CSIRO Middleware Technology Evaluation Series

Evaluating J2EE Application Servers

Borland Enterprise Server v5.02
INTERSTAGE Application Server v4.0
JBoss v2.4.3
SilverStream Application Server v3.7.4
WebLogic Server v6.1
WebSphere Application Server v4.0

Version 2.1

CSIRO

June 2002

Who is CSIRO?

The Commonwealth Scientific and Industrial Research Organisation (CSIRO) is Australia's world-renowned research and development organization. Over many years, CSIRO has established a global reputation for high quality research and innovation in many areas, including Information Technology.

The *Software Architectures and Component Technologies (SACT)* group, based in Sydney, specializes in working with enterprise middleware products. SACT focuses on the use of COTS technologies in building large-scale, mission-critical enterprise systems. SACT works with industry clients who wish to evaluate, build and deploy enterprise systems using state-of-the-art middleware technology. Their work contributes to the Middleware Technology Evaluation (MTE) project, which produces qualitative and quantitative independent technology evaluation reports for end-user organizations. More information on SACT can be found at http://www.cmis.csiro.au/SACT.

Authors

Paul Brebner, Shiping Chen, Ian Gorton, Jeffrey Gosper, Lei Hu, Anna Liu, Doug Palmer, Shuping Ran

Product experts

Borland Enterprise Server:	Lei Hu, Paul Brebner, Jeffrey Gosper, Shuping Ran
INTERSTAGE Application Server:	Shiping Chen
JBoss:	Shiping Chen, Doug Palmer
SilverStream Application Server:	Paul Brebner, Lei Hu
WebLogic Server:	Lei Hu, Anna Liu
WebSphere Application Server:	Shiping Chen, Shuping Ran

Editors

Paul Brebner, Ian Gorton

Biographies of Authors

Paul Brebner Paul Brebner is a Software Research Engineer in the Software Architectures and Component Technologies (SACT) R&D group at CSIRO in Canberra. He specializes in Object Oriented methods and tools, software processes, requirements analysis, architecture design and evaluation, Java, Enterprise Java Beans, and technology evaluation. Paul has a MSc in computer science (machine architecture and artificial intelligence) from Waikato University (NZ), and 6 years post-graduate research in knowledge based systems in Sydney (UNSW). He has extensive experience with logic programming, UNIX kernel system programming, object oriented design, and software process improvement. He currently contributes to the MTE project J2EE AppServer evaluations, and is researching ebXML, Web Services, and software architectures.

Shiping Chen Shiping Chen received a BEng. in Electrical Engineering from University of Technology, Harbin China in 1985, a MsEng. in Computer System Engineering from Chinese Academy of Sciences in 1990 and a PhD in Computer Science from the University of New South Wales, Australia in 2001. From 1990 to 1999 he worked on system simulation, real-time control, parallel computing and CORBA-based Internet gaming systems at SIA, UTS and ACCESS. Currently, he is working in SACT as a research engineer. His key skills and research interests are building benchmarking applications using middleware technology, management and configuration for web-based large-scale distributed systems and performance tuning of application servers and database.

Ian Gorton Ian Gorton has over 12 years R&D experience in this area gained at CSIRO, IBM Transarc, the University of New South Wales and Microsoft Australia. He has published numerous papers in journals and conferences, presented tutorials at OOPSLA and TOOLS conferences, and has written a book for Addison-Wesley on enterprise transaction processing systems. He is now the Chief Architect at the Pacific Northwest National Laboratory, US Department of Energy.

Jeffrey Gosper Jeffrey Gosper is a Senior Research Engineer in the Software Architectures and Component Technologies (SACT) R&D group at CSIRO in Sydney. He has recently joined the group leaving his position as Senior Software Consultant with the Tripos Software Consultancy Service (UK). In this former role he was responsible for complete life cycle software development and project management in informatics solutions primarily targeting data/application integration via distributed object computing. His skills also include customer interfacing, technology evangelism, requirements gathering, object oriented analysis and design, as well as data analysis and visualization with implementation skills in both Java and VB. He holds an MSc in Advanced Computing from Kings College London and a Ph.D. in Organic Chemistry from the University of Sydney. He has extensive experience in teaching and research (being a university lecturer for almost 10 years) and has published a number of papers in journals and conferences.

Lei Hu Lei Hu is a Senior Software Engineer in the Software Architectures and Component Technologies (SACT) R&D group at CSIRO in Sydney. He specializes in object-oriented analysis, design and programming skills, parallel and distributed software modelling and performance evaluation, and middleware technologies. He holds a PhD in Computer Engineering from the University of NSW. During his career, he has written a large amount of code, ranging from MSDOS TSR programs, to UNIX and WindowsNT/2000 system programs.

Anna Liu is a Senior Research Engineer in the Software Architectures and Component Technologies (SACT) R&D group at CSIRO in Sydney, and is the MTE project manager. She specializes in gathering requirements for middleware technology, quality attribute based architecture analysis and design, and technology evaluation. Anna holds a PhD in Computer Engineering from UNSW, and has published numerous research papers on object-oriented software architectures. She has also presented tutorials at TOOLS Pacific in 2000, WICSA 2001, and will be an invited industry track speaker at ICSE 2002. She has also been a major contributor to SACT's consulting work. She has recently spent 3 months as a Resident Affiliate at the Software Engineering Institute at Carnegie Mellon University in Pittsburgh, and holds honorary lecturing positions at the University of Technology, Sydney and University of Sydney.

Doug Palmer Doug Palmer is a Software Engineer in the SACT group in Canberra, specialising in the design of test and architecture tools. He holds a PhD in Computer Science from the University of Melbourne. During his career, he has worked in areas as diverse as software games and financial software. His skills include object-oriented analysis and design, messaging system architectures and the construction of distributed test platforms.

Shuping Ran is a Senior Research Scientist in the Software Architectures and Component Technologies (SACT) R&D group at CSIRO in Canberra. Her main research focus is on architecture analysis and design, and technology evaluation for building large-scale enterprise and business-to-business systems. Dr Ran has numerous years of R & D experience working at CSIRO, DSTO, Australian National University, Computer System Research Institute of China, and IIE of Mexico. She has published numerous research papers and has been a strong contributor to SACT's consulting work.

The authors may be contacted at
CSIRO Mathematical and Information Sciences
Building E6B
Macquarie University
North Ryde NSW 2113
AUSTRALIA

Tel + 61 2 9325 3278
Fax + 61 2 9325 3200
E-mail: Jeffrey.Gosper@cmis.csiro.au
Web: http://www.cmis.csiro.au/SACT/

The MTE project is supported by the

Test-IT

industry development fund scheme,

Australian Department of

Communications, Information Technology and the Arts

Table of Contents

1 Introduction

NOTE: This is an extract from the full J2EE Application Server Evaluation Report related to the StockOnline benchmark (and general evaluation methodology).

0.1. Aims of the Study

Formatted: Bullets and Numbering

Complexity is a fact of life in the software industry. Large software systems are inherently complex and expensive to build. Despite advances in mainstream software development technology, such as object-orientation and components, there seems little likelihood of the situation changing in the foreseeable future. The insatiable demand of business for 'bigger and better' applications seems certain to ensure this is the case.

The exponential growth of the Internet has added further fuel to the complexity fire. Less than a decade ago, virtually every enterprise-scale business systems could be designed with a high degree of advanced knowledge about the number of users and transaction load that they would support. This made system sizing and capacity planning *relatively* straightforward. It was even possible in most cases to predict growth needs, and build systems with the capacity to scale to higher levels of usage.

All these assumptions go out of the window when business applications are exposed to the Internet. The ubiquity of Internet access means that, for all practical purposes, limitless numbers of users may converge on a system at the same time. Industry folklore is rife with stories of Internet e-business sites breaking under unexpected client surges in demand. Such occurrences are of course not good for business. And on the Internet, a competitor's site is just a mouse-click away.

It's worth quickly mentioning a couple of examples of the load that some WWW sites now support. The Wimbledon Tennis tournament experienced almost 1 billion WWW accesses in 1999, with 420,000 hits per minute (7000 per second) during one particular match. The volumes at the Sydney Olympic Games WWW site exceeded the Wimbledon volumes by a significant amount. And bear in mind, Internet access is used by a small proportion of the globe's population at this stage. Things have only just started.

It's hardly surprising that the software industry has quickly realized that robust, reliable and scalable technology is needed to support their enterprise, e-business systems. Middleware technology, the plumbing of many Internet systems, has emerged in various guises as a base infrastructure for running advanced e-business systems. A number of product vendors offer enterprise middleware technologies that have been developed over a number of years, and are now seen as viable infrastructures for e-business applications.

Our aim in the Middleware Technology Evaluation (MTE) project is to shed some light upon the strengths and weaknesses of the major competing enterprise middleware technologies on the market. At a superficial level, most products offer very similar sets of features, and of course all these features are better than the competitors!

Experience shows this not to be the case. Underneath the covers, there are massive differences in the exact features various products provide. These differences manifest themselves in all areas of the products, from base middleware support, to higher level services such as transactions, security and system management. Unfortunately, it is not uncommon for applications to rely on certain feature to provide a mission-critical piece of functionality, only to find this feature deficient in some way. The inevitable consequences are schedule delays and budget overruns while fixes are obtained and workarounds implemented. Some applications simply don't survive.

We hope that this report, and the accompanying product-specific reports in the MTE series, will help IT professionals to more comprehensively understand the key issues in building enterprise systems, and how various products offer solutions in these areas. Enterprise middleware technologies are large, complex products, typically with well over 1000 APIs for accessing the various services provided. Consequently they require a serious investment in time and effort in order to appreciate fully what they do well, and what they do less well.

We have invested several person-years effort in working extensively with the products covered, writing applications and testing them to see how they perform. The resulting analysis of the product architectures and features is the most in-depth that has ever, to our knowledge, been presented. The accompanying performance analysis shows how the products perform under normal and stressful transaction loads, and in some cases, where they break.

Therefore, armed with the information contained in the MTE reports, it will be possible for an IT organization to significantly reduce their costs in technology evaluation and adoption. Perhaps even more importantly, this will reduce the potentially massive downstream costs of making a less than optimal decision.

0.2. Overview of the Approach

Formatted: Bullets and Numbering

The evaluation approach taken in this and other MTE reports revolves around two different but complementary activities, namely:

Qualitative architectural analysis

Quantitative performance analysis

Both activities are driven by working with every product to build applications. With each product, at the very minimum an example test application is created. The test application is explained in detail in Section 3 of this report. This test application is important, as by implementing the same application with the same component architecture on each technology, there is a stable reference application that we can use as a basis for performance comparison.

The test application has a number of key ingredients that are tested in each product. These will be explained later in considerable detail, but the basic components required are:

- Middleware infrastructure for implementing a 3-tier application
- A transaction service
- A name or directory service
- A client request load-balancing mechanism

These components are typically at the core of most n-tier applications that are constructed using middleware. They provide the message transport, database coordination and connectivity, location transparency and scalability required for systems that operate in demanding enterprise environments.

When the test application is up and running, the various architectural alternatives and deployment options that each product offers are examined. By experimenting with these different configurations, we gain an in-depth understanding of the product architecture, the various strengths and weaknesses, and areas where the technology constrains application design. The results of these activities feed in to the qualitative architectural analysis provided for each product in the MTE reports.

Next, the test application is put through a series of demanding performance tests. The tests utilise[1] the same hardware and software infrastructure (eg machines, network, and database) for each product. The aim of these tests is to measure how a technology performs under both normal and heavy transaction loads. The results obtained give deep insights in to the performance and scalability potential of the various products. They also make it possible in some cases to draw some broad conclusions about the relative performance merits of various products. These results feed in to the quantitative analysis provided for each product in the MTE reports.

This whole approach is driven by some fundamental principles to which we adhere:

Technological Agnosticism: We have no fundamental belief that any technology is for some reason *better* or *worthier* than any others. Different technologies have different strengths and weaknesses. By striving to expose these strengths and weaknesses, it should be possible for IT organizations to achieve a better fit between products and their use in enterprise solutions.

Rigorous Investigations: The findings contained in the MTE reports are all based on detailed and rigorous explorations of the technologies. Applications are run, their performance analyzed and anomalies resolved, often with the help of product support organizations. The individual product report, results and conclusions are then passed to respective vendor to read and comment upon. This exposes areas where some minor modifications need to be made to ensure that each report accurately represents the product.

Factual: The reports contain observations based solely on fact. These cover product descriptions, configurations and performance results obtained in our test environment. The limitations of what is observed are also clearly stated so that results and conclusions are not interpreted out of context. We occasionally speculate about the behaviour of components or configurations that we've been unable to fully explore. This is however always explicitly stated, and only done when we are extremely sure of the validity of the speculations.

0.3. Products Covered

Formatted: Bullets and Numbering

In this report we have worked with the following products:

- Borland Enterprise Server (BES) v5.02
- INTERSTAGE Application Server (IAS) v4.0

[1] Whenever possible - certain product requirements, features and restrictions force minor variations in test set-up. These are fully documented in the performance results sections.

- SilverStream Application Server (SS) v3.7.4
- WebLogic Server (WLS) v6.1[2]
- WebSphere Application Server (WAS) v4.0
- JBoss v2.4.3[3]

The products with new versions evaluated since the last report (v1.1) are Borland Enterprise Server, SilverStream, WebLogic and WebSphere.

0.4. Limitations of the Study

Formatted: Bullets and Numbering

There are some important limitations regarding the performance analysis results that need to be understood up front.

Most importantly, they are only valid for the software and hardware infrastructure, application architecture and program code that we have used. While every effort has been made to make this as representative as possible of real usage scenarios, J2EE application servers can be used in many weird and wonderful ways, and no single application can cover all possibilities.

A good analogy[4] for how to interpret the performance results can be found in the automobile industry. When you look to buy a new car, each different model usually has a fuel consumption rating. This tells a potential buyer the number of gallons/litres of fuel expected to be consumed per mile/kilometre in both city and highway driving. This is of course an indicative measure, derived from driving tests performed in a controlled environment. As the small print usually says, "This represents indicative fuel consumption, but your experience may vary depending on your driving".

Just as there are many ways to drive a car and many environments to drive cars in, there are many ways to use the components of middleware products, and many different applications that can be created using middleware. Testing all of these is practically and economically infeasible, just as is testing a car's fuel consumption on all possible roads with all possible drivers. Car buyers happily recognize the limitations of these measures, and make sensible decisions based on their experience. We hope the IT industry can interpret and utilize our performance results in a similar manner.

It is also worth emphasizing at this stage that we are not performing a benchmarking study in the strict sense of the term. The aim of the performance analysis tests is not to see how fast, in absolute terms, we can make the client response times in the demonstration application with each product. The aim is rather to impose a fixed application architecture, database structure and defined client load, and see how the system behaves. The aim is to stress the products in a number of areas, and see what affects this has.

For this reason, the performance figures in Section 5 should be treated very carefully. It will only *ever* be valid to directly compare sets of results with the same application written using different middleware technology, but using the same database code and application architecture, and running under the same constraints on the same hardware.

Of course, if we removed some of the constraints we've imposed, it would be simple to dramatically improve throughput and response time. This could easily be done through

[2] WLS 7.0 is now released, but was unavailable at the time of our evaluation. BEA claims that 7.0 has enhanced performance over 6.1.
[3] Jboss v3 was is alpha release only at the time of evaluation.
[4] Thanks to Eric Newcomer for this analogy.

redesign and performance tuning, and in fact the MTE reports describe many of the techniques that are applicable, and shows what performance improvements can be obtained when some constraints are relaxed. Faster hardware would also obviously improve performance.

It is important that this message is clearly understood by the reader. Presenting cold, hard performance figures gives great insight in to a technology. It is crucial however these figures are understood in their context, and not used outside of it.

1. Evaluation Methodology

Formatted: Bullets and Numbering

1.1. Qualitative Evaluation Methodology

While working with each of the products, a set of qualitative evaluation criteria was created, based on our product analysis and hands-on experience. The evaluation team then initially rated each product against these criteria. After initial ratings were set, considerable internal discussion took place to ensure that the allocated scores were as consistent and representative as possible. Further detailed investigations and clarification with vendors was sought when required.

Section 3 of this report presents the results of the qualitative evaluation of the products we have tested. Product features are broken down in to 6 categories, and within each category a number of evaluation criteria are applied. The evaluation criteria attempt to highlight differences in the products, and show the relative strengths and weaknesses of each.

Underlying this qualitative evaluation is a detailed feature matrix, containing over 200 individual requirements points for application servers. The evaluations allocated to these individual criteria are summed to produce the scores for the higher-level categories shown in this report. This feature matrix is not however included in this report, as it is valuable intellectual property that we make available to clients through direct consulting engagements. The commercial reality of partially public funded R&D organizations dictates that we behave in this manner.

Note there is no attempt to add up the scores across all categories to produce an overall 'winner'.

We believe this exercise would produce misleading and totally meaningless outcomes. There is no absolute *best or worst* in this style of technology evaluation. Relative merit changes depending on application and business requirements, costs, skills base, risks, and so on. This section should be used to identify product strengths and weaknesses that are important in an organization's business context.

Like any qualitative evaluation, there is a degree of subjectivity in some of the assessments. We have attempted to be as impartial as is possible, and each set of evaluations have been discussed within the MTE team and agreed upon. Inevitably, there will be some that do not agree with these assessments. We are of course happy to have our views optimized and corrected where something has been missed.

We have used the following scale to rate product features:

0 Not supported/available

1 Not available in the packaged product, but available as an additional component or from a 3rd party, at additional cost.

2 Limited or partial support for a feature. Possible, but not ideal.

3 Feature is supported but has with some limitations or caveats

4 Feature is well supported, with minor limitations

5 The product could do little more to support this feature

More detailed descriptions of the various product features we have evaluated are contained in the separate MTE reports for the individual products.

1.2. Quantitative Methodology

Formatted: Bullets and Numbering

The quantitative methodology adopted in this study aims to analyze the various products through a behavioural and performance analysis approach. This involves:

1. Building a demonstration application using each technology.

2. Running a defined set of transactions and various client loads against the demonstration application. The tests for each product are run on the same hardware and software environment.

3. Measuring various aspects of the performance of the application

The demonstration application tests various aspects of each product. These include at least the underlying middleware, the naming and load balancing service, the transaction service and the database connection interface. The same code base is used to implement the application.

The application code is heavily instrumented, which enables us to record the time taken for various significant events in the system. Examples of these are transaction response time, transaction commit time, and so on.

After an initial set of tests has been run, the performance results are analyzed. In effect, the results obtained tell 'a story' about the product's behaviour. The raw performance figures give us strong indications of where the product performs well, and where there are weaknesses. A number of configuration variations[5] are then tested, and their effect on the application performance measured. This process usually involves discussions with the product vendors. Eventually a final test configuration is reached, and we report these performance figures.

Therefore, the approach we adopt is basically performance-driven. By building a demonstration application, we get to see what works well, not so well, and often, what does not work at all. This is a level of analysis is unique to the MTE project. Anyone can tell from reading manuals that all the products we work with have, for example, transaction services. Only we can tell how well each transaction service performs, what restrictions it

[5] Only variations that do not invalidate the basic test structure are explored.

places on an application, what design and deployment trade-offs are possible, and how to best configure it for maximum performance.

1.3. Stock-Online Application

Formatted: Bullets and Numbering

1.3.1. Requirements

Stock-OnLine is a simulation of a simple on-line stockbroking system. It enables subscribers to buy and sell stock, inquire about the up-to-date prices of particular stocks, and get a holding statement detailing the stocks they currently own. From a subscriber's perspective, the following services are offered:

Create Account: A person wishing to enrol with Stock-Online can create themselves a subscriber account with the service provider.

Update Account: A subscriber can modify their allocated credit limit.

Query Stock Value: A subscriber can query the current price for a given stock. A unique identifier code, or a mnemonic code can be used to identify the stock value to be retrieved.

Buy Stock: A subscriber can place an order to buy a given number of a specified stock. If successful, a transaction record is created for later processing.

Sell Stock: A subscriber can place a request to sell a specified number of any stock item they have purchased through the Stock-Online. If successful, a transaction record is created for later processing.

Get Holding Statement: A subscriber can request a statement of all the stock they have purchased through Stock-Online and still retain ownership of.

1.3.2. Database Design

Formatted: Bullets and Numbering

In a real implementation of an on-line stockbroking system, the database would need to store many details in order to track customers, their transactions, payments, and so on. In the example used for performance tests a simple database design has been used that contains the minimum tables and fields to allow the system to sensibly operate.

The *SubAccount* table holds basic information on a subscriber to the system. The SQL statement to create the table is as follows:

```
CREATE TABLE SubAccount (
   sub_accno int NOT NULL ,
   sub_name varchar (30) NOT NULL ,
   sub_address varchar (60) NOT NULL ,
   sub_credit int NULL
)
```

The primary key for *SubAccount* is the subscriber's account number, *sub_accno*. For simplicity, the name and address fields are not broken down in to their constituent parts (eg street, city, last name). The *sub_credit* field holds the amount of credit a subscriber has with the Stock-OnLine system.

When a new account is created, the system needs to allocate a new, unique account number. To achieve this, we rely upon appropriate mechanism provided by the databases. Specifically, the Oracle *Sequences* and SQL/Server *Identity* features are utilised to provide this functionality.

The *StockHolding* table contains information on the amount of a given stock that a subscriber holds. The primary key is *{sub_accno, stock_id}*. The SQL statement to create the table is:

```
CREATE TABLE StockHolding (
   sub_accno int NOT NULL ,
   stock_id int NOT NULL ,
   amount int NOT NULL
)
```

The *StockHolding* table is needed as the single source of information on which stock items a subscriber holds. While this information could be calculated from the individual transaction information, this relies on all the transactions a subscriber ever performs remaining on-line in the database 'forever'. In high-volume transaction systems, individual transaction records are typically archived off after a given period, making them slow (or impossible) to access.

Information about each stock that a subscriber can trade through Stock-Online is held in the *StockItem* table. The primary key is *stock_id*, and the other fields represent the stock's trading code, company name, current value and recent high and low values. The SQL statement to create the table is as follows:

```
CREATE TABLE StockItem (
   stock_id int NOT NULL ,
   name varchar (30) NOT NULL ,
   code char (4) NOT NULL ,
   current_val float NOT NULL ,
   high_val float NULL ,
   low_val float NULL
)
```

Finally, there is the *StockTransaction* table, which contains information on each transaction that a subscriber performs. The primary key is *trans_id,* which is generated in a similar manner to new accounts through the use of a database mechanism; again Oracle *Sequences* and SQL/Server *Identity* columns are used. The *trans_type* is a code that represents a buy or a sell transaction. Other fields record the subscriber who performed the transaction, the stock item sold or purchased, the amount of stock involved, the price and the date of the transaction. The SQL statement for this table is:

```
CREATE TABLE StockTransaction (
   trans_id int NOT NULL ,
   tran_type varchar (3) NOT NULL ,
   sub_accno int NOT NULL ,
   stock_id int NULL ,
   amount int NULL ,
   price float NULL ,
   date varchar (8) NULL
)
```

1.3.3. Transaction Business Logic

Formatted: Bullets and Numbering

An overview of the business logic for each transaction is given below. The descriptions focus on the database activity that each transaction performs, as these are the most expensive operations. Essentially each numbered action below represents one or more SQL operations. Exception cases are not described, even though these are handled in the application.

Create Account:

1. Get a new account key from an Oracle Sequence

2. Insert a new *SubAccount* record for the new subscriber

Update Account:

1. Update the subscriber's record in the *SubAccount* table

QueryByID:

1. Use the supplied stock identifier to retrieve the current, high and low values from the *StockItem* table using the primary key

BuyStock:

1. Get a new transaction key from a Sequence

2. Read the *StockItem* table to retrieve the current price of the stock the subscriber wishes to purchase

3. Read the *SubAccount* table to retrieve the credit limit for the subscriber, and ensure they have sufficient credit to make the purchase

4. If the subscriber has not purchased this stock item before, insert a new record in the *StockHolding* table to reflect the purchase. If they do hold this stock already, update the record that already exists in the *StockHolding* table.

5. Insert a new record in the *StockTransaction* table to create a permanent record of the purchase.

Sell Stock:

1. Get a new transaction key from an Oracle Sequence

2. Read the *StockItem* table to retrieve the current price of the stock the subscriber wishes to sell

3. Read the *StockHolding* table to ensure that the subscriber has sufficient holdings of this stock to sell the specified amount

4. Update the *StockHolding* table to reflect the sale of some of this stock item by the subscriber

5. Insert a new record in the *StockTransaction* table to create a permanent record of the sale.

Get Holding Statement:

1. Read the *StockHolding* table and retrieve up to 20 *StockHolding* records for the subscriber

Clearly then, the *Buy* and *Sell* transactions are more heavyweight in their demands for database operations. *CreateAccount* and *Update* perform database modifications, but are relatively lightweight. The remaining 3 transactions are all read-only, and will execute quickly as they only perform a single database operation.

1.4. Test Configurations

Formatted: Bullets and Numbering

Figure 1 shows the basic EJB test configuration. Multi-threaded client test drivers were configured to run on a single dedicated client machine. We monitored the machine to ensure that there was no paging on the client machine.

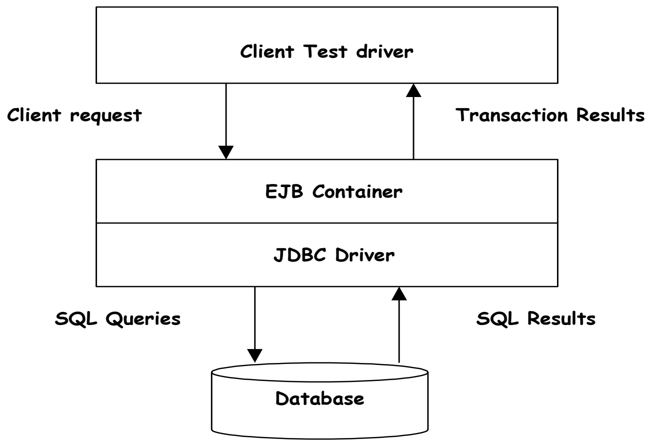

FIGURE 1 BASIC EJB TEST SETUP

A number of different EJB configurations were built and tested to give insights how different application architectures perform. These are:

Stateless session bean only: a single stateless session bean supports all transactions, and executes the JDBC calls directly in its methods. The bean uses the container managed transaction model.

Stateless session bean and Entity Beans: a single stateless session bean supports all transactions for clients, but uses container managed persistence entity beans to access data from the database. The session bean basically acts as a façade for the entity beans.

1.5. Test Environment

Formatted: Bullets and Numbering

1.5.1. Test Platform

The test environment is based on multiple-cpu Intel machines (Dell PowerEdge 6450/700) with the following specifications:

- 4 x Xeon 700MHz Intel CPUs
- 4 GB of main memory
- Win2k 5.0.2195, Pack 2

The Oracle 8.1.7. database is running on an a similarly configured machine (IBM Server 8681) but with 8 CPUs.

An isolated 100 MBit switched Ethernet runs between the test machines. The basic test configurations are shown below in Figure 2.

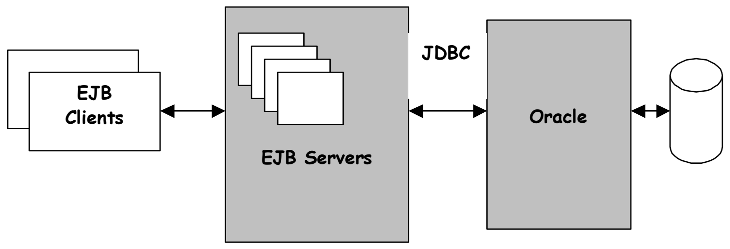

FIGURE 2 BASIC TEST CONFIGURATIONS

This configuration deploys the application servers on a physically different machine to the database. All database communications now takes place across the network. This configuration is scalable, as additional machines can be deployed as needed to host application servers in the middle tier. The performance of the network and database server machine is obviously crucial in this configuration. It requires the network, processor and disk capacity to support a growing number of middle tier application servers.

1.5.2. Client Test Behaviour

Formatted: Bullets and Numbering

Each client thread performs a number of iterations of a fixed *transaction mix*. The transaction mix represents the concept of one complete business cycle at the client side. It consists of a combination of different transaction types, namely:

1	**CreateAccount**
3	**Buy**
3	**Sell**
1	**Update**
30	**QueryById**
5	**GetHoldingStatement**

This transaction mix comprises 43 individual transactions, and is a combination of mainly read-only (81%) transactions and some update (19%) transactions. Different transaction mixes can be used to model different types of business cycle. The aim of this transaction mix is to represent a reasonably typical WWW site transaction load, where over 80% of transactions are read-only (browse) operations. Each client performs this transaction mix 10 times.

The flow chart in Figure 3 illustrates the main logic of the client process for a single transaction mix iteration.

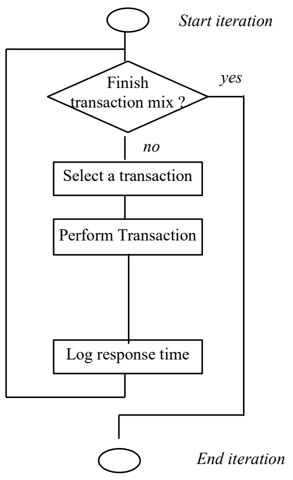

FIGURE 3 CLIENT BEHAVIOUR

Essentially, the client loops through the transaction mix, selects a transaction type to perform, performs the selected transaction, attempts to commit the transaction and then logs the performance measurements.

Notice the client doesn't wait or pause between each transaction invocations. Once a transaction is complete, every client immediately starts the next selected transaction. A wait period is often known as a *think* or *sleep time*, and is meant to simulate the delay between transaction submissions from a single user. A *think* time effectively lowers the transaction rate that an application must process from a given number of clients.

However, in most WWW systems with EJB back-ends, the interactive client code doesn't call the business logic layer directly – this is done by 'proxy' client components somehow launched by the WWW server (e.g. JSPs/ASPs, servlets, etc). These proxy clients do not 'think'. Therefore, our performance tests do not have *think* times in the clients. Each client works flat out, starting a new transaction as soon as the previous one has finished.

Furthermore, in our performance tests, we are more concerned with transaction processing rates at the server side, and less with the number of simultaneous clients a technology can support. As the results show, around 100-200 clients produce a high, sustained transaction load when there is no *think* time. With *think* times, many more clients would need to be run to produce an equivalent transaction load.

1.5.3. Database Population

Formatted: Bullets and Numbering

Prior to each performance test, the database was populated with initial test data. The following table shows the database tables and their cardinality (rows) used for the performance tests.

Table	**Number of Records**	**Comment**
SubAccount	3000	Initially, the system has 3000 subscriber accounts. The field 'sub_name' is a random string of length 5 characters. The field 'sub_address' is a random string of length 10 characters. The field 'sub_credit' is a random integer between 1000 and 100000. This is the initial credit of the subscriber.
StockItem	3000	The system has 3000 unique stock items. The field 'name_varchar' is a random string of length 6 characters. The field 'code_char' is a random string of length 2 characters. The field 'current_val', 'high_val', and 'low_val' are random real number between 0.0 and 100.0.
StockHolding	3000*10	Initially, each subscriber holds 10 different stock items. The stock items are randomly selected from the StockItem table. The field 'amount' is a random integer between 1 and 1000.
StockTransaction	0	Initially, there are no transaction records.

For every performance test, all the records generated previously are deleted. This is to ensure a test is not contaminated with previous ones.

1.6. Performance Analysis

Formatted: Bullets and Numbering

The test code is instrumented at a number of points to obtain performance information. Three main measures are recorded in every test run:

Average client response time: This represents the mean response time for every transaction from the client's perspective. It encompasses the network round-trip time to the server and the time taken by the server object to process the request and access the database on the client's behalf

Total client test time: This represents the time taken for a client thread or process to execute a complete test script. From this figure we can calculate the overall system throughput in terms of transactions per second (tps).

We have also written a performance monitor program that displays in real time the number of transactions per second that the system is executing. This allows us to monitor the system's behaviour as the test progresses, and clearly indicates if problems occur during a test run.

Collectively, these measures make it possible to gain an excellent understanding of how a technology behaves. The performance results *tell a story*, and point is to areas and components of products that represent particular strengths and weaknesses. Often, this leads us to insert additional measurement code to clearly show up which individual operations are particularly fast, slow or interesting in some other way.

1.7. Limitations of Quantitative Evaluations

Formatted: Bullets and Numbering

Before closely examining the results obtained, it's important to understand what conclusions *cannot* be drawn from the observed performance. Like any performance tests, the test software and environment exhibits certain application characteristics, which may or may not be representative of the needs of other applications.

The list below therefore states some dangerous assumptions:

- The performance tests have been performed on Windows 2000 running on a specified hardware platform. The results therefore illustrate the performance that can be expected on these operating system and hardware configuration. This is extremely useful information, as the results from the various technologies can be compared with competing products running on the identical hardware and software platform. However, it is not valid to draw conclusions about how any middleware technologies may perform on different hardware and/or operating system combinations.

- The results are specific to Oracle v8.1.7. on Windows 2000. We have tuned the databases to give good performance. The Oracle configurations are available on request. More importantly, we have ensured that the configuration has remained unchanged across tests with all the middleware technologies we have analyzed. As with operating systems, it is not valid to draw conclusions about the likely performance of other database products from the results in this report.

- The Stock-Online application is deliberately designed to exhibit certain application features. Other applications have other *quirks*, which will place different demands on the middleware, database and operating system. It's therefore important that these application-specific features and their effect on performance are fully understood.

- These results were not derived for, and are not to be interpreted as an absolute measure of performance, and can only be correctly interpreted relative to the results of other products tested under the same conditions.