BEA WebLogic
Server™

Capacity Planning

About This Document

This document helps you choose the appropriate hardware for a customer solution built with the WebLogic Server™ e-commerce platform.

The document is organized as follows:

- Chapter 1, "Planning for Capacity Requirements," introduces capacity planning and provides information on factors to consider.

- Chapter 2, "Examining Results from the Baseline Applications," presents the characteristics of three baseline applications and provides baseline metrics for several different hardware configurations.

- Chapter 3, "Determining Hardware Capacity Requirements," guides you through the steps of generating hardware capacity requirements.

- Appendix A, "Benchmarking Conclusions," provides graphical illustrations of benchmarking conclusions.

Audience

This document is written for who are responsible for capacity planning and long-range hardware planning for solutions built with WebLogic Server. It is assumed that readers know Web technologies, object-oriented programming techniques, and client/server computing.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File—Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at http://www.adobe.com.

Related Information

The BEA corporate Web site provides all documentation for WebLogic Server. Information related to capacity planning is available in these topics:

- "Preparing to Install WebLogic Server: System Requirements" in the *BEA WebLogic Server Installation Guide*.

- *BEA WebLogic Server Performance and Tuning*

Information on topics related to capacity planning is available from numerous third-party software sources, including the following:

- Capacity Planning for Web Performance: Metrics, Models, and Methods. Prentice Hall, 1998, ISBN 0-13-693822-1 at http://www.cs.gmu.edu/~menasce/webbook/index.html.

- Configuration and Capacity Planning for Solaris Servers at http://www.sun.com/books/catalog/Wong.

- J2EE Applications and BEA WebLogic Server. Prentice Hall, 2001, ISBN 0-13-091111-9 at http://www.phptr.com.

- Web portal focusing on capacity-planning issues for enterprise application deployments at http://www.capacityplanning.com/.

Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at http://www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number

- Your company name and company address

- Your machine type and authorization codes

- The name and version of the product you are using

- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Usage
Ctrl+Tab	Keys you press simultaneously.
italics	Emphasis and book titles.

Convention	Usage

`monospace text`	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard.
	
	Examples:
	
	`import java.util.Enumeration;`
	
	`chmod u+w *`
	
	`config/examples/applications`
	
	`.java`
	
	`config.xml`
	
	`float`

`monospace italic text`	Variables in code. *Example*: `String CustomerName;`
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Example*s: LPT1 BEA_HOME OR
{ }	A set of choices in a syntax line.

[ ]	Optional items in a syntax line. *Example*:
	
	`java utils.MulticastTest -n name -a address`
	`      [-p portnumber] [-t timeout] [-s send]`

|	Separates mutually exclusive choices in a syntax line. *Example*:
	
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`      password {application} {source}`

Convention	Usage
. . .	Indicates one of the following in a command line: • An argument can be repeated several times in the command line. • The statement omits additional optional arguments. • You can enter additional parameters, values, or other information
. . .	Indicates the omission of items from a code example or from a syntax line.

Contents

About This Document

Planning for Capacity Requirements

The following sections provide an introduction to capacity planning for the WebLogic Server e-commerce platform:

Introduction to Capacity Planning

BEA WebLogic Server runs on hardware ranging from low-end PCs to high-end mainframes. The process of determining what type of hardware and software configuration is required to meet application needs adequately is called capacity planning. This document provides capacity planning efforts for solutions on WebLogic Server, with a focus on server hardware requirements.

Note: Capacity planning is not an exact science. Every application is different and every user behavior is different. This document is intended as a guide for developing capacity planning numbers and encourages you to err on the side of caution.

Any and all recommendations generated by this guide should be adequately verified before a given system is placed into production. There is no substitute for adequately testing a prototype for capacity planning numbers.

Capacity Planning Factors

A number of factors influence how much capacity a given hardware configuration will need in order to support a WebLogic Server instance and a given application. The hardware capacity required to support your application depends on the specifics of the application and configuration. You should consider how each of these factors applies to your configuration and application.

The following sections discuss several of these factors. Understanding these factors and considering the requirements of your application will aid you in generating server hardware requirements for your configuration. Consider how much your application differs from a baseline application numbers provided in Chapter 2, "Examining Results from the Baseline Applications" Consider the capacity planning questions in Table 1-1.

Table 1-1 Capacity Planning Factors and Information Reference

Capacity Planning Questions	For Information, See:
Is WebLogic Server well-tuned?	"Tuning Your WebLogic Server Deployment" on page 1-7
How well-designed is the user application?	"Database Server Capacity and User Storage Requirements" on page 1-5
Is there enough bandwidth?	"Network Load" on page 1-6
How many transactions need to run simultaneously?	"Concurrent Sessions" on page 1-5

Capacity Planning Questions	For Information, See:
Is the database a limiting factor? Are there additional user storage requirements?	"Database Server Capacity and User Storage Requirements" on page 1-5
What is running on the machine in addition to WebLogic Server?	"Network Load" on page 1-6
Do clients use SSL to connect to WebLogic Server?	"SSL Connections and Performance" on page 1-4
What types of traffic do the clients generate?	"RMI and Server Traffic" on page 1-4
What types of clients connect to the WebLogic Server application?	"Programmatic and Web-based Clients" on page 1-3
Is your deployment configured for a cluster?	"Clustered Configurations" on page 1-6

Programmatic and Web-based Clients

Primarily, two types of clients can connect to WebLogic Server:

- Web-based clients, such as Web browsers and HTTP proxies, use the HTTP or HTTPS (secure) protocol to obtain HTML or servlet output.

- Programmatic clients, such as Java applications and applets, can connect through the T3 protocol and use RMI to connect to the server.

The stateless nature of HTTP requires that the server handle more overhead than is the case with programmatic clients. However, the benefits of HTTP clients are numerous, such as the availability of browsers and firewall compatibility, and are usually worth the performance costs.

Programmatic clients are generally more efficient than HTTP clients because T3 does more of the presentation work on the client side. Programmatic clients typically call directly into EJBs while Web clients usually go through servlets. This eliminates the work the server must do for presentation. The T3 protocol operates using sockets and has a long-standing connection to the server.

A WebLogic Server installation that relies only on programmatic clients should be able to handle more concurrent clients than an HTTP proxy that is serving installations. If you are tunneling T3 over HTTP, you should not expect this performance benefit. In fact, performance of T3 over HTTP is generally 15 percent worse than typical HTTP and similarly reduces the optimum capacity of your WebLogic Server installation.

RMI and Server Traffic

What types of server traffic do the clients generate? If you are using T3 clients, most interaction with the server involves Remote Method Invocation (RMI.) Clients using RMI do not generate heavy traffic to the server because there is only one sender and one listener.

RMI can use HTTP tunneling to allow RMI calls to traverse a firewall. RMI tunneled through HTTP often does not deliver the higher degree of performance provided by non-tunneled RMI.

SSL Connections and Performance

Secure sockets layer (SSL) is a standard for secure Internet communications. WebLogic Server security services support X.509 digital certificates and access control lists (ACLs) to authenticate participants and manage access to network services. For example, SSL can protect JSP pages listing employee salaries, blocking access to confidential information.

SSL involves intensive computing operations. When supporting the cryptography operations in the SSL protocol, WebLogic Server can not handle as many simultaneous connections.

Note the number of SSL connections required out of the total number of clients required. Typically, for every SSL connection that the server can handle, it can handle three non-SSL connections. SSL reduces the capacity of the server by about 33 to 50 percent depending upon the strength of encryption used in the SSL connections. Also, the amount of overhead SSL imposes is related to how many client interactions have SSL enabled. WebLogic Server includes native performance packs for SSL operations.

WebLogic Server Process Load

What is running on the machine in addition to a WebLogic Server? The machine may be processing much more than presentation and business logic. For example, it could be running a Web server or maintaining a remote information feed, such as a stock information feed from a quote service.

Consider how much of your WebLogic Server machine's processing power is consumed by processes unrelated to WebLogic Server. In the case in which WebLogic Server (or the machine on which it resides) is doing substantial additional work, you need to determine how much processing power will be drained by other processes. When a Web server proxy is running on the same machine as WebLogic Server, expect anywhere from 25 to 50 percent of the computing capacity.

Database Server Capacity and User Storage Requirements

Is the database a bottleneck? Are there additional user storage requirements? Often the database server runs out of capacity much sooner that WebLogic Server does. Plan for a database that is sufficiently robust to handle the application. Typically, a good application's database requires hardware three to four times more powerful than the application server hardware. It is good practice to use a separate machine for your database server.

Generally, you can tell if your database is the bottleneck if you are unable to maintain WebLogic Server CPU usage in the 85 to 95 percent range. This indicates that WebLogic Server is often idle and waiting for the database to return results. With load balancing in a cluster, the CPU utilization across the nodes should be about even.

Some database vendors are beginning to provide capacity planning information for application servers. Frequently this is a response to the three-tier model for applications.

An application might require user storage for operations that do not interact with a database. For example, in a secure system disk and memory are required to store security information for each user. You should calculate the size required to store one user's information, and multiply by the maximum number of expected users.

Concurrent Sessions

How many transactions must run concurrently? Determine the maximum number of concurrent sessions WebLogic Server will be called upon to handle. For each session, you will need to add more RAM for efficiency. BEA Systems recommends that you install a *minimum* of 256 MB of memory for each WebLogic Server installation that will be handling more than minimal capacity.

Next, research the maximum number of clients that will make requests at the same time, and how frequently each client will be making a request. The number of user interactions per second with WebLogic Server represents the total number of interactions that should be handled per second by a given WebLogic Server deployment. Typically for Web deployments, user interactions access JSP pages or servlets. User interactions in application deployments typically access EJBs.

Consider also the maximum number of transactions in a given period to handle spikes in demand. For example, in a stock report application, plan for a surge after the stock market opens and before it closes. If your company is broadcasting a Web site as part of an advertisement during the World Series or World Cup Soccer playoffs, you should expect spikes in demand.

Network Load

Is the bandwidth sufficient? WebLogic Server requires enough bandwidth to handle all connections from clients. In the case of programmatic clients, each client JVM will have a single socket to the server. Each socket requires bandwidth. A WebLogic Server handling programmatic clients should have 125 to 150 percent the bandwidth that a server with Web-based clients would handle. If you are interested in the bandwidth required to run a web server, you can assume that each 56kbps (kilobits per second) of bandwidth can handle between seven and ten simultaneous requests depending upon the size of the content that you are delivering. If you are handling only HTTP clients, expect a similar bandwidth requirement as a Web server serving static pages.

The primary factor affecting the requirements for a LAN infrastructure is the use of in-memory replication of session information for servlets and stateful session EJBs. In a cluster, in-memory replication of session information is the biggest consumer of LAN bandwidth. Consider whether your application will require the replication of session information for servlets and EJBs.

To determine whether you have enough bandwidth in a given deployment, look at the network tools provided by your network operating system vendor. In most cases, including Windows NT, Windows 2000, and Solaris, you can inspect the load on the network system. If the load is very high, bandwidth may be a bottleneck for your system.

Clustered Configurations

Clusters greatly improve efficiency and failover. Customers using clustering should not see any noticeable performance degradation. A number of WebLogic Server deployments in production involve placing a cluster of WebLogic Server instances on a single multiprocessor server.

Large clusters performing in-memory replication of session data for Enterprise JavaBeans (EJBs) or servlets require more bandwidth than smaller clusters. Consider the size of session data and the size of the cluster.

Application Design

How well-designed is the application? WebLogic Server is a platform for user applications. Badly designed or unoptimized user applications can drastically slow down the performance of a given configuration from 10 to 50 percent The prudent course is to assume that every application that is developed for WebLogic Server will not be optimal and will not perform as well as benchmark applications. I increase the maximum capacity that you calculate or expect.

WebLogic Server documentation and Customer Support centers can help you design the most efficient applications for WebLogic Server. For the latest information, visit the WebLogic Server Web sites.

Tuning Your WebLogic Server Deployment

Tune your WebLogic Server deployment according to *BEA WebLogic Server Performance and Tuning*. If the server is not tuned, expect about a 33% decrease in performance compared to the baseline numbers in Chapter 2, "Examining Results from the Baseline Applications."

Next Steps

The next step is to examine the characteristics and baseline results from sample applications. This step can assist you in generating capacity planning requirements for your application. Proceed to Chapter 2, "Examining Results from the Baseline Applications."

Examining Results from the Baseline Applications

The following sections provide information about baseline applications:

MedRec Benchmark Overview

The MedRec sample application was used for the Light and Heavy-weight application tests. The difference between the Light and Heavy-weight application tests is the communication protocol used between client and server. For the Light-weight application test, the HTTP 1.1 protocol is used from the clients. For the Heavy-weight application test, the HTTPS 1.1 protocol is used, which involves secure/encrypted data communication between client and server.

Avitek Medical Records (or MedRec) is a WebLogic Server sample application suite that concisely demonstrates all aspects of the J2EE platform. MedRec is designed as an educational tool for all levels of J2EE developers; it showcases the use of each J2EE component, and illustrates best practice design patterns for component interaction and client development.

The MedRec application provides a framework for patients, doctors, and administrators to manage patient data using a variety of different clients. Patient data includes:

- Patient profile information—A patient's name, address, social security number, and login information.

- Patient medical records—Details about a patient's visit with a physician such as the patient's vital signs and symptoms as well as the physician's diagnosis and prescriptions.

The MedRec application suite consists of two main J2EE applications. Each application supports one or more user scenarios for MedRec:

- medrecEar—Patients log in to the patient Web Application (patientWebApp) to edit their profile information, or request that their profile be added to the system. Patients can also view prior medical records of visits with their physician. Administrators use the administration Web Application (adminWebApp) to approve or deny new patient profile requests. medrecEar also provides all of the controller and business logic used by the MedRec application suite, as well as the Web Service used by different clients.

- physicianEar—Physicians and nurses log in to the physician Web Application (physicianWebApp) to search and access patient profiles, create and review patient medical records, and prescribe medicine to patients. The physician application is designed to communicate using the Web Service provided in medrecEar.

The medrecEAR application is deployed to a single WebLogic Server instance called MedRecServer. The physicianEAR application is deployed to a separate server, PhysicianServer, which communicates with the controller components of medrecEAR by using Web services.

Changes to MedRec and Transactions Used for Benchmarking

For capacity planning benchmarking purposes, the WebLogic Server 8.1 MedRec application has been optimized. The patient registration process was made synchronous and two-phase commit is omitted in the modified application. The client running the LoadRunner benchmarking software is used to generate the load using the following sequence of transactions:

1. A patient connects to the MedRec application and opens the Start page.

2. The patient completes the registration process.

3. A physician connect to the online system and signs in.

4. The physician performs a lookup for a patient.

5. The physician creates a new visit for the patient, creates and deletes prescriptions.

6. The physician logs out.

7. A patient signs in and edits the user profile.

8. The patient makes changes to her profile and updates the information.

9. The patient logs out.

Each action in the sequence above is treated as a transaction for computing the throughput. For each client, a unique patient ID is generated for every iteration.

Overview of the Baseline Applications

BEA produced baseline capacity planning results as a starting point. These baseline numbers were generated using three different applications:

- Light application software—MedRec application consisting of `search`, `select`, `insert`, and `update` transactions using the HTTP protocol.

- Heavy application software—MedRec application consisting of `search`, `select`, `insert`, and `update` transactions using the HTTPS (secure) protocol.

You can use the metrics provided for each of these applications to set the expectation about WebLogic Server throughput, response time, number of concurrent users, and overall impact on performance based on customer choice of hardware and configuration.

The measurements in the systems under study were captured without adding a think-time factor to the transaction mix.

Note: For a capacity planning profile using the WebAuction application on standard hardware, see Chapter 15, "Analysis of Capacity Planning," in *J2EE Applications and BEA WebLogic Server*, Prentice Hall, 2001, ISBN 0-13-091111-9 at http://www.phptr.com.

Configuration for MedRec Baseline Applications (Light and Heavy)

For all measurements, the client applications and the Oracle database were kept on a 4x400 Mhz Solaris processor. A WebLogic Server intance was started on each processor/configuration combination listed. Baseline numbers from the capacity measurement runs for each of three applications are listed in tabular form, after the description of the application.

The baseline numbers were collected using a three-tier configuration, separating the client applications (running LoadRunner benchmarking software), WebLogic Server containing the MedRec baseline application, and the Oracle database server. The database in the systems under study is an Oracle 8.1.7 database. All measurements were taken using a 1 GB network.

Figure 2-1 Three-Tier Configuration
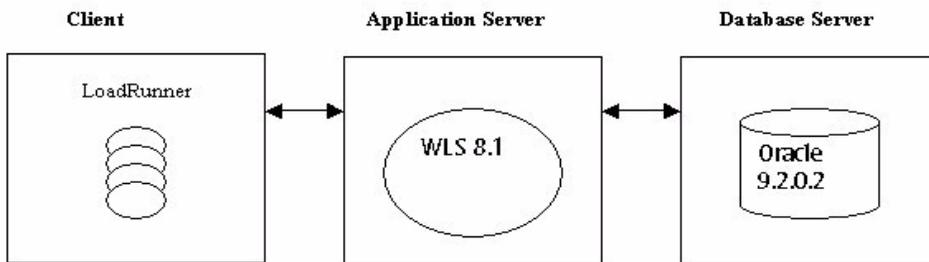
WebLogic Server Configurations

The baseline numbers were collected using the following product versions and settings:

- BEA WebLogic Server 8.1 GA product.
- Sun JDK141_02 with the following Java output on a Solaris platform:

  java version "1.4.1_02-ea"

  Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_02-ea-b01)

Java HotSpot(TM) Client VM (build 1.4.1_02-ea-b01, mixed mode)

- BEA JRockit on the Windows platform.

  java version "1.4.1_02"

  Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_02)

  BEA WebLogic JRockit(R) Virtual Machine (build 8.1-1.4.1_02-win32-CROSIS-20030317-1550, Native Threads, Generational Concurrent Garbage Collector)

- Default WebLogic JVM configuration, except for the following:

  JVM configuration—Initial and maximum heap size set to 1G

  Thread count—Set to 50, the optimal value found.

  JDBC configuration:

    Used Oracle 9.2.0.2 Thin Driver

    JDBC Connection pool size set to 50 (initial and maximum), the optimal value found.

    JDBC StatementCacheSize set to 50.

  Cluster configuration—One Managed Server for each node with the Administration Server running on the first node.

Measured Configurations

The following configurations were measured:

- WebLogic Server 8.1, Sun JDK141_02: 1-, 2-, 4- and 8-processor configuration on 750 MHz processor with 1 GB memory settings and one instance of WebLogic Server with BEA's best tuning.

- WebLogic Server 8.1, Sun JDK141_02: 1-, 2-, 3- and 4-node configuration on 400 MHz processor with 1 GB memory settings and one instance of WebLogic Server per node with BEA's best tuning. The LoadRunner clients performed the load balancing.

- WebLogic Server 8.1, JRockit version 141_02: 1-, 2-, 3- and 4-node configuration on 700 MHz processor with 1 GB memory settings and one instance of WebLogic Server per node with BEA's best tuning. The LoadRunner clients performed the load balancing.

Database Configurations

Oracle 9.2.0.2.0 was used on an eight-way 750 MHz E6800. To improve performance, the Oracle logs (8GB) were placed on a RAID array of four disks striped together and another RAID array of four disks striped for all the other database files.

Database configurations were as follows:

- Host—Solaris 9

- CPU—8X 750 MHz UltraSparc III – Sun E6800

- Memory—8GB

- Disk—2 X 18GB + 4 x 18GB RAID Array

- Software—Oracle9i Enterprise Edition Release 9.2.0.2.0

Client Configurations

Client configurations were as follows:

- Host—Windows 2000 Advanced Server

- CPU—4X 700 MHz Poweredge 6400

- Memory—4 GB

- Disk—2 X 18GB

- Software—LoadRunner 7.02

Network Configuration

A one gigabite (1 GB) network was used.

LoadRunner Configurations

LoadRunner 7.02 was used for client load generation on a 4X700 MHz Windows 2000 box with 4 GB of memory. No think time is used for the test. HTTP/HTTPs version 1.1 was used.

About Baseline Numbers

The baseline numbers produced by the benchmarks used in this study should not be used to compare WebLogic Server with other application servers or hardware running similar benchmarks. The benchmark methodology and tuning used in this study are unique.

A number of benchmarks show how a well-designed application can perform on WebLogic Server. These benchmarks are available from BEA Systems. For more information, contact your BEA Systems sales representative.

Measured TPS for Light MedRec Application on UNIX

The measurements for the light MedRec application on UNIX were obtained using Solaris hardware and the HTTP protocol. Table 2-1 lists the number of clients in the first column, and hardware configuration with transactions per second in the remaining columns. For this application on UNIX, the designation for the configuration number is light medrec UNIX n where n is the configuration number (lmUn).

(TPS = Transactions Per Second)

Table 2-1 Number of Clients x TPS for Light MedRec Application on UNIX

Processor in MHz Config #	1x750 lmU1	2x750 lmU2	4x750 lmU3	8x750 lmU4	4x400 1-node lmU5	4x400 2-node lmU6	4x400 3-node lmU7	4x400 4-node lmU8
Number of Clients	TPS	TPS	TPS	TPS	TPS	TPS	TPS	TPS
1 client	16.20	19.15	21.45	22.63	15.69			
4 clients	17.20	29.80	47.95	58.12	34.54	53.57	58.67	63.33
10 clients	16.74	30.12	53.87	80.96	38.87	76.82	102.48	
20 clients	17.04	30.54	55.76	83.84	39.17	81.14	120.73	154.76
40 clients	16.72	30.81	54.02	84.66	39.32	81.65	122.50	163.23
80 clients	16.83	30.71	54.33	84.22	39.02	80.64	121.47	163.84

100 clients	16.85	30.71	54.36	84.09	38.78	80.24	122.26	163.86
150 clients							121.38	162.71
200 clients								163.10
Max TPS	**17.20**	**30.81**	**55.76**	**84.66**	**39.32**	**81.65**	**122.50**	**163.86**
Appserver CPU Utilization for Max TPS	**98%**	**97%**	**95%**	**91%**	**95.09%**	**95.58%**	**95.35%**	**95.22%**
DB Server CPU Utilization for Max TPS	**<1%**	**1.30%**	**2.25%**	**3.91%**	**1.70%**	**3.64%**	**6.21%**	**9.70%**
DB Server Disk Utilization for Max TPS	**6%**	**12%**	**19%**	**28.24**	**13.94**	**27.79**	**40.58**	**53.25**

Table 2-2 summarizes the TPS achieved for each processor type/configuration combination measured, and identifies the number of concurrent clients running to achieve the measured TPS result.

Table 2-2 Measured TPS for Light MedRec Application on UNIX

Processor Type, Configuration	Config #	Measured TPS	Number of Clients
Solaris 1x750 MHz	lmU1	17.20	4
Solaris 2x750 MHz	lmU2	30.81	40
Solaris 4x750 MHz	lmU3	55.76	20

Solaris 8x750 MHz	lmU4	84.66	40
Solaris 4x400 – 1 Node	lmU5	39.32	40
Solaris 4x400 – 2 Node	lmU6	81.65	40
Solaris 4x400 – 3 Node	lmU7	122.50	40
Solaris 4x400 – 4 Node	lmU8	163.86	100

Measured TPS for Heavy MedRec Application on UNIX

The measurements for the heavy MedRec application on UNIX were obtained using Solaris hardware and the HTTP protocol. Table 2-3 lists the number of clients in the first column, and hardware configuration with transactions per second in the remaining columns. For this application on UNIX, the designation for the configuration number is <u>h</u>eavy <u>m</u>edrec <u>U</u>NIX n (hmUn).

(TPS = Transactions Per Second)

Table 2-3 Number of Clients x TPS for Heavy MedRec Application on UNIX

Processor in MHz Config #	4x750 hmU1	4x400 hmU2
Number of Clients	**TPS**	**TPS**
1 client	12.18	8.76
4 clients	27.49	20.70
10 clients	32.13	23.00
20 clients	31.08	23.07
40 clients	31.39	23.35
80 clients	31.43	23.44
100 clients	31.81	23.29
Max TPS	**32.13**	**23.44**
Appserver CPU Utilization for Max TPS	**93.00%**	**94.06%**
DB Server CPU Utilization for Max TPS	**1.16%**	**0.89%**
DB Server Disk Utilization for Max TPS	**12.23%**	**9.22%**

Measured TPS for Light MedRec Application on Windows 2000

The measurements for the light MedRec application on Windows 2000 were obtained using the HTTP protocol. Table 2-4 lists the number of concurrently running clients in the first column, and hardware configuration with transactions per second in the remaining columns. For this application on Windows 2000, the designation for the configuration number is light medrec Windows n (lmWn).

Table 2-4 Number of Clients x TPS for Light MedRec Application on Windows 2000

Processor in MHz Config #	Win2K 4x700 1-Node lmW1	Win2K 4x700 2-Node lmW2	Win2K 4x700 3-Node lmW3	Win2K 4x700 3-Node lmW3
Number of Clients	TPS	TPS	TPS	TPS
1 client	33.50			
4 clients	78.53	107.58	112.36	122.83
10 clients	80.24	149.09	205.13	
20 clients	79.28	148.12	226.53	297.35
40 clients	77.36	148.78	223.53	299.11
80 clients	76.16	146.75	224.38	291.82
100 clients	76.38	144.99	220.02	292.22
150 clients			219.38	291.17
200 clients				292.87
Max TPS	**80.24**	**149.09**	**226.53**	**299.11**
Appserver CPU Utilization for Max TPS	**92.51%**	**92.42%**	**92.40%**	**92.82%**
DB Server CPU Utilization for Max TPS	**3.32%**	**7.63%**	**14.63**	**23.44%**
DB Disk Server CPU Utilization for Max TPS	**26.32%**	**44.63%**	**70.25%**	**88.98%**

The following table summarizes the measured TPS achieved for each processor type/configuration combination measured, and identifies the number of concurrent clients running to achieve the measured TPS result.

Table 2-5 Measured TPS for Light MedRec on Windows 2000

Processor Type, Configuration	Config #	Measured TPS	Number of Clients
Windows 2000 4x700 MHz – 1 Node	lmW1	80.24	10
Windows 2000 4x700 MHz – 2 Node	lmW2	149.09	10
Windows 2000 4x700 MHz – 3 Node	lmW3	226.53	20
Windows 2000 4x700 MHz – 4 Node	lmW4	299.11	40

Measured TPS for Heavy MedRec Application on Windows 2000

The following are measurements for the heavy MedRec application on Windows 2000. All tests on all operating systems use the HTTPS (secure) protocol. The JVM used for heavy MedRec application is the Sun Microsystems JDK.

Table 2-6 lists the number of clients in the first column, and hardware configuration with transactions per second in the remaining columns. For this application on Windows 2000, the designation for the configuration number is heavy medrec { U | W } n (hmUn or mmWn).

Table 2-6 Number of Clients x TPS for Heavy MedRec Application on Windows 2000

Processor in MHz Config #	Win2K 4x700 hmW1
Number of Clients	**TPS**
1 client	16.29
4 clients	38.62
10 clients	38.27
20 clients	37.58
40 clients	37.37
80 clients	36.86
100 clients	36.92
Max TPS	**38.62**

Next Steps

After examining the characteristics and baseline results from sample applications, the next step is to compare your application to one or more of the baseline samples. Then proceed to Chapter 3, "Determining Hardware Capacity Requirements." These steps can assist you in generating capacity planning requirements for your application.

Determining Hardware Capacity Requirements

The following sections provide information on how to determine your server hardware capacity requirements.

Assessing Your Application Performance Objectives

At this stage in capacity planning, you gather information about the level of activity expected on your server, the anticipated number of users, the number of requests, acceptable response time, and preferred hardware configuration. Capacity planning for server hardware should focus on maximum performance requirements and set measurable objectives for capacity.

For your application, take the information that you derive from Chapter 2, "Examining Results from the Baseline Applications," to see how your application differs from one of the baseline applications. For example, if you are using the HTTPS protocol for a business application similar to MedRec, you should examine the metrics provided for the heavy MedRec application. Perform the same logical process for all of the factors listed in "Capacity Planning Factors" on page 1-2.

The numbers that you calculate from using one of our sample applications are of course just a rough approximation of what you may see with your application. There is no substitute for benchmarking with the actual production application using production hardware. In particular, your application may reveal subtle contention or other issues not captured by our test applications.

Calculating Hardware Requirements

To calculate hardware capacity requirements:

1. Evaluate the complexity of your application, comparing it to one or more of the applications described in Chapter 2, "Examining Results from the Baseline Applications." The example in "Guidelines for Calculating Hardware Requirements" on page 3-3 identifies this value as the Complexity Factor. If your application is about as complex as one of the baselines, your Complexity Factor = 1.

2. Consider what throughput is required for your application. In the example, this is called the Required TPS (transactions per second).

3. Take the preferred hardware TPS value from the appropriate table. The example in "Guidelines for Calculating Hardware Requirements" on page 3-3 identifies this value as the Reference TPS.

Guidelines for Calculating Hardware Requirements

The number of computers required is calculated as follows:

Number of boxes = (Required TPS) / (Reference TPS / Complexity Factor)

For example, if your assessment shows:

- Your application is twice as complex as the Light MedRec application; the Complexity Factor = 2.

- The requirement is for a 400 TPS; the Required TPS = 400.

- The preferred hardware configuration is Windows 2000 using 4x700 MHz processors.

- The Reference TPS is 205, from Table 2-3, configuration number lmW1.

The number of boxes required is approximately equal to:
400/(205/2) = 400/102.5 = next whole number, 3.90 rounded up = 4 boxes.

Always test the capacity of your system before relying on it for production deployments.

Best Practices When Calculating Hardware Requirements

BEA recommends several best practices related to capacity planning.

- Tune and optimize your application. During the testing process, plan to take steps to optimize the application you will be running on WebLogic Server. See *BEA WebLogic Server Performance and Tuning*.

- Be conservative when making your capacity estimates. Plan for peak loads that your application might experience. If absolute reliability is preferable to hardware cost savings, increase your estimates.

- Consider future growth requirements. WebLogic Server is extensible. Your implementation might grow over time, as more and bigger clients are added.

- Stress test your application under real-world conditions. Prototype software and tools are available such as LoadRunner (http://www-heva.mercuryinteractive.com) and eLoad(R) (http://www.empirix.com). Some hardware vendors and independent third-party companies offer test laboratories where you can test the hardware on which you plan to deploy.

Capacity Planning Guide FAQs

This section contains some frequently asked questions and answers.

Mainframes and Other Types of Hardware

What about mainframes or other types of hardware? How do I determine capacity planning information for those systems? Consider the mainframe as a multiple of many individual PCs.

Heterogeneous Client Types

What should I assume about heterogeneous client types? How about programmatic along with HTTP clients? When in doubt, assume the worst. In this case, assume that 100% of the clients will be HTTP clients and develop your capacity planning numbers appropriately.

Improving Application Performance

How do I get my application to run faster? Tune WebLogic Server first, using the *BEA WebLogic Server Performance and Tuning*, included in BEA Systems documentation. Second, consider using a tuning tool such as jProbe from sitraka at http://www.sitraka.com to find bottlenecks in the code. Also, check the latest in the WebLogic Server documentation and customer support FAQs to see up-to-date application tuning and design guides.

Determining Memory Requirements

How do I determine how much memory I need? BEA Systems recommends that you install a minimum of 256 MB of memory for each machine running WebLogic Server that will be handling more than minimal capacity. If you are expecting a very heavy load, BEA Systems recommends that you increase your memory substantially.

A WebLogic Server deployment tracks session objects in memory, either to RAM or swapping to disk. There must be sufficient RAM/disk to store all the session objects. This RAM is accessible to Java through the Java heap.

For more information on memory requirements, see *BEA WebLogic Server Performance and Tuning*.

Allowing for Think Time

"Think time" is the time spent by a user between transactions or surfing from one page to another. The BEA capacity planning benchmarks do not account for think time. Rather, the benchmarks send out each subsequent request as soon as the previous one finishes. However, in a real world scenario, a user typically spends a little time between pages.

Given an estimated think time (ETT) in seconds and anticipated transactions per second (TPS) there is a general calculation you can make to estimate the number of concurrent users your system can support:

Number of concurrent users = (ETT x TPS) + TPS

For example, with a TPS value of 100:

- For an estimated think time (ETT) = 0 seconds

  Number of concurrent users = (0 x 100) + 100 = 100

- For an ETT = 1 second

  Number of concurrent users = (1 x 100) + 100 = 200

- For an ETT = 10 seconds

 Number of concurrent users = (10 x 100) + 100 = 1100

Benchmarking Conclusions

This section provides illustrations of benchmarking conclusions based on tests performed:

Solaris Platform Summary - Linear Scalability

The following conclusions were reached based on linear scalability (increasing the number of processors on the same box).

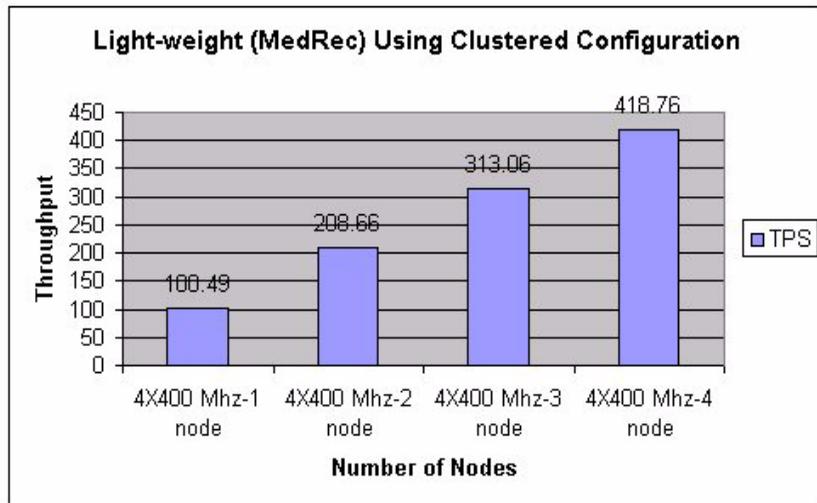**Figure A-1   Light-weight (MedRec) Using HTTPS**
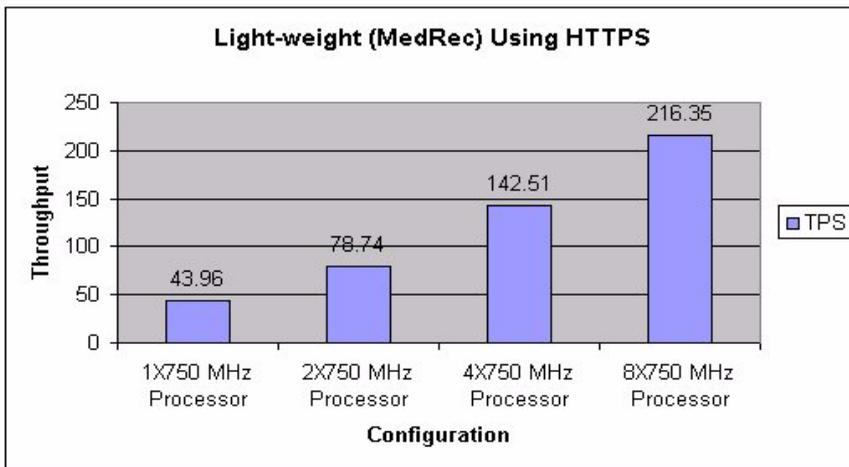
- The throughput of a two-processor configuration is 1.79 times that of a one-processor configuration.

- The throughput of a four-processor configuration is 1.80 times that of a two-processor configuration.

- The throughput of an eight-processor configuration is 1.52 times that of a four-processor configuration.

Solaris Platform Summary - Horizontal Scalability

The following conclusions were reached based on horizontal scalability (increasing the number of boxes, using a clustered configuration).

Figure A-2   Light-weight (MedRec) Using Clustered Configuration
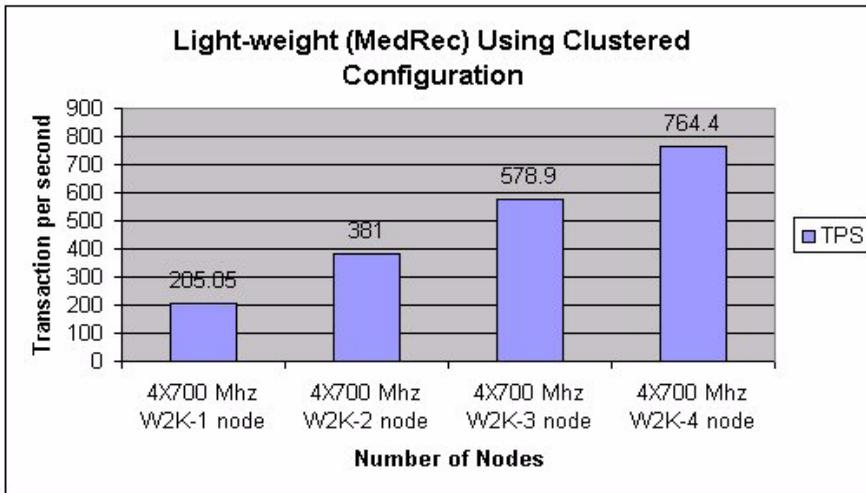

The throughput of a one-node to four-node cluster shows linear scalability.

Windows Platform Summary - Horizontal Scalability

The following conclusions were reached based on horizontal scalability (increasing the number of boxes, using a clustered configuration).

Figure A-3   Light-weight (MedRec) Using Clustered Configuration

- The throughput of a two-node configuration is 1.85 times that of a one-node cluster.

- The throughput of a three-node configuration is 2.82 times that of a one-node cluster.

- The throughput of a four-node configuration is 3.72 times that of a one-node cluster.

Figure A-4   Heavy-weight (MedRec) Using HTTPS
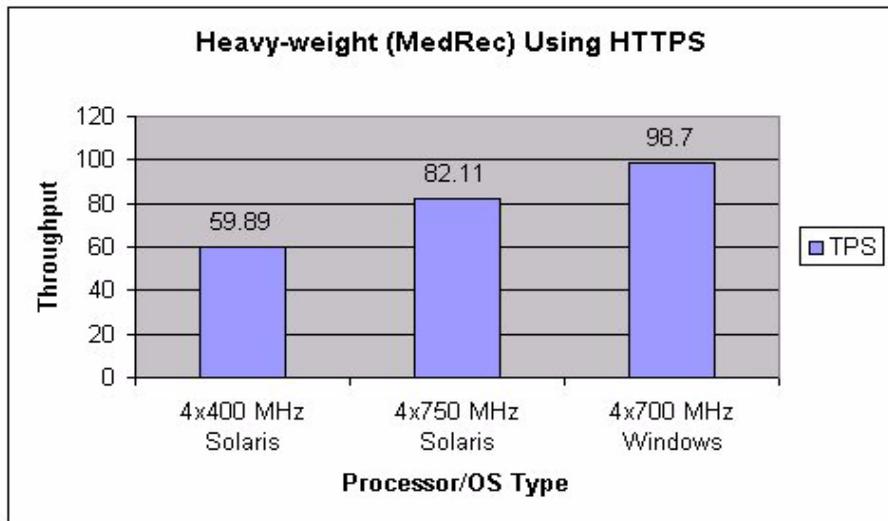
- HTTPS test on 4X400 Mhz Solaris configuration gives 60% performance of HTTPS test.

- HTTPS test on 4X750 Mhz Solaris configuration gives 58% performance of HTTPS test.

- HTTPS test on 4X700 Mhz Windows 2000 configuration gives 49% performance of HTTPS test

Benchmarking Conclusions