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

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 Group Leader of the Software Architectures and Component Technologies (SACT) at the CSIRO and is heading the Middleware Technology Evaluation project. Jeff only recently took up this role after being tempted back to Australia from the UK, where he had resided the last decade. Previous to the working at the CSIRO, Jeff was a Senior Software Consultant with the Tripos Software Consultancy Service (UK). In this 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 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. from the University of Sydney. He has extensive experience in teaching and research (being a university lecturer for almost 10 years) and has published many papers as well as speaking at numerous industry and scientific 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. 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

0. Executive Summary

J2EE-based application servers are increasingly becoming the engine rooms of Web-enabled applications. They provide the scalable, high-performance Java infrastructure for processing many simultaneous requests from users for Internet application services. They also offer a consistent design and deployment model, and various software components that aim to greatly ease the development of complex Internet-based applications.

Numerous product vendors offer technologies that support the various components of the J2EE specification. These technologies have been designed, developed and in some cases have evolved in very different ways, and there is a large range in quality of the features and tools that the various products support.

This crowded product marketplace creates a challenge for IT departments who have to evaluate and select an application server that is sufficient for their needs. All business applications are different in some ways, and there is rarely, if ever, a 'one-size-fits-all' product match. Different J2EE products have different sets of strengths and weaknesses and costs, and consequently will be better suited to some types of applications than others.

The difficulty for IT departments is in understanding these strengths and weaknesses early in the development cycle for a project, and choosing an appropriate technology. Unfortunately, this is not an easy task, and the risks and costs associated with selecting an inappropriate technology are high.

This report aims to help reduce the risks and costs of J2EE application server selection. It provides the most detailed analysis, evaluation and comparison published on six leading application server technologies, from IBM, Fujitsu, SilverStream, BEA, Borland and the open source JBoss product.

We have worked extensively with these products by building test applications to help us understand what they are capable of. Based on this analysis, the report compares the products in qualitative terms through a detailed feature set comparison, and in quantitative terms through performance and stress testing of an application tested on each product in an *identical* deployment environment. This provides us with the critical ability to directly compare the performance of the Application Servers alone, unlike the other recent J2EE benchmark, ECPerf[1], which may be run on completely different test environments, and is therefore open to varying interpretations. The focus of the quantitative comparisons is the J2EE EJB container, which forms the core component of most application server deployments.

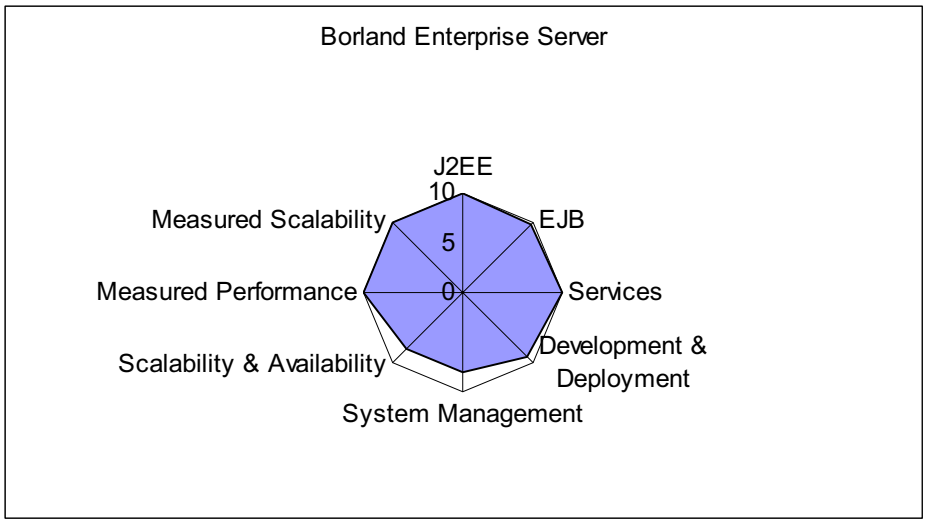
[1] See published ECPerf results at www.theserverside.com/ecperf. For a discussion of how ECPerf relates to the results in this report see www.cmis.csiro.au/adsat/j2eev2.htm

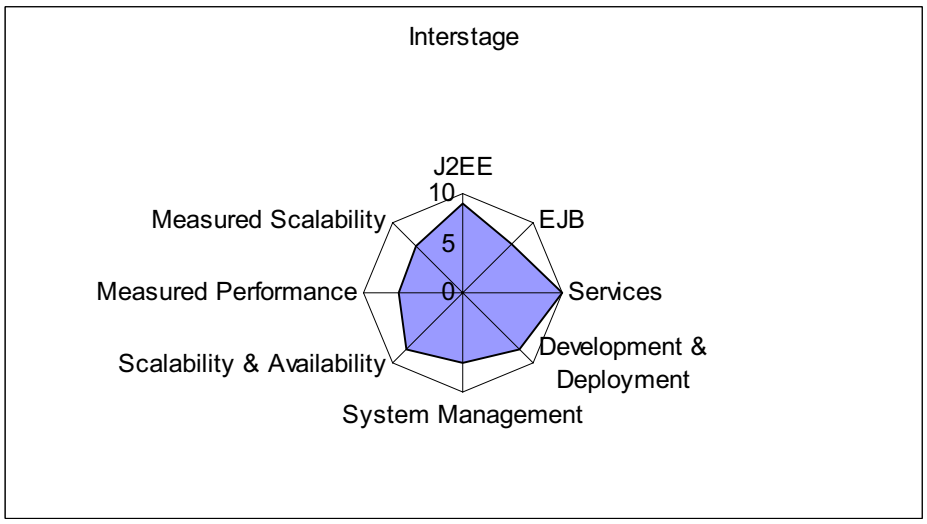
The products were evaluated qualitatively against 6 overall categories: J2EE Support, EJB Support, J2EE Services, Development and Deployment, System Management, and Scalability and Availability. A coarse grain summary of the overall results is shown below, and the detailed results and evaluation criteria are presented in the main body of the report. Overall, all the products provide a good level of support for base J2EE features, and with the increasing maturity of J2EE Application Servers we notice a convergence of features supported.

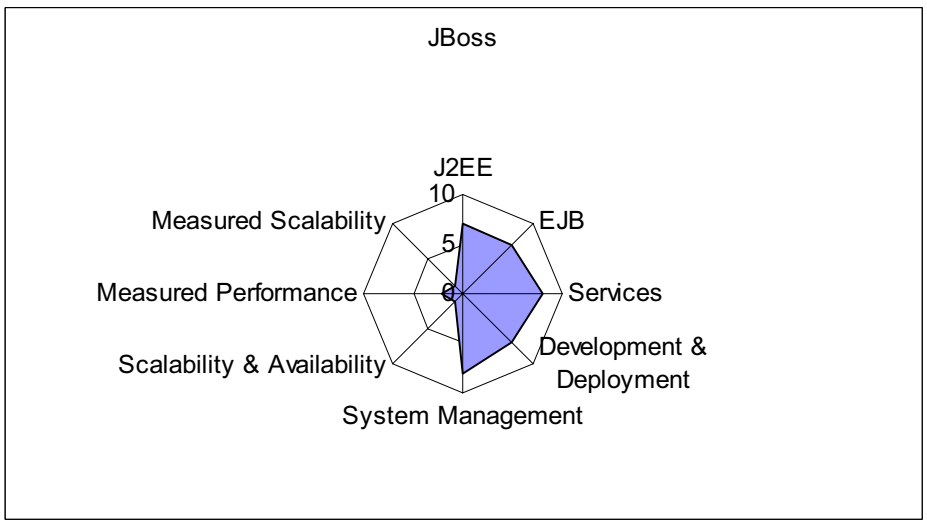
In terms of performance, it should be first noted that all these products should provide acceptable levels of response times and throughput for all but the most demanding applications. The test suite used in this evaluation is designed to stress the technologies, as this reveals most about their internal architecture and build quality. Stress testing helps differentiate between the levels of performance a product can produce, and how the products stand up to very high request loads.

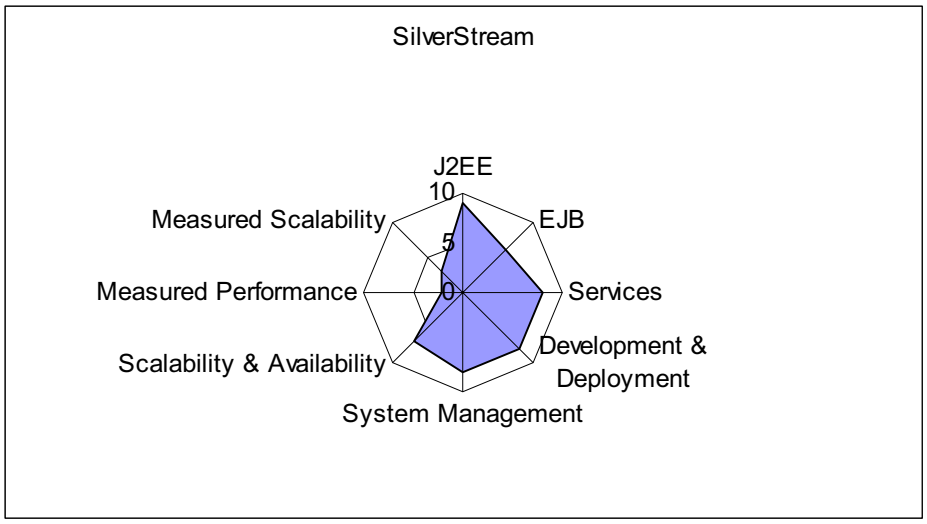
With the current versions of the products on the new hardware configuration used for this update of the report we notice a greater divergence in product performances. All the available performance and scalability results for each product have been combined into normalised values ("Measured Performance" and "Measured Scalability") shown below. However, the main body of the report must be consulted in order to understand the significance of this summary data.

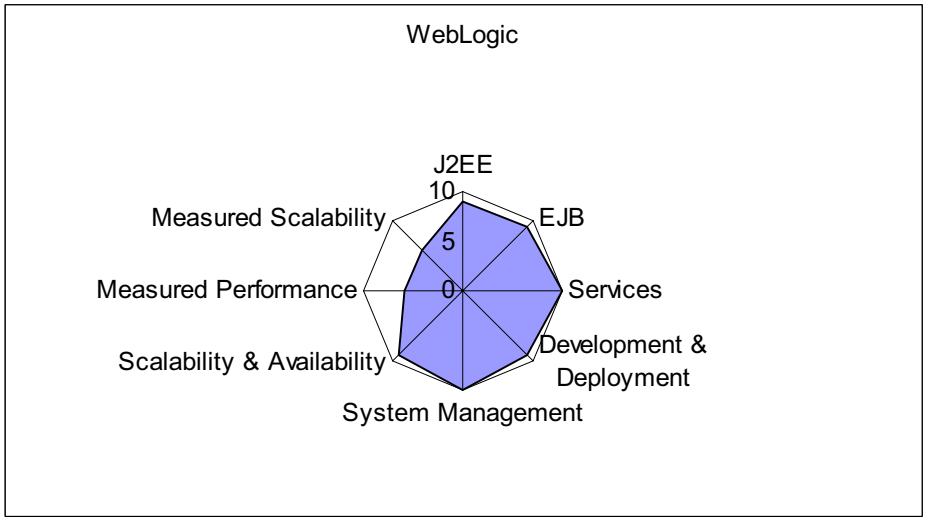
The following graphs summarise both the qualitative and quantitative rankings for each product.

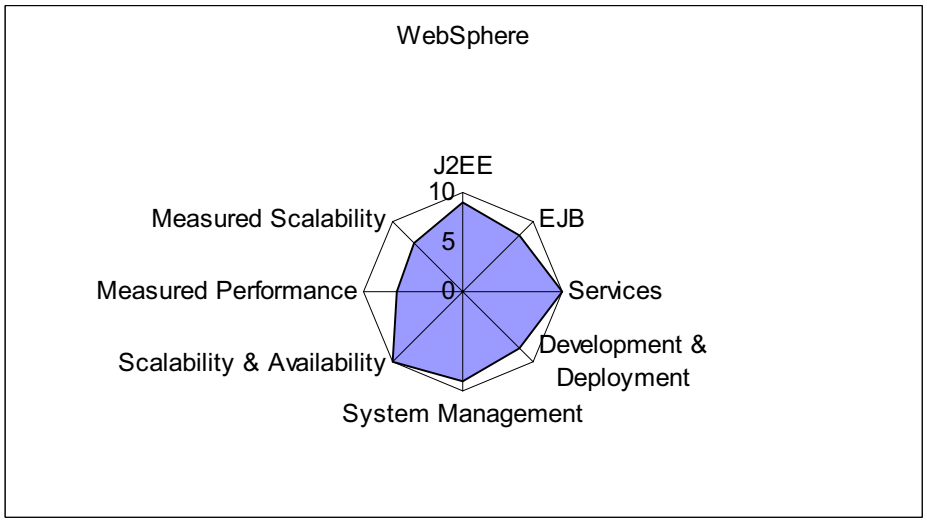

Borland Enterprise Server
J2EE
10
5
0
EJB
Services
Development &
Deployment
System Management
Scalability & Availability
Measured Performance
Measured Scalability


Interstage
J2EE
10
5
0
EJB
Services
Development &
Deployment
System Management
Scalability & Availability
Measured Performance
Measured Scalability


JBoss
J2EE
10
5
0
EJB
Services
Development &
Deployment
System Management
Scalability & Availability
Measured Performance
Measured Scalability


SilverStream
J2EE
10
5
0
EJB
Services
Development &
Deployment
System Management
Scalability & Availability
Measured Performance
Measured Scalability


WebLogic
J2EE
10
5
0
EJB
Services
Development &
Deployment
System Management
Scalability & Availability
Measured Performance
Measured Scalability


WebSphere
J2EE
10
5
0
EJB
Services
Development &
Deployment
System Management
Scalability & Availability
Measured Performance
Measured Scalability

Borland Enterprise Server has emerged as the clear leader. For performance and scalability Borland Enterprise Server is in a class of its own, it provides excellent support for virtually all features, and it ranks first in the majority of functional areas.

Nevertheless, there is no 'absolute best' or 'absolute worst' in technology comparisons. This way of thinking massively oversimplifies a complex problem, and therefore the report makes no attempt to declare absolute winners and losers. The key is for potential users of a J2EE application server to understand their requirements, and select a technology that matches these. Cost is one factor that is almost always influential in this process, and is one we have not taken in to account.

Also, the evaluations represent a comparison of technologies taken at a 'snapshot in time'. These products evolve and improve, and there is no way to predict how their strengths and weaknesses will change. At least three of the products evaluated in this version of the report have changed position in the relative performance and scalability rankings since the last version. For this reason, we intend to regularly publish updates to this report when a product moves to a new major version, and to include new products from alternative vendors.

1. Introduction

1.1. Aims of the Study

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.

1.2. Overview of the Approach

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[2] 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.

1.3. Products Covered

In this report we have worked with the following products:

- Borland Enterprise Server (BES) v5.02
- INTERSTAGE Application Server (IAS) v4.0
- SilverStream Application Server (SS) v3.7.4
- WebLogic Server (WLS) v6.1[3]
- WebSphere Application Server (WAS) v4.0

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

[3] 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.

- JBoss v2.4.3[4]

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

1.4. Limitations of the Study

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[5] 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 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.

[4] Jboss v3 was is alpha release only at the time of evaluation.
[5] Thanks to Eric Newcomer for this analogy.

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.

2. Evaluation Methodology

2.1. 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.

2.2. Quantitative Methodology

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

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

2.3. Stock-Online Application

2.3.1. 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.

2.3.2. Database Design

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
)
```

2.3.3. Transaction Business Logic

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.

2.4. Test Configurations

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.

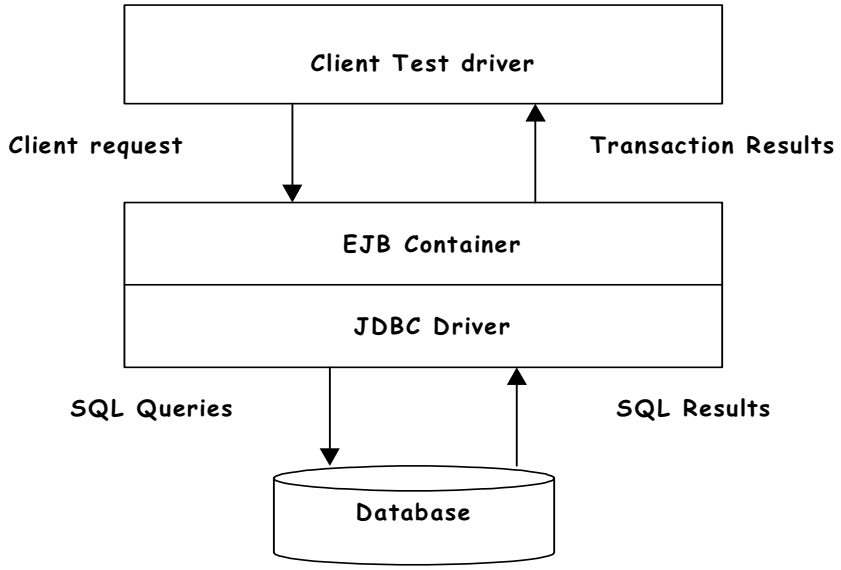


FIGURE 1 BASIC EJB TEST SETUP

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.

2.5. Test Environment

2.5.1. 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.

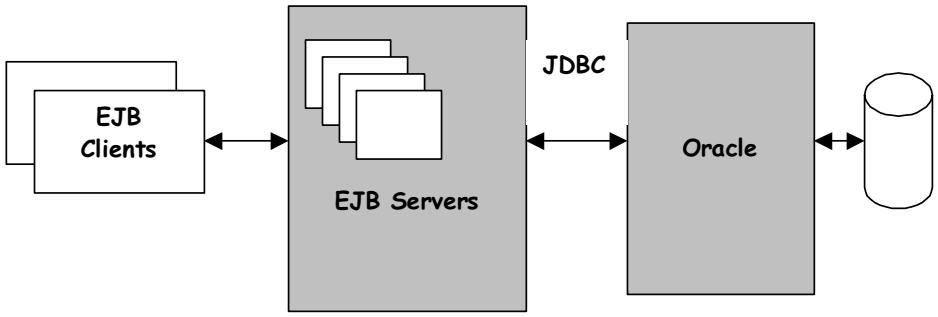


FIGURE 2 BASIC TEST CONFIGURATIONS

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.

2.5.2. Client Test Behaviour

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.

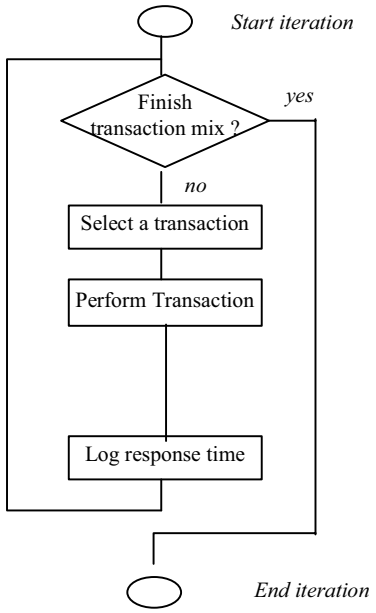


FIGURE 3 CLIENT BEHAVIOUR

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.

2.5.3. Database Population

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.

2.6. Performance Analysis

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.

2.7. Limitations of Quantitative Evaluations

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.

3. Qualitative Evaluation

3.1. Introduction

This section provides a qualitative evaluation of the products we have tested. Product features are broken down into six categories, and within each category a number of evaluation criteria are applied. Underlying each evaluation criteria are a number of low level evaluations, which are not shown in the report for reasons explained in Section 2. Overall, the evaluation criteria attempt to highlight the differences in the products, and show the relative strengths and weaknesses of each.

Note there is no attempt to add up the scores across all the different 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. However, because we have determined the ratings based on a combination of low-level features and our direct experience of the relative product strengths, we believe that they provide an accurate representation of the products as is possible.

The evaluation criteria are rated on a scale of zero to five with the following interpretation:

0 Not supported.

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

2 Limited or partial support. Possible, but not ideal.

3 Supported, but with some limitations or caveats.

4 Well supported, with only minor limitations.

5 Excellent support, the product could do little more.

Except where explicitly stated we are only evaluating features that are included in the product as packaged. However, vendors may choose to provide different packaging options, and not all components may be available in all packages. For example, a *Development* version may not contain clustering support, but an *Enterprise* version may. Therefore entry-level versions of the products evaluated here may not necessarily contain all the features indicated. We have however used the most comprehensive and fully featured versions available from vendors in these evaluations.

In some cases additional components are available to support extra functionality, but these come at an additional cost, or from 3rd party suppliers. These additional components are generally not included in the evaluations in this section simply because:

- We have not used them in the evaluation work.
- As we have not used them, we cannot rigorously evaluate their features.

An exception to this rule is IDEs. If a vendor recommends, supports or supplies (even at extra cost) an IDE that can be integrated with an application server, then we have included this as part of the product evaluation.

Application servers are a moving target and often have many changes introduced between major versions. To the best of our knowledge the evaluation criteria accurately reflect the version of the products evaluated, but are likely to be different for subsequent releases.

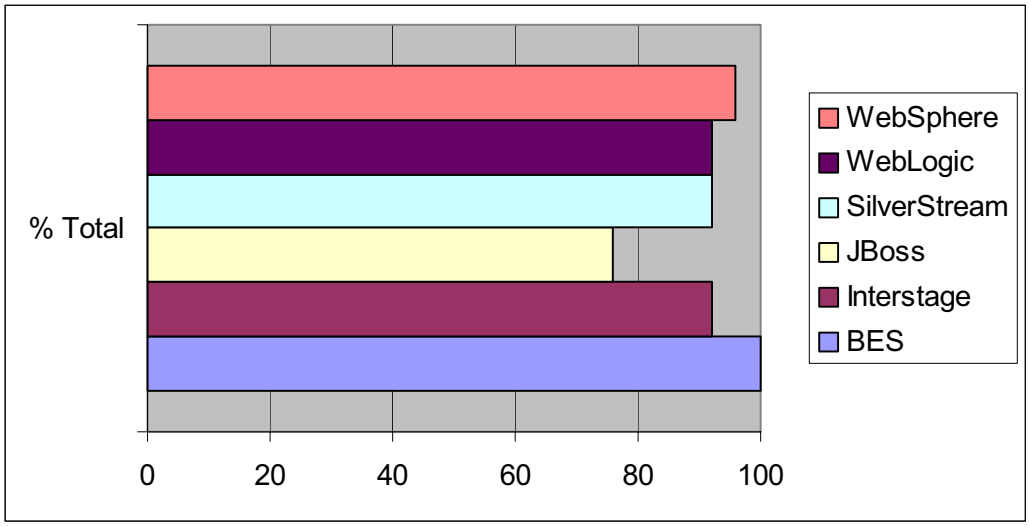
The table below summarizes the qualitative evaluation of the products against 6 overall categories. A score of 10 represents the highest score available, and 1 the lowest. This summary is a coarse grain representation of the overall results.

	J2EE Support	EJB Support	J2EE Services	Scalability and Availability	Development and Deployment	System Management
Borland Enterprise Server	**10**	**9**	**10**	**8**	**9**	**8**
INTERSTAGE App. Server	**9**	7	**10**	**8**	**8**	7
JBoss	7	7	**8**	**1**	7	**8**
SilverStream App. Server	**9**	**6**	**8**	7	**8**	**8**
WebLogic App. Server	**9**	**9**	**10**	**9**	**9**	**10**
WebSphere App. Server	**9**	**8**	**10**	**10**	**8**	**9**

3.2. Summary of Evaluations

3.2.1. J2EE Support

Borland Enterprise Server is out in front with excellent support, whereas the other products have varying degrees of support for J2EE Application Clients, explicit J2EE containers or J2EE module types. The recent releases of WebSphere v4.0 and INTERSTAGE v4.0 have reduced the gaps between products[7], although JBoss is still lagging behind slightly.



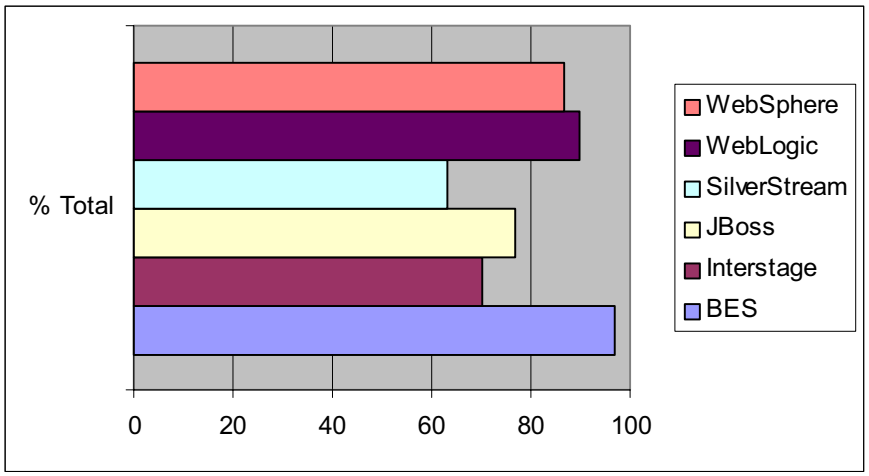
3.2.2. EJB Container and Bean features

CMP Entity bean support is a major differentiator between products, and has an impact on both useability and performance. Some products adhere closely to the EJB specification (E.g. Borland Enterprise Server supports the three commit options explicitly), whereas others have different features and options for optimising performance. However, all of the products require an effort to understand the features and their impact on CMP Entity bean performance.

Three products now support EJB 2.0 (Borland Enterprise Server, WebSphere and WebLogic).

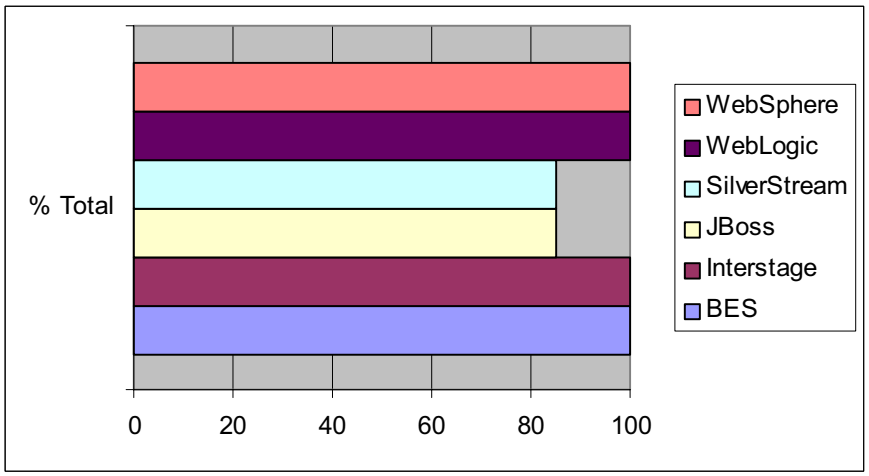
Borland Enterprise Server has uniformly good EJB container support, followed closely by WebLogic, and WebSphere. Note that this does not invariably equate to better overall performance, but does provide more deployment and performance options.

[7] This is also true for other categories.



3.2.3. Services

All the products evaluated have good support for J2EE services.

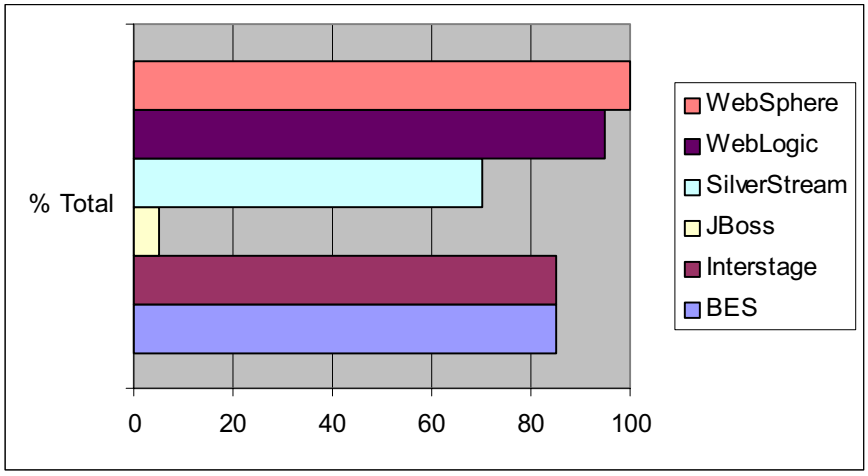


3.2.4. Scalability and Availability

Because clustering, failover and load balancing are outside the scope of the J2EE specification, products show a wide range of abilities in these areas. Basic products support only simple session-level load balancing (a client uses only the first allocated server). More sophisticated products support method-level load balancing and failover (a client may be allocated to multiple servers).

WebSphere and WebLogic are in front of the rest of the field in this category. They both have extensive support for load balancing, clustering and failover and are candidates for deployment into demanding enterprise environments. The other products have at least one weakness in this area, typically in their support for complex load balancing (Borland Enterprise Server and SilverStream), or failover (INTERSTAGE and SilverStream). Note that JBoss has no support for clustering, failover or load-balancing across multiple instances of the application server (either on the same or multiple machines) so rates

poorly in this category. Version 3.0, in alpha testing at the time of writing, promises to fix this.



3.2.5. Development and Deployment

The better products provide complete EJB lifecycle tool support in an integrated environment, for both server and client side development. Few products provide complete tool support for every task[8].

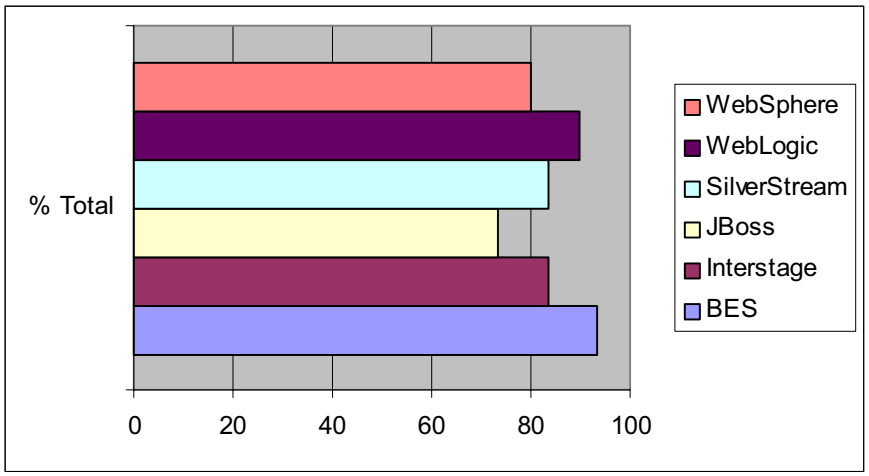
On the client side the J2EE specification suggests that varying degrees of tool support for J2EE application container installation, application client generation, deployment and live-update will be provided. In this respect some products provide no tool support and others completely automate the process. This would make a critical difference if thousands of clients have to be installed and maintained.

To adequately determine if the tool support provided is sufficient for the type of development/deployment required, some hands on experience and testing against a number of typical scenarios is recommended. Our experiments show that there are significant differences in the number of steps required to do various EJB related tasks across products. For example, one product took half the number of steps (60) as another (120) to develop the deployment descriptor and deploy the same bean.

TogetherSoft Control Centre has a (3rd party) JBoss plugin. To provide a fair comparison for development, JBoss was evaluated as if the developer was using this collection of 3rd party products to provide a complete develop-deploy-debug IDE. JBoss by itself comes with only a few development tools.

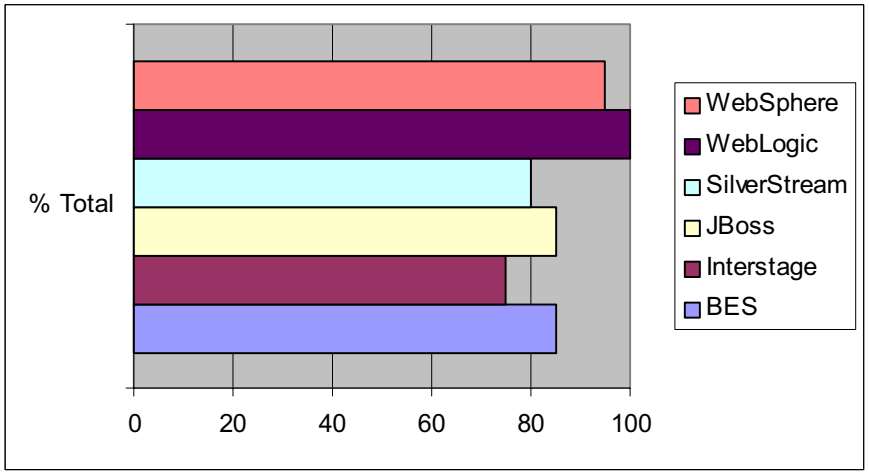
Most products now provide adequate tool support for EJB development and deployment, but many are still let down by bugs encountered during the development and deployment cycle.

[8] E.g. Even SilverStream, which provides a good tool environment for just about everything, still requires initial command line deployment of J2EE application clients onto the server.



3.2.6. System Management

Most products have reasonably sophisticated management tools, which is essential because monitoring and debugging application server deployments is a crucial and often difficult task. However, there are some differences in the way in which products are installed and updated, and in the level of detail available for monitoring deployed EJBs.



3.3. J2EE support

This set of evaluation criteria covers J2EE support, including components, protocols, modules, services and containers. Borland Enterprise Server has excellent support for the constituent parts of J2EE. INTERSTAGE, JBoss, WebLogic and WebSphere do not provide complete support for all module types. SilverStream is only let down by incomplete explicit support for J2EE containers.

Feature	BES	IAS	JBoss	SS	WLS	WAS
J2EE Components supported	5	5	4	5	5	5
J2EE Protocols supported	5	5	4	5	5	5
J2EE Modules supported	5	3	4	5	4	4
J2EE Services supported	5	5	4	5	5	5
J2EE Container support	5	5	3	3	4	5

Brief explanatory notes on the evaluation criteria:

J2EE Components supported: J2EE components are application-level software units. J2EE defines four types of components, with five instances: Servlets, JSP (both Web components), EJB, Applets, and J2EE application clients.

J2EE Protocols supported: J2EE protocols include: RMI-IIOP, HTTP, SSL, RMI, JavaMail, and JMS.

J2EE Modules supported: A J2EE module is a software unit that consists of one or more J2EE components of the same container type and one deployment descriptor of that type. J2EE Module types are: Enterprise Application Archive (EAR), Web Application Archive (WAR), Java Archive (JAR), Client Application Archive (CAR).

J2EE Services supported: JDBC 2.0, JNDI, Corba, JTA, XML DD.

J2EE container support: A container provides lifecycle management and other services to components. J2EE containers are EJB, Web (Web, JSP, Servlet), Applet, and client. The J2EE specification does not require support for explicit containers (they may just be implemented as part of the server process), but more sophisticated products will provide them for added flexibility, possibly with the ability to run them in their own JVM, or standalone on same or different machines.

3.4. EJB Container and Bean features

These criteria evaluate important features of the EJB container, and Session and Entity bean support. In this category products will exhibit a wide range of scores, as many of these features are not mandated by the EJB specification. In general however, all products have a good range of features for EJB support. Only INTERSTAGE and SilverStream rate quite low in their support for entity bean caching. INTERSTAGE's is limited in this area and SilverStream only supports Commit Mode C with no object pool.

Feature	BES	IAS	JBoss	SS	WLS	WAS
EJB Version support	5	4	4	4	5	5
EJB container	5	3	3	3	5	4
Session Bean support	4	3	4	3	5	5
CMP Support	5	4	4	3	4	4
Entity Bean Pool and Cache	5	2	5	1	4	5
CMP Entity Finder methods	5	5	3	5	4	3

Brief explanatory notes on the evaluation criteria:

EJB Version support: What EJB specification does it comply with? (Best is EJB 2.0).

EJB container: Overall EJB container architecture. Support for core features, but also: Can EJB services be run in and out of the EJB container? Can EJBs be deployed/redeployed without restarting the container/server? Can deployment settings be changed in deployed EJBs without having to redeploy them or restart the container/server? Is there in-built EJB container support for primary key generation? Is there support for plug-able O/R mapping tools?

Session Bean support: Does the container support Session bean pooling? Can the minimum and maximum number of instances in the pool be set? Is stateful session bean passivation/activation to/from secondary storage supported? Is stateful session bean failover supported?

CMP Support: Can the CMP Engine be customised or replaced? Does the engine support optimistic concurrency, and are there user settings? Does the container optimise `ejbStore` calls either automatically by detecting if fields haven't changed since loading or through deployment settings (e.g. readonly). How sophisticated is the CM Persistence support (bean/database mapping capabilities)?

Entity bean Pool and Cache: Is a Passivated bean pool, a Ready pool (Activated bean pool) or Cache (activated bean pool with data) supported? Are commit options A, B, and C explicitly supported? Are other optimisations supported?

CMP Entity Finder methods: Is there support for lazy (just primary key) and eager (primary key and data) finder object creation? Do multi-object finder methods load all required rows in one database access? Are finder methods automatically generated by the container (all the deployer has to do is add a "where" clause using the deployment tool)?

3.5. Services

Features for J2EE services such as transactions, database access and security are captured in this category.

Feature	BES	IAS	JBoss	SS	WLS	WAS
Transaction support	5	5	5	4	5	5
Database support	5	5	4	4	5	5
Security	5	5	5	5	5	5
Naming services	5	5	3	4	5	5

Brief explanatory notes on the evaluation criteria:

Transaction support: Is JTA/JTS supported? Are bean managed, container managed and client managed transactions supported? Is there support for transaction attributes? Is XA (two-phase commit) supported?

Database support: Does the server pool database connections? Can the minimum/maximum number of connections, and connection timeout be specified? Is JDBC supported? Is there support for overriding the default database isolation level?

Security: Is authentication supported? Is method, application or role based authorization supported? Is secure client communication supported? Is auditing/logging supported?

Naming services: How well supported is JNDI? Does it offer application environment configuration, support for resource factory management? How well supported are directory services and integration of external directory services (E.g. LDAP)? Can the naming service be replicated?

3.6. Scalability and Availability

Clustering, load balancing and failover are critical features for enterprise systems in order to ensure high availability and scalability. Because these are not specified in J2EE there is room for vendors to differentiate their products in this category. Both WebSphere and WebLogic are outstanding in these *enterprise class* features, and only JBoss is inadequate due to its lack of support for scaling out and failover in the current version.

Feature	BES	IAS	JBoss	SS	WLS	WAS
Load balancing	3	5	0	3	5	5
Clustering	5	4	0	5	5	5
Failover	4	3	0	3	5	5
Replication and concurrency	5	5	1	3	4	5

Brief explanatory notes on the evaluation criteria:

Load balancing: Is load balancing provided? Can the load balancer be customised or replaced? Can the load balancing strategy be configured or customised? How many load balancing strategies are supported?

Clustering: How easy is clustering to set up? Is there complete/partial tool support? Can all components be replicated (on different machines) for improved scalability? How effective is clustering at run-time?

Failover: Is failover supported? Is automatic failover provided for all clustering components? Is method or session level failover supported? Client code requires no modifications to cope with failover of any cluster component.

Replication and concurrency: To provide flexible product scalability a number of implementations of J2EE are possible. Can multiple instances of containers be run? Can multiple servers be run on a machine, and can J2EE services be run as separate processes, on same or different machines? Can concurrency be limited in each server (e.g. by limiting Orb threads)?

3.7. Development and Deployment

These features assess the ease of development and deployment with particular emphasis on the tool support offered by each product. This is perhaps the most subjective area in this assessment. However, we include it as tool support and usability is an area in which vendors attempt to distinguish themselves, and our experience with the range of products is extremely varied.

The following points should be noted, and the extra cost of tool purchase, integration and maintenance needs to be taken into account where required:

- We have assumed that JBuilder is used in conjunction with Borland Enterprise Server.
- VisualCafe is used with WebLogic.
- The SilverStream toolkit is extensive and bundled with the application server; there is also extensive support for 3rd party IDEs.
- WebSphere is integrated with VisualAge. However, there is no support for integration with other IDEs for Entity Beans.
- INTERSTAGE also has tool support for development and deployment, but these tools are dedicated to INTERSTAGE and lack support for CAR modules.
- As well as JBuilder and TogetherSoft Control Centre (which requires an extra JBoss plugin), JBoss supports VisualAge and Forte.

Feature	BES	IAS	JBoss	SS	WLS	WAS
General tool support	5	5	2	5	5	3
EJB Development support	5	5	4	4	5	4
EJB Deployment support	5	4	4	5	5	3
Other tool support	4	3	3	5	3	4
System Robustness During Development	4	4	5	3	4	5
Modularity and Integration	5	4	4	3	5	5

Brief explanatory notes on the evaluation criteria:

General tool support: How many, how sophisticated and how usable are the tools provided as packaged? Is there support/integration for 3rd party IDEs and tools? Does the resulting set of tools offer an integrated development, deployment, and debugging environment?

EJB Development support: How easy is it to create a new EJB jar file or import an existing JAR/classes? Is there tool support for creating vendor specific deployment information? Is it easy to create CMP Entity bean deployment descriptors, including mapping persistent fields to database tables and columns? Are CMP Entity beans/fields automatically mapped to default table/column names?

EJB Deployment support: What is the level of tool support for EJB deployment? Can EJBs be deployed from the development tool or IDE? Can EJBs be deployed to remote servers and clusters from the tools?

Other tool support: Is there tool support for data source specification and deployment? Is there tool support for remote debugging and profiling? Is there tool support for J2EE application client CAR file generation, verification and deployment? Is there tool support for J2EE application client container installation and automatic update of CAR files?

System Robustness During Development: How stable were the products (server and tools) during development, in terms of their effect on developer productivity? Did bugs interfere with development, deployment or testing? Were workarounds, patches or new versions required? Are there outstanding issues and problems?

Modularity and Integration: Can the web server, JVM and other components of the server be replaced? Is there support for EAI adaptors, and the Java Connection Architecture[9].

3.8. System Management

Issues concerning system configuration, management and monitoring are evaluated in this category. Overall, all of the products offer a good range of features in these categories.

Feature	BES	IAS	JBoss	SS	WLS	WAS
Server installation	5	3	5	4	5	5
Server and services administration	4	3	4	4	5	5
EJB administration and monitoring	3	4	3	3	5	4
Debugging and logging	5	5	5	5	5	5

Brief explanatory notes on the evaluation criteria:

Server installation: Server is easy to install and configure. This includes provision of all major components, complete automation of install, speed of installation, how cleanly it uninstalls, how easy it is to upgrade with new versions, and ease of specifying database drivers and getting it working with the required databases.

Server and services administration: All servers and services can be located and monitored, started and stopped. Server, service and container properties can be viewed and changed. Server and container performance can be monitored. Are Java Management Extensions (JMX) supported?

EJB administration and monitoring: Deployed EJBs can be redeployed, undeployed, monitored, and removed. The number of EJB instances and life-cycle states

[9] Full JCA support will require J2EE 1.3.

can be monitored. Deployed EJB deployment settings can be examine directly from management console?

Debugging and logging: Debugging, statistics, trace, event and logging levels can be viewed and changed. Log files and exceptions can be examined.

4. Quantitative Evaluation

4.1. Results Summary

4.1.1. Test Configurations

Each product has been tested in 4 application configurations, namely:

1. Single application server machine, session bean only code
2. Clustered (2) application server machines, session bean only code
3. Single application server machine, session bean and CMP entity bean code
4. Clustered (2) application server machines, session bean and CMP entity bean code

With each product, extensive experimentation was performed to configure the application server to perform as well as possible in the test environment. This typically meant finding a good mix of connections for the database and application server threads. By setting these values where possible to known maximum sizes we were able to control the resources the application servers utilized so they were able to handle increasingly large client loads. Not surprisingly, the exact settings to achieve strong performance tended to be different for each product, but were all pretty much in the range of 20-50 threads and database connections.

The tests were also monitored to ensure the client applications, network, and database server were not influencing test performance. In all tests reported except one[10], the CPU utilization on the client machine was under 20%, the network load averages about 5%, and database CPU a maximum of 20-40%.

For each product, we have attempted to use all the features provided by the technology to gain greatest performance. However, there are some rules that the tests were not allowed to violate:

- Container managed transactions must be used for all EJB methods.
- Transactions are not configured to use two phase commit and XA.
- Entity bean commit option A is generally not very useful as it assumes exclusive access to the database, and precludes clustering. It was allowed for the StockItem bean as it is readonly.
- The stateless session bean code uses prepared SQL statements.

[10] Borland Enterprise Server Clustered SB test used higher than expected client and database utilisation, see 4.2 .

4.1.2. Stateless Session Beans

Figure 4 shows the application throughput achieved by each product tested running on a single server machine. Borland Enterprise Server is the fastest product by a substantial margin (50% faster than the runner up, WebSphere), and exhibits exemplary scalability with no drop in throughput. WebSphere and Weblogic exhibit good scalability with only 7% decrease in throughput at 1000 clients. INTERSTAGE is fourth, but is significantly less scalable, dropping off by 23% at 1000 clients.

There is nothing to pick between JBoss and SilverStream with the slowest performance (well under half of the leaders), and degrading significantly (up to 40% slower) as client load increases. Both JBoss and SilverStream are unable to utilise all the CPU capacity of the servers even at 100 clients, and a decreasing amount with increasing load.

Depending on load, the best product (Borland Enterprise Server) is between 3.7 and 5.3 times faster than the slowest products.

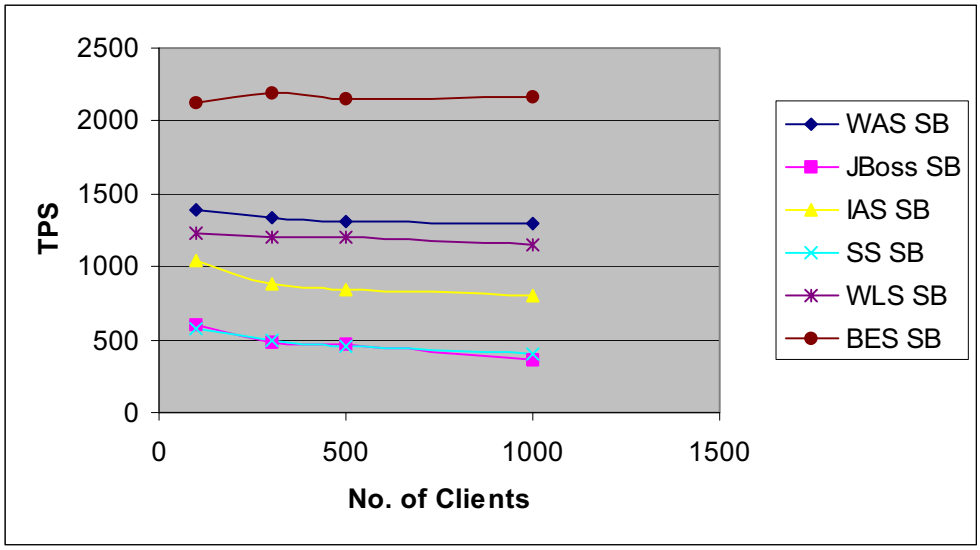


FIGURE 4 STATELESS SESSION BEAN – SINGLE SERVER THROUGHPUT

Figure 5 shows the throughput achieved by all products when the test case is scaled out to include two application server machines in a clustered configuration[11]. Once again the Borland technology performs strongest, followed by WebSphere, and then WebLogic Server and INTERSTAGE (up from 4th place for single server) tied at 3rd place. Borland Enterprise Server is 1000TPS ahead of WebSphere (40% faster), and is again notable for its scalable clustered performance.

The capacity of the clustered machines provides a more scalable and higher performing configuration for SilverStream, but it lags behind the other technologies.

[11] JBoss is not included as the version tested does not support clustering.

The best product (Borland Enterprise Server) is between 3.35 and 4.8 times faster than the slowest product, slightly less difference than the single server results. It is also significant to note that the single server SB results for Borland Enterprise Server are equal to or better than the clustered SB results for all products except WebSphere – i.e. Borland Enterprise Server provides the same or better throughput with half the middle tier resources.

These are not the best possible Borland Enterprise Server clustered results, due to database and client saturation (indicated by use of dashed line). See the discussion in section 4.2.

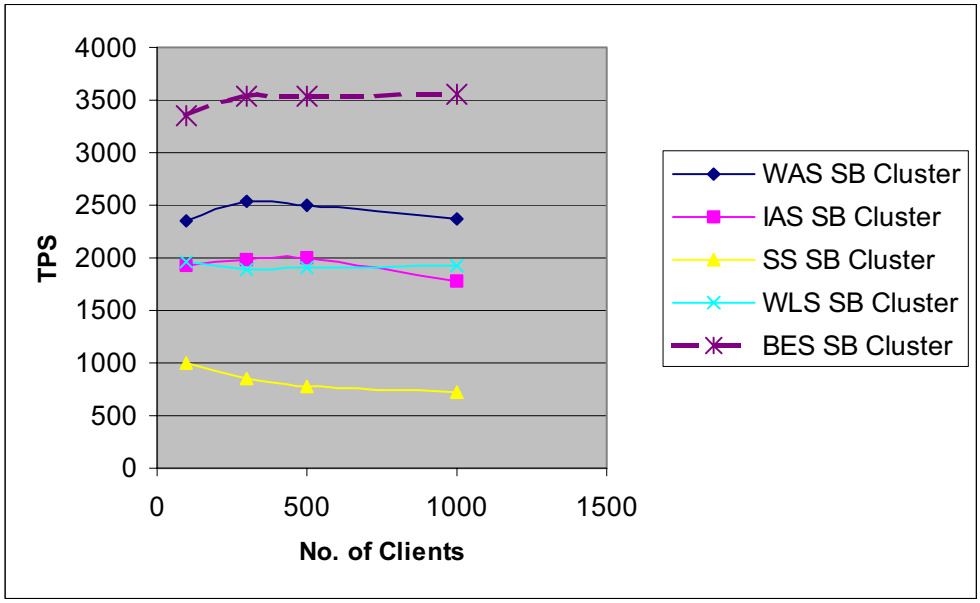


FIGURE 5 STATELESS SESSION BEAN – 2 SERVER CLUSTER THROUGHPUT

In theory it would be desirable if doubling the processing capacity available to the application would also double the throughput, a 100% increase in throughput. Figure 6 shows that WebSphere clustering gives a maximum increase of 90% throughput, SilverStream 80%, and WebLogic 70%, and Borland Enterprise Server 64%[12]. INTERSTAGE by comparison exhibits a better than perfect best-case improvement of 140%. This is not as impossible as it looks, as if a server is heavily saturated it is impossible to tell what improvement will result from providing more resources. In the single server tests, the INTERSTAGE machine is heavily saturated.

[12] Borland Enterprise Server SB clustering performance was prematurely limited due to saturation of the database.

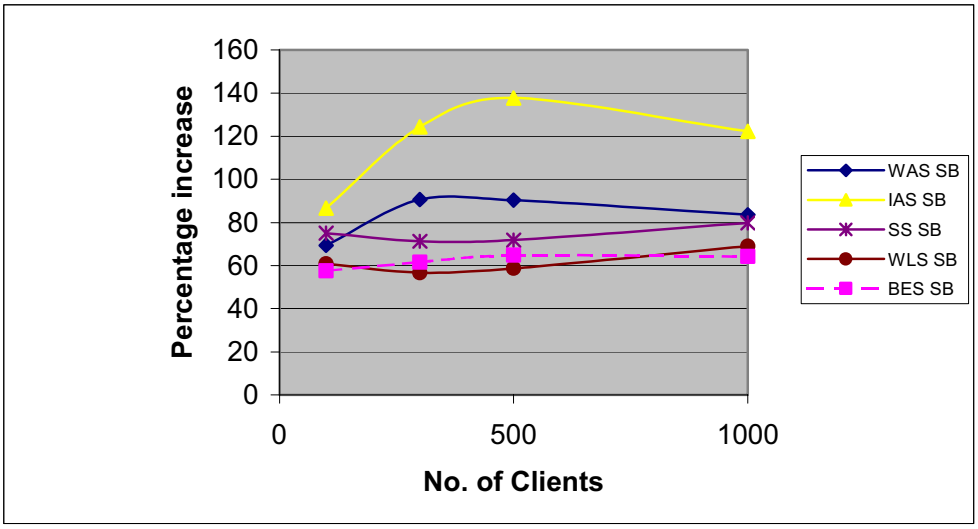


FIGURE 6 SCALABILITY OF CLUSTERED SB CONFIGURATION COMPARED TO SINGLE SERVER

4.1.3. CMP Entity Beans

Figure 7 shows the performance of all products tested using container managed persistence (CMP) entity beans[13]. First, the variation in performance across products is notable, with the best (Borland Enterprise Server) providing almost an order of magnitude better performance than the worst (JBoss). Borland Enterprise Server and INTERSTAGE both have the same performance at 100 clients (1200TPS), but Borland Enterprise Server rapidly pulls out in front (peaking at 1254TPS), and INTERSTAGE drops rapidly to 850TPS (30% less) until flattening out at 500 clients. Borland Enterprise Server again shows unbeatable scalability with no drop in throughput at 1000 clients. WebSphere has good performance and scalability, consistently maintaining about 900TPS across all client loads, but is 30% slower than Borland Enterprise Server. WebLogic Server follows with a peak of 800TPS, and also shows good scalability as the client load grows (dropping only to 715TPS or 10% at 1000 clients).

JBoss does not provide particularly competitive performance in this test. Both JBoss and WebLogic Server are unable to utilise the total CPU capacity in these tests (using only 60% cpu, decreasing with client load, and 84% cpu, respectively).

The best peak CMP performance (1254TPS, BES) is 43% slower than the Borland Enterprise Server Session Bean performance (2186TPS), but only slightly behind the best peak Session Bean performance of the other products (1400TPS, WebSphere). Session beans have a reduced advantage over CMP Beans for some other products.

[13] SilverStream was excluded due to problems running the CMP test.

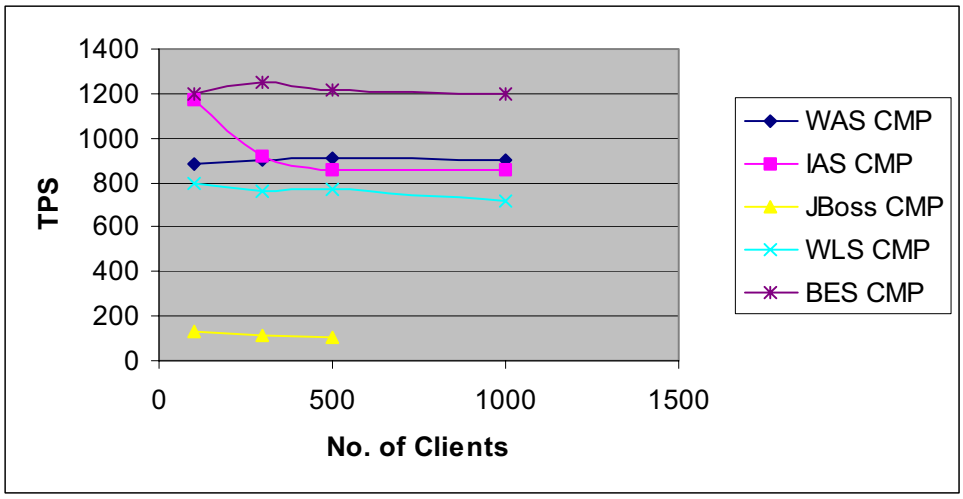


FIGURE 7 CMP – SINGLE SERVER THROUGHPUT

Figure 8 shows the performance results from scaling out the CMP test to run on 2 server machines in a cluster. INTERSTAGE benefits dramatically from the additional capacity of the cluster, and provides slightly better performance than Borland Enterprise Server, and significantly better performance than WebLogic Server and WebSphere, although with slightly worse scalability. The peak throughput is 2360TPS at 300 clients, which is 70% faster than WebLogic Server. However, by 1000 clients the TPS has dropped by 12% to 2035.

Borland Enterprise Server peaks at 2179TPS, and again tops the scalability stakes with a variation in TPS of only a few percent, finally overtaking INTERSTAGE by 1000 clients. WebSphere and WebLogic provide similar performance and demonstrate good scalability. At 100 clients WebSphere achieves 1524TPS and WebLogic 1440TPS, both dropping by about 6% at 1000 clients.

Significantly, the best clustered CMP performances (2360TPS, INTERSTAGE, and 2179 TPS, BES) are better than or comparable to the majority of the clustered Session Bean performances. Only the Borland Enterprise Server clustered SB result is substantially faster (3500TPS).

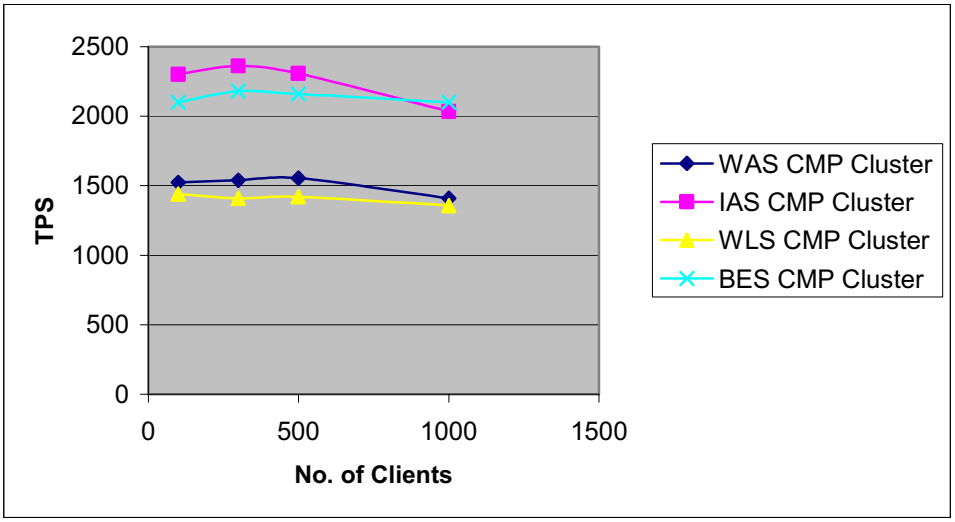


FIGURE 8 CMP – 2 SERVER CLUSTER THROUGHPUT

Comparing the clustered CMP results with single server results we once again notice that INTERSTAGE has better than perfect scalability with a maximum of 169% increase in throughput (See Figure 9). WebLogic Server shows good clustering scalability with a range of 80-90%, Borland Enterprise Server at 77%, and WebSphere in the range 56-70% (but decreasing with increasing client load).

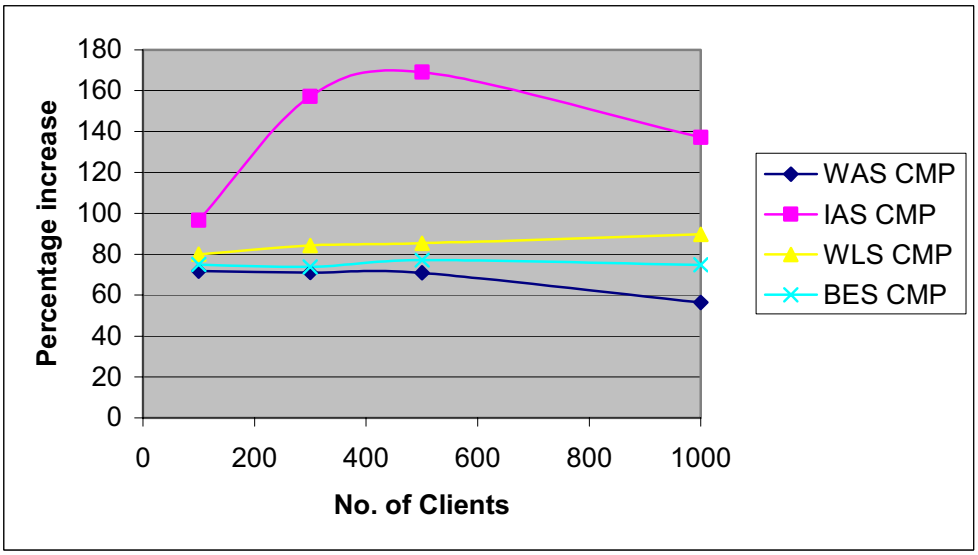


FIGURE 9 SCALABILITY OF CLUSTERED CMP CONFIGURATION COMPARED TO SINGLE SERVER

A piece of folk-law going around JavaOne in 2001 was "Real performance only comes by banning Entity Beans."[14] The two versions of Stock-Online allow us to accurately determine the merits of this assertion. We have already noted that the best CMP results across all the products are comparable with the majority of Session bean results, but what if we compare SB and CMP for the *same* products? Figure 10 shows the percentage increase in TPS for SB over CMP for the same products (both single server and clustered) at 500 clients. The product with the biggest difference between SB and CMP performance is JBoss, with SB almost 350% faster than CMP. The other products have less difference. WebSphere, is 57%/68% faster (single and clustered respectively), WebLogic Server 37%/30% faster, and Borland Enterprise Server 43%/39% faster. The strangest result is INTERSTAGE, where *CMP* on a single server is 2% *faster* than Session beans, and clustered CMP 19% *faster*.

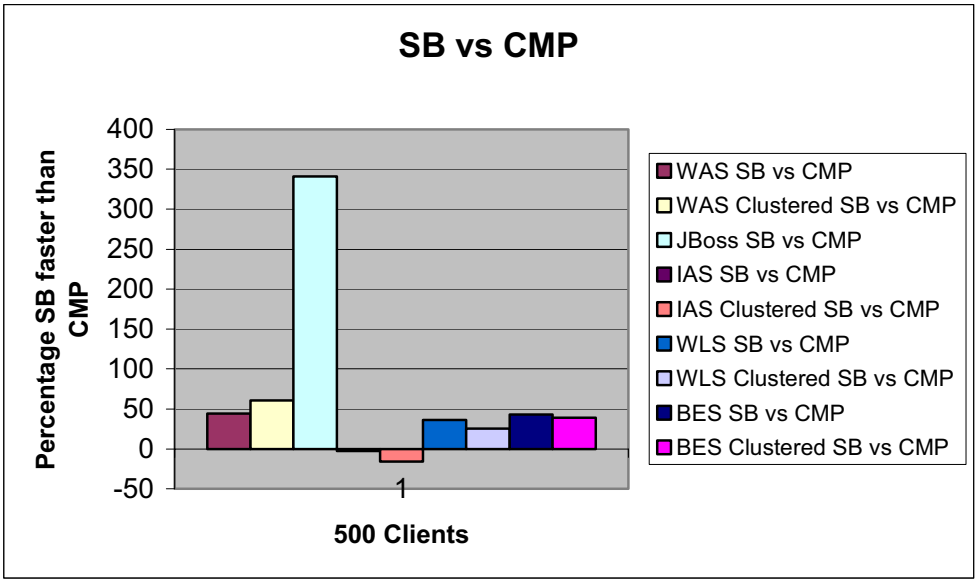


FIGURE 10 SESSION BEAN PERFORMANCE CF CMP ENTITY BEANS

4.1.4. Performance and Scalability Summary

Figure 11 summarises the six products that were evaluated quantitatively in terms of their raw performance (on a single server) and scalability (with increasing client load, and clustered on two servers). Borland Enterprise Server has a substantial lead over the other products in both performance and scalability in this version of the report. As the actual quantitative results show, however, there is no outright winner for all the tests (although Borland Enterprise Server came close in this version of the report), and specific combinations of applications, products and hardware configurations will perform differently.

[14] http://www-106.ibm.com/developerworks/java/library/j-j1wrap.html

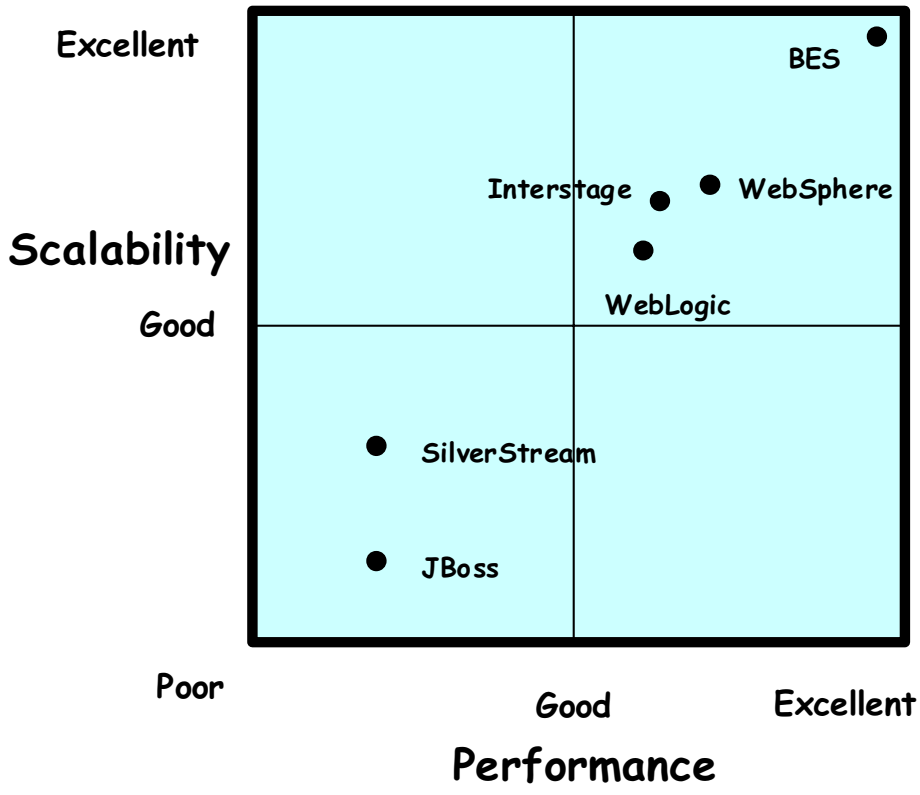


FIGURE 11 PERFORMANCE AND SCALABILITY SUMMARY

4.2. Borland Enterprise Server

The previous version of this report (v2.0), omitted Borland Enterprise Server performance results due to slower than expected performance on the new quad cpu test hardware, and our inability to explain and rectify the problem. We have since been able to fix the performance problems and have rerun the results. The performance was dramatically improved by changing the Sun JVM HotSpot option from "server" to "client" in the Borland Enterprise Server configuration file for the partitions. It is possible that Borland Enterprise Server was encountering the Sun *server* HotSpot JVM scalability limitations that we have previously reported[15]. Using the *client* HotSpot engine produced a large jump in performance, but required multiple JVMs to be run per server (1 per cpu) to get optimal performance.

4.2.1. Product-Specific Configuration Parameters

For the Borland Enterprise Server tests, the following configurations were used. These gave the best performance in our test environment.

All tests

- Sun JDK 1.3.1_03 HotSpot client engine. The server engine isn't as scalable, and the client engine giving a huge improvement of 400% over the server engine. There was also no discernable difference between JDK 1.3.1_01 and 1.3.1_03.
- 512MB heap size
- Client JVMs/processes: 8 per server (therefore 16 for clustered tests).
- 4 Partitions per server (JVMs with 1 EJB container per partition).
- CPU affinity set to true. Each JVM is bound to a cpu giving a 50% improvement to CMP throughput, and 40% for SB.
- JDBC 2.x type 4 driver (from Oracle)

Stateless Session Beans

- Maximum ORB threads set to 5

Container Managed Persistence

- Commit option C for all beans, except StockItem which used A.
- Maximum ORB threads set to 3
- CheckExistenceBeforeCreate set to false

[15] www.cmis.csiro.au/adsat/jvms.htm

4.2.2. Application Performance

Figure 12 shows the throughput for all Borland Enterprise Server tests. The obvious thing to note is that the scalability for all configurations is exemplary, with throughput actually increasing in some cases with increasing client load. Starting from the lowest result first, the single server CMP result is a constant 1200 TPS (90% server cpu, 9% db cpu), and the clustered CMP almost double this at 2100 TPS (75% higher, 90% each server cpu, 15% db cpu).

Coincidentally, the single server SB result is indistinguishable from the clustered CMP result, but using 85% server cpu and 37% db cpu. The clustered SB configuration gives a maximum of 3560TPS (at 1000 clients), an increase of 64% over the single server SB result. However, we noted that the single client machine was saturating, so the number of client machines was increased to 3 (each running at about 70% cpu). Also, the database was close to saturation, even though the two servers running Borland Enterprise Server were using only70% cpu (up to 60% spare capacity overall).

Our testing approach so far has been driven by two assumptions: (1) The middle-tier is being tested, so the database cannot be a bottleneck, and (2) using a fixed hardware configuration – we are interested in the best performance on the prescribed hardware. In this case however there are potentially two factors limiting the true clustered SB performance for Borland Enterprise Server: database and clients. A dashed line is used for the Borland Enterprise Server SB Cluster results to indicate that this is the lowest measured throughput. The removal of the database/client bottlenecks may allow the 60% spare middle-tier cpu capacity to be utilised bringing the Clustered throughput up to as much as 5600TPS (in theory).

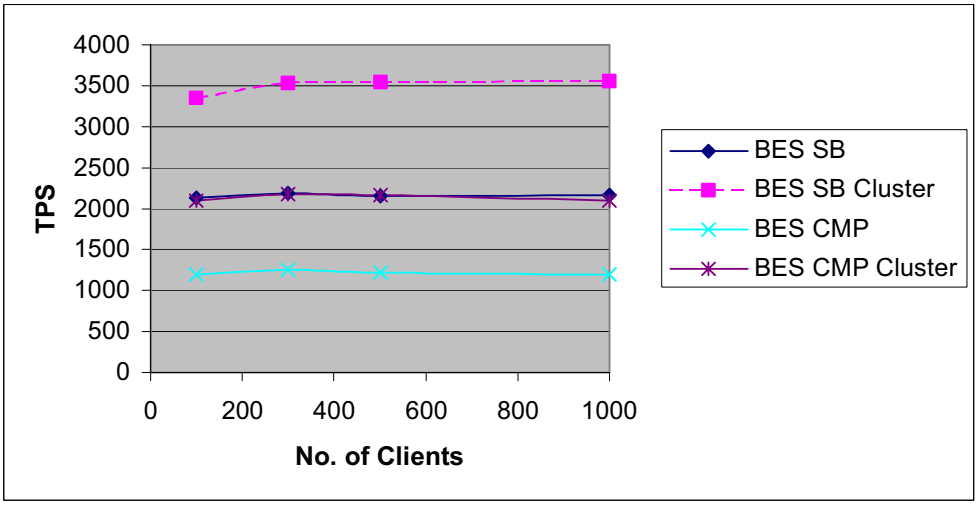


FIGURE 12 BORLAND ENTERPRISE SERVER APPLICATION THROUGHPUT

4.2.3. Performance Analysis

The raw performance of the Borland Application Server is exceptional in all test cases. It would appear that one of the reasons the product performs so well is that it is able to

extract more performance from the Oracle database due to optimizations that eliminate unnecessary database commits. The *partition* feature new to Borland Enterprise Server 5.0 (an evolution of explicit EJB container support in previous versions) easily allows multiple JVMs to be run per server thereby mitigating possible scalability limitations of the Sun client JVM. As a consequence it was also necessary to bind the processes to the processors.

In addition, the Borland Enterprise Server clustering features work well and allow the product to scale out to use additional hardware resources. The only caveat is that in a dynamic environment, the clustering mechanism may not be sufficient to adequately load balance the request load over the available EJB resources.

With no apparent architectural flaws revealed during testing, this product sets the standard for performance and scalability.

4.3. INTERSTAGE Application Server

4.3.1. Product-Specific Configuration Parameters

INTERSTAGE allows the EJB container to be configured in a number of ways to tune performance. After experimentation, we settled on the following configuration:

Stateless Session Beans Tests

- thread pool size = DB connection pool size = 32 ~ 96

Container Managed Persistence Tests

- thread pool size = DB connection pool size = 32 ~ 64
- entity bean pool sizes:
    - stubaccount = 10,000
    - stockitem = 10,000
    - stocktransaction = 10,000
    - stockholding = 10,000

INTERSTAGE has no explicit setting for the entity beans cache option at bean deployment time, as only commit mode C is supported. Instead, it provides a similar setting, known as instance management modes. There are four options for this setting, and in our tests we used the following:

- subaccount = ReadWrite
- stockitem = ReadOnly
- stocktransaction = ReadWrite
- stockholding = ReadWrite

For all tests, we set:

- Prepare Statement Cache Size = 100,000 (using default)
- JDK 1.3.0 Java HotSpot(TM) Client VM provided by Fujitsu
- JVM heap size = 512 MB

INTERSTAGE makes it necessary to run multiple instances on the same machine in order to achieve high performance. After considerable experimentation, we decided to use 1 server instance with bigger thread/DB pool size (64~96) for light client load (100~300) and 2 server instances with smaller thread/DB pool sizes (32~64) for heavy client load (500~1000).

4.3.2. Application Performance

The application throughput achieved in all the INTERSTAGE test configurations is shown in Figure 13. For the stateless session bean tests, peak performance for a single machine is 1037 tps, and 1935 tps for the cluster test. This represents near linear scalability. In particular, the INTERSTAGE server machines are saturated for all single and clustering tests.

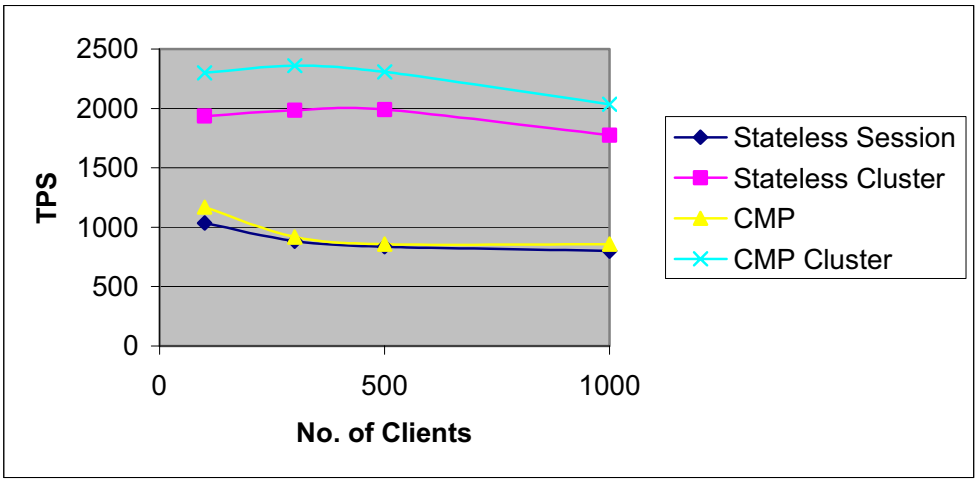


FIGURE 13 INTERSTAGE APPLICATION THROUGHPUT

The CMP tests exhibit peak performance with 100 clients at 1208 tps for a single machine, and 2360 tps for 300 clients in the clustering tests. Surprisingly, as the client load increases, there was no significant performance degradation as observed in INTERSTAGE v3.x.x testing. Again, the INTERSTAGE server machines are saturated.

4.3.3. Performance Analysis

INTERSTAGE Application Server provides excellent performance and scalability in these tests. In particular, INTERSTAGE 4.0's CMP entity beans out-performed the session beans. Note that our transaction mix consists of mostly read-only transactions. We expect that INTERSTAGE exploits this fact by efficiently reusing the read-only beans in cache. These results demonstrated that it is possible for CMP beans to over-perform session beans for a specific class of applications with careful tuning.

While tuning the thread pool size is important for performance, INTERSTAGE requires multiple EJB containers to run on a machine for optimal performance. This is likely due to the fact that the INTERSTAGE EJB container is tightly linked with the INTERSTAGE ORB.

4.4. JBoss

4.4.1. Product-Specific Configuration Parameters

JBoss configuration allows for a number of pluggable components and interceptors to be added or removed.[16] With the exception including the embedded Tomcat Java server, to provide a complete J2EE system, the JBoss configuration was left as distributed.

In general, JBoss seems somewhat insensitive to performance tuning. Provided that fairly sensible pool sizes, cache sizes and other resources are used, less than 2% performance variation was observed over a wide range of parameter values. For the JBoss tests, the following parameter settings were used -- these gave the best performance in our test environment:

Common

- 30 pooled database connections
- JDBC Oracle thin type 4 driver
- 256MB heap size
- Sun JDK 1.3.0_02 Java HotSpot(TM) Client VM
- Optimised intra-bean calls

Stateless Session Beans

- 200 pooled session beans

Container Managed Persistence

- 200 pooled entity beans and 200 pooled session beans
- Commit option B used with a 5000 bean cache
- Tuned CMP updates
- Read-only annotation on the stock item
- JAWS persistence engine

JBoss supports an `isModified` method in addition to tuned updates for CMP. There was very little difference in performance between tuned updates and an `isModified` method; in some cases, tuned updates outperformed `isModified`. Since `isModified` is a doubtful design practice and provides no performance improvement, it was not used.

JBoss also supports a read-ahead optimisation on CMP finders. In the configuration tested, this option does not provide any noticeable increase in performance.

[16] For example, a different persistence engine could be used to replace the packaged JAWS engine.

JBoss is very resource efficient, particularly for stateless session beans. While a 256Mb heap was used for best performance, it was possible to run the 100 client stateless session tests in a 32Mb heap with less than 1% performance degradation.

4.4.2. Application Performance

Figure 14 shows the performance of JBoss in both the stateless session bean and container managed persistence test configurations. Peak performance for the stateless session test is 600 TPS with 100 clients. Peak performance for the CMP test is 135 TPS with 100 clients. The application server machine runs at approximately 70-80% cpu utilization for SB tests, and 50-60% for CMP tests, at 100 clients. As the number of clients increases, one processor continues to run at high utilization, while the other processors decline, the average cpu dropping to 50% (SB) and 30% (CMP) by 1000 clients. The database machine is 20% utilized for the stateless session tests and only 10% for the CMP tests. Trying to run the 1000 client CMP test on JBoss failed as too many exceptions are produced.

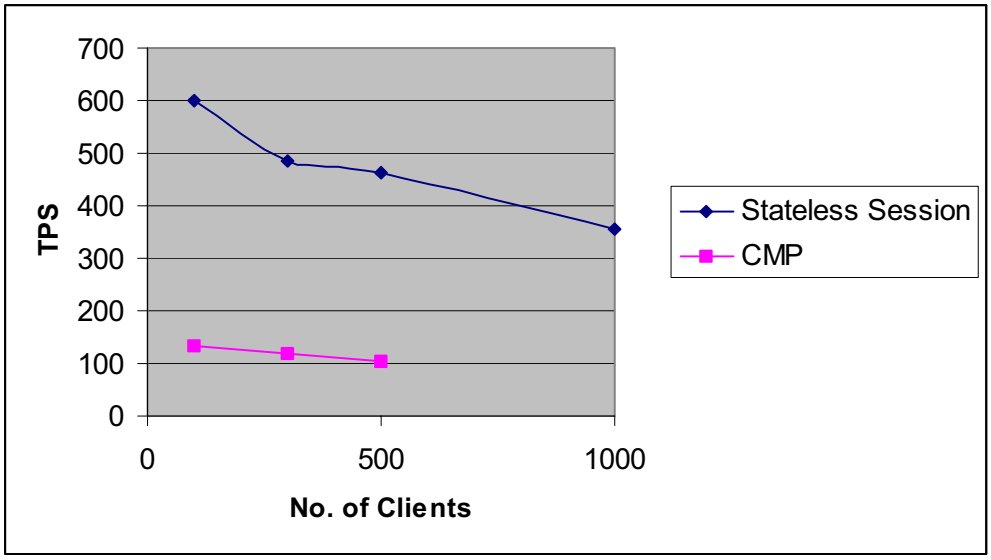


FIGURE 14 JBOSS THROUGHPUT

JBoss shows an unusual pattern of a high throughput on the first test run followed by a sharp decline in performance to a stable level in subsequent runs. Experiments with run sizes show a pattern of increasing disparity as the number of transactions in a single run increases. Since JBoss is an open-source project, it was possible to examine and modify the source code to test for likely causes of this phenomenon. However, code modifications have not yet halted this behaviour and it appears to be part of some complex interaction between the JBoss components. Only the stable performance figures have been used when reporting results.

4.4.3. Performance Analysis

At client loads of around 100 per application server, JBoss provides good stateless session bean performance. Performance however declines as the client loads increase. This means that a JBoss application may not be able to scale to handle very large bursts in traffic. The CMP application performance is quite low in comparison; it appears that the persistence manager is both demanding in terms of CPU resources and somewhat inefficient in managing entity beans with the B and C commit options.

There has been a general drop in performance since JBoss 2.2.2. There has been a minor drop in stateless session bean performance, and a significant drop in CMP performance. JBoss 2.4.3 introduced a number of new areas of functionality, so the reduction in performance is likely to be a traditional feature-performance trade-off

Since there is no clustering capability in the version of JBoss evaluated, it is not possible to spread load across several systems. JBoss also shows increasing processor asymmetry as load increases, so multi-processor machines can only provide a certain level of load balancing. This explains why JBoss performs almost identically on the current h/w environment (peak of 600TPS, dropping to 350TPS, quad cpu server) compared with the previous dual cpu servers (peak of 512TPS, dropping to 357TPS).

4.5. SilverStream Application Server

4.5.1. Product-Specific Configuration Parameters

The SilverStream tests used the following configuration parameters. These were settled upon after experimentation to discover the best settings to achieve highest performance.

- Java HotSpot(TM) Client VM (build 1.3.0_02, mixed mode)
- Maximum of 20 database connections
- Maximum of 20 beans in Session bean pool
- Three drivers tried were tried, namely the SilverStream OCI driver, the Oracle thin client (type 4) and Oracle OCI driver. The SilverStream driver was found to perform marginally faster and hence was used to gather results.
- Only 1 SilverStream server per machine used.
- CMP Entity beans only have option C

4.5.2. Application Performance

With a single SilverStream server, the stateless session bean gives the peak performance of 575 tps with 100 clients (See Figure 15). As the client loads grows to 500, the performance drops away, reducing by about 22%. The clustered stateless session bean configuration shows good scalability. With 100 clients, the throughput is 1006 tps, a 75% increase over a single server. With 500 clients, a 72% increase is observed, which is also good.

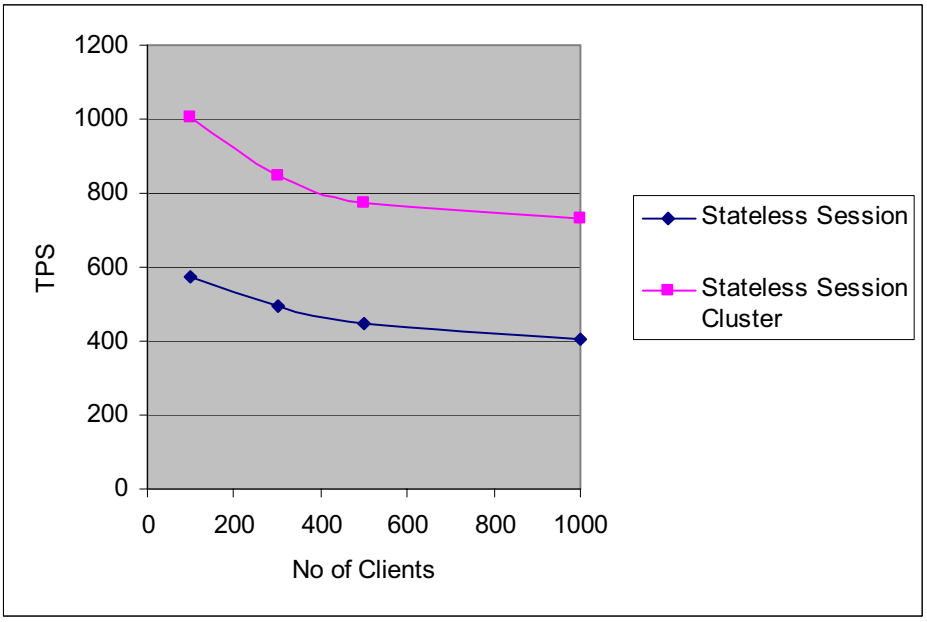


FIGURE 15 SILVERSTREAM APPLICATION THROUGHPUT

In this version of SilverStream server, we cannot successfully run our CMP Entity bean benchmark code as the application server hangs. However, there is no problem with only one client. Therefore, it is most likely that there is a bug in thread synchronization inside the application server. We have reported this problem to SilverStream, but because of the tight deadline for this version of the report there is insufficient time for SilverStream to provide a patch.

4.5.3. Analysis of Performance

The performance is best at client loads of around 100 per application. Performance however declines as the client load increases. Careful investigations show that the measured throughput values are not steady during the runs. At the beginning of a test, the performance of 1000 client threads is actually the same as that of 100 client threads. However, with 1000 clients the performance gradually decreases, thus reducing the average throughput. We are unable to explain this phenomenon at the moment.

Another observation is that the CPU usage is quite low on the server machine and database machine (32% and 10% respectively). It appears that there is heavy internal resource contention in the application server. If this problem could be fixed, much greater performance increase could be expected.

4.6. WebLogic Server

4.6.1. Product-Specific Configuration Parameters

The results presented show the overall throughput in terms of transactions per second. Test configurations included:

- Sun JDK1.3.1, Java HotSpot(TM) Client VM
- 100 - 1000 clients
- 20 threads. Experimentation demonstrated that this gave better performance on our testing environment than the default of 15, or higher numbers such as 30, 50, and so on.
- 1 WLS EJB container with a JVM heap size of 256MB. We experimented with clustering containers on the same machine but found that a single container provided the optimal throughput in the test environment. Clustering on the same machine only makes the CPUs busier, but with no performance gain.
- Initial connection pool size is set to 10, and maximum size is set to 40. Only 20-22 physical connections are actually created by the server. The Oracle thin driver supplied by BEA with WLS is used.
- The `IsModified` method is used for CMP tests.
- Default setting for the data source's WLS prepared statement cache.

It should be noted that the prepared statement cache settings are undocumented. We have tried different values, but did not find any impact on the performance.

4.6.2. Application Performance

Figure 16 charts the application throughput obtained with WLS v6.1. With a single machine running WLS, the stateless session bean tests show a peak performance of 1225 tps, and the CMP tests achieve 800 tps. Hence the use of an application architecture based on entity beans achieves approximately 2/3rds of the performance of the simpler session bean architecture.

Interestingly, unlike the tests of previous WLS versions, such as v6.0, where the server CPU is the performance bottleneck, in this version's (v6.1) tests, the WLS machine CPU is not fully saturated. In the stateless session bean tests, the CPU usage is 88%; and in the CMP tests, the CPU usage is 84%. The database CPU was approximately 20% utilized, and hence not the bottleneck. We have tried various configuration parameters, but could not make the server CPU usage higher. It appears that a bottleneck is due to some internal contention within the application server and Java Virtual Machine.

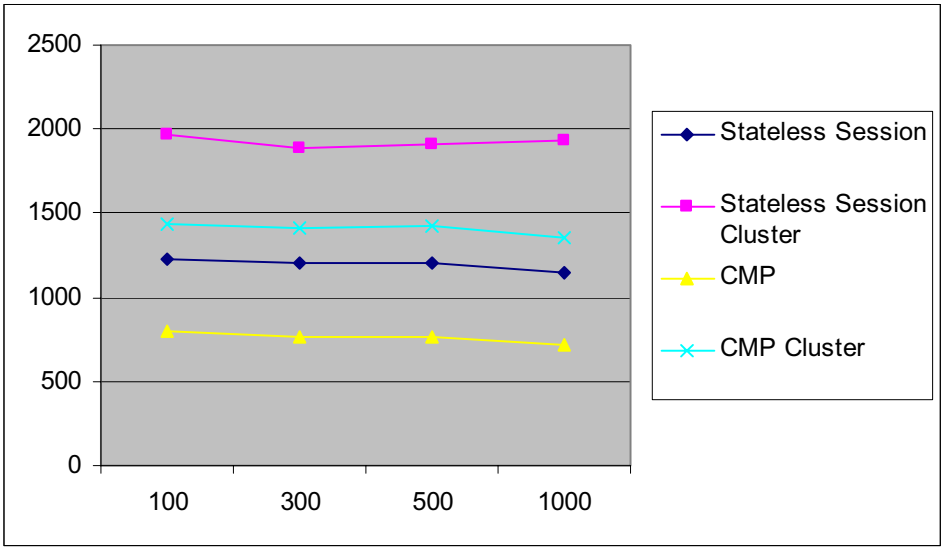


FIGURE 16 WLS THROUGHPUT

When the stateless session bean tests are clustered across 2 WLS machines, a peak performance of 1970 tps is obtained. This is approximately a 61% increase over the single machine peak. The WLS machines were at an average of 78% CPU utilization, and the database server was about at 71% CPU utilization.

It is worth pointing out that BEA has improved the clustered CMP performance considerably. Our clustered CMP tests show very good performance, peaking at 1440, which is approximately 85% increase over the single machine peak. The average WLS CPU utilization is 78%, which is the same as that of the clustered stateless bean tests. However, compared with the 71% database CPU utilization observed from the clustered stateless session bean tests, the database CPU utilization of the clustered CMP tests drops down to 40%. This indicates that with our single database server more WLS instances can be added in to the clustered group to provide more processing capability. The number of server machines that can be added solely depends on when the database server becomes saturated.

4.6.3. Performance Analysis[17]

WLS exhibits excellent performance and scalability across all tested configurations. There is actually no or very minimal performance decrease as the client load increases from 100 to 1000. This means WLS applications should be able to handle sudden bursts of transactions and provide continuing stable throughput.

[17] Our performance tests were conducted on WebLogic 6.1. Published ECPerf figures for WebLogic server 7.0 beta are available at www.theserverside.com/ecperf. WLS 7.0 is now released, but was unavailable at the time of our testing. BEA claims that 7.0 has enhanced performance over 6.1.

4.7. WebSphere Application Server

4.7.1. Product-Specific Configuration Parameters

The WebSphere tests used the following configuration parameters. Again, these were settled upon after experimentation to discover the best settings to achieve highest performance on the test environment.

All Tests J2RE 1.3.0 provided by IBM

Container Managed Persistence Tests

- 50 server threads and 50 database connections per container, reduced to 25 for cluster tests
- 512 MB heap size
- JDBC2.x driver type 4 (from oracle)
- Mode C cache, pool size 1000 per EJB, with read-only methods set where appropriate.
- Invoking entity beans from the session bean is set as 'call by reference'.

Session Bean Tests

- 50 server threads and 50 database connections per container, reduced to 25 for cluster tests
- 512 MB heap size
- JDBC2.x driver type 4 (from oracle)
- Prepared Statements are used, with a cache size of 1000.

For the clustering tests, a round-robin load balancing strategy was selected. In all tests, the administration database was hosted in a DB2 database on a server machine.

4.7.2. Application Performance

The application throughput achieved with WebSphere in all test cases is shown in Figure 17. Overall performance is strong across all test cases.

For the stateless session bean tests, peak throughput for the single server machine test is 1392 tps, and for the cluster peak throughput is 2538 tps. The bottleneck in all these tests appears to be the WebSphere server machines, because for both single server and clustering server tests, all testing machines running WebSphere application servers reached 100% CPU usage. This shows that WebSphere is able to make efficient use of multiple CPU machines.

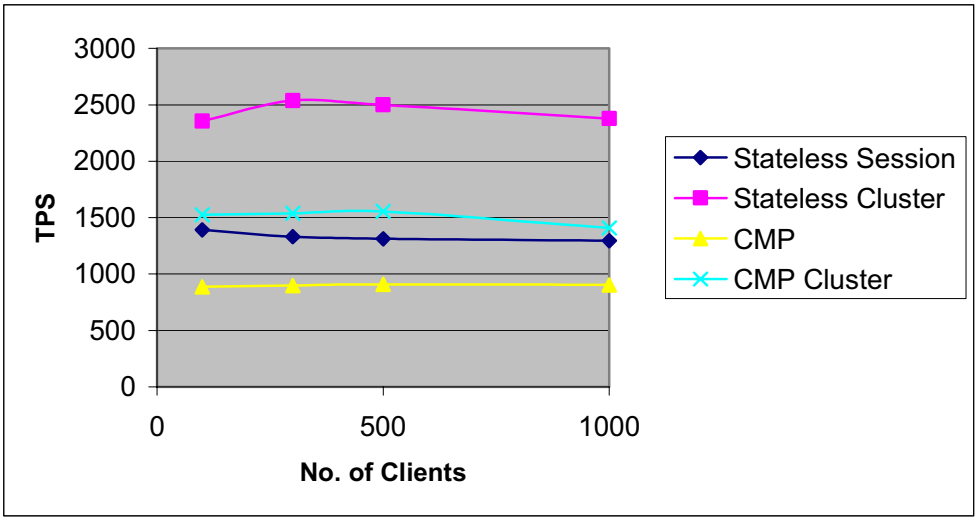


FIGURE 17 WAS APPLICATION THROUGHPUT

The CMP test performance peaks at 910 tps for a single server, and 1555 tps for a cluster. At these levels the WebSphere server machines were again saturated. Across the range of client loads tested, the scalability achieved by the clustered configuration is essentially linear, which is an excellent outcome. Interestingly, no significant performance drops were observed as the client load grows – this was not the case with WebSphere 3.5.x.

4.7.3. Performance Analysis

Overall, the performance of WebSphere is excellent across all tests. This is especially true of the stateless session bean results. This may benefit from WebSphere's efficient thread management and prepared statement cache. WebSphere's CMP performance is reasonable and competitive with other products. In both cases, WebSphere demonstrated good scalability. The speedup for stateless session bean and CMP entity bean tests is between 1.7 and 1.9, which is close to a linear speedup. This indicates that WebSphere is able to scale out efficiently by using additional hardware resources.

Since WebSphere is able to saturate the server machines for all tests, the server machine capacity is the bottleneck in these tests. WebSphere should therefore be able to achieve higher performance if more powerful machines are available.

In general it is difficult to determine the extent to which AppServer performance is helped or hindered by the underlying JVM. In the case of WebSphere, however, we believe that using the IBM JVM is a significant advantage. We investigated the relative performance of the hardware and JVMs used in this and the previous version of the report by running some JVM benchmarks and made a number of discoveries[18]. First, the IBM JVM is faster than the Sun JVM on both versions of the hardware – in some cases greater than 20% faster. Second, the IBM JVM is more scalable with increasing numbers of CPUs. The Sun JVM does not scale well with new objects created in multiple-threads on the quad cpu servers – in some cases it is slower than the dual cpu servers. Multiple–threads and new objects are typical of AppServer EJB usage and we expect this to have an impact on

[18] For JVM benchmarks and results see www.cmis.csiro.au/adsat/jvms.htm

performance. IBM has addressed the problem of scalable object allocation through the use of split heaps[19]. The AppServer performance results and our JVM benchmarks confirm that IBM has spent considerable effort developing a highly scalable server-side JVM which they view as a core component of their AppServer offering[20].

[19] http://www-106.ibm.com/developerworks/ibm/library/j-berry/?open&l=907,t=gr,p=bob

[20] A history of the IBM JVM can be found at http://www-106.ibm.com/developerworks/ibm/library/j-berry/sidebar.html

5. Product Overviews

5.1. Borland Enterprise Server EJB Service

5.1.1. Overview

Release 5.0 of the previously named "Borland Application Server" has been incorporated in the *Borland Enterprise Server* (BES) product range made up of the Web, VisiBroker (Corba) and AppServer (J2EE) editions. We evaluated the Borland Enterprise Server AppServer edition. Borland Enterprise Server 5.0 is a 1.3 compatible implementation[21]. It supports Servlets 2.3, JSP 1.2, EJB 2.0, and JDBC 2.0, JMS, IDL, JNDI, JTS, JavaMail, JAF, XML, and Security.

Borland Enterprise Server is built on top of Borland's VisiBroker CORBA technology. This allows it to leverage the underlying CORBA features such IIOP enabled communications, the Portable Object Adapter (POA), Objects-by-value (OBV), and RMI-over-IIOP. In addition, Borland Enterprise Server includes a XA-compliant Java Transaction Service (JTS 1.0) that operates with JDBC 2.0.

Connectivity to legacy systems using the J2EE Connector Architecture is also provided. This makes it possible to access a variety of ERP and CRM systems, TP monitors, and backend systems such as SAP, CICS, and MQSeries.

New in Borland Enterprise Server 5.0 are *partitions*, EJB 2.0 and J2EE 1.3 support, Web services, JAAS 1.0 (Java Authentication and Authorization Service) support, integrated Java/C++ visibroker, integrated security, and integrated SonicMQ JMS.

[21] http://java.sun.com/j2ee/compatibility.html

5.1.2. Application Server Architecture

Figure 18 shows the architecture of the Borland Enterprise Server[22].

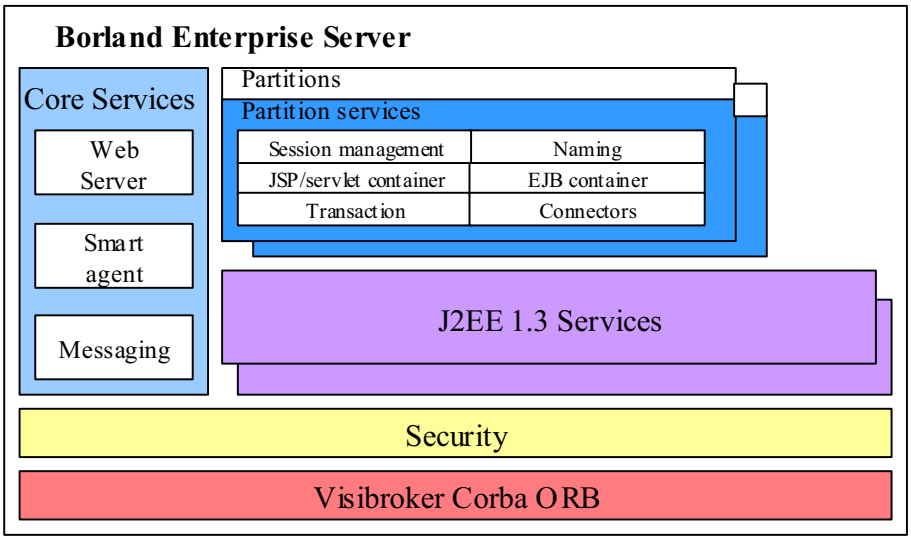


FIGURE 18 ARCHITECTURE OF BORLAND ENTERPRISE SERVER

Some of the key services are described below:

Object Request Broker: The foundation of this environment is Borland's CORBA product, VisiBroker, which supports the CORBA 2.4 specification and IIOP.

J2EE 1.3 Services: These are APIs for J2EE 1.3 services including servlets, JSP, JNDI, JTA, EJB 2.0, JCA, JDBC, JAAS, RMI-IIOP, JAXP, JavaMail, and JMS.

Web server: This is a fully integrated Tomcat 4.0.1 web container, and Apache 1.3 web server, with support for Java Servlets 2.3 and Java Server Pages 1.2.

Naming Service: This is a Java Naming and Directory Interface (JNDI) implementation that is implemented as a wrapper around the VisiBroker Naming Service.

Security: Security is bundled in Borland Enterprise Server 5.0, and includes strong 128-bit data encryption, flexible access control, user authentication supporting a variety of authentication realms and support for JSSE, JAAS, JCE, SSL, TLS, and X.509 standards.

Transaction Management*:* There are two transaction engines provided by Borland Enterprise Server. The VisiTransact TM is a JTS 1.0 compliant transaction engine supporting 2 phase commit, JDBC 2.0, and XA for distributed transactions. In contrast, the Lite TM does not provide support for two-phase commit transactions. If XA is not required, then better performance is likely with the Lite TM running in the partitions.

[22] Based on diagram in http://www.borland.com/bes/papers/partitioning.pdf

Partitions: New in Borland Enterprise Server 5.0 is the concept of *partitions*. Previously containers and services could be replicated in the same AppServer, and some services (E.g. Naming, session, transaction) could be run in the same processes as EJB containers. Partitions now allow containers and selected services and configurations to be grouped together, run in the same JVM, and optionally replicated. This allows for both the separation and replication of deployed applications for improved flexibility, performance, scalability, redundancy and ease of management.

Partitions provide an architecture that has some advantages over conventional application servers. With this architecture it is possible to deploy different applications within the same server and ensure they do not conflict as each partition results in a separate process and has its own configuration, memory, etc. Performance gains may also result from combining web and EJB containers in the same partition.

5.1.3. EJB Partition Features

In Borland Enterprise Server 5.0 EJBs are deployed to *partitions* instead of EJB containers, although partitions have an explicit EJB container. EJB containers can run within the Application Server (in partitions), independently, or embedded in applications. Due to the fact that Borland supports explicit partitions/containers, many configuration settings and properties can be set per partition. These include trace levels, class loader policies, stateful bean passivation timeouts, enabling JPDA Remote Debugging and run-time monitoring.

The following services can be run in a partition:

Session Storage (JSS): Stateful Session Storage service.

JDataStore Service (JDB): A local persistent store that can be used for CMP (alternatively, an external JDBC data source is required).

Lite Transaction Manager Service (JTS): The partition hosts a local one-phase commit transaction manager.

Naming Service (JNS): The partition hosts a local naming service.

5.1.4. Session Beans

Stateless

According to the EJB specification, stateless session bean instances have two states in their life cycle: *Does Not Exist* and *Method-Ready Pool*. The Borland Enterprise Server container currently implements a relatively simple approach to manage stateless session beans. Essentially the container creates as many stateless session beans as needed to handle the number of concurrent requests for the bean's services.

The container continues to create beans until the number of beans reaches the maximum number of threads configured in the underlying ORB. If the ORB threads limit is configured, then the number of beans created will not exceed the maximum number of threads that the container can run. Also, these beans do not have a timeout associated

with them. Therefore they will never be removed and remain in the pool ready to receive future requests on the session bean

Stateful

Stateful session beans have four states in their life cycle: *Does Not Exist, Method Ready, Method Ready in Transaction,* and *Passive*. The Borland Enterprise Server container does periodic passivation to secondary storage of non-active stateful session beans. This period is configurable (*PassivationTimeout*) with the default being 5 seconds. Another configuration option in the bean's deployment descriptor (*session timeout*) controls the amount of time the bean remains in secondary storage before being removed altogether. This configuration option is specific to Borland, and the default is 0, meaning never remove the bean.

5.1.5. Entity Beans

Borland Enterprise Server 5.0 has full support for EJB 2.0 (Message-driven beans, CMP 2.0, local interfaces, RMI-over-IIOP interoperability). However, as StockOnline is currently based on EJB 1.1, we did not evaluate the EJB 2.0 support directly.

The CMP finder methods can be set to either *verify* only (which just gets primary key - useful if many objects are returned but few are used), or *load* which gets the primary key and the data in one database access. There is also container support for the automatic generation of primary keys using Entity beans, Oracle sequences, or SQL Server Identity columns.

Borland Enterprise Server 5.0 has a sophisticated and highly optimised CMP engine, which provides full support for the three transaction commit-time options (A, B, C), and has a Pooled state pool and Ready state cache. The default size for these is 1000 instances, and is configurable. Concurrent entity bean access from multiple clients, as well as transactions, are managed using optimistic concurrency. There is fine-grain control over the concurrency behaviour, and the equivalent of field level locking can be achieved.

5.1.6. Transaction Service

There are two types of transaction managers available in Borland Enterprise Server 5.0:

Lite Transaction Manager

VisiTransact Transaction Manager

Lite TM is Borland's implementation of Java Transaction Services (JTS). It is the default transaction engine for partitions. It supports only a one-phase commit protocol, but can handle commits to multiple databases by setting the "allow unrecoverable completion" option, then the container makes a one-phase commit call to each participating database during the transaction commit process. This service is sufficient and efficient for all scenarios except driving transactions that require two-phase commit.

VisiTransact TM is an implementation of the Java Transaction Service (JTS) and CORBA Object Transaction Service specifications, as defined by Sun and OMG respectively. It is implemented on top of the VisiBroker ORB, and provides a transaction service, recovery and logging, XA integration with databases, and administration facilities.

5.1.7. Database Connectivity

Borland Enterprise Server 5.0 supports standard JDBC 1.x and JDBC 2.0 drivers. Therefore, there are two possible types of data sources: JDBC 1 data source and JDBC 2 data source. Borland Enterprise Server does JDBC connection pooling, and provides properties for both Jdbc 1 and Jdbc 2 datasources to adjust the prepared statement cache size, and the database connection limit.

Database Connection Pooling is built-in for all modes of data access (BMP, CMP, Session Beans, Servlets, JSPs, or any other client). The Borland Enterprise Server Jdbc connection pool (jdbc 1 and 2) has timeouts for enforcing the maximum duration for which a connection can be associated with a transaction, and the amount of time after which an unused connection is removed from the pool.

For Jdbc2 datasources, the properties are specified using DDEditor in jndi-definitions.xml file. There is support for overriding the default database isolation level using Borland Enterprise Server DDEditor. If no-isolation level is specified the default isolation level of the underlying driver is used.

Because some JDBC drivers are not fully Jdbc2 compliant, Borland Enterprise Server provides a way of dynamically exposing jdbc1 drivers as jdbc2 datasources using a dynamic wrapper.

5.1.8. Development Tools

Borland Enterprise Server 5.0 is supplied with a Deployment Descriptor and a Deployment Wizard.

The Deployment Descriptor Editor (DDE) can edit an EJB Jar file, or an XML DD file. The DDE can be used to create new EJB deployment descriptors from scratch, or modify existing files. The DDE allows a number of deployment fields to be changed, including the environment, EJB References, security roles, resource references, CMP database mappings, finders, properties and data sources. EJB jar files are deployed using the Borland Enterprise Server Deployment Wizard. It creates the required stubs and skeletons for the beans in the jar, validates the deployment descriptor and deploys the jar to the specified partitions. EJB jar files can be deployed, reloaded and removed without shutting the partitions down.

Apart from these tools, Borland Enterprise Server is well integrated with JBuilder 6.0 for development and deployment descriptor configuration. The configured EJB jar file can be deployed using the Borland Enterprise Server deployment wizard. A third party IDE can also be used for development. This development and deployment environment is flexible and easy to manage.

5.1.9. Scalability

Borland applications servers can be configured to form clusters of application servers based on IIOP. The clustering is achieved by using Borland's naming service. The naming service has built-in support for clustering, which forms the foundation for federation, replication, load balancing and fail over.

The naming service can be fully replicated with multiple copies using the same name store. One server is designated as the master, and the others run as slaves in standby mode.

When the master fails, a *slave* server takes over transparently. This is accomplished using a combination of bootstrapping protocols that are part of the CORBA Interoperable Naming Service (INS) specification, and Borland's own VisiBroker *Smart Agent* Technology. This failover scheme requires that the naming data is persistent. To this end, Borland Enterprise Server includes pluggable data store support that includes Borland's pure Java JDataStore, Oracle, Sybase and JDBC resources, or LDAP-based directories.

The default built-in load-balancing algorithm is *round robin*. Alternative load balancing algorithms can be programmed and specified at the cluster creation time. If one of the objects in a cluster fails, the Borland Enterprise Server runtime can automatically substitute it with the object references of one of the replicas in the cluster. Figure 19 shows one possible clustering configuration.

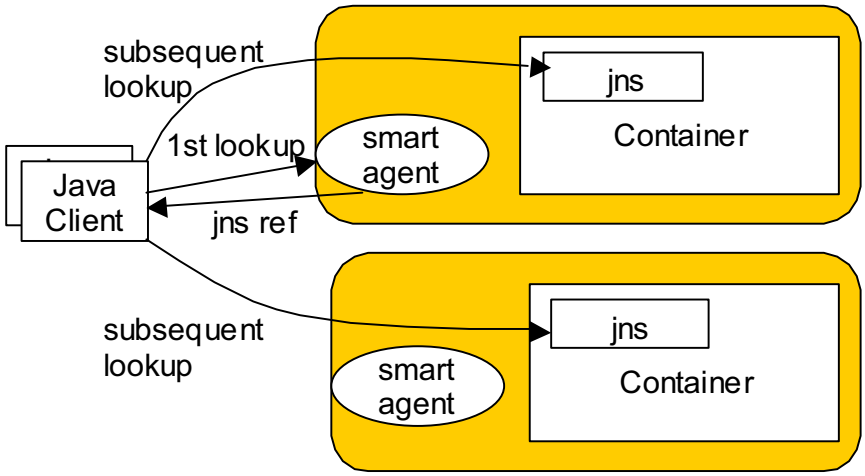


FIGURE 19 EXAMPLE BORLAND ENTERPRISE SERVER CLUSTER CONFIGURATION

5.1.10. System Management

The Borland Enterprise Server management console is the central management tool for Borland Enterprise Server applications. It allows servers on the network to be located; servers and services to be started, stopped and monitored; partitions to be created and cloned, containers and EJB jar files and Beans to be deployed and removed; server and container properties to be set; and server and container performance to be monitored.

One of the useful features that did not make it from BAS 4 to Borland Enterprise Server 5.0 is the ability to view and edit the deployment descriptors of deployed beans directly from the management console. Such editing requires the use of the separate 'Deployment Descriptor Editor' tool. Borland do provide such a tool, as described above, however the locating of the deployed jar is left up to the user. The simple ability of launching this tool to edit the descriptors a bean of interest from within the management console (such a context menu) would be useful to restore. Another useful feature that has not made it into version 5.0 is the ability to monitor the number of instances of beans as well as their states. Again restoring this ability would be useful.

Server debugging and statistics, and container tracing levels can be turned on from the console, and the resulting log files examined. There are also facilities for monitoring the performance and memory usage of a partition.

Figure 20 depicts a screen shot of the Borland Enterprise Server management console showing general information on an entity bean.

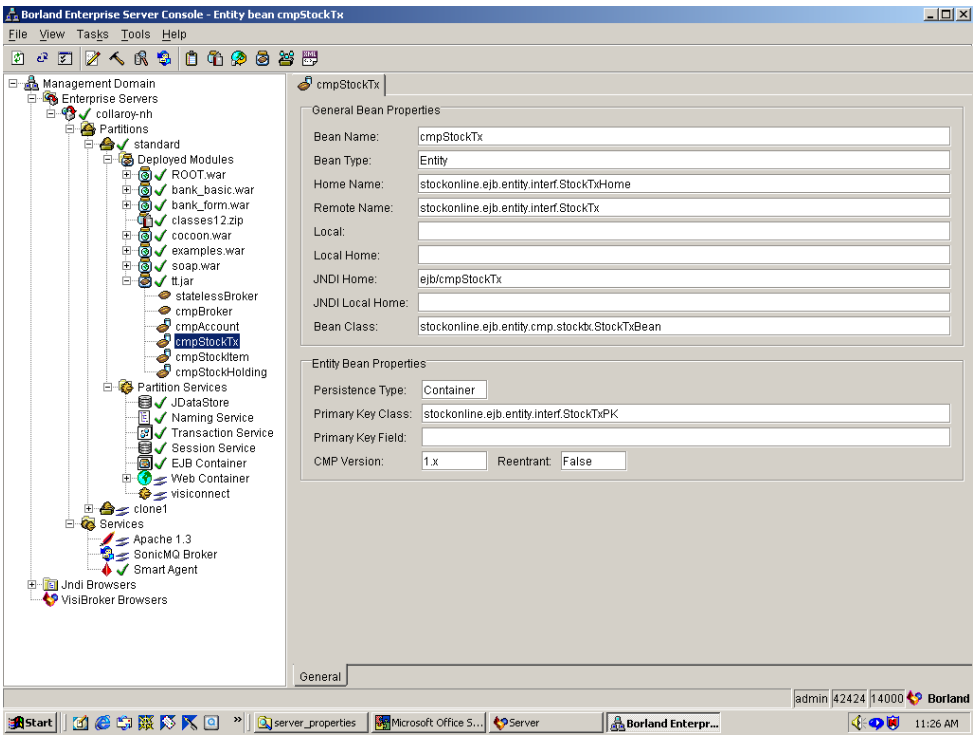


FIGURE 20 SCREEN SHOT OF BORLAND ENTERPRISE SERVER MANAGEMENT CONSOLE

5.2. INTERSTAGE EJB Service

5.2.1. Overview

INTERSTAGE Application Server V4.0 is a J2EE 1.2 compatible product. The technologies supported are:

- Servlet 2.2
- JSP 1.1
- EJB 1.1
- JDBC 2.0
- JTS/JTA
- JavaMail

In addition, it supports early implementations of the J2EE 1.3 and EJB 2.0 technologies like JMS/MDB (Message Driven Beans) and JCA for connectivity with legacy applications. It also provides a SOAP gateway (supporting SOAP 1.1) for interoperability with .NET.

The INTERSTAGE product comes with an integrated development environment tool called APWorks. The IDE tool can develop and deploy applications for all J2EE functions of the application server. It also has support for EAR files and can remotely deploy applications. The IDE can assist in defining the container-managed fields as well as the finders for the CMP beans. It uses the datastore to check the values at development time rather than deployment or runtime.

Figure 21 shows the J2EE architecture of INTERSTAGE.

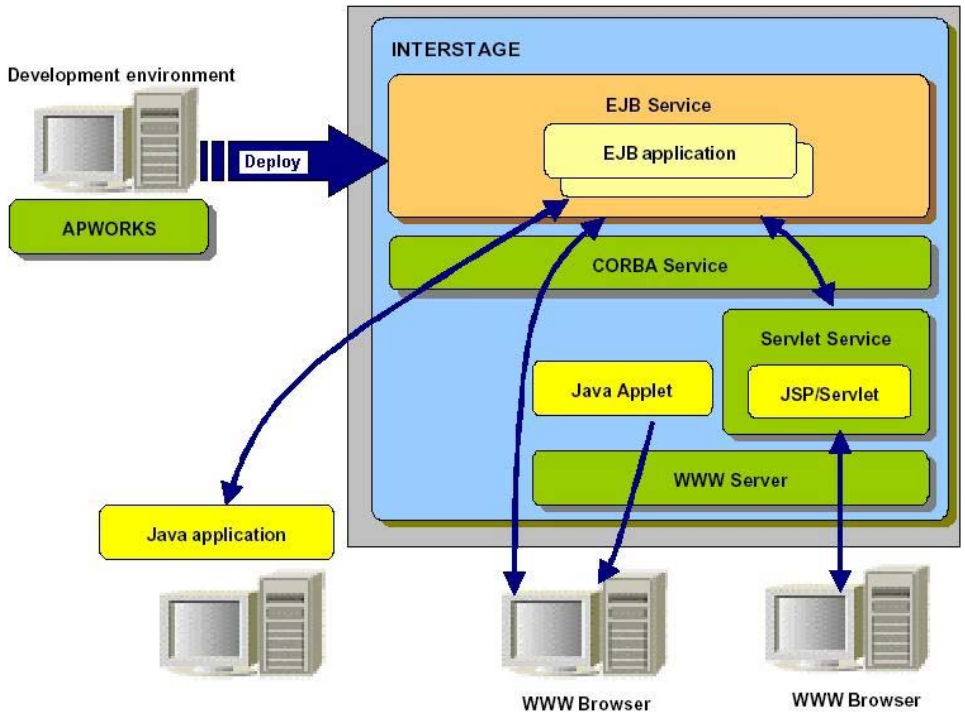


FIGURE 21 INTERSTAGE J2EE ARCHITECTURE

Applications that are developed as applets can connect to the server without needing to have the client EJB installed if using the Portable ORB service. The Portable ORB and the EJB client service are automatically downloaded to the client browser when needed.

The INTERSTAGE EJB Service comprises an EJB container implementation with support for session and entity beans, RMI-over-IIOP, database connection pooling, a transaction service and JNDI. An overview of the architecture is shown in Figure 22.

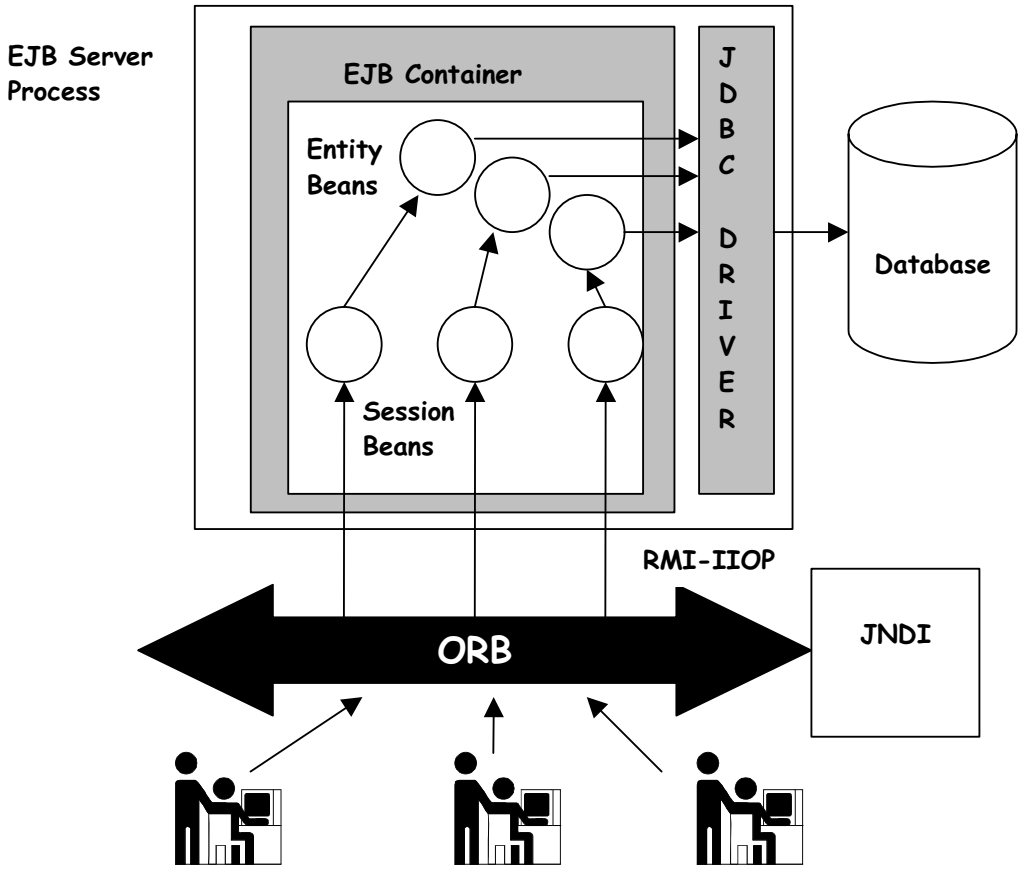


FIGURE 22 EJB SERVICE OVERVIEW

These features are explained in more detail in the sections that follow.

5.2.2. EJB Container Features

The EJB Service comprises a run-time environment for session and entity beans. EJBs run in the INTERSTAGE EJB container, which provides life-cycle, transaction and database connectivity services for beans. The EJB Service can support EJBs that adhere to both the EJB1.1 and EJB 1.0 specification (with some restrictions), and supports RMI-over-IIOP for remote communications.

The EJB container has a number of configuration settings that enable its behaviour to be tuned at run-time. The key configuration parameters revolve around cache sizes for entity beans, and the number of threads running in an EJB container (maximum is 16).

The EJB container is built around an architecture that is similar to the EJB2.0 proposed enhancement for Local interfaces of enterprise java beans. It has a grouping policy in which multiple beans that are part of the same application can be run in the same JVM. This grouping policy is totally transparent to the application developer (it is part of

deployment and assembling tasks) and it is also compliant with EJB1.1. Each group must have an entry bean. This bean can be either a session (stateless or stateful) or an entity bean. Beans within a group can remotely call beans located in other JVMs.

Bean groups run in one JVM, thus avoiding network calls and allowing for database and transaction optimisations. The INTERSTAGE architecture provides connection pooling and connection sharing for both JDBC1.x and JDBC2.x type drivers. JDBC2.x type can override the connection pool provided with INTERSTAGE by setting up its own or an external connection-pooling provider.

5.2.3. Session Beans

INTERSTAGE supports both stateless and stateful session beans. There are some points to note when using session beans - these are covered in the following sections.

Stateless Session Beans

The maximum number of stateless session beans that a container should create can be configured in the EJB application definition. If the number of concurrent client requests exceeds the maximum number of session beans, requests are queued until a bean instance becomes available.

In INTERSTAGE V4.0 stateless beans can use container-managed transactions. Also, a stateless session bean can be used as a group entry. This will allow clients to access the EJB application (including other beans) through a stateless bean.

Stateful Session Beans

The container maintains a session timeout interval for stateful session beans. When the client does not call the bean within the timeout interval, the container deletes the bean. If the client subsequently makes a call on a deleted bean, it receives a `RemoteException` error. The default value of zero disables timeout monitoring, and thus relies on well-behaved clients to remove beans when they complete the processing. The timeout value is set using the INTERSTAGE EJB Customize tool.

5.2.4. Entity Beans

INTERSTAGE supports both bean-managed and container-managed entity beans. The EJB container maintains an entity bean pool for each different bean type. The pool size is configurable.

In INTERSTAGE v4.0, entity beans can be configured as a bean group that is created and managed by a stateful or stateless session bean. This is the recommended configuration, as it allows for performance optimizations by the container. The entity beans can also be called directly from EJB client code. However, applications exposing entity bean interfaces to clients are unlikely to be high performance and scalable.

In the case when the client directly accesses the entity bean, INTERSTAGE provides a way to recycle resources for entity beans that are not used for long periods of time. The feature is implemented using a timer that is activated whenever the entity instance is not in the middle of a transaction (however, transaction timeout has precedence). The timeout value can be configured using the *EJB Customize tool* (a value of zero disables this feature).

In addition, INTERSTAGE allows an *instance management mode* to be defined for each entity bean type. The aim of the instance management mode is to allow the container to perform optimizations based on the usage pattern of an entity bean in an application. Three mode settings are permissible, and can be selected from the *EJB Customize* tool:

ReadWrite: This is the default mode. An entity bean will be modified within a single transaction.

ReadOnly: This mode enables the container to cache entity beans for subsequent reuse across transactions, speeding up access for data that does not change.

Sequential: This mode can improve memory usage when collections of beans are read and searched sequentially. It is effective for batch or large data processing.

NoCache: This mode tells the container to create a new entity bean even though one may already exist in memory for the same data in the database. This mode is available only for backward compatibility with INTERSTAGE V2.0 and it is not recommended for later versions.

5.2.5. Rapid Invocation

The INTERSTAGE EJB Service has the *ObjectDirector* ORB as its underlying transport mechanism for the RMI-over-IIOP protocol. This means a call from one bean to another by default will pass through the ORB layer. *Rapid Invocation* makes it possible for EJBs running in the same JVM to communicate without incurring the overheads of the ORB communications. This provides faster application performance.

The *EJB Customize* tool is used to set up rapid invocation. This requires the designer to nominate *Rapid Invoking Beans* and *Rapid Invoked Beans*. The former can be a stateful or a stateless session bean, while the latter may be stateful session beans or entity beans. Together, the definition of these bean attributes forms a bean group that can be deployed on the same JVM and use optimized communications.

Entity beans can also be configured as *Rapid Invoking Beans*, but they cannot contain any *Rapid Invoked Beans* and must use container-managed transaction; they can invoke other beans located in the same server machine or another. There is a performance drawback when using directly accessed entity beans in an application as all the calls to an entity are done using RMI/IIOP.

Only beans configured for rapid invocation may utilize the container's transaction service, entity bean run-time management and JDBC connection pooling features.

5.2.6. Transaction Service

The transaction service can only be used between *Rapid Invoking Beans* and *Rapid Invoked Beans*. A *Rapid Invoking Bean* can use the `UserTransaction` interface to explicitly begin and complete a transaction or they can also use container-managed transactions. Within this transaction scope, *Rapid Invoked Beans* (entity or stateful beans) can be called. *Rapid Invoked Beans* can be configured with the *Mandatory* transaction attribute. In this case, when a bean with the *Mandatory* attribute is called from within a transaction scope, the transaction context is automatically propagated to the called bean, which becomes a transaction participant.

When the transaction originator (the *Rapid Invoking Bean*) attempts to commit the transaction, the transaction service will ask all beans that have participated in the transaction to commit. Beans that have accessed the database will at this stage attempt to commit any updates they have made.

Two transaction managers can be used with INTERSTAGE.

First, there is the Local Transaction Manager that runs in process with the EJB container. The local transaction service commit protocol does not extend to using a two-phase commit with the database management systems in the transaction. This means that if, for example, two entity beans update different databases in the same EJB transaction it is possible to get a mixed outcome (i.e. one commits, one aborts). For this reason, when using local transaction manager, INTERSTAGE EJB Service applications should only be designed to perform updates to single database.

Second, the OTS service provides two-phase commit functionality for a distributed database application. The OTS service supports the XA interface for implementing Global Transactions that can span multiple databases located on different server machines.

5.2.7. Database Connectivity

The EJB Service can be configured to initialize a database connection pool at start-up. Again, this is performed using the *EJB Customize* tool. Pooled database connections can only be used if rapid invocation is set. Once initialized, the connections in the pool remain open until the EJB Service terminates.

INTERSTAGE provides connection pooling and connection sharing for both JDBC1.x and JDBC2.x drivers. In case of JDBC2.x, the connection pool can be replaced with an external provider that can implement a different algorithm. Connection sharing allows the container to reuse connections that are participating in the same transaction, improving the resource usage.

5.2.8. Development Tools

INTERSTAGE is supplied with an NT-only Java development environment called APWORKS. However, any Java IDE that supports the EJB v1.1 specification can be used.

5.2.9. Scalability

The EJB Service has support for both *scale-up*[23] and *scale-out* [24]of applications.

In terms of scaling up an EJB application, multiple EJB servers can be run on the same server machine. Client requests are load-balanced across the replicated EJB servers using the underlying ORB load-balancing functionality. No special features or configuration are needed to deploy this configuration.

In order to scale-out an application to run replicated EJB servers across multiple machines, the INTERSTAGE Naming Service load-balancing option needs to be used. Another option is to use Network Access Server (NAS), an optional product tightly integrated with INTERSTAGE that provides better support for load balancing and security

[23] Running the application on a more powerful machine
[24] Running the application across multiple machines

through access control functionality. For more information about NAS see the next section.

The EJB Service may also exploit the Microsoft Clustering Service. This makes it possible to introduce hardware hot-standby in to an application. When a machine fails, the clustering service transfers processing to the standby machine. In order for this to operate successfully, machines need to share physical disks.

5.2.10. System Management

The EJB Service has no specific system management tools available in v4.0. However, INTERSTAGE's *SystemWalker/CentricMgr* network management tools can be used to manage EJB servers. These powerful GUI-based tools can be used to monitor and configure, and to set up the automatic operation of an INTERSTAGE system that is distributed over many machines. INTERSTAGE components must be linked with system management libraries to be visible to this management tool.

The Network Access Server (NAS) is a new product available to INTERSTAGE V4.0. It has two components:

- **Traffic Director (TD)**: The TD tool is used to monitor Internet traffic and to provide load balancing and fail over support. A powerful feature implemented within Traffic Director is the capability to dynamically control the bandwidth of different types of network traffic, from the application level down to network protocol levels (RMI/IIOP, SOAP, HTTP/S, TCP/IP). This can ensure, for example, that management tools will always have access to a busy server and take appropriate measures even if normal applications use all of their bandwidth. The bandwidth can be dynamically altered; for example, if an allocated bandwidth is not used, it can be re-distributed to a busier logical line. Bandwidth can also be scheduled to different levels for different periods of time. These features are implemented using special agents located on each server controlled by the Traffic Director.

- **Security Access:** this function is provided using a packet filter service. The filtering service can allow or disallow passing of network data using the following security policies:

    - IP address
    - Service (TCP/UDP port number)
    - Protocol

The NAS supports hot-standby to ensure high reliability operation. It also provides a runtime management environment composed of a *Policy Server* and a *Management Console*. These tools can be run from different machines. The Management Console has a monitoring function that can use SNMP and email to send information about abnormal situations, or when a monitored function reaches a threshold. Things like CPU load, memory usage and disk I/O for each controlled server can be monitored from NAS.

5.3. JBoss EJB Service

5.3.1. Overview

JBoss[25] is an open-source application server. JBoss distributions and source-code are freely available for use by downloading from the JBoss/Sourceforge web sites. JBoss is available under the GNU LGPL license, meaning that, while the JBoss application server is covered by the free software licensing requirements, proprietary applications can be run in the JBoss application server.

JBoss itself is free to use. Free support is available in the form of mailing lists and discussion forums, both very active; these lists are under no obligation to provide either timely or any support, however. Since the source code is available, bug-fixes can be done "in house" by a knowledgeable programmer, if one is available. In production, and depending on the criticality of the application, JBoss may need to be backed by a support contract. The JBoss Group, the commercial aspect of the JBoss developers, provides support contracts.

The JBoss EJB Service comprises an EJB container implementation with support for session and entity beans, database connection pooling, a transaction service and JNDI. There is also support for JMS, JavaMail, JCA and security using the JAAS framework. Federated naming and security services are possible, using an LDAP server. JDK 1.2.2 and 1.3 clients are both supported. JBoss is a pure Java implementation and runs on the following platforms: Windows 9x, Windows NT, Windows 2000, Solaris, Linux.

Architecture

JBoss has a somewhat unusual architecture. Each bean is placed in a container, which also acts as the server-side `EJBObject` and `EJBHome` that handles remote invocations. Client objects are dynamic proxies that are mapped to bean instances by the container. The JBoss architecture is illustrated in Figure 23: a message in a client application is delivered to a dynamic proxy; the proxy sends the message to the bean container, along with the identity of the bean being invoked; the container readies the bean from a pool of beans, passes the message through a stack of interceptors and invokes the appropriate method on the bean, using Java reflection. The interceptors are used to manage the transactional behaviour of the invocation, metrics gathering, security, logging and other features. The stack of interceptors for a particular bean can be specified at deployment time.

The JBoss architecture is resource-efficient – since the server does not need to keep a corresponding `EJBObject` for each client object – and highly configurable and elegant. Under very high load, the single point of entry for each bean type represents a potential bottleneck, leading to a processor asymmetry

[25] http://www.JBoss.org

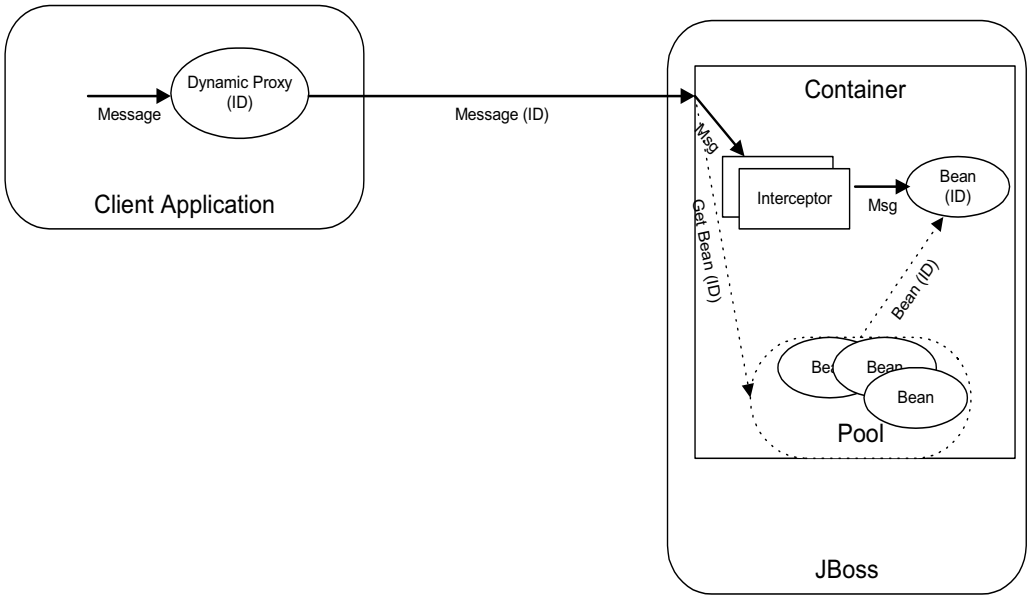


FIGURE 23 JBOSS ARCHITECTURE

5.3.2. EJB Container Features

Each container (and, therefore, bean) can have a number of configuration settings, which enables its behaviour to be tuned. The key configuration parameters allow pool and cache sizes, usage policies and passivation policies to be set. Each container can also have a stack of interceptors specified, allowing per-bean management of security and monitoring. Pools of database connections can also be managed in a similar way. JBoss provides no way of configuring the number of active threads in the application server.

An optimization allows beans within the JBoss JVM to call other beans directly by reference, bypassing the container architecture. This optimization can improve performance markedly for complex applications. The optimization can produce unexpected behaviour, however, as some objects that should be copied if the EJB specification were followed exactly are returned by reference.

5.3.3. Session Beans

JBoss supports both stateless and stateful session beans. Each container maintains a pool of beans, with configurable minimum and maximum pool sizes.

Stateful session beans are cached. The cache size and passivation policies are configurable on a per-bean basis.

5.3.4. Entity Beans

JBoss supports both bean-managed and container-managed entity beans. Each container maintains a pool of beans, with configurable minimum and maximum pool sizes. Entity beans are also cached, with the cache size and management policies configurable on a per-bean basis.

Using the JAWS persistent manager, the `tuned-updates` parameter can be set, to allow persistence optimization. If the `isModified()` method is defined on an entity bean, this method will also be called before an entity bean is stored, to see if the bean needs to be written back to the database. Additional parameters are `read-only` for beans that are never modified by the application and `read-ahead` for eager loading of beans during finder calls.

Commit options A, B and C are all supported. In addition, a non-standard option D is supported, where the cached bean data is refreshed from the database at regular intervals.

5.3.5. Transaction Service

JBoss supports distributed transactions across multiple resources. Two-phase commit is supported if all drivers support the JDBC 2.0 optional package. If not all drivers support this package, then a two-phase commit is simulated but there is still the possibility of an inconsistency.

5.3.6. Database Connectivity

JBoss comes with drivers and pools for the Hypersonic and InstantDB already configured. Other JDBC drivers are available from database vendors and the JBoss documentation contains instructions for configuring Oracle, DB2, Sybase, Postgres, Interbase, mySQL, Microsoft Jet and Microsoft SQL Server.

Database connections are pooled, with multiple pools possible. The pool sizes and management policies are specified in the JBoss configuration files. JBoss can be configured to close idle database connections after a period of inactivity.

5.3.7. Development Tools

Support for development tools for EJB development within JBoss itself is limited. The documentation describes how to integrate with Jbuilder and VisualAge for Java for remote debugging. Otherwise, no application development tools are provided or specifically integrated with. However, the minimal amount of additional information needed to deploy EJBs into JBoss and hot-deployment makes the development cycle relatively simple.

The JBoss development team uses the tools developed by the Jakarta project, such as *ant*, to develop JBoss and EJBs. This is in keeping with the general open-source philosophy, but presents a barrier to programmers starting with JBoss. A full IDE environment featuring complete build, deployment, debug and server control is available by using the TogetherSoft Control Center and a JBoss-specific plugin.

5.3.8. Scalability

JBoss provides no clustering or failover facilities, so scale-out is impossible. Under load JBoss also develops an asymmetry in processor usage, meaning that scale-up will be limited to a certain extent by the speed of a single processor.

5.3.9. System Management

JBoss has a highly modular architecture and uses the JMX management system to allow management of JBoss components. The JBoss configuration file is, essentially, a list of

managed beans (Mbeans) that the JBoss system can load and unload. A primitive web-based UI can be used to examine and manage the Mbeans running inside JBoss.

Assembly and deployment simply consists of adding optional JBoss-specific configuration information to the provided EJB jar and placing it in a deployment directory. JBoss detects changes to the deployment and loads (or reloads) the EJBs. There is no need to run special deployment tools, although adding JBoss-specific configuration information is often a good idea.

JBoss 2.4.3 comes with a simple monitor application for collecting performance information and statistics. A large amount of logging information can also be generated, but this information needs to be manually analysed.

5.4. SilverStream EJB Service

5.4.1. Overview

SilverStream has positioned itself as a supplier of J2EE framework and middleware products under the *eXtend* and *jBroker* umbrellas. The jBroker range of middleware products are SilverStream jBroker Web (Java/Web services), jBroker Orb (Java Corba ORB), jBrokerMQ (Java/JMS), jBroker Transaction Manager (Java OMG OTS 1.2 with XA), and jBroker Console (for managing all jBroker products). The SilverStream eXtend J2EE products are eXtend Workbench (J2EE and Web Services), eXtend Director (J2EE framework for personalization, workflow, content management, user profiling, wireless delivery, security, and portals), eXtend UDDI Server (JEDDI 1.0), and eXtend Application Server 3.7.4. (J2EE AppServer).

We evaluated the SilverStream eXtend Application Server 3.7.4, which is J2EE 1.2 compatible[26]. It supports the following standard J2EE components:

- Server side: Servlets (2.2), JSP (1.1), EJB (1.1)
- Client side: Applets, Java application clients (CARs and Client container)

SilverStream supports the following standard J2EE services:

- Naming (JNDI)
- Deployment (server and client side components)
- Transactions
- Security

SilverStream supports the following communications technologies and protocols:

- Web: TCP/IP, HTTP 1.0, SSL 3.0
- Corba (2.3): RMI-IIOP (+SSL)
- Messaging: JavaMail, JMS (1.0.2)

SilverStream supports the following J2EE module types[27]:

- J2EE applications: Enterprise Archive Files (EARs)
- Web, EJB and client application modules (WARs, JARs, and CARs)

SilverStream supports the following J2EE clients:

- Web clients: Java Server Pages, Servlets, Applets

[26] http://java.sun.com/j2ee/tested_config/SilverStream.html
[27] J2EE modules are collections of components with deployment information

- J2EE Application clients

J2EE Application clients are hosted in client containers (SilverJ2EEClient), which provide access to J2EE services, and are portable across J2EE servers. The SilverJ2EEClient container supports RMI-IIOP and RMI-IIOP over SSL, and dynamically loads the current version of client jar files from the server (with authentication).

SilverStream provides the following Enterprise Java Beans support:

- RMI-IIOP access using jBroker ORB (Java based ORB).
- Security

EJBs are fully integrated with the SilverStream security model that provides:

- Authentication against a variety of directory services (E.g. LDAP, Sun NIS+, Windows NT Domains, SilverStream Security Directory)
- Client-based x.509 certificates.
- Access authorization for EJB calls and resource manager access.
- Secure communication with remote clients with SSL 3.0 (RSA and DSA public key/private key encryption between client and server).

SilverStream also has an integrated full text search engine (Fulcrum SearchServer), which maintains a full text index of HTML and other documents stored in database tables. Full text queries can be included in SQL.

5.4.2. Application Server Architecture

SilverStream is built on top of the jBroker Orb.

SilverStream stores both application and system data in database tables. The SilverMaster database is used:

- As a component/module repository
- To store application server configuration settings
- To store log files.

The J2EE (1.2) specification defines 4 *logical* container types:

- EJB
- Web (For web pages, JSP, and Servlets)
- Applet
- Application client.

Vendors may choose to implement these as physically distinct containers, which can be run as separate processes, or in a simple case as one container (probably implicit) in one server.

SilverStream seems to have taken this approach, as there is no mention of containers in the documentation or externally visible in the product itself[28]. We therefore assume that all server side components run in the one server process. It is not possible to deploy multiple containers in one server, or set container properties that are different to the server properties. Because there are no explicit containers, EJBs are associated with a database, and deployed into that database.

The various clustering components (SilverStream Servers, Load Manager, Dispatchers, Cache Manager), however can be run as separate processes, and on different machines and platforms.

5.4.3. EJB Container Features

Because SilverStream doesn't distinguish between the EJB container and the server, all services and Management console settings therefore apply to the entire server (apart from deployment plan settings that are specific to EJBs).

SilverStream supports database connection pools, and the minimum/maximum number of connections is configurable from the management console.

5.4.4. Session Beans

SilverStream supports stateless session bean pooling (a pool of unused/idle stateless session beans). The maximum number of beans in the pool can be specified in the deployment plan. The number of beans increases on demand to the maximum.[29]

SilverStream does not support passivation[30], or consequently failover, for stateful session beans.

5.4.5. Entity Beans

SilverStream only supports commit option 'C' CMP Entity Beans, and with the following features:

- CMP extensions (using sub-classing)
    - Can modify ejbLoad() and ejbStore() to do complex things
    - Can write own finder methods to do complex finds.
- Persistent field mappings
    - Map to columns in "primary" table

[28] Except for the J2EE Application client container

[29] We conducted some tests of this feature, keeping the number of database connections constant (20) while varying the maximum Pool size. We discovered only a small difference in throughput (10% maximum at 100 clients, Pool varying from 1 to 500).

[30] This seems to be optional in the EJB 1.1. Specification (section 6.6).

    - Map to columns in "related" tables (Read only)

- Map to persistent fields in other entity beans.
- Finder methods
    - For each bean (not each finder however), can set finders to lazy (only gets primary key), or non-lazy (gets entire database record).
- Pooling
    - No ready or passivated pool. Only supports basic commit option C without any object pooling[31].
- EjbStore
    - Only writes to database if data values have changed. The comparison may be expensive, however, so methods that don't modify fields can be identified in the deployment plan.
- Concurrency
    - Optimistic concurrency is supported.

5.4.6. Transaction Service

SilverStream provides transaction support for a single database only. It does not support distributed transactions (i.e. No XA support).

5.4.7. Database Connectivity

SilverStream supports JDBC 2.0, types 1, 2 and 4.

SilverStream connects to Oracle using a Type 2 JDBC driver written and provided by SilverStream. This driver interfaces with the Oracle SQL Net Client (installed on the SilverStream server machine) to achieve connectivity to the Oracle Server. Integrated drivers are supplied for:

- Oracle 7 and 8
- SQL Server 6.5/7.0
- Informix 5.0 and 6.0
- Sybase Adaptive server 11/12 (Using Jconnect driver)
- IBM DB2

[31] This is a basic implementation, and the Entity bean test results reflect this.

5.4.8. Development Tools

SilverStream has a set of comprehensive and very useable tools and wizards (approximately 19), including Jar builder, Deployment plan designer, a debugger and profiler.

The SilverStream Designer is a collection of tools including:

- EJB Designer/Deployment wizards
- Business Object Designer
- Table Designer
- Distributed Debugger
- EJB and Media Store
- Source Management
- Consoles
- Page designer (HTML development)
- Form Designer
- View Designer

There is also extensive support for external IDEs including JBuilder, Inline, Visual Café (WebGain), and Dreamweaver. EJBs can be deployed from JBuilder and Inline.

In SilverStream, EJBs are not deployed to "containers" but to databases. EJBs are deployed in SilverStream as follows:

- Import existing Jar files into a database using the Designer tool.
- Write the *deployment plan* (vendor specific deployment information) with the designer tool, and deploy to a server.
- Datasources for each database are automatically created.
- Deploy jar files to the database that they were imported into using the Designer tool.
- The deployed jar file cannot be modified. However, a new deployment plan can be produced, and then deployed without having to remove the running Jar file or restart the server.

SilverStream has sophisticated support for J2EE Application client deployment. The J2EE Application container can be downloaded from the SilverStream server and installed on a client machine using a Web Browser. The client jar file is initially deployed onto a server (although there is no GUI support for this), and is then automatically deployed and updated on the client machines.

Enterprise Data Connectors for PeopleSoft, Lotus Notes, and SAP are supported, along with tool support (Wizards) to enable business objects to connect to them.

5.4.9. Scalability

The SilverStream *Enterprise* AppServer supports scalability by server clustering. A cluster consists of:

- SilverStream Servers
- Load Manager
- Dispatchers
- Cache Manager

The cluster components are configured as shown in Figure 24.

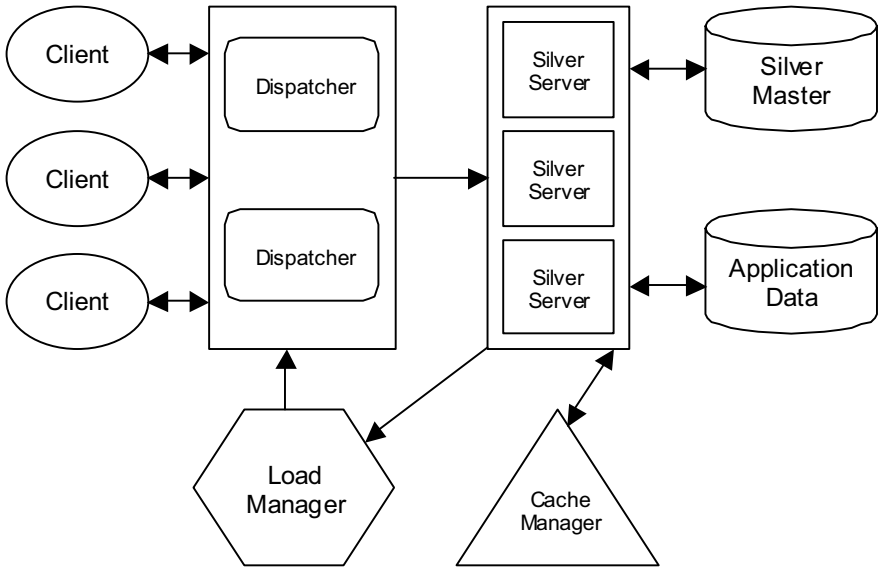


FIGURE 24 SILVERSTREAM CLUSTERING CONFIGURATION

All clustering components can be run as separate processes, on separate machines, and on different platforms.

Clustering also provides failover capacity. Failover of "all" cluster services is provided, although there is no automatic failover support for the dispatcher. Dispatchers can be replicated, but requests are not automatically redirected from a failed dispatcher to another.

Load balancing is *round-robin* by default, but the relative number of hits for each server can be modified (statically).

5.4.10. System Management

The SilverStream Management Console (SMC) is a Java program which can be run in a browser or stand-alone and manages all aspects of the application server and server clusters. Figure 25 shows an example screen shot from this tool.

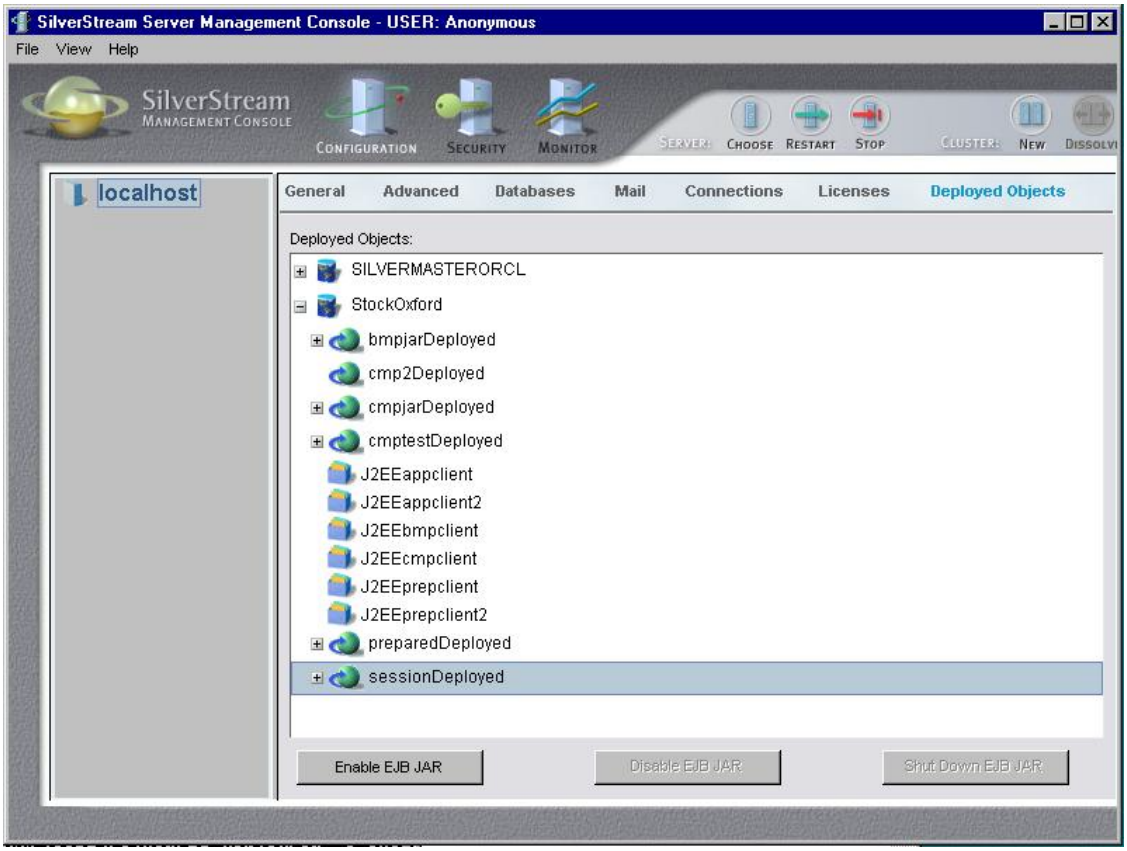


FIGURE 25 SILVERSTREAM SERVER MANAGEMENT CONSOLE

SMC supports general administration including:

- Setting minimum/maximum database connections.
- Setting min/max/optimal number of client connections (pool of idle threads) for HTTP clients.
- Setting memory and disk data cache sizes.

SMC supports Monitoring of Charts, Logs and Statistics. SMC also supports clustering, making it easy to configure or dissolve clusters of servers.

5.5. WebLogic Server EJB Service

5.5.1. Overview

The WLS v6.1 is J2EE 1.2 compatible. The WLS EJB Service comprises an EJB container implementation with support for session and entity beans, RMI-over-IIOP, database connection pooling, a transaction service and JNDI. An overview of the architecture is shown in Figure 26.

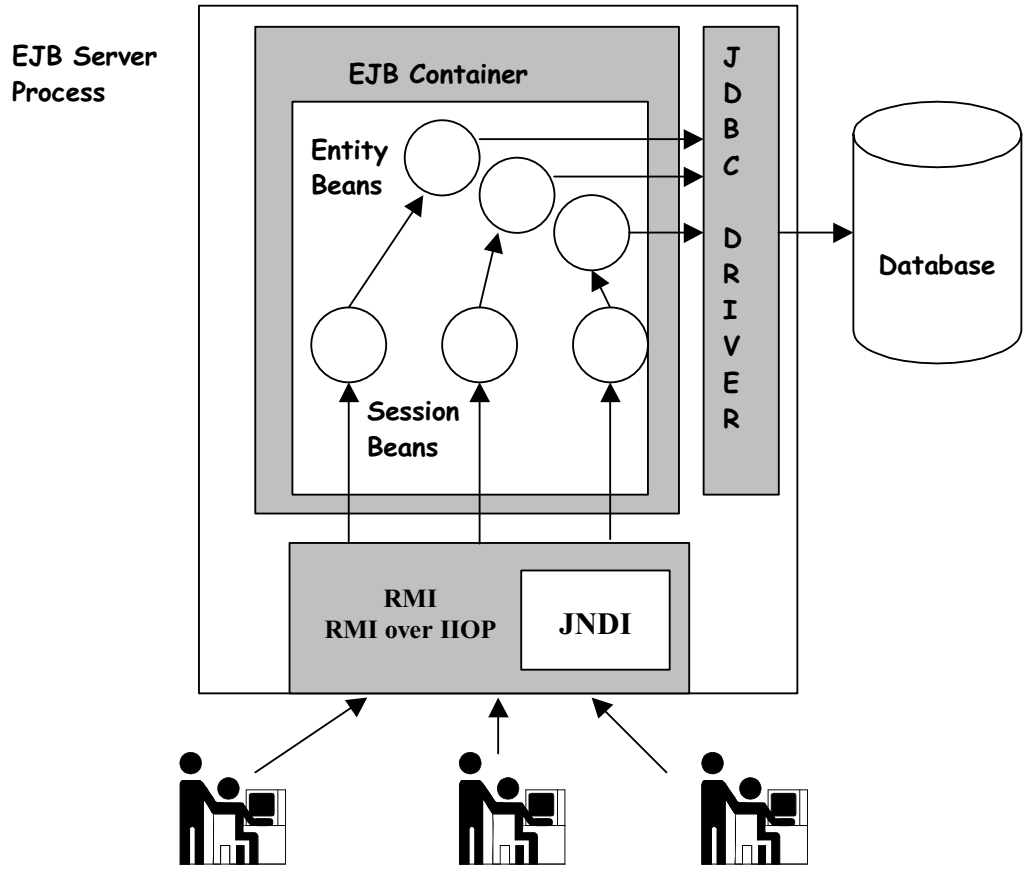


FIGURE 26 EJB SERVICE OVERVIEW

These features are explained in more detail in the sections that follow. The explanations assume an understanding of the basic EJB model, and aim to point out where WLS value-adds or varies from the specification.

5.5.2. EJB Container Features

The EJB Service comprises a run-time environment for session and entity beans. EJBs run in the WLS EJB container, which provides ready to use lifecycle, transaction and database connectivity services.

WLS 6.1 supports J2EE 1.3 and therefore EJB 2.0. Listed below are some of the new EJB features in this release, which are important and useful:

- Non-transactional entity bean caching
- Automated Generated Primary Key Support
- J2EE Application client support
- Cascade deletion support

The WLS EJB container has a number of configuration settings that enable its behaviour to be tuned at run-time. The key configuration parameters revolve around cache sizes for entity beans, the number of threads running in an EJB server, and the number of database connections in the JDBC connection pool.

5.5.3. Session Beans

WLS supports both stateless and stateful session beans. There are some WLS specific performance tuning issues to note when using session beans - these are covered in the following sections.

Stateless Session Beans

WebLogic Server uses a free pool to improve performance and throughput for stateless session EJBs. The free pool stores unbound stateless session EJBs. Unbound EJBs are instances of a stateless session EJB class that are not processing a method call at a given time.

When a client calls a method on a stateless EJB, WLS obtains an instance from the free pool. The EJB remains active for the duration of the client's method call. After the method completes, the EJB is returned to the free pool.

By default, no stateless session EJB instances exist in WLS at startup time. As clients access individual beans, WLS initializes new instances of the EJB class. To improve initial response time when accessing EJBs, you can specify how many inactive instances of the EJB that should be put in the free pool when WLS is started by assigning the number you want to the `initial-beans-in-free-pool` deployment descriptor element, in the `weblogic-ejb-jar.xml` file. This way, initial client requests can be satisfied by just activating the bean from the free pool rather than creating the bean from the heap and then activating it. By default, `initial-beans-in-free-pool` is set to 0 (zero).

You can also specify the maximum number of beans in the free pool by assigning a value to the `max-beans-in-free-pool` deployment element. The maximum number of beans that can exist in the free pool is also limited by the heap size.

Stateful Session Beans

WebLogic Server uses a cache of bean instances to improve the performance of stateful EJBs. The cache stores active EJB instances in memory so that they are immediately available for client requests.

No stateful EJB instances exist in WebLogic Server at startup time. As clients look up and obtain references to individual beans, WebLogic Server initializes new instances of the required EJB class and stores them in the cache. To achieve high performance, WebLogic Server reserves the cache for EJBs that clients are currently using and EJBs that were recently in use. When EJBs no longer meet these criteria, they become eligible for

passivation. Passivation is the process a J2EE application server uses to remove an EJB from cache while preserving the EJB's state on disk. WebLogic Server also provides a `max-beans-in-cache` deployment element, which provides some control over when EJBs are passivated.

The EJB developer has to make sure that a call to the `ejbPassivate()` method leaves a stateful session bean in a state where WebLogic Server can serialize its data elements and passivate (persist) the bean's instance. During passivation, WebLogic Server attempts to serialize all data members of the bean that are not declared `transient`.

5.5.4. Entity Beans

WebLogic Server supports both bean-managed and container-managed entity beans. The EJB container maintains an entity bean pool for each different bean type. The pool size is configurable.

The Entity Bean lifecycle as defined by the EJB v1.1 Specification is not the most optimal in terms of performance. The operations `ejbLoad()` and `ejbStore()` may be executed unnecessarily in many situations depending on the application logic. This depends when and how often update operations are made to an entity bean.

Consequently, WLS provides several optimization mechanisms through configuration parameters. These parameters affect the run-time behaviour of the container, and in most cases, remove redundant or unnecessary database access operations (ie. `ejbLoad()` and `ejbStore()`).

Using db-is-shared to limit calls to `ejbLoad()`

In the case where only a single WebLogic Server instance ever accesses an EJB's underlying data, the parameter *db-is-shared* can be set to "false" in the bean's `weblogic-ejb-jar.xml` file. When `db-is-shared` set to "false," WebLogic Server calls `ejbLoad()` for the bean only when:

- A client first references the EJB
- The EJB's transaction is rolled back

Using is-modified-method-name to limit calls to `ejbStore()`

WebLogic Server calls `ejbStore()` at transaction commit time when an EJBs persistent fields were actually updated. To use `is-modified-method-name`, EJB providers must develop an EJB method that "cues" WebLogic Server when persistent data has been updated. If no data has been changed during a transaction, the store operation is not called.

Using delay-updates-until-end-of-tx to change `ejbStore()` behaviour

By default, WebLogic Server updates the persistent store of all beans in a transaction only at the completion (commit) of the transaction. However, if an isolation level READ_UNCOMMITTED is set, other database users should view the intermediate results of in-progress transactions. In this case, `delay-updates-until-end-of-tx` can be set to false to change WebLogic Server's `ejbStore()` default behaviour.

Read-write cache strategy

The read-write strategy defines the default caching behaviour for entity EJBs in WebLogic Server. With read-write entity EJBs, multiple clients can use the same bean instance in transactions, and data integrity is ensured. For read-write EJBs, WebLogic Server loads EJB data into the cache at the beginning of each transaction. WebLogic Server calls `ejbStore()` at the successful commit of a transaction.

Read-only cache strategy

The read-only cache strategy can be used for entity EJBs that are never modified by an EJB client, but may be updated periodically by a source external to the J2EE environment. WebLogic Server never calls `ejbStore()` for a read-only entity EJB. `ejbLoad()` is called initially when the EJB is created. This strategy attempts to ensure minimal database access overhead is incurred.

5.5.5. Transaction Service

Since v6.0, WebLogic Server has supported distributed transactions across multiple resource managers.

However, WebLogic Server v5.1 and earlier did not provide a two-phase commit protocol. This means that if, for example, two entity beans update different databases in the same EJB transaction, it is possible to get a mixed transaction outcome (i.e. one commits, one aborts). For this reason, WLS applications should be limited to using a single database or resource manager. EJBs that engage in distributed transactions (transactions that make updates in multiple databases) cannot guarantee that all branches of the transaction commit or roll back as a logical unit.

As an alternative, WebLogic Enterprise/Tuxedo supports a full two-phase commit protocol, which can be accessed from WLS applications through the WLE Connectivity component.

5.5.6. Database Connectivity

WLS provides a number of JDBC drivers for Oracle, SQL Server and Informix. There are also other JDBC drivers available from the database vendors.

The EJB Service is configured to initialize a database connection pool at start-up. This is specified in the domain-wide `config.xml` configuration file. There are several attributes of the `JDBCConnectionPool` element in the XML file you can use to control how a connection pool is created, and how it grows and shrinks.

The `InitialCapacity` attribute allows you to set the number of physical database connections to create when configuring the pool. If the server is unable to create this number of connections, the creation of the connection pool will fail. The `MaxCapacity` attribute allows you to set the maximum number of physical database connections in a connection pool. The `CapacityIncrement` attribute specifies the number of connections that can be added at one time when the server needs more connections to service the client's requests. The server will ensure that the pool size does not exceed the maximum number of physical connections as set by `MaxCapacity`. The `ShrinkingEnabled` attribute indicates whether or not the pool can shrink back to the value of `InitialCapacity` when connections are detected to not be in use. When this attribute is set to true, the `ShrinkPeriodMinutes` attribute controls the number of minutes to wait before shrinking a connection pool that has incrementally increased to meet demand.

The connection pool attributes, especially `MaxCapacity`, need to be set carefully. It is advisable that the number of connections in the pool equal the number of active concurrent client sessions that require JDBC connections. The BEA document, *Tuning WebLogic Server*, indicates that "the pool capacity is independent of the number of execute threads in the server. There may be many more ongoing user sessions than there are execute threads." This is not very true. Notice that the number of active concurrent client sessions does depend on the number of execute threads. As a result, the number of connections will not exceed the number of execute threads too many. Normally, it is a few more than the number of threads.

5.5.7. Development Tools

The WebGain Studio product can be used to rapidly develop WebLogic Server applications. Given that BEA Systems currently holds a large share of WebGain, it can be anticipated that an even tighter integration of WebGain Studio (the IDE) and WebLogic Server (the deployment environment) will occur.

We have developed WebLogic Server applications using both Jbuilder and VisualCafé with ease (JDK 1.1.7a for WLS v5.1, JDK 1.3.0 for v6.0, and JDK 1.3.1 for v6.1). However, our experience shows that using command line scripts is the most straightforward way to build and deploy applications.

5.5.8. Scalability

The EJB Service has support for both *scale-up*[32] and *scale-out* [33]of applications.

In terms of scaling up an EJB application, various JVM (eg. heap size) and WLS configurations (eg. system threads, JDBC connection pool size) can be set that will more fully utilise system resources such as memory and multi-processor power. Further, multiple EJB servers can be run on the same server machine by configuring a cluster on a single machine. Client requests are load-balanced across the replicated EJB servers using the WLS clustering functionality. The configuration for a cluster on one machine is the same as that for setting up a cluster across multiple machines.

In order to scale-out an application to run replicated EJB servers across multiple machines, the WLS Clustering Service needs to be used. Basically, each WLS server instance holds a unique IP address, and for each WLS cluster, a unique multicast address (or group) must be assigned.

This multicast group is used by the clustered servers to communicate with one another. A server can request the JNDI tree update via multicast address. A server also periodically (typically 10 seconds) multicasts "heartbeat" signals to tell the other cluster members that it is still alive.

A performance issue with WLS v5.1 clustering service is that the different WLS server instances need to share a common physical disk directory. Although this is easily configurable, it is not a very scalable solution. In the WLS v6.0 and v6.1 Clustering service, the need for different WLS Server instances to share the same disk directory is removed. We have presented some clustering service test results in section 4.

[32] Running the application on a more powerful machine
[33] Running the application across multiple machines

5.5.9. System Management

BEA provides a powerful and easy-to-use tool, the WebLogic Server Administration Console, for managing and monitoring EJB applications. The console is so powerful that you can do almost all your server administration tasks on it. Here is a brief list of WebLogic console capabilities:

- View WebLogic Server configuration properties, including properties set in the configuration file (`config.xml`) as well as WLS internally runtime property/parameters
- Monitor messages in the WebLogic Server log and the JDBC log
- View status of, and statistics for, WebLogic Server resources, including workspaces, events, JNDI, EJB, JDBC, security and file system objects
- View connected users status
- Deploy and remove various WLS objects such as connection pools, EJBs and servlets.
- Configure clusters of WLS instances

Implemented as web pages written in JSP, the Administration Console can be used on a remote machine away from the running WLS server. From our experience, besides the administration functions, the BEA console also provides many useful monitoring functions during system development and debugging stages.

5.6. WebSphere EJB Service

5.6.1. Overview

WebSphere 4.0 Advanced Edition (AE) is J2EE 1.2 compatible. It is a component of IBM's WebSphere Application Server series consisting of a standard edition, advanced edition and an enterprise edition. The advanced edition includes the features provided by the standard edition as well as support for J2EE technology.

WebSphere 4.0 AE supports the following server side component types:

- Web services open standard, including SOAP, UDDI and WSDL
- Some J2EE 1.3 support, such as JMS and JCA
- Java servlets 2.2, with exception of the J2EE extensions to the specification
- Java Server Pages 1.1

WebSphere 4.0 AE supports the following client types:

- HTTP based clients (e.g. Java Applets)
- Java applications
- C++ EJB clients

The Advanced Edition also supports the following communication technologies and protocols:

- Remote Method Invocation (RMI)
- Internet Inter-ORB protocol (IIOP)
- RMI over IIOP
- Java Messaging Service 1.0.2

The following services are part of WebSphere AE:

- Deployment support for EJBs, Java servlets and JSPs that support application-level partitioning and load balancing
- Java Transaction API 1.1 integration with JMS to MQSeries
- Database connection pooling using JDBC (2.0)
- Support for distributed transactions and transaction processing
- Naming service

- Security management
- System Administration

WebSphere has an impressive set of administration tools in the product. These include:

- A GUI based Admin Console
- A Web browser client
- XML configuration – for automated administration
- A Java Tcl Command Line System Management Tool for script based administration

WebSphere also includes support for the eXtensible Markup Language (XML), namely:

- XML parser for Java
- SAX 1.0 parser
- DOM 1.0 parser
- DTD for JSPs
- XSL support

In terms of security, the WebSphere Security Service can be configured at user and group level, or at method level. The following security technologies are supported:

- Partial support of J2EE Java security API
- x.509 certificates
- HTTP/SSL
- RMI/IIOP/SSL
- RSA and public key encryption

Support for directory and naming services include:

- JNDI with new cache enhancement
- CORBA CosNaming
- LDAP
- Use of native OS registry or LDAP for users management and security

IBM also incorporates a HTTP server into WebSphere, based on the Apache Web Server. This includes a GUI-driven administration tool and support for integration with LDAP and SNMP.

There are a range of miscellaneous features that also form part of the advanced edition of WebSphere. These include:

- Support of Java 2 Software Development Kit v1.3
- Integration with IBM VisualAge for Java that enables remote testing and debugging Web-based applications
- Tivoli Ready modules
- Dynamic web caching (ie. Servlet/JSP response caching)
- Object Level Tracing

At the time of writing, WebSphere AE runs on the following platforms:

Operating Systems Supported	*Database Systems Supported*
Windows NT 4.0 **Windows 2000** **Solaris 7.0 or 8.0** **AIX 4.3.3.7** **AS/400** **HP-UX 11.0** **Linux (ReadHat 7.1 or SuSE 7.1)**	**Oracle** **DB2** **Sybase** **SQL-Server via Merant** **Informix** **InstantDB**

5.6.2. Application Server Architecture

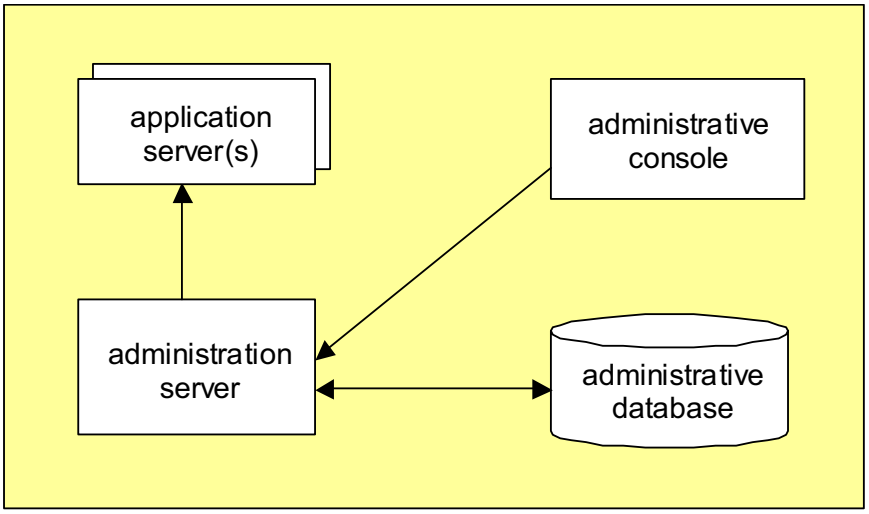


FIGURE 27 COMPONENTS OF THE WEBSPHERE APPLICATION SERVER ENVIRONMENT

WebSphere Application Server (WAS) provides central administration of EJB servers and other resources. In a WAS application, an *administrative domain* is a collection of host

machines called managed nodes. Each managed node runs an *administration server* (administration servers are also EJB servers).

A node's administration server is responsible for configuring, monitoring, and managing the resources on that node. Resources include components such as EJB servers, containers, deployed beans, JSP files, Java servlets, and applications. Resources also include objects such as method groups or policies that are used to define security for resources in the domain. Resource beans are in fact container-managed persistent (CMP) entity beans. The persistent data associated with a resource (for example the name, current state, and executable of an EJB server) is stored in a central data repository, the administrative database.

The administration server communicates with a repository server to access, define, and modify resource information stored in the repository. An administration server also communicates with other (remote) administration servers to delegate tasks and to respond to requests. IBM's DB2 relational database, which is packaged with WAS AE, can act as the repository server. It is also possible to use any other database supported by WebSphere. Figure 27 shows the components of a WAS application configuration.

The administrative server provides the services that are used to control resources and perform tasks on the administrative database. In addition to the server start/stop/restart functionality and monitoring capabilities, the administrative server also provides shared services for:

- Naming
- Transaction monitoring
- Security

5.6.3. EJB Container Features

A WAS application can comprise one or more EJB servers. Each server can have one or more EJB containers, as illustrated in Figure 28. EJB containers insulate the enterprise beans from the underlying EJB server and provide a standard application programming interface (EJB API) between the beans and the container. The EJB container provides the standard J2EE services such as transactions, security, naming, and persistence. WAS has a component called Enterprise Java Services (EJS) that supports the deployment and management of Enterprise JavaBeans in EJB containers.

WAS provides a data source that manages a pool of database connections, as well as an associated cache of prepared statement objects. Prepared statements are cached separately for each connection that executes them. More than one data source can be configured.

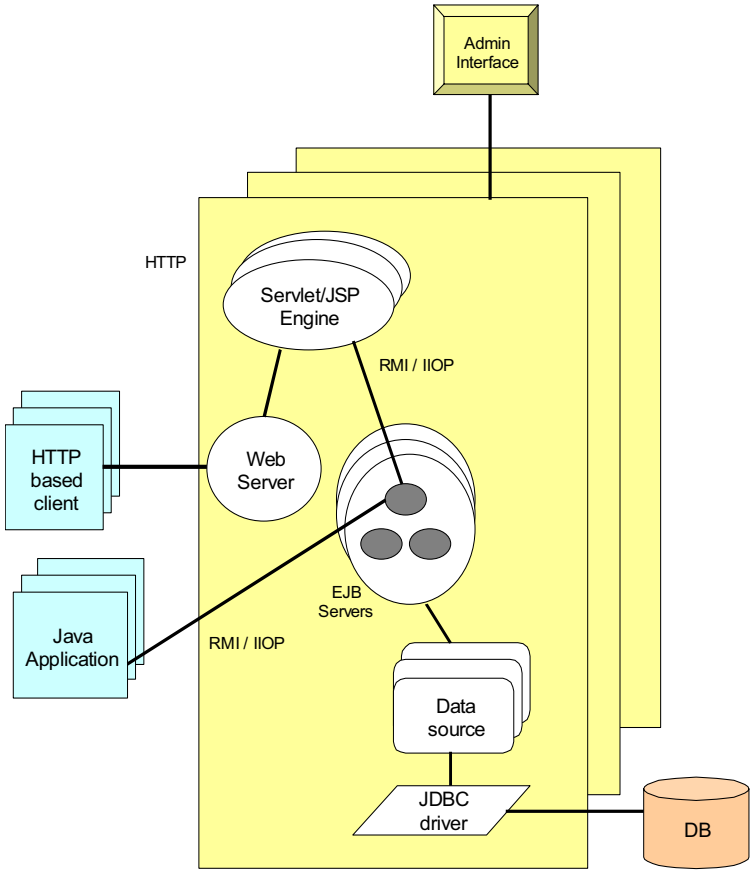


FIGURE 28 WAS EJB ENVIRONMENT

WAS AE v4.0 is fully J2EE 1.2 compatible with early support for some features specified in J2EE 1.3. They are:

- Java Message-driven bean (JMS) is supported in WebSphere AE v4.0.
- WebSphere AE v4.0 supports J2EE Connector Architecture (JCA) and provides a set of IBM application adapters, which allow integration with many WebSphere Host On-Demanding applications, including SAP, PeopleSoft, Oracle ERP Financials and J.D. Edwards.

5.6.4. Session Beans

WAS AE supports stateless and stateful session beans and the associated session bean life-cycle defined in J2EE.

Stateless session beans remain in the ready state after creation of its in memory object. EJB removal is caused by a client or the container invoking a remove method defined in the bean's interface. A container can implicitly call a remove method on an instance after

the lifetime of the session bean has expired. The lifetime is set in the deployment descriptor with the *timeout* attribute.

The WAS container has a sophisticated algorithm for managing which EJB instances are retained in memory. When a container determines that a stateful session bean instance is no longer required in, it invokes the bean instance's *ejbPassivate* method and moves the bean instance into a reserve pool. The bean's state information is then stored on secondary storage. If a client invokes a method on a passivated instance of a stateful session bean, the container re-activates the instance by restoring the instance's state and then invoking the bean instance's *ejbActivate* method. When this method returns, the bean instance is again in the ready state.

5.6.5. Entity Beans

WAS supports both Container-Managed Persistence (CMP) and Bean-Managed Persistence (BMP) entity beans. The following needs to be considered when developing and deploying entity beans:

- Need to create a *finder helper* interface for each CMP entity bean that contains specialized finder methods (other than *the* `findByPrimaryKey` method).
- The primary key class of a CMP entity bean must override the `equals` method and the `hashCode` method inherited from the `java.lang.Object` class.
- The *run-as identity* and *access control* deployment descriptor attributes are not used for entity beans.
- When Visual Age for Java can be used to create the deployed JAR file, WebSphere provides a deployment tool, *ejbdeploy*, to help generate the deployed JAR file. The tool can be applied either manually or automatically from the admin console.
- For CMP, if the automatic deployment process in the console is used, the order and names of the columns in the generated table are not guaranteed to match the table configuration needed by the associated database. Basically this is because the console deployment process makes certain assumptions about the order of container-managed fields. If automatic deployment is used within the console, a number of conventions must be followed and naming restrictions adhered to.
- EJBs that participate in an inheritance hierarchy must be deployed in a single JAR file, and the inheritance hierarchy must be installed and removed as a unit. Also, the JNDI name of each bean in the hierarchy must be unique within its container.

5.6.6. Transaction Service

WAS' transaction service is implemented as a Java wrapper for an implementation of the OMG OTS version 1.1. Note that the EJB transaction model does not support transaction nesting.

In the EJB server environment, transactions are handled by three main components of the transaction service:

- A transaction manager interface that enables the EJB server to control transaction boundaries within EJBs based on the transactional attributes specified for the beans.

- An interface (UserTransaction) that allows an EJB or an EJB client to manage transactions. The container makes this interface available via the name service.

- Transaction coordination using the X/Open XA interface to enable a transactional resource manager (such as a database) to participate in a transaction controlled by an external transaction manager.

WAS supports both JTA and JTS. It provides distributed transaction support for EJBs run in an EJB container as well as to servlets and JSPs run in a web container. Hence the WebSphere EJB container provides the infrastructure for EJBs to participate in a distributed transaction.

5.6.7. Database Connectivity

The table below lists some WAS' database connectivity support and configuration information:

DB	*Class*	*URL Prefix*	*JTA*	*2PC*
DB2 6.1/7.1	COM.ibm.db2.jdbc.app.DB2Driver	jdbc:jta:db2	true	yes
Oracle 8i (8.1.6)	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@<db_host name>:<port_number>	false	no
Sybase12	Com.sybase.jdbc2.jdbc.SybDriver	jdbc:sybase:Tds:<db_hostn ame>: < port_number>	true	yes

JDBC Version 2.0 provides a standard two-phase commit API for multiple databases and a mechanism to look up data sources via the Java Naming and Directory Interface (JNDI). The `javax.sql.DataSource` interface that is defined in the JDBC 2.0 extension specification is a factory for JDBC connections. WebSphere supports JDBC 2.0 drivers and its implementation of the data source object also provides connection pooling to boost performance.

Drivers in the above table are considered fully supported because they can be used as repositories for the WebSphere configuration information and as the backing store for CMP EJBs. The developer does not have to write the JDBC code for those drivers. Other databases with JDBC drivers can be used by servlets and BMP EJBs, but not by CMP EJBs or as an administrative database.

5.6.8. Development Tools

IBM strongly recommends the use of IBM's VisualAge for Java for EJB development for reasons described in Entity Bean section. Third party IDE's can be used, but it is easier to use VisualAge for CMP bean development and deployment descriptor configuration.

WAS provides an offline deployment descriptor configuration tool *ejbdeploy,* which makes certain assumptions about database table names, i.e the entity bean name (Not JNDI name) must be the same as the table name and *vice versa*. This makes it difficult using this tool to configure the deployment descriptor for CMP entity beans to work with existing databases. If this limitation is removed in the next version, WebSphere would be free from Visual Age. The deployment tool can also be activated within the administration console.

VisualAge for Java is tightly integrated with WAS. It provides a powerful development, deployment and remote debugging environment. It supports team-based development and version management as well as having features especially for EJB development.

With WAS, the association of the container-managed data members of entity beans with the associated database table columns can be done in three ways:

1. **bottom-up:** CMP fields are derived from the database schema and entity bean code developed afterwards

2. **top-to-bottom:** CMP entity beans are developed first, and database tables created via VisualAge using information in the entity beans

3. **middle-out:** association is done by mapping the CMP fields to the database schema derived by connecting to the existing database.

On one hand the tight integration of WAS with Visual Age offers a powerful development and deployment environment. On the other hand this integration makes it difficult for WAS to be used with a third party IDE.

Notice that when writing the report, IBM has released another integrated J2EE development/deployment tool*, WebSphere Studio application developer*. Although IBM claims to support both VisualAge and WebShpere Studio currently, we feel VisualAge would be replaced with (or integrated into) WebSphere Studio in long term, because there are many overlaps between the two products and it seems that WebSphere Studio is more powerful and provides more consistent interforace than VisualAge. For details about WebSphere Studio, please refer to http://www-3.ibm.com/software/ad/adstudio.

5.6.9. Scalability

WAS has a Work Load Management (WLM) component to support application scalability. The WLM optimizes the distribution of client processing tasks. It also provides failover in terms of redirection of client requests to other servers when a server is not available.

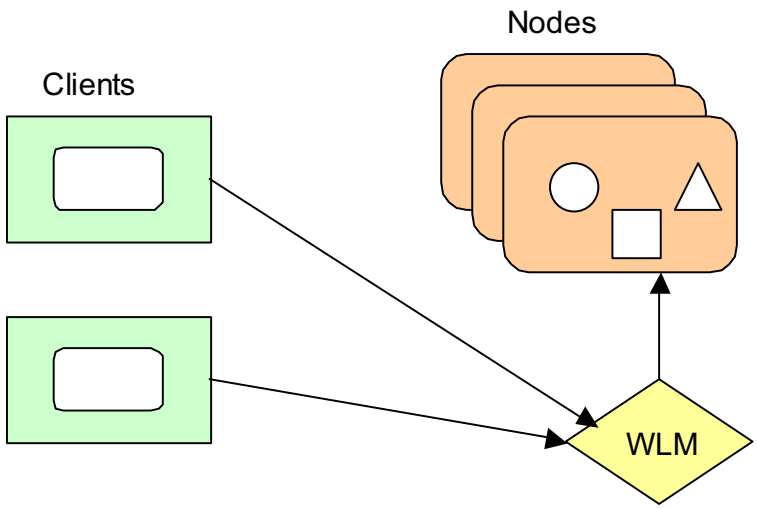


FIGURE 29 WAS APPLICATION SYSTEM RUN TIME ARCHITECTURE

Figure 29 depicts the basic application server system architecture when using WLM. Each node represents a physical machine with WAS installed, and having one or more EJB servers running. The WLM component runs on each node, and supports application scalability by optimizing the distribution of client processing tasks. It also provides failover in terms of redirection of client requests to other servers when a server is not available.

The WLM operates on identical copies of objects – known as *clones*. The clones can be enterprise beans, servlets, containers, or EJB servers. The workload is distributed among the clones, and fails over to other clones when one of them fails. This is achieved by the use of smart bean proxies, which actually contain references to multiple clones of the same replicated beans.

Four workload management policies are supported, namely:

1. round robin
2. round-robin-prefer-local
3. random
4. random-prefer-local

The *prefer-local* suffix in these names indicates that a local server will be given preference if the client resides on the same physical machine.

The following procedure needs to be followed to implement the WLM component:

1. Create clones to replicate instances of objects such as enterprise beans, servlets, and application servers.

2. EJBs and servlets have additional steps for implementing workload management.

- While deploying a bean using WebSphere AE v4.0 deployment tool, wlm-enable is set as default. No additional action is required. For previously (3.5.x) deployed JAR files with no wlm feature, a *wlmjar* utility can be used to transform the jar file to wlm-enabled jar file. In fact, this transformation is to generate additional helper classes required by the workload management to wrap the existing beans.

- For servlets, use servlet redirection to route client requests to remote servlet clones.

  Web clients request service from applications. They make their requests to Web servers. If a requested application resides on an application server remote to a Web server, the Web server must use *a servlet redirector to* relay the request to the remote application server. Workload management is built-in to the servlet redirector.

Administration servers can also participate in workload management for failover support of administration servers. There are configuration properties in the application server configuration file for this purpose.

5.6.10. System Management

WAS offers various ways to perform system management tasks such as creating or configuring servers, drivers, data sources and deploying beans. The following is a list of the facilities:

- GUI based Admin Console
- Web browser client
- XML config – for automated administration
- Java Tcl Command Line System Management Tool for script based administration

There is an IBM performance monitoring tool available integrated in WebSphere Admin Console, called the *Resource Analyzer*. The application server can be configured to indicate which components or objects (eg. Server, bean, ORB etc) are to be monitored, and the level of details of performance data to be collected (i.e. none, low, medium, high). *The Resource Analyzer* then connects to the application server, collects the specified data and presents the performance data in form of charts or tables. The data can also be stored in a file by the Resource Analyzer for post- analysis. Figure 30 shows a screen shot of the WAS system administration console.

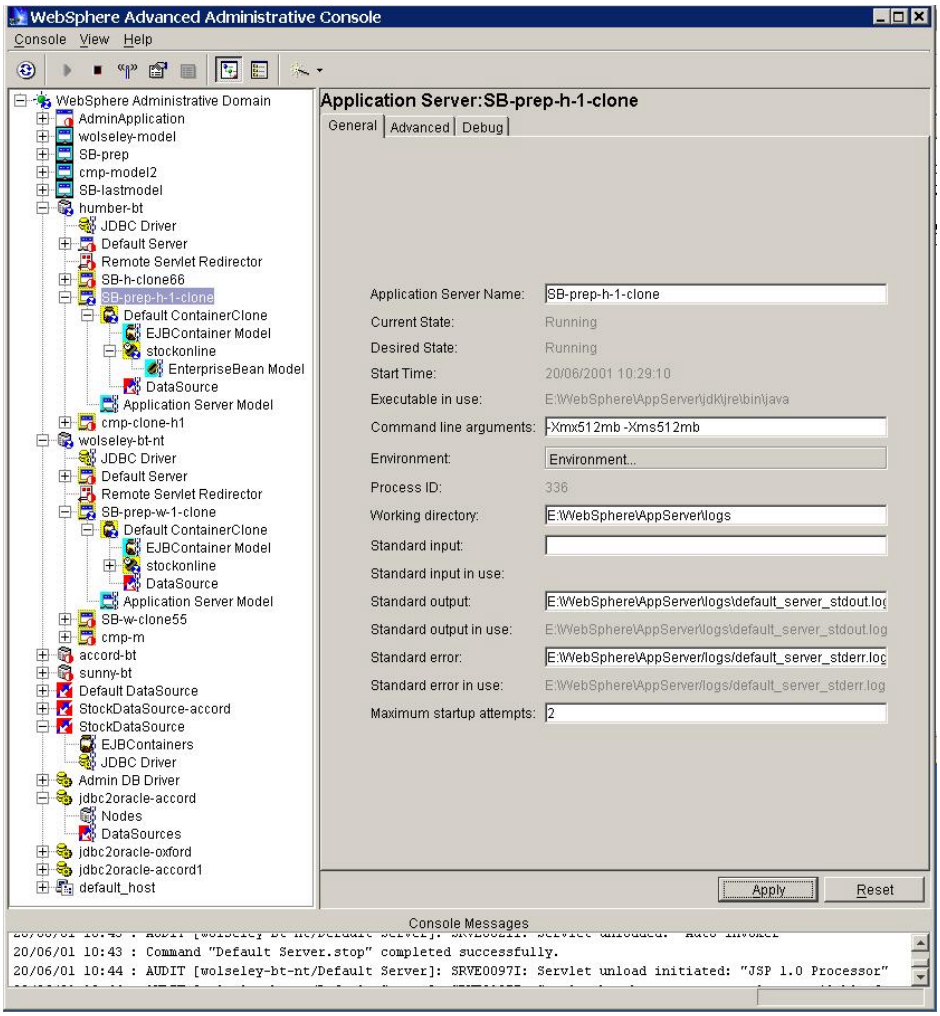


FIGURE 30 SCREEN SHOT OF WAS ADMINISTRATION CONSOLE