at the University of Queensland, Australia. The tool has some features that has not been completed and is still under developments for enhancements. Based on Roast which is a framework to support automated testing of Java Application Programming Interfaces (API), ConAn is built as an extension to support concurrency testing of Java API. Long, Hoffman, and Strooper (2001) introduce the ConAn tool and its testing technique. Testers are required to understand the test scripts and commands of the ConAn and Roast tools. Testers also need to specify a sequence of method calls and the order of these methods calls. Generally, a ConAn file that contains ConAn and Roast test scripts is written to test a particular Java class. When the ConAn file is compiled and run, the test scripts generate a java test class and some other classes. This Java test class is used to test the original Java class.

The ConAn tool provides a compatible framework for concurrency testing of concurrent Java components. However, it does have some drawbacks. Firstly, testers need to understand the test scripts. Secondly, a number of features have not been implemented such as the Integrated Development Environment (IDE) that allows ConAn to be used straightforwardly, for instance, it will be easier to edit and debug the source code. Thus, it is fundamental to have a plug-in for ConAn in Eclipse so that the tool becomes more capable to run on different platforms. To achieve this, Strooper (2005) identifies a number of tasks that needs to be completed. For example, an XML format for specification of ConAn test cases needs to be designed, and a framework for running ConAn generated tests with JUnit is required. In addition, a plug-in for Roast is needed. Another disadvantage of ConAn is the lack of support and help features. Although the tool provides the help documentation, it is impossible to discuss some ConAn issues with other users except contacting the developers directly. This would be better if there is a discussion board to receive questions and suggestions. Because of these difficulties,

ConAn is not used to test the sample applications that have been demonstrated in section 5.2.

6.3.3 Summary

GroboTestingJUnit and ConAn are evaluated to identify their strengths and weaknesses. Depending on the purpose of the test, testers can make a decision of which tool should be used. Concurrency testing can be performed by the combination of different tools to achieve the best result. The strengths and weaknesses of GroboTestingJUnit and ConAn are summarised in Table 4.

Tools / **Features**	**GroboTestingJUnit**	**ConAn**
Multithreaded Support	√	√
IDE Plugability	√	
Clear and Easy to Use API	√	√
Java Language Support built on top of JUnit	√	
Graphical Test Result Analysis (i.e. graphs, charts)		
Code Coverage Feature	√	
Hierarchical Unit Tests	√	
MIT Open Source License	√	√
XML Format		
Roast and ConAn Test Scripts		√
Maintainability and Support	√	

Table 4: GroboTestingJUnit and ConAn

7. Scalability Testing

Scalability was previously mentioned in section 2.2 and its details are explained in this section. This section is divided into three parts. The first part explains a number of reasons for maintaining scalability and some techniques to achieve scalability in distributed systems. The second part shows three sample web based systems which are tested by using TestMaker, one of the open source software, in order to highlight the helpful role of scalability in the context of testing. The third part presents three scalability testing tools such as TestMaker, OpenSTA, and httperf to identify the strengths and weaknesses of each tool.

7.1 Scalability in Distributed System

Distributed systems tend to get larger when there is an increase in the number of workstations and resources. Neuman (1994) categorises three different dimensions that are used to measure the scalability of a system, which are the size of the system, geographical scalability, and administrative scalability. These measurements ensure that although the users and resources may lie far apart, it is still easy to manage. Scalability testing provides the information regarding whether the system can still function in a fast manner under heavy load. This information assists the IT manager to decide whether additional servers should be included.

There are a number of issues caused by scalability and these are mentioned by Tanenbaum and Steen (2002). Centralized services, data, and algorithms can become problematic when the number of users increases. Although having a single server to serve a large number of users is not a good idea, adding additional servers to increase the system performance is also not practical in some situations. For example, the information on the server is highly confidential, such as information about bank accounts, adding additional servers create more chances for security attacks. Another issue is geographical scalability which also slows down the communication between clients and server.

There are some techniques that can help to maintain and improve scalability in distributed systems. The most common technique is replication which adds more and faster processors as well as design components to be scalable (Emmerich, 1997). Several copies of the same server are made available, the requests are sent to the servers based on their physical location, the loads of the server, or can be sent randomly. Consistency should be taken into consideration when this technique is used. It is to ensure that users can only view the up-to-dated information and the information appears the same to all users. This technique is not suitable when handling sensitive data, which was mentioned in the previous section.

Neuman (1994) proposed another technique which is known as distribution. Distribution allows a component to be split into smaller parts and these parts are spread across the system. For instance, a name space is divided into different parts and assigned part of the naming database to different servers. Hence, it reduces the number of queries and updates to be processed (Neuman, 1994), also each request is handled faster since the size of the database is smaller. Caching technique is also introduced by Neuman (1994) which allow the result to be remembered, thus, additional requests for the same information can be reduced.

According to Tanenbaum and Steen (2002), the hiding communication latencies technique can be used to enhance the performance of a system. That is to say, after the client sends the request, the client continues to do other tasks rather than await for the response from the server. When the response arrives, the application is interrupted to complete the previous task.

7.2 Systems: OLC Assets, DCRR, and Employee Directory

The OLC Assets, the DCRR, and the Employee Directory are web based systems and their functionalities are described in sections 6.2.1, 6.2.2, and 6.2.3, respectively. These three systems are chosen for the scalability test which is to test how each server reacts when there is an increase in number of users. These tests aim to highlight the role of scalability testing in analysing system performance. These tests also explain how scalability testing can assist testers in making important decisions such as whether the current server(s) are able to serve an increase in the number of concurrent users, or whether additional servers should be implemented in order to handle a high volume of users.

The three systems are tested using a range of concurrent virtual users (2 to 500 virtual users) in one minute. Because of the limited resources, the maximum number of virtual users was set to 500. The output graphs (Figure 20, 23, and 27) which start from 25 virtual users are redrawn based on the original graphs because they do not show the ratio between numbers of virtual users appropriately. The original graphs are generated by TestMaker, one of the open source software that supports scalability testing. Further information about the original graphs can be found in Appendix 5, 6, 7. The information that can be observed from these graphs is described below.

Each graph describes the number of successful transactions per second versus the number of virtual users. The term “transactions per second” is defined as the number of pages per second a server can generate (Lampert & Lvov, 2006). It can be seen from the graph that as the number of users rises, the number of transactions per second gradually increases. This indication shows that the server is still able to handle a large amount of concurrent requests. On the other hand, if the number of transactions per second drops as the number of users increases, this indicates that there is a concurrency problem and hence a scalability problem. Although the results from testing the three web based systems verify that the servers are still able to manage a large amount of concurrent requests, it is

believed that when the number of users keeps growing, the server will fail to handle these requests at some point.

7.2.1 OLC Assets

Everyday, the Otago Language Centre receives a number of new items to facilitate students in studying English. These items can be library books, hardware, software, TVs, desks, chairs, etc. These items are entered into the Centre's database for future maintenance. OLC Assets System is a web based system that grants users access to enter details of each item into the database by completing a simple form. Figures 17 and 18 present the two main interfaces where a user can enter an item into the database. In addition, users can view the table of existing items as described in Figure 19. The performance of the system is tested under the situation where there are multiple users who have entered a number of items into the database.

Figure 17: Main Page - OLC Assets System

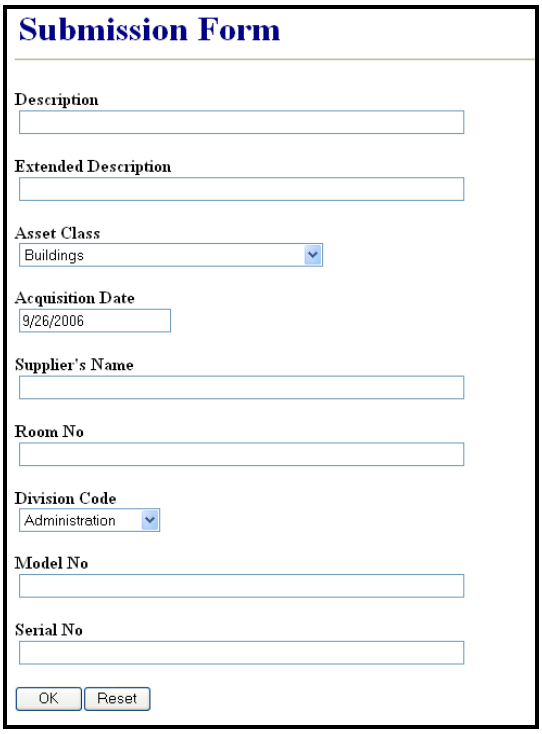
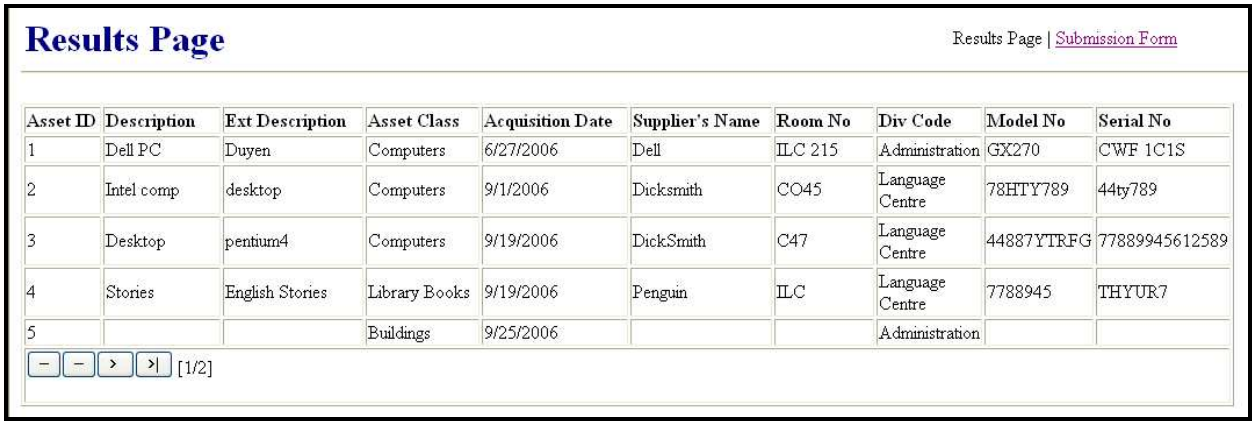


Figure 18: Submission Form - OLC Assets System



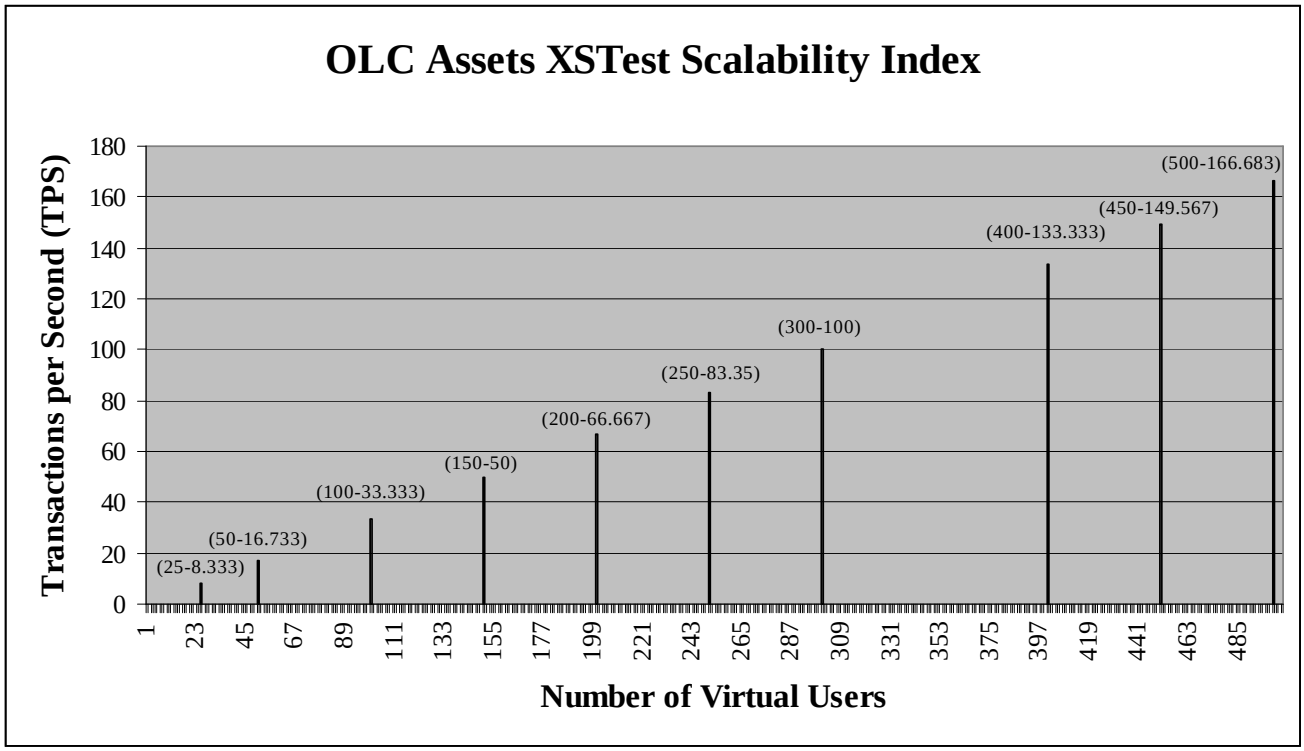
Results Page

Results Page | Submission Form

Asset ID	Description	Ext Description	Asset Class	Acquisition Date	Supplier's Name	Room No	Div Code	Model No	Serial No
1	Dell PC	Duyen	Computers	6/27/2006	Dell	ILC 215	Administration	GX270	CWF 1C1S
2	Intel comp	desktop	Computers	9/1/2006	Dicksmith	CO45	Language Centre	78HTY789	44ty789
3	Desktop	pentium4	Computers	9/19/2006	DickSmith	C47	Language Centre	44887YTRFG	77889945612589
4	Stories	English Stories	Library Books	9/19/2006	Penguin	ILC	Language Centre	7788945	THYUR7
5			Buildings	9/25/2006			Administration		

[1/2]

Figure 19: Results Page - OLC Assets System



Number of Virtual Users	25	50	100	150	200	250	300	400	450	500
Transactions per Second (TPS)	8.333	16.733	33.333	50	66.667	83.35	100	133.333	149.567	166.683

Figure 20: OLC Assets XSTest Scalability Index

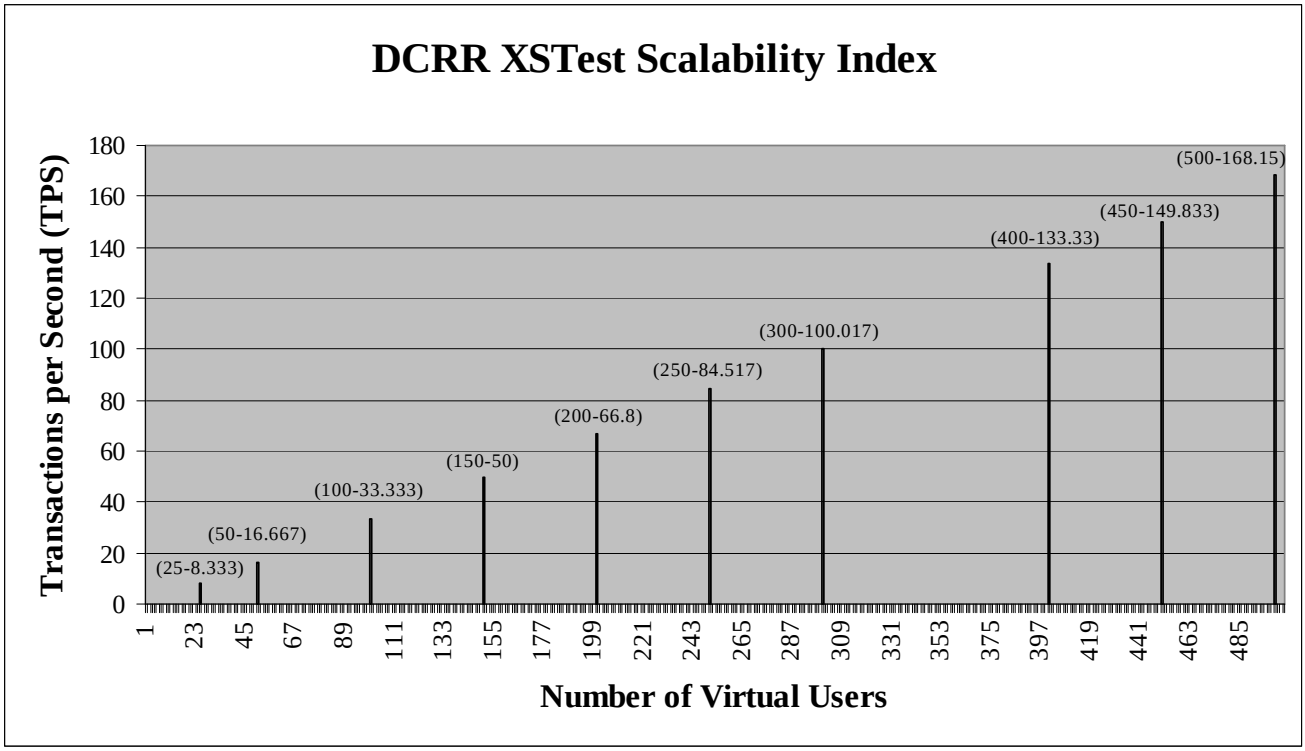
It can be observed from Figure 20 that when there are 25 users, the transaction per second is 8.333 and when there are 500 users, the transaction per second increases to 166.683.

7.2.2 DCRR

DCRR is a web based system that provides users access to search for a particular restaurant, to comment about a restaurant via an electronic evaluation form, and to add a new restaurant into the database. Figures 21 and 22 show the two main interfaces of the DCRR system. A number of virtual users accessing the system to add a new restaurant is taken as the scenario for this performance testing.

Figure 21: Search for a Restaurant - DCRR System

Figure 22: Add a New Restaurant - DCRR System



Number of Virtual Users	25	50	100	150	200	250	300	400	450	500
Transactions per Second (TPS)	8.333	16.667	33.333	50	66.8	84.517	100.017	133.33	149.833	168.15

Figure 23: DCRR XSTest Scalability Index

It can be observed from Figure 23 that when there are 25 users, the transaction per second is 8.333 and when there are 500 users, the transaction per second increases to 168.15.

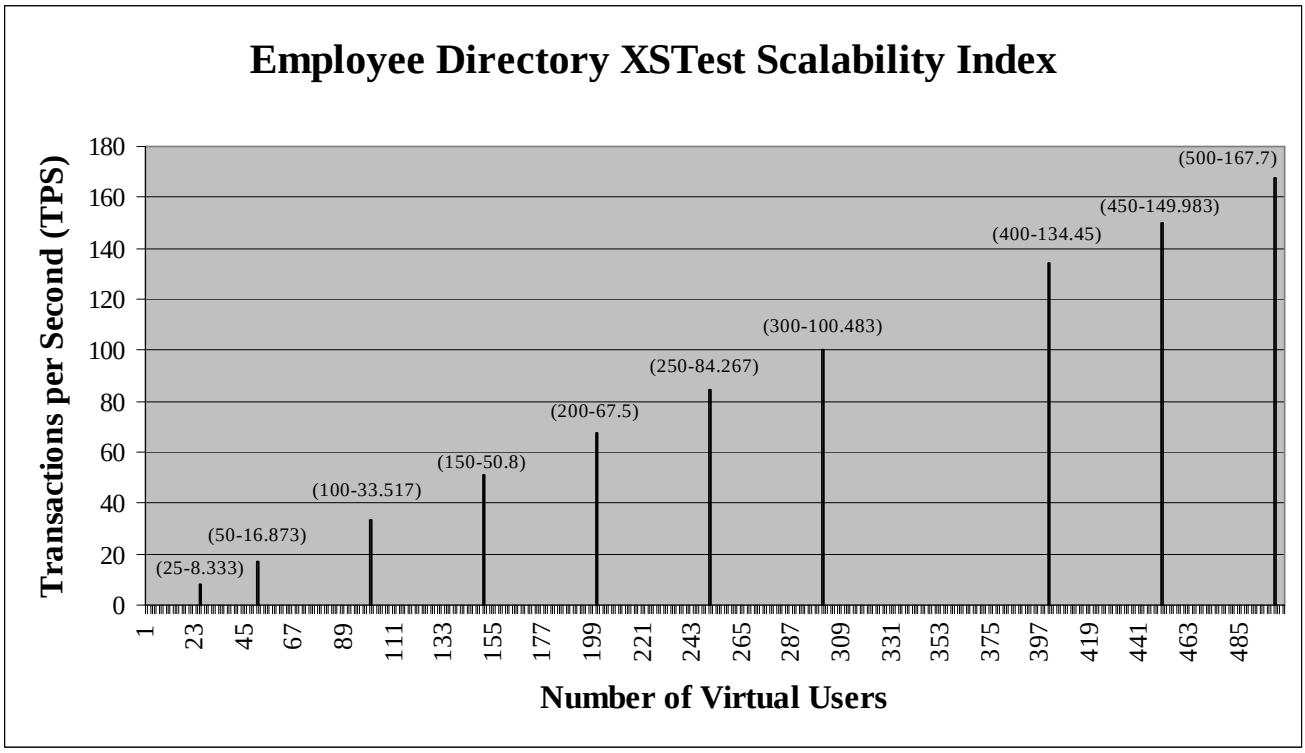
7.2.3 Employee Directory

Employee Directory is a web based system that allows users to search for a particular employee and view the details of that employee. Three main interfaces of the system are shown in Figures 24, 25, and 26. Performance testing is conducted in the situation where there are an increasing number of users searching for a particular employee.

Figure 24: Main Search Page - Employee Directory System

Figure 25: Result Page - Employee Directory System

Figure 26: Details of an Employee - Employee Directory System



Number of Virtual Users	25	50	100	150	200	250	300	400	450	500
Transactions per Second (TPS)	8.333	16.873	33.517	50.8	67.5	84.267	100.483	134.45	149.983	167.7

Figure 27: Employee Directory XSTest Scalability Index

It can be observed from Figure 27 that when there are 25 users, the transaction per second is 8.333 and when there are 500 users, the transaction per second increases to 167.7.

7.3 Tools: TestMaker, OpenSTA, and httperf

Section 6.2 illustrated three web based systems that were tested. It becomes clear that scalability testing can help to provide valuable information about the performance of a system. Because of this, there are a number of available scalability testing tools that can be used. This section examines the advantages and disadvantages of three open source scalability testing tools which are TestMaker, OpenSTA, and httperf. The functionalities of these tools are explained below.

7.3.1 TestMaker (version 4.4)

TestMaker is a comprehensive testing framework that is used for scalability, functionality, and load testing. TestMaker can create a variety of test cases. Based on the information provided in TestMaker documentation (2006), TestMaker can support the Open Internet Protocols (HTTP, HTTPS, SOAP, XML-RPC, SMTP, POP3, IMAP, etc.). Test cases are from the WSDL definition of a SOAP-based Web Service, the forms found in an HTML Web page, and JUnit test cases. The software allows testers to create a number of virtual users and with the support from TestNetwork, testers can generate up to 10,000 or more concurrent test agents. TestMaker generates a clear output graph that represents the performance of the server based on the number of transactions per second versus the number of virtual users. The original output graphs can be viewed in Appendices 5, 6, and 7.

Apart from the advantages that are provided in the open source version, the commercial version (TestMaker Platform) offers some additional features such as the XSTest Pro that delivers scalability index helps to identify scalability problems for capacity planning. Service Monitor System can send notifications via emails to notify when a service fails or responds too slowly. Reports (charts and graphs) are generated to show the scalability index, result timing distribution, and performance timing by operation. This is an extremely useful feature since it is the quickest way to analyse the performance of the system. Another feature is TestNetwork, which was also mentioned previously.

TestNetwork allows testers to generate a large amount of virtual users to test a specific web service.

On the contrary, TestMaker does have some disadvantages. According to Appendices 5, 6, and 7, graphs are generated based on the number of transactions per second. In doing so, it assumes that all transactions are the same. However, there is no standard transaction because a transaction can be long, short, or complex. There is a possibility that a complex transaction may take shorter time to complete and this is not reflected on the graph. In addition, test cases should be performed more than one time to get the correct outputs.

7.3.2 OpenSTA (version 1.4.3)

OpenSTA is a completely free framework for performance testing Web Application Environments (WAEs). It is designed to create and run HTTP/S load tests in order to assess the performance of WAEs. OpenSTA provides a very simple GUI to create a new test, new collector, and new script.

A new script is written in Script Control Language (SCL) and created by recording a list of activities that are carried out by the users. These activities are decided based on how testers would like users to use the web application. Scripts are then modified so that it can be used in tests to function as one or more virtual users during the time the test is run. According to OpenSTA documentation (2005), the process of creating collectors involves "deciding which host computers or other devices to collect performance data from and the type of data to collect". After scripts and collectors are created, they are added into a test. Each test can be edited and controlled by using a number of features that are provided on the GUI. These features include description about the test, start time, number of iterations, host name, number of virtual users, duration of the test, delay between each iteration, etc. During the time a test is run, it can be monitored and the result is displayed in graph when the test completes. Graphs can be customised to improve the presentation of data. Finally, the software provides a very good help documentation together with online help and commercial support. One disadvantage is that OpenSTA can only support

HTTP and HTTPS protocols, it can only run on Windows OS. Another disadvantage of OpenSTA is that testers have to understand the SCL language.

7.3.3 httperf

httperf is a tool for measuring web server performance. A test is run by using the command line to specify hostname, port, page address, rate, number of connections, and timeout. Timeout features help testers to define the number of seconds that each client is willing to wait for the response from the server. When a test completes, the result is generated in the form of plain text. Mosberger and Jin (1998) explain that the result output consists of six groups. These groups cover the overall results, results pertaining to the TCP connections, results for the requests that were sent, results for the replies that were received, CPU and network utilization figures, and a summary of the errors that occurred. A performance graph can be also generated by httperf to illustrate the server performance.

httperf has some weaknesses such as it only supports the HTTP protocol and can only run on a Linux OS. In addition, Mosberger and Jin (1998) point out that the testers have to start httperf on each client machine, collect and summarise each client-result. They also suppose that a single command line that can control multiple clients could help in improving httperf.

7.3.4 Summary

TestMaker, OpenSTA, and httperf are chosen amongst other testing tools to identify their advantages and disadvantages. The strengths and weaknesses of each tool are summarised in the following table.

Features \ Tools	TestMaker	OpenSTA	httperf
Unix OS	√		√
MacOS X	√		
Window OS	√	√	
Support JUnit Test	√		
Commercial Version	√		
Open Source Version	√	√	
Extensible Library of Protocols (HTTP, HTTPS, SOAP, XML-RPC, SMTP, POP3, IMAP)	√	Only HTTP & HTTPS	Only HTTP
Friendly and Easy to Use GUI	√	√	
Support J2EE, .NET	√		
Support for Test Result Analysis (i.e. graphs, charts)	√	√	
Object Oriented	√		√
Maintainability and support	√	√	
Agent Recorder	√	√	
Sample Test Agent	√		
Run Test from Command Line	√		√
Support for Virtual Users	√	√	√
P2P Support			
GPL Open Source License	√	√	
Script Control Language (SCL)		√	
Transaction per Second	√		
Calculate the rate of data transfer (bytes/second)		√	
Timeout Management		√	√

Table 5: TestMaker, OpenSTA, and httperf

8. Fault Tolerance Testing

Fault tolerance was introduced briefly in section 2.2 as one of the key characteristics of distributed systems. This section is divided into two parts. The first part identifies a number of failures that can occur in distributed systems in conjunction with the solutions to achieve fault tolerance. The second part outlines some possible situations where failures might be occurred in two sample distributed systems which are CalME, and OLC Assets.

8.1 Fault Tolerance in Distributed System

Generally, distributed systems aim to perform without being seriously affected by any failures that might happen in the system. If there is a failure, the system is still able to recover and the users are unable to recognise this failure. Tanenbaum and Steen (2002) explain that a system is said to be fault tolerant if it maintains the four characteristics such as availability, reliability, safety, and maintainability. Availability ensures that the system is available to be used at any given moment. Reliability allows the system to perform continuously without failure. Safety makes sure that in the event of failure and when the system is unable to operate, it should not cause any undesirable behaviour that comprises the safety of the system and the users. Finally, maintainability measures how straightforward a failure can be repaired. There are different types of failures that can happen. Cristian, Dolev, Strong, and Aghili (1987), and Tanenbaum and Steen (2002) classify these failures and they are listed below.

- ***Communication Failure*** happens when a message may be delayed, lost, duplicated, reordered, and corrupted. The client cannot locate the server; the server may be crashed after receiving request, and the client may be crashed after sending a request.
- ***Byzantine Failure*** relates to the Byzantine Generals Problem which was introduced by Lamport, Shortak, and Pease (1982) and based on the Byzantine army concept. The Byzantine failure occurs when a server does not behave in the correct manner. For instance, server produces an output which it should not produce or does not produce any outputs.

- ***Omission Failure*** occurs when a server fails to respond to a request based on different reasons such as connection failures.
- ***Crash Failure*** is a subclass of omission failure, it occurs when a server systematically omits all outputs and nothing is heard from that server anymore.
- ***Timing Failure*** occurs when a server either omits the specified output, or responds too early or too late. In the situation where the server responds too early, this causes trouble for recipient if there is not enough buffer space to hold all the incoming data. On the other hand, if the response is too slow, then it will affect the performance of the system.
- ***Response Failure*** happens when the server's response is incorrect. The server might provide the wrong reply to a request or it reacts unexpectedly to a request.

There are a number of techniques that can help to achieve fault tolerance in distributed systems. The most common technique is redundancy. Developers can apply information redundancy, time redundancy, and physical redundancy. These three types of redundancy are explained by Krzyzanowski (2002). Information redundancy provides fault tolerance through replicating or coding the data. Time redundancy allows an operation to be performed several times. Finally, physical redundancy deals with hardware or software, it runs an extra hardware or software at the same time to provide the correct output.

Another technique is *checkpointing* and *rollback*. *Checkpointing* saves system states at regular intervals. Developers should decide how often to save and how much to save. This prevents from loosing a large amount of data when a failure occurs. Additionally, *rollback* recovery helps the system to rollback to the previous state, this requires a stable storage (Saluja, 2005).

8.2 Systems: CalME and OLC Assets

Fault tolerance testing is a process of investigating how a system performs under certain faulty conditions. The most popular technique when testing fault tolerance is applying the fault injection techniques. Ghosh and Mathur (1999) proposed that fault injection is carried out by inserting faults at certain locations in the code of the program and

monitoring how the system behaves when these faults occur. The following two systems demonstrate some possible situations that might happen when trying to perform fault tolerance testing.

8.2.1 CalME

The CalME application was described in section 5.2.2. CalME is built by applying the P2P architecture. The communication between peers is tested for fault tolerance. The objective is to monitor how the system reacts when a peer goes offline unexpectedly. The test is carried out with three peers online. One of the peers goes offline but the communication between the other two peers is still maintainable. The two remaining peers receive a notification notifying that the other peer has left the network. The test shows that when a peer is offline, this does not affect the network and the communication between other peers.

8.2.2 OLC Assets

The OLC Assets system was described in section 6.2.1. A fault can be created by removing one of the servers to assume that the server crashed while serving the clients. The system should be monitored continuously in order to observe how another backup server can immediately handle all the requests without the loss of information in the event of failure.

9. Conclusion and Future Work

Designing, building, and especially testing distributed system is not a simple process, but it is not impossible. There are a number of open source and commercial testing tools that support developers and testers in testing distributed systems. This reduces the complexity and the heavy burden of testing.

This dissertation presented a number of issues that are identified when building distributed system together with the key solutions to these issues. The dissertation also emphasised the important role of distributed system testing by examining some sample distributed systems in the context of concurrency testing, scalability testing, and fault tolerance testing. The dissertation not only introduces some open source testing tools that can be put in practice but also compares and contrasts these tools against a number of criteria.

The work conducted in this dissertation provides a useful resource for developers and testers, who would like to make use of programming knowledge and exploit available testing tools to enhance the performance of distributed systems. However, all the features needed for distributed system testing are not completely supported by open source tools. Thus, in the testing of future distributed systems, it is hoped to develop a testing framework that can be integrated as part of the common IDE which can be used to generate, execute, and analyse the required test cases that satisfy most of the issues which were noted throughout this dissertation.

10. References

Albrecht, M. (2003). *Multithreaded Tests with JUnit*. Retrieved June 6, 2006, from http://today.java.net/pub/a/today/2003/08/06/multithreadedTests.html

Amiri, K. (2005). *Timestamp Ordering & Database Recovery*. Retrieved September 17, 2006, from http://www.doc.ic.ac.uk/~amiri/DB05/Lecture8-9.pdf

Ambler, S. W. (2006). *Agile Database Techniques – Effective Strategies for the Agile Software Developer*. America: John Wiley & Sons.

Anderson, R. J. (2001). *Security Engineering: A guide to Building Dependable Distributed Systems*. Canada: John Wiley & Sons, Inc.

Burback, R. L. R. (1998). *No Global Clock*. Retrieved August 15, 2006, from http://www-db.stanford.edu/~burback/dadl/node91.html

Cohen, F. (2004). *Java Testing and Design: From Unit Test to automated Web Tested*. America: Prentice-Hall.

Cristian, F., Dolev, D., Strong, R., & Aghili, H. (1987). Atomic Broadcast In A Real-Time Environment. In Simons, B. & Spector, A. (Eds), *Fault-Tolerance Distributed Computing* (pp. 51-71). Germany: Springer-Verlag.

Dorfman, M. (1997). Requirements Engineering. In Thayer, R. H. & Dorfman, M. (Eds.), *Software Requirements Engineering* (pp. 7-22). IEEE Computer Society Press.

Dunedin City Restaurant Review. (2005). Retrieved May 25, 2006, from http://info-nts-12.otago.ac.nz:8080/DCRR/Main_Page.jsp

Dustin, E. (2002). *Effective Software Testing: 50 Specific Ways to Improve Your Testing*. America: Addison Wesley Professional.

Eickelmann, N. S., & Richardson, D. J. (1996). An Evaluation of Software Test Environment Architectures. *IEEE Proceedings of ICSE, 18*, 353-364.

Emmerich, W. (1997). *Distributed System Principles.* UK: University College London. Retrieved June 20, 2006, from http://www.cs.ucl.ac.uk/staff/ucacwxe/lectures/ds98-99/dsee3.pdf

Employee Directory System - University of Otago Language Centre. (2006). Retrieved July 15, 2006, from http://139.80.15.230/Emp_dir

Fagerström, J. (1988). Design and Test of Distributed Applications. *Software Engineering – Proceedings of the 10th International Conference*. 88-92.

Ghosh, S. & Mathur, A. P. (1999). *Testing for Fault Tolerance*. Retrieved July 10, 2006 from, http://citeseer.ist.psu.edu/ghosh99testing.html

Goel, A. (2004). *Advances in Distributed Systems*. Canada: University of Toronto.

GroboTestingJUnit. (2004). *JUnit Extension*. Retrieved July 4, 2006, from http://groboutils.sourceforge.net/testing-junit/

GroboUtils Project. (2003). Retrieved July 4, 2006, from http://groboutils.sourceforge.net/

Haddad, P. & Donoghue, T. (1998). Choosing an Approach for Locking. Retrieved July 20, 2006, from http://developer.apple.com/documentation/LegacyTechnologies/WebObjects/WebObjects_5/Topics/ProgrammingTopics.2a.html

Hierons, R. M. (2004). *Testing a Distributed System: Generating Minimal Synchronised Test Sequences That Detect Output-Shifting Faults*. Retrieved August 25, 2006, from http://people.brunel.ac.uk/~csstrmh/papers/ist01a-pre.pdf

Hillston, J. (2002). *Distributed Systems*. UK: The University of Edinburgh.

IEEE Standards Board 610.12-1990. (1990). *IEEE Standard Glossary of Software Engineering Terminology*. America: The Institute of Electrical and Electronic Engineers.

Indiana University – University Information Technology Services. (2006). *What is a Distributed Application?* Retrieved June 20, 2006, from http://kb.iu.edu/data/adob.html

Ireland, A. (2006). *The Software Testing Life-Cycle*. Edinburg: Heriot-Watt University.

Katzan, H. J. (1979). *Distributed Information Systems*. America: Petrocelli Books, Inc.

Krzyzanowski, P. (2002). *Fault Tolerance*. Retrieved May 11, 2006, from http://www.cs.rutgers.edu/~pxk/rutgers/notes/pdf/fault-tol.pdf

Lampert, D. & Lvov, S. (2006). *Database vs DbPool in Netscape ES 3*. Retrieved August 10, 2006, from http://www.weizmann.ac.il/lvov/sergei/

Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*. 4, 3, 382-401.

Long, B., Hoffman, D., & Strooper, P. (2001). A Concurrency Test Tool for Java Monitors. *16th IEEE International Conference on Automated Software Engineering (ASE01)*, 421.

Miller, R. W. & Collins, C. T. (2001). *Acceptance Testing*. Retrieved June 10, 2006, from http://citeseer.ist.psu.edu/527940.html

Mosberger, D. & Jin, T. (1998). *httperf – A Tool for Measuring Web Server Performance*. Retrieved June 15, 2006, from ftp://ftp.hpl.com/pub/httperf/

MSDN. (2006). *Integration Testing*. Retrieved June 10, 2006, from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsent7/html/vxconintegrationtesting.asp

Myers, G. J. (1978). *The Art of Software Testing*. America: A Wiley-Interscience Publication.

Nesterenko, M. & Jin, M. (2002). *Classification of Distributed Systems. Advanced Operating Systems*. Retrieved August 10, 2006, from http://deneb.cs.kent.edu/~mikhail/classes/aos.s02/l01DStaxonomy.PDF

Neuman, B. C. (1994). Scale in Distributed Systems. *Readings in Distributed Computing Systems*. IEEE Computer Society Press.

NTP: The Network Time Protocol. (2006). Retrieved September 5, 2006 from http://www.ntp.org/ntpfaq/

OLC Assets System - University of Otago Language Centre. (2005). Retrieved July 15, 2006, from http://139.80.15.230/assets

Open System Testing Architecture 1.4.3. (2005). Retrieved July 26, 2006, from http://OpenSTA.org

Pfleeger, C. (1997). *Security in Computing* (2nd ed.). New Jersey: Prentice Hall.

Pfleeger, S. (1997). *Software Engineering: Theory and Practice*. America: Prentice Hall.

Pressman, R. (2001). *Software Engineering: A Practitioner's Approach*. Boston: McGraw Hill.

Raishe, T. (1999). *Black Box Testing*. Retrieved August 15, 2006, from http://www.cse.fau.edu/~maria/COURSES/CEN4010-SE/C13/black.html

Rupp, N. A. (2003). *Multithreaded Tests with JUnit.* Retrieved June 6, 2006, from http://today.java.net/pub/a/today/2003/08/06/multithreadedTests.html

Saluja, K. K. (2005). *High Level Fault-Tolerance: Checkpointing and Recovery*. Retrieved September 2, 2006, from http://homepages.cae.wisc.edu/~ece753/slides/Lecture-Set_9.pdf

Strooper, P., Duke, R., Wildman, L., Goldson, D., & Long, B. (2004). *Practical Tools and Techniques for the Testing of Concurrent Software Components*. Retrieved June 10, 2006, from http://www.itee.uq.edu.au/~testcon/

Strooper, P. (2005). *Tool Support for Testing Concurrent Java Components*. Australia: The University of Queensland.

Tanenbaum, A. S., & Steen, M. V. (2002). *Distributed Systems – Principles and Paradigms*. America: Prentice-Hall.

Taylor, R. N., Redmiles, D. F., & Hoek, A. V. D. (2006). *Decentralized Architectures*. Retrieved June 15, 2006, from http://www.isr.uci.edu/architecture/archdecent.html

TestMaker. (2006). Retrieved August 5, 2006, from http://www.PushToTest.com

Turnbull, M. (1987). Support for Heterogeneity in the Global Distributed Operating System. *ACM Press*, 21, 11-22.

Ulrich, A. W., Zimmerer, P., & Chrobok-Diening, G. (1999). Test Architectures for Testing Distributed Systems. *Proc. of Software Quality Week.* 1-6.

Wake, W. C. (2000). *The Test/Code Cycle in XP: Part 1, Model*. Retrieved August 25, 2006, from http://xp123.com/xplor/xp0002/index.shtml

Warfield, A., Coady, Y., & Hutchinson, N. (2001). Identifying Open Problems in Distributed Systems. Retrieved June 10, 2006, from http://www.cs.unibo.it/ersads/papers/warfield.pdf.

Williams, L. (2004). *White-Box Testing*. Retrieved April 15, 2006, from http://agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf.

11. Appendix

11.1 Appendix 1: DeleteAndOrderCarTest.java

```
/*
 * Created on July 20, 2006
 */
package testCases;
import java.io.FileNotFoundException;
import GUI.OrderACar;
import datastore.CarsData;
import domain.SportsCar;
/**
 * @author Owner
 */
public class DeleteAndOrderCarTest {
    public static void main(String[] args) throws FileNotFoundException {
        //System.setOut(new PrintStream(new
FileOutputStream("CreateGroupOut2.txt")));
        //System.setErr(new PrintStream(new
FileOutputStream("CreateGroupErr2.txt")));
        DelACar();
        OrdACar();
    }
    static SportsCar c1 = new SportsCar("Sport", "Isuzu", "Legend", "Family",
"White", 10000, "Manual", 1000);
    static SportsCar c2 = new SportsCar("Sport", "Honda", "Honda", "Vigor",
"red", 20000, "Manual", 2000);
    static SportsCar c3 = new SportsCar("Sport", "Daewoo", "Daewoo", "Integra",
"blue", 30000, "Manual", 3000);
    static SportsCar c4 = new SportsCar("Sport", "Fiat", "Fiat", "Legend",
"black", 40000, "Manual", 4000);
    private static void DelACar() {
        CarsData.addCar(c1);
        CarsData.addCar(c2);
        CarsData.addCar(c3);
        CarsData.addCar(c4);
        System.out.println("Available car list.");
        for(int i=0; i<CarsData.carsList.size(); i++) {
            System.out.println(CarsData.carsList.get(i));
        }
        System.out.println("About to remove car: ");
        System.out.println(CarsData.carsList.get(2));
        CarsData.carsList.remove(2);
        System.out.println("1 car is removed!");
        System.out.println("Available car list after 1 car is remove: ");
        for(int i=0; i<CarsData.carsList.size(); i++) {
            System.out.println(CarsData.carsList.get(i));
        }}
    private static void OrdACar() {
        System.out.println("-----------");
        OrderACar orderC3 = new OrderACar("Order A Car",2);
        System.out.println("Order car 3");
        System.out.println("Actual car is car 4");
        System.out.println(CarsData.carsList.get(2));
        System.out.println("-----------");
        System.out.println("Order car 4");
        OrderACar orderC4 = new OrderACar("Order A Car",3);
        for(int i=0; i<CarsData.carsList.size(); i++) {
            System.out.println(CarsData.carsList.get(i));}}}
```

11.2 Appendix 2: CreateGroupTest1.java and CreateGroupTest2.java

```
/*
 * Created on July 22, 2006
 */
package test;

import gui.dlgCreateGroup;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.Calendar;
import java.util.Vector;
import org.globalse.fragme.ControlCenter;
import dataAccess.CalendarInterface;
import domain.Group;

/**
 * @author Owner
 */
public class CreateGroupTest1 {

    public static void main(String[] args) throws Exception {
        ControlCenter.setUpConnections("forad587","Adam");
        //System.setOut(new PrintStream(new
FileOutputStream("CreateGroupOut1.txt")));
        //System.setErr(new PrintStream(new
FileOutputStream("CreateGroupErr1.txt")));

        long ti = System.currentTimeMillis();
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(ti);
        int milliseconds = calendar.get(Calendar.MILLISECOND);
        int seconds      = calendar.get(Calendar.SECOND);
        int minutes      = calendar.get(Calendar.MINUTE);
        int hours        = calendar.get(Calendar.HOUR_OF_DAY);

        if(hours == 18 && minutes == 18) {
            while (seconds <30 && milliseconds != 0) {
                System.out.println("Hours: " + hours);
                System.out.println("Minutes: " + minutes);
                System.out.println("Seconds: " + seconds);
                System.out.println("Milliseconds: " + milliseconds);
                System.out.println("Executing...");
                TestGroup1();
            }
        }
    }
    private static boolean TestGroup1() throws Exception {
        Vector mems = new Vector();
        System.out.println("Adam creates an INFO444 group!");
        Group gr1 = new Group();
        if(dlgCreateGroup.checkAvailableGroup("INFO444") == true) {
            CalendarInterface.createGroup("Information 444","Information
444",mems);
        }
        return true;
    }
}
```

```
/*
 * Created on July 22, 2006
 */
package test;

import gui.dlgCreateGroup;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.Calendar;
import java.util.Vector;
import org.globalse.fragme.ControlCenter;
import dataAccess.CalendarInterface;
import domain.Group;

/**
 * @author Owner
 */
public class CreateGroupTest2 {

    public static void main(String[] args) throws Exception {
        ControlCenter.setUpConnections("sunji568","Jim");
        //System.setOut(new PrintStream(new
FileOutputStream("CreateGroupOut2.txt")));
        //System.setErr(new PrintStream(new
FileOutputStream("CreateGroupErr2.txt")));

        long ti = System.currentTimeMillis();
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(ti);
        int milliseconds = calendar.get(Calendar.MILLISECOND);
        int seconds      = calendar.get(Calendar.SECOND);
        int minutes      = calendar.get(Calendar.MINUTE);
        int hours        = calendar.get(Calendar.HOUR_OF_DAY);

        System.out.println("Hours: " + hours);
        System.out.println("Minutes: " + minutes);
        System.out.println("Seconds: " + seconds);
        System.out.println("Milliseconds: " + milliseconds);

        if(hours == 18 && minutes == 18) {
            while (seconds < 30 && milliseconds != 0) {
                System.out.println("Hours: " + hours);
                System.out.println("Minutes: " + minutes);
                System.out.println("Seconds: " + seconds);
                System.out.println("Milliseconds: " + milliseconds);
                System.out.println("Executing...");
                TestGroup2();
            }
        }
    }
    private static boolean TestGroup2() throws Exception {
        Vector mems = new Vector();
        System.out.println("Jian creates another INFO444 group concurrently!");
        Group gr2 = new Group();
        if(dlgCreateGroup.checkAvailableGroup("INFO444") == true) {
            CalendarInterface.createGroup("Information 444","Information
444",mems);
        }
        return true;
    }
}
```

11.3 Appendix 3: CreateAppTest1.java and CreateAppTest2.java

```
/*
 * Created on July 22, 2006
 */
package test;

import java.util.Calendar;
import org.globalse.fragme.ControlCenter;
import dataAccess.CalendarInterface;
import domain.Appointment;

/**
 * @author Owner
 */
public class CreateAppTest1 {

    public static void main(String[] args) throws Exception {
        ControlCenter.setUpConnections("forad587","Adam");
        //System.setOut(new PrintStream(new
FileOutputStream("CreateGroupOut2.txt")));
        //System.setErr(new PrintStream(new
FileOutputStream("CreateGroupErr2.txt")));

        long ti = System.currentTimeMillis();
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(ti);
        int milliseconds = calendar.get(Calendar.MILLISECOND);
        int seconds      = calendar.get(Calendar.SECOND);
        int minutes      = calendar.get(Calendar.MINUTE);
        int hours        = calendar.get(Calendar.HOUR_OF_DAY);

        if(hours == 18 && minutes == 34) {
            while (seconds < 30 && milliseconds != 0) {
                System.out.println("Hours: " + hours);
                System.out.println("Minutes: " + minutes);
                System.out.println("Seconds: " + seconds);
                System.out.println("Milliseconds: " + milliseconds);
                System.out.println("Executing...");
                createNewApp1();
            }
        }
    }

    private static void createNewApp1() throws Exception {
        Appointment ap1 = new Appointment();
        ap1 = CalendarInterface.createAppointment("forad587", "15/06/2006
09:00","60","CO450","INFO401 meeting","Group Project");
        System.out.println("Appointment " + ap1+ " has been created by Adam.");
    }
}
```

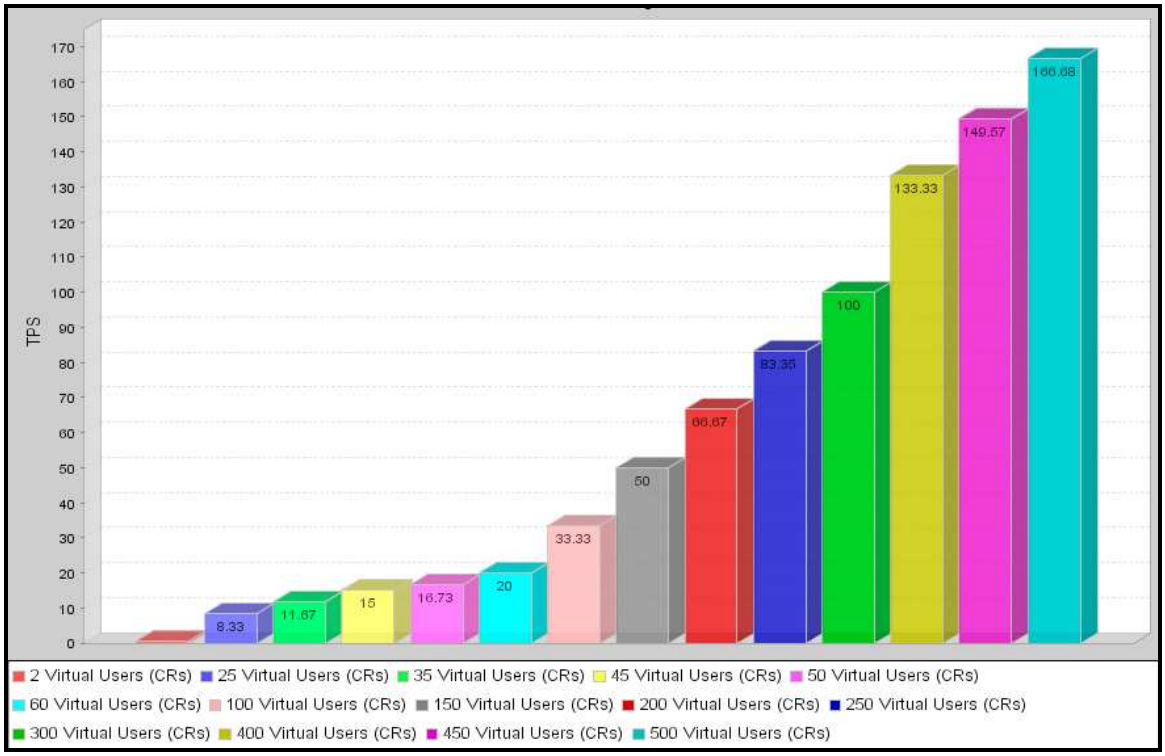
```
/*
 * Created on July 22, 2006
 */
package test;

import java.util.Calendar;
import org.globalse.fragme.ControlCenter;
import dataAccess.CalendarInterface;
import domain.Appointment;

/**
 * @author Owner
 */
public class CreateAppTest2 {

    public static void main(String[] args) throws Exception {
        ControlCenter.setUpConnections("sunji568","Jian");
        //System.setOut(new PrintStream(new
FileOutputStream("CreateGroupOut2.txt")));
        //System.setErr(new PrintStream(new
FileOutputStream("CreateGroupErr2.txt")));

        long ti = System.currentTimeMillis();
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(ti);
        int milliseconds = calendar.get(Calendar.MILLISECOND);
        int seconds      = calendar.get(Calendar.SECOND);
        int minutes      = calendar.get(Calendar.MINUTE);
        int hours        = calendar.get(Calendar.HOUR_OF_DAY);

        if(hours == 18 && minutes == 34) {
            while (seconds < 30 && milliseconds != 0) {
                System.out.println("Hours: " + hours);
                System.out.println("Minutes: " + minutes);
                System.out.println("Seconds: " + seconds);
                System.out.println("Milliseconds: " + milliseconds);
                System.out.println("Executing...");
                createNewApp2();
            }
        }
    }

    private static void createNewApp2() throws Exception {
        Appointment ap2 = new Appointment();
        ap2 = CalendarInterface.createAppointment("sunji568", "15/06/2006
09:00","60","CO450","INFO401 meeting","Group Project");
        System.out.println("Appointment " + ap2 + " has been created by
Jian.");
    }}
```

11.4 Appendix 4: EditExistingCategoriesTestA.java and EditExistingCategoriesTestB.java

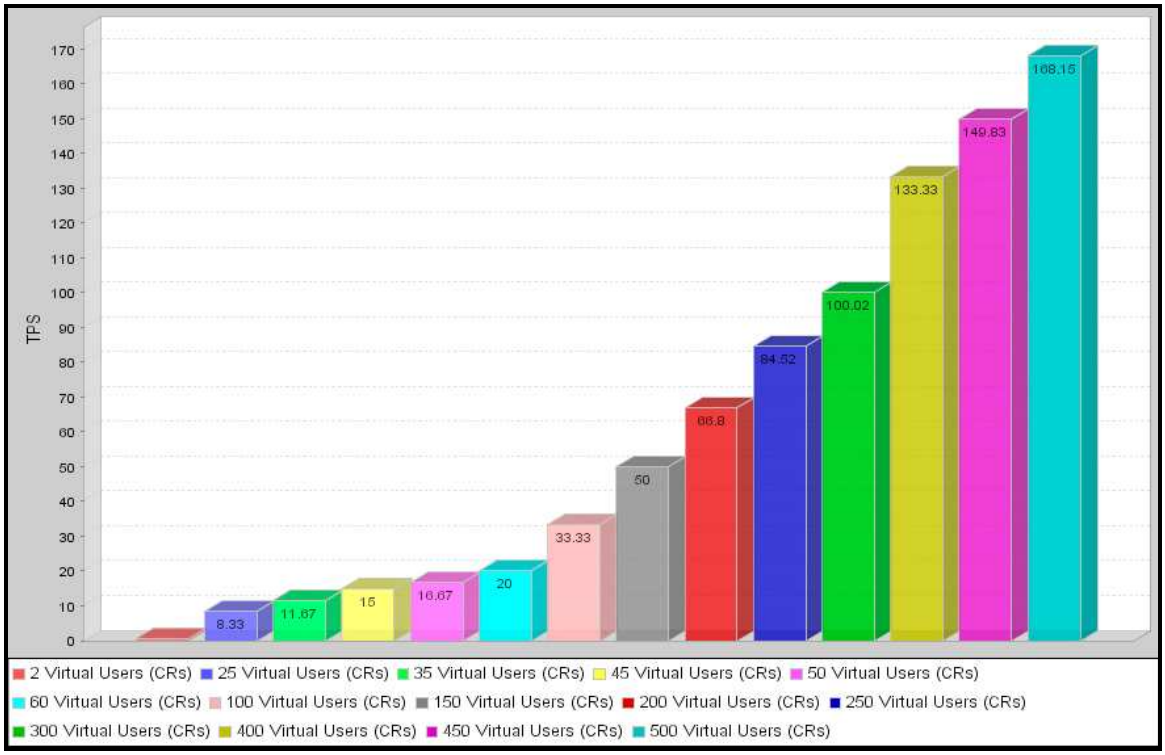
```
/*
 * Created on Aug 25, 2006
 */
package tests;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.Calendar;
import server.Category;
import gui2.AddCategoriesFrame;
import gui2.EditCategoriesFrame;
import gui2.EditRemoveCategoriesFrame;
/**
 * @author Owner
 */
public class EditExistingCategoriesTestA {
    /**
     * @param args
     * @throws FileNotFoundException
     */
    public static void main(String[] args) throws FileNotFoundException {
        //System.setOut(new PrintStream(new
FileOutputStream("CreateGroupOut2.txt")));
        //System.setErr(new PrintStream(new
FileOutputStream("CreateGroupErr2.txt")));
        long ti = System.currentTimeMillis();
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(ti);
        int milliseconds = calendar.get(Calendar.MILLISECOND);
        int seconds      = calendar.get(Calendar.SECOND);
        int minutes      = calendar.get(Calendar.MINUTE);
        int hours        = calendar.get(Calendar.HOUR_OF_DAY);

        AddCategoriesFrame a = new AddCategoriesFrame();
        a.jButtonSaveActionPerformed();

        if(hours == 21 && minutes == 57) {
            while (seconds < 50 && milliseconds != 0) {
                System.out.println("Hours: " + hours);
                System.out.println("Minutes: " + minutes);
                System.out.println("Seconds: " + seconds);
                System.out.println("Milliseconds: " + milliseconds);
                System.out.println("Editing...");
                addCategories1();
            }
        }
    }

    private static void addCategories1() {
        System.out.println("Original Data: " + Category.getCategoriesMap());
        EditCategoriesFrame cf = new EditCategoriesFrame();
        EditRemoveCategoriesFrame ercf = new EditRemoveCategoriesFrame();
        cf.setId("1");
        cf.setName("Folders");
        cf.setDescription("Same colours & brand but different size");
        cf.SaveButtonClick();
        System.out.println(Category.getCategoriesMap());}}
```

```
/*
 * Created on Aug 25, 2006
 */
package tests;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.Calendar;
import gui2.AddCategoriesFrame;
import gui2.EditCategoriesFrame;
import gui2.EditRemoveCategoriesFrame;
import server.Category;

/**
 * @author Owner
 */
public class EditExistingCategoriesTestB {

    public static void main(String[] args) throws FileNotFoundException {
        //System.setOut(new PrintStream(new
FileOutputStream("CreateGroupOut2.txt")));
        //System.setErr(new PrintStream(new
FileOutputStream("CreateGroupErr2.txt")));

        long ti = System.currentTimeMillis();
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(ti);
        int milliseconds = calendar.get(Calendar.MILLISECOND);
        int seconds      = calendar.get(Calendar.SECOND);
        int minutes      = calendar.get(Calendar.MINUTE);
        int hours        = calendar.get(Calendar.HOUR_OF_DAY);

        AddCategoriesFrame a = new AddCategoriesFrame();
        a.jButtonSaveActionPerformed();

        if(hours == 21 && minutes == 57) {
            while (seconds < 50 && milliseconds != 0) {
                System.out.println("Hours: " + hours);
                System.out.println("Minutes: " + minutes);
                System.out.println("Seconds: " + seconds);
                System.out.println("Milliseconds: " + milliseconds);
                System.out.println("Editing...");
                addCategories2();
            }
       }
    }
    private static void addCategories2() {
        System.out.println("Original Data: " + Category.getCategoriesMap());
        EditCategoriesFrame cf = new EditCategoriesFrame();
        EditRemoveCategoriesFrame ercf = new EditRemoveCategoriesFrame();
        cf.setId("1");
        cf.setName("Folders");
        cf.setDescription("Different colours & size but the same brand.");
        cf.SaveButtonClick();
        System.out.println(Category.getCategoriesMap());
    }
}
```

11.5 Appendix 5: OLC Asset XSTest Scalability Index



11.6 Appendix 6: DCRR XSTest Scalability Index



11.7 Appendix 7: Employee Directory XSTest Scalability Index