



BEA WebLogic Portal™®

Capacity Planning Guide

Version 10.2
February 2008

Contents

Capacity Planning for WebLogic Portal

Capacity Planning Factors to Consider	2
Performance Testing Suggestions	3
Hardware Configuration and Performance Requirements	3
Clustered Configurations	4
Simulated Workload	5
Concurrent Sessions	6
Tuning WebLogic Server	6
Application Design	6
SSL Connections and Performance	7
WebLogic Server Process Load	7
Database Server Capacity	8
Network Load	9
Selecting Your JVM	10
Performance Results	10
Test Applications	11
HP Linux Hardware and Server Configurations	14
Sun Solaris Hardware and Server Configurations	15
Comparing 10.2 Results Against Previous Releases	16
Portal Framework Benchmark Results	16
Portal Framework Concurrent User Results	18
WSRP Benchmark Results	21

Content Management Benchmark Results	23
Other Resources	29

Capacity Planning for WebLogic Portal

BEA WebLogic Portal is the leading Java portal for the delivery of a personalized web experience with rich, Web 2.0 interactions. Thousands of solutions for business have been delivered with the WebLogic Portal for companies large and small. The WebLogic Portal sets the standard for the delivery of enterprise-class portal solutions and is designed to meet the scalability and performance requirements for some of the most demanding deployments. WebLogic Portal solutions range from departmental applications to some of the largest corporate deployments in existence. Deployment architectures may include few, independent servers or sophisticated cluster configurations with state replication – WebLogic Portal is designed with the flexibility to serve the solution demands. The architecture and physical deployment of any given implementation depends on numerous factors which this document will cover. The process of evaluating the hardware and software configuration needed for a given deployment is called capacity planning.

This document covers the steps involved with capacity planning for WebLogic Portal 10.2 and will serve as a baseline set of measurements so that more accurate estimates can be made for capacity planning by our customers.

Capacity planning is not an exact science. Every application deployment and user is unique and testing cannot anticipate all the interactions that may result. This document is a guide for developing capacity planning numbers not a prescription or recommendation for any specific deployment. While designing your solution and testing the configuration, careful consideration must be given to business expectations, periodic fluctuations in demand, and application constraints – few deployments are identical. Customers need to plan carefully, test methodically, and incorporate conservative principles such as deploying with extra capacity – not a bare minimum. Before deploying any application into a production environment the application

should be put through a rigorous performance testing cycle. For more information on performance testing see [“Approaches to Performance Testing”](#).

Note: Any and all recommendations provided in this guide should be adequately verified before a given system is moved into production. As stated above, the data published in this document is meant to represent the specific configuration that was tested. There are a number of factors that come into play when determining how much capacity a system can support and thus there is no substitute for adequately testing a prototype to obtain your own capacity planning numbers. If you do not have a group experienced in physical architecture, capacity planning, performance testing, and disaster recovery planning you should strongly consider getting assistance.

Capacity Planning Factors to Consider

Numerous factors must be weighed to estimate the demands of any given WebLogic Portal deployment. The hardware capacity required to support your application depends on the specifics of the application, server hardware, network infrastructure, and portal configuration. You should consider how each of these factors applies to your implementation.

[Table 1](#) provides a checklist for capacity planning. The following sections discuss each item in the checklist. Understanding these factors and considering the requirements of your application will aid you in generating server hardware requirements for your configuration.

Table 1 - Capacity Planning Factors and Information Reference

Capacity Planning Questions	For Information, See:
Have you performance tested your application?	“Performance Testing Suggestions” on page 3
Does the hardware meet the configuration requirements?	“Hardware Configuration and Performance Requirements” on page 3
Is WebLogic Portal configured for clustering?	“Clustered Configurations” on page 4
Is the simulated workload adequate?	“Simulated Workload” on page 5
How many users need to run simultaneously?	“Concurrent Sessions” on page 6
Is WebLogic Portal well-tuned?	“Tuning WebLogic Server” on page 6
How well-designed is the user application?	“Application Design” on page 6
Do clients use SSL to connect to WebLogic Portal?	“SSL Connections and Performance” on page 7

Table 1 - Capacity Planning Factors and Information Reference

Capacity Planning Questions	For Information, See:
What is running on the machine in additional to WebLogic Portal?	“WebLogic Server Process Load” on page 7
Is the database a limiting factor?	“Database Server Capacity” on page 8
Is there enough network bandwidth?	“Network Load” on page 9
What JVM is used and with what parameters?	“Selecting Your JVM” on page 10

Performance Testing Suggestions

Capacity planning is the last step in the performance testing process. Before an application is ready to be sized for a production deployment it should go through an iterative performance testing process to ensure that bottlenecks are addressed and that response behaviors meet requirements.

Running benchmarks against the application will set a baseline set of measurements so that as features are added and removed from the application the impact of those changes can be objectively measured.

Profiling the application during development will help flush out performance problems or performance hotspots that could turn into major issues down the road. Catching these kinds of problems early will significantly reduce the overhead in trying to fix them later.

Production monitoring systems are frequently useful additions to production deployments and QA testing to help identify and quantify specific problem areas. Anything that will be included in the production deployment, such as a monitoring tool, load balancer, or intrusion detection software, should be replicated in the testing environment if possible. Many testing configurations include synthetic network traffic generators to make testing as accurate as possible.

Recommendation

Read [Approaches to Performance Testing](#) on BEA’s dev2dev site.

Hardware Configuration and Performance Requirements

The operating systems and hardware configurations that BEA supports for WebLogic Portal 10.2 are documented in [WebLogic Platform 10.2 Supported Configurations](#).

When performance characteristics for any given application deployed in WebLogic Portal are not met the first question that should be asked in this situation is if the hardware is sufficient. This is the single most important factor when determining how well the system will scale. During BEA's internal performance testing, WebLogic Portal was CPU bound, so the performance of the system will depend on how fast each CPU is and how many total CPUs there are.

BEA's internal performance testing indicated a direct relationship between the performance of the system and the overall clock-speed of the CPU(s). By adding more CPUs, or faster CPUs the capacity of the system will typically increase. Additionally, by clustering machines WebLogic Portal will gain additional scalability due to the addition of CPUs to the overall application deployment. Newer processor technology is also a big factor in determining how a system will perform. For instance, in the results section there is a series of data from tests conducted with different CPU architectures and substantial differences can result.

Recommendation

Get the fastest CPUs possible and grow the size of the cluster as needed.

Clustered Configurations

Is the WebLogic Portal Server deployment configured to support clusters? Clusters can provide session protection and fail over via state replication when configured, in addition to horizontal load distribution. Customers who choose to implement clusters should see negligible degradation unless their application stores large amounts of data in the session and that session is replicated across the cluster.

If you are using a web server to forward requests to a WebLogic Server cluster, sometimes the bottleneck can be the web server. This can happen when using the supplied `HttpClusterServlet` and a proxy server, or one of the supported plug-ins. If the response time does not improve after adding servers to the cluster and the web server machine shows a high CPU utilization, consider clustering the web server or running the web server on more powerful hardware. The web server should be largely I/O bound (including disk utilization and network utilization) rather than CPU bound.

Recommendation

Based on capacity tests with tuned applications, WebLogic Portal is typically CPU-bound. When deciding how much hardware to buy for a production environment, the speed of the processor(s) should be the top priority.

In most cases, WebLogic Server clusters scale best when deployed with one WebLogic Server instance for every two CPUs. However, as with all capacity planning, you should test the actual deployment with your target portal applications to determine the optimal number and distribution of server instances.

Simulated Workload

When trying to determine the performance requirements of your system you will need to take into account the expected workload on the application. For example, a typical banking application experiences heavy traffic (a high number of concurrent sessions) during the “peak hours” of 9 AM and 5 PM. So when doing capacity estimates it is best to test with workloads that will closely mimic the anticipated workload. Additionally, some testing groups will include actual end user testing cycles to identify usage patterns.

Several workload factors can influence the overall performance of the system and depending on how these factors are tested, very different results will be produced. The first is the anticipated “think-time” for users of each application. Think-time is defined as the pause between requests by a user who is active on the system. For example, if a user clicks to see their bank account balance, they may not click again for 30 seconds, thus the think-time is 30 seconds. This think-time should be averaged across all users (because “expert” users will have shorter think-times and “novice” users will have much longer times). Think times will vary from application to application and system testing should reflect this. Decreasing the think-time will impact on the performance characteristics of the system. For example, if all of the users are added to the system at once, a “wave” effect will occur where the response times may be high during the initial requests then improve dramatically as users continue to navigate through the system. Adding users in a staggered fashion will prevent this from happening and provide more consistent performance from the system. Some randomization of the think-time will make the test more “real-world” and usually helps decrease the “wavy” behavior producing more consistent and accurate results.

Recommendation

When testing the system to determine capacity requirements, make sure that the workload of the simulated users accurately reflects what the system would experience in the production environment. Again, this is best accomplished through actual end user timings and modeling test from usage patterns. Pay close attention to excessive simulated workload that may produce inaccurate results.

Concurrent Sessions

Determine the maximum number of concurrent user sessions for your WebLogic Portal. To handle more users, you will need to have adequate CPU capacity and RAM for scalability. For most supported configurations 1GB of RAM is the minimum configuration and 2GB is recommended in production for each WebLogic Portal instance.

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. This can be done by using some fraction of the total number of users or better still, by measuring an existing system if one already exists. The number of user interactions per second with WebLogic Portal represents the total number of interactions that should be handled per second by a given Portal deployment.

The maximum number of transactions in a given period should also be considered to ensure spikes in demand can be handled. If the demand is close to the maximum capacity for the system, then additional hardware should be added to increase the overall system performance and capacity. For capacity information about concurrent users see [“Portal Framework Concurrent User Results”](#) on page 18.

Tuning WebLogic Server

WebLogic Server should be tuned using the available tuning guide.

Recommendation

For more information about tuning Server, see

- [WebLogic Server Performance and Tuning](#)
- [Top Tuning Recommendations for WebLogic Server](#)

Application Design

A well tested and deployed architecture can easily be crippled by poorly built or non-optimized user applications. The best course is to assume that each application developed for WebLogic Portal will add overhead and will thus not perform as well as benchmark applications. As a precaution, you should take into account these features of the application and add additional capacity to your system.

The size of the portal deployment (calculated by adding up the number of distinct books, pages, and portlets) may have a significant impact on the performance and capacity of the system. As

the portal size increases, the control tree grows along with it and very large ones can have a noticeable performance impact.

The use of multi-level menus negates many of the benefits of the Portal Control Tree Optimizations because the tree must be navigated in order to build the menu structure. This is fine with smaller portals, but for larger portals this will have a significant impact on the performance and scalability of the system and will thus require more hardware resources in the deployment.

Recommendation

Breaking large portals into several smaller Desktops is recommended to optimize the performance of the system. Additionally, profiling with either a “heavy-weight” profiler or a run-time “light-weight” profiler is strongly recommended to find non-optimized areas of code in your application. The use of multi-level menus is discouraged for large portals.

For more information about designing portals, see [Designing Portals for Optimal Performance](#).

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 cannot handle as many simultaneous connections.

You should 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 33-50% 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.

Recommendation

Implement SSL using hardware accelerators or disable SSL if it is not required by the application.

WebLogic Server Process Load

What is running on the machine in addition to a WebLogic Portal? The machine where a WebLogic Portal is running 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; however, this configuration is not recommended.

Consider how much of your WebLogic Portal machine's processing power is consumed by processes unrelated to WebLogic Portal. In the case in which the WebLogic Portal (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.

BEA recommends that the average CPU utilization on the WebLogic Portal server when executing benchmark tests be in the range of 85 to 95% as a cumulative statistic for that machine. For example, if the machine has multiple processors then the average for both processors should be between the above percentages. This allows the machine to operate at near peak capacity, but also allows for other system processes to run and not drive the CPU to 100%. During production additional CPU overhead should be given to the system to accommodate spikes in traffic so that SLAs around response times are maintained. A good rule of thumb for production should target no higher than 70% CPU utilization to ensure extra capacity is available for unanticipated events.

Additionally, if any third party applications, services, or processes are deployed in addition to WebLogic Portal, BEA recommends deploying those applications, services, or processes on separate hardware machines.

When dealing with a clustered WebLogic Portal deployment a load balancing solution must be considered. With load balancing in a cluster, the user sessions across the nodes should be about even. If the distribution is not even then that points to a problem with either the WebLogic Portal configuration or the load balancer configuration.

Recommendation

If a cluster of servers is required to meet the capacity demands of the system then a load balancer should be implemented to distribute load across the machines.

All third party applications and services should be off-loaded onto separate hardware.

Database Server Capacity

Is the database a bottleneck? Are there additional user storage requirements? Many installations find that their database server runs out of capacity much sooner than the WebLogic Portal does. You must plan for a database that is sufficiently robust to handle the application. Typically, a good application will require a database that is 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 a high CPU utilization for WebLogic Portal CPU. This is a good indication that your WebLogic Portal is spending much of its time idle and waiting for the database to return.

Some database vendors are beginning to provide capacity planning information for application servers. Frequently this is a response to the 3-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.

There are additional ways to prevent the database from being the bottleneck in the system and one of those ways is by implementing caching at the database layer. WebLogic Portal uses many different caches to avoid hitting the database. If during performance testing the database is determined to be a bottleneck then it might be useful to tune the WebLogic Portal caches to take some of the load off the database.

Recommendation

See the WebLogic Portal [Database Administration Guide: Performance Considerations](#) and [Sizing Considerations](#) documentation for sizing and other performance related considerations.

Review the [WebLogic Portal Cache Reference](#) for more information about database caches.

Network Load

Is the bandwidth sufficient? Network performance is affected when the supply of resources is unable to keep up with the demand. WebLogic Server requires a large enough bandwidth to handle all of the connections from clients it is required to handle. If you are handling only HTTP clients, expect a similar bandwidth requirement to a web server serving static pages.

In a cluster, by default, in-memory replication of session information shares the same network as the HTTP clients. An alternative to the standard network topology would be to change the physical network with a different channel for internal cluster communication and a second channel for external traffic. See [Configuring Network Resources](#) for details. Although the WebLogic Portal framework does not create large amounts of session data it is possible for a custom application to add significant overhead in this area. Additionally, a high load of concurrent users with frequent requests will also lead to network saturation. Consider whether your application and business needs require the replication of session information. Finally, the combination of lots of concurrent users and frequent requests to the server should be estimated to determine if the network can handle the anticipated load.

To determine if you have enough bandwidth in a given deployment, you should look at the network tools provided by your network operating system vendor. There are plenty of free and commercial tools available including build-in applications for Windows and Solaris to help measure this. Additionally, most hardware load balancing solutions provide network statistics. If only one load balancer is used, it too may become a bottleneck on the system if the load is very high.

Recommendation

BEA recommends running a gigabit LAN and implementing one or more server load balancers to optimize network traffic.

Selecting Your JVM

What JVM will be used? What parameters will be used? How much heap is required to get the best performance out of the application? Different applications may perform better on one JVM or another. WebLogic Portal supports BEA's JRockit and Sun's HotSpot JVMs. In general, BEA's JRockit JVM performed better during "Benchmark" tests on Intel processors with Linux as the OS, however HotSpot performed slightly better as the cluster size increased during "Concurrent User" tests.

The JVM parameters can have a dramatic impact on the performance of the system. Please see the [BEA JRockit Reference Manual](#) for a list of all of the parameters and where they may be used.

The size of the heap will also impact the performance of the system. Larger applications may need larger heap sizes. Additionally, a high number of concurrent users will require a larger heap size to prevent the system from running out of memory.

Recommendation

In all cases with JRockit it is recommended that `-Xgc:parallel` be used and with HotSpot `-XX:MaxPermSize` with a minimum of 128m be used. Depending on your application the memory requirements may be quite high. In all cases a set of benchmark tests should be run with the different settings to determine what is best for your application.

Performance Results

There are two types of performance test results in the following sections; one test to assess throughput and another to determine the maximum number of concurrent users supported by the

system. The differences between these tests are numerous and thus comparing a data-point from one type of test to another is not recommended.

The first set of data is referred to as “Benchmark Results.” This set of tests were run to determine a baseline for the throughput of system measured in pages returned per second. The goal of these tests is to determine the maximum throughput of the system in various configurations where the portal size, portlet type, and JVM are varied.

The second set of data is referred to as “Concurrent User Results” because it is more closely related to the sort of tests run in production-like systems. The goal of this type of test is to determine the maximum number of concurrent users (actively clicking through the Portal) for a given response time (often referred to as a Service Level Agreement.)

Each test is driven by a LoadRunner script that allows each user to log-in once and then click through the pages (for all but the Very Small portal, there were 50 page/book clicks) and then repeat at the first page when the last page is reached. The very small portal has 8 pages, so there were 8 clicks. This continues until the test duration is complete.

Test Applications

The following sections discuss the applications used in the performance testing. The applications include:

- [Portal Framework Application](#)
- [WSRP Application](#)
- [Content Management Application](#)

Portal Framework Application

The portal framework test application is deployed to the cluster as an EAR that contains `.portal` and `.portlet` files. Form-based authentication is used for each Portal so that a user is registered. The portals themselves vary in size and portlet type. Each portal tested includes portlets of only one type, including JSP, page flow, and JSR168. The portlets used are considered simple portlets such as “Hello World”-type portlets. Tree optimization is enabled for all of the portals.

Entitlements and user customizations are disabled. Session replication using the flag “`replicated_if_clustered`” is configured for all tests. Because all of the users are required to log-in and then did not log-out, a session was maintained for each user for the duration of the test.

The portal sizes vary with the following parameters:

Table 2 - Tested WebLogic Portal Sizes

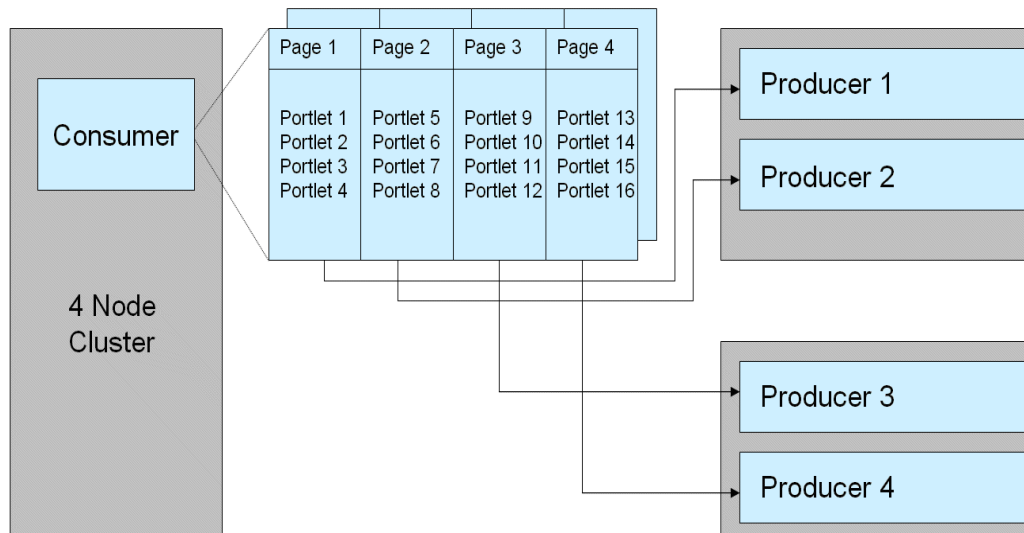
Portal Size	Number of Books	Number of Pages	Number of Portlets
Very Small	1	8	64
Small	5	50	500
Very Large	40	400	4000

With the exception of the Very Small portal (which has 8 portlets per page) each portal has 10 portlets per page.

WSRP Application

The WSRP test application is deployed to federated clusters as an EAR that contains `.portal` and `.portlet` files. Form-based authentication is used for each Portal so that a user is registered. Each portal has one book containing eight pages; each page has from 1 to 4 remote portlets. Multiple instances of the same remote portlet are used for each page (i.e., page 1, portlet 1 shares the same remote portlet definition as portlets 2, 3, and 4 on that same page). Each remote portlet accesses a portal Producer located on a remote machine on the same network subnet. The portlets located on pages 1 and 5 are configured to point at the same Producer. The same pattern of configuration was applied for portlets on pages 2 and 6, 3 and 7, and 4 and 8. Graphically this is represented in Figure 1.

Figure 1 WSRP Cluster Configuration



All portlets are page flow portlets. The remote portals are designed to provide approximately 40KB of HTML content to the consumer portlets. Tree optimization is enabled for all portals, while entitlements and user customizations are disabled. Session replication using the flag “replicated_if_clustered” is turned on for all tests. Because all of the users are required to log-in and then do not log-out, a session is maintained for each user for the duration of the test. The WSRP SSO feature using the WSRP SAML identity asserter is not used to propagate user identity between the Consumer and Producer.

The tests application varies over various portlet sizes and features as follows:

- Eight, 16, or 32 total portlets on eight portal pages
- Caching on and off

When caching is enabled, the cache TTL is set to a time period longer than the duration of the test.

The F5 Networks Big-IP load balancer is used to balance load on the consumer cluster only. The producers are not clustered or load balanced in any way.

Content Management Application

The content management test application is deployed to a cluster as an EAR that contains JSP files which hit the content APIs directly. These JSPs are designed to test the performance of node creation, node access, paginating over nodes in a result set, the security overhead of reading a node, and the concurrent reading and writing of nodes in a BEA content repository. There are 2 types of node content that are used, simple and complex. Simple content is comprised of five string properties and one binary property. Complex content has two string, two calendar, one boolean, one long, one binary and one nested properties. The nested property is itself made up of three string, two long and two calendar properties. Repository management is disabled as are repository events. The repository is not read only and search is disabled. These tests vary over features as follows:

- Number of users creating nodes
- Number of users reading nodes
- Binary (0 or 10KB) size added to the content
- Number of total nodes in the repository
- Simple and Complex node types
- Reading the node by node Id or node name
- Number of results in a paginated list
- Number of entitled nodes in the database
- Administrator/Non-admin user

In general the nodes are created one level below the repository root. The exception to this is the nested node in the complex type, this is created as a grandchild of the root.

HP Linux Hardware and Server Configurations

The HP Linux tests varied over several different cluster configurations in which there were two or four physical machines in a cluster. Each physical machine had Portal running in a single JVM. In order to keep hardware acquisition cost a constant from previous releases, the 10.2 performance tests were run on upgraded hardware. The upgraded hardware was purchased at the same price point as the previous hardware. See section [“Comparing 10.2 Results Against Previous Releases” on page 16](#) for details on comparing 10.2 performance numbers with previous releases.

- Administration and Managed Servers: HP ProLiant DL360 G5 – Dual 2.66 GHz Xeon, 4 GB RAM, 15K RPM SCSI Disks, HyperThreading enabled, RedHat Enterprise Linux AS 5.0, Gigabit NIC
- Database Server: HP ProLiant DL380 G4 – Dual 3.4 GHz Xeon, 4 GB RAM, 15K RPM SCSI Disks, HyperThreading enabled, Red Hat Enterprise Linux AS 4 Update 4, Oracle 10R2, Gigabit NIC
- Load Balancer: F5 Networks Big-IP 3400
- LoadRunner Controller: HP ProLiant DL360 G4 – Dual 3.6 GHz Xeon, 3.5 GB RAM, 15K RPM SCSI Disks, HyperThreading enabled, Windows 2003 Server Enterprise Edition SP1, LoadRunner 7.8, Gigabit NIC
- BEA JRockit JVM with `-Xms2048m -Xmx2048m -Xgc:parallel` setting.
- Sun HotSpot JVM with `-server -Xms2048m -Xmx2048m -XX:MaxPermSize=128m` setting.

Sun Solaris Hardware and Server Configurations

The Sun Solaris tests used four and eight CPU configurations in which there were one and two physical machines. Each machine had two running managed servers, which translates into two portals and two JVMs on each physical machine, for a total of two and four managed servers in the cluster. Each server has four CPUs and the data is presented in the table by CPU count.

- Administration Server: Sun Fire v240, 2 x 1.02GHz, 4GB RAM, 10K RPM SCSI Disks, Sun Solaris 10
- Managed Servers: Sun Fire v440, 4 x 1.02GHz, 8GB RAM, 10K RPM SCSI Disks, Sun Solaris 10, Gigabit NIC
- Database Server: HP ProLiant DL380 G4 – Dual 3.4 GHz Xeon, 4 GB RAM, 15K PRM SCSI Disks, HyperThreading enabled, Windows 2003 Server Enterprise Edition SP1, Oracle 9.2.0.6, Gigabit NIC
- Load Balancer: F5 Networks Big-IP 1500
- LoadRunner Controller: HP ProLiant DL320 G3 – 3.6 GHz Pentium 4, 2 GB RAM, 15K RPM SCSI Disk, HyperThreading enabled, Windows 2003 Server Enterprise Edition SP1, LoadRunner 7.8, Gigabit NIC
- JVM: Hotspot with `-server -Xms2048m -Xmx2048m -XX:MaxPermSize=128m` setting.

Comparing 10.2 Results Against Previous Releases

Due to hardware upgrades in this release, when compared against WLP 10.0, you can expect to see anywhere from even to 4x performance increases on tests performed on HP Linux. This is accounted for by the increase from dual-core, 2 chip G4's to quad-core, 2 chip G5's, as well as an improved front-side bus. The configurations (JSP, JSR168) that are CPU intensive see the most benefit from this hardware upgrade. Page Flows, which are constrained by the physical memory on the machine, see less benefits.

Portal Framework Benchmark Results

“Benchmark” tests are designed to show the maximum throughput of the system under different conditions. We varied over the type of portlets in the portal and the size of the portal as well as the JVM. For each configuration the goal is to saturate the server to achieve maximum throughput. The WebLogic Portal servers reached between 85 and 95 percent CPU utilization which is the optimal range for maximum throughput.

To achieve maximum throughput, zero seconds of “think-time” was used, which is to say that the time between a response from the server and the subsequent request was zero seconds. With this type of workload it is very easy to saturate the server and achieve maximum throughput in a short period of time with relatively few users.

For the Benchmark tests a ratio of 10 virtual users (VUsers in LoadRunner) were used per CPU. The Benchmarks were run on two hardware configurations, HP Linux and Sun Solaris. Since all of the Linux machines tested were configured with two CPUs, for each node in the WebLogic Portal cluster, 20 virtual users were used per machine. The Sun Solaris machines tested had 4 CPUs and thus 40 virtual users were used per machine. These users were “ramped-up” (added to the system) over 25 minutes followed by a steady-state (where no additional users were added but the existing users continued to access the system) that lasted an additional 10 minutes.

Note: The baseline numbers produced by the Benchmarks used in this study should not be used to compare WebLogic Portal with other portals or hardware running similar Benchmarks. The Benchmark methodology and tuning used in this study are unique.

This section includes results from the following configurations:

- [HP Linux Hardware and Server Configurations](#)
- [Sun Solaris Hardware and Server Configurations](#)

HP Linux Results

The servers were set to auto-tune which has been a new feature since WebLogic Server 9.0. The JDBC connection pools were set to start at five connections with the ability to grow to 25. These tests were run with zero seconds of think time so that the servers would become saturated quickly.

Table 3 - Linux Benchmarks - Throughput in Pages Per Second

Portal Type	Portal Size	JVM	4 CPU's	8 CPU's
JSP	Very Small	Jrocket	1097	1589
		Hotspot	1182	1742
	Small	Jrocket	636	1224
		Hotspot	880	1284
	Very Large	Jrocket	773	1123
		Hotspot	856	1258
JSR 168	Very Small	Jrocket	1069	2331
		Hotspot	1645	2510
	Small	Jrocket	919	1589
		Hotspot	1066	1424
	Very Large	Jrocket	741	1228
		Hotspot	841	1323
Page Flow	Very Small	Jrocket	1004	1097
		Hotspot	952	1262
	Small	Jrocket	399	833
		Hotspot	455	865
	Very Large	Jrocket	373	690
		Hotspot	436	751

Sun Solaris Results

The servers were set to auto-tune which has been a new feature since WebLogic Server 9.0. The JDBC connection pools were set to start at five connections with the ability to grow to 25. These tests were run with zero seconds of think time so that the servers would become saturated quickly.

Table 4 - Solaris Benchmarks - Throughput in Pages Per Second

Portal Type	Portal Size	4 CPU's	8 CPU's
JSP	Very Small	167	323
	Small	125	245
	Very Large	107	215
JSR 168	Very Small	114	230
	Small	84	169
	Very Large	77	152
Page Flow	Very Small	112	214
	Small	56	106
	Very Large	54	101

Portal Framework Concurrent User Results

This set of performance test results are also known as “Capacity Planning” results because they are best suited for determining what the overall capacity of the system is by measuring how many concurrent users can run on a given set of hardware. These tests are designed to mimic real-world user loads and thus show a more accurate representation of the system than the standard “Benchmark” tests.

Based on feedback from our customers the most common SLAs are 2 second and 5 second response times. Our goal was to determine how many users WebLogic Portal could support across various configurations with those SLAs. If your given SLA is higher, then the number of

supported users will also be higher, although estimating that number would be difficult to do without actually running additional tests.

For Capacity Planning tests the think-time is also meant to mimic real-world production systems being accessed by so-called “expert users.” This should be considered a very high workload for the system and in many other configurations the request times by the end users will not be “expert” like. The think-time for these tests was randomized at 5 seconds +/- 25% (between 3.75 and 6.25 seconds.) Whereas a non-export like system might state that the think-time is closer to 30 seconds averaged across all users. The think-time for the system has a dramatic impact on the overall capacity of the Portal. A higher think-time will allow many more users on the system. You can see in the “Benchmark” configuration there was only 10 users per CPU required to saturate the system, but with think-time it could take hundreds if not thousands of users per CPU to have the same impact.

The workload for Capacity Planning tests is vastly different than that of the above “Benchmark” tests. Because the number of users required to meet the minimum SLAs is much higher (due to think-time) the duration of the tests must be extended. The number of users for each configuration was ramped-up over the course of one hour and for each configuration a different number of users was added at a constant rate every 30 seconds. We chose one hour because the system responded better and thus supported more users than with shorter ramp-up schedules. A high number of users was added to the system until they were all running at roughly the one hour mark.

This test established how many concurrent users the test portal could support with a given response time. Goal response times of two seconds and five seconds were used. The number of concurrent users listed in the table represent the maximum number of running concurrent users under 2 or 5 seconds. This test used the HP Linux configuration, see [“HP Linux Hardware and Server Configurations” on page 14](#). Each server has two CPUs and the data is presented in the table by CPU count.

This section reports the following results:

Table 5 - Concurrent User Results - Number of Users Per Service Level Agreement (2 seconds or 5 seconds)

Portal Type	Portal Size	JVM	2 Second SLA		5 Second SLA	
			4 CPU's	8 CPU's	4 CPU's	8 CPU's
JSP	Very Small	Jrocket	10474	17906	12869	23048
		Hotspot	7080	14959	9600	25230
	Small	Jrocket	6249	11576	7299	14145
		Hotspot	4440	8040	7530	13160
	Very Large	Jrocket	5474	10240	6493	12691
		Hotspot	3700	8519	5149	13555
JSR168	Very Small	Jrocket	8001	15811	10647	20847
		Hotspot	6124	14110	9064	20163
	Small	Jrocket	5032	10393	6072	12716
		Hotspot	3058	9160	5040	13040
	Very Large	Jrocket	4939	9441	5959	11610
		Hotspot	3080	8155	4860	11515
Page Flow	Very Small	Jrocket	3220	4707	3360	5142
		Hotspot	1918	4296	2938	6048
	Small	Jrocket	765	1128	859	1262
		Hotspot	532	1420	556	1622
	Very Large	Jrocket	692	1455	747	1718
		Hotspot	452	1400	512	1630

NOTE RE: PageFlow Portlet Results

PageFlow portlets can have additional memory requirements which may affect performance. This is documented in more detail in the [Tuning for PageFlow Portlets](#) section of [The Performance Tuning Guide](#) and in the [Portal Development Guide](#). The numbers in this section are subject to these limitations.

WSRP Benchmark Results

WSRP Benchmark tests are designed to show the maximum throughput of a federated system under different conditions. The number of portlets in the portal, caching, as well as the JVM are all varied over during these tests.

For each configuration the goal is to saturate the infrastructure to achieve maximum throughput. The WebLogic Portal servers reach between 85 and 95 percent CPU utilization which is the optimal range for maximum throughput. The number of producers is fixed for these tests at two. As additional managed servers are added to the consumer cluster, the producers are not able to keep in step, and eventually become the system bottleneck. The CPUs of the producer clusters increase beyond the optimal range and affect the overall performance of the consumer cluster. In the case of these tests, the optimal configuration is two producer machines to three managed servers in the consumer cluster.

To achieve maximum throughput, zero seconds of “think-time” is used, which is to say that the time between a response from the server and the subsequent request is zero seconds. With this type of workload it is very easy to saturate the server and achieve maximum throughput in a short period of time with relatively few users. For these tests a ratio of 50 virtual users (VUsers in LoadRunner) are used per CPU to produce maximum load.

These users are “ramped-up” (added to the system) over 40 minutes followed by a steady-state (where no additional users were added but the existing users continued to access the system) that lasts an additional 20 minutes.

There are a myriad of configuration parameters on various JVMs. Each of these can have a specific effect on the overall performance of an application. For JRockit JVM and Sun’s HotSpot JVM we run with the following JVM parameters:

- JRockit: -Xms2048m -Xmx2048m -Xgc:parallel
- HotSpot: -Xms2048m -Xmx2048m -XX:MaxPermSize=128m

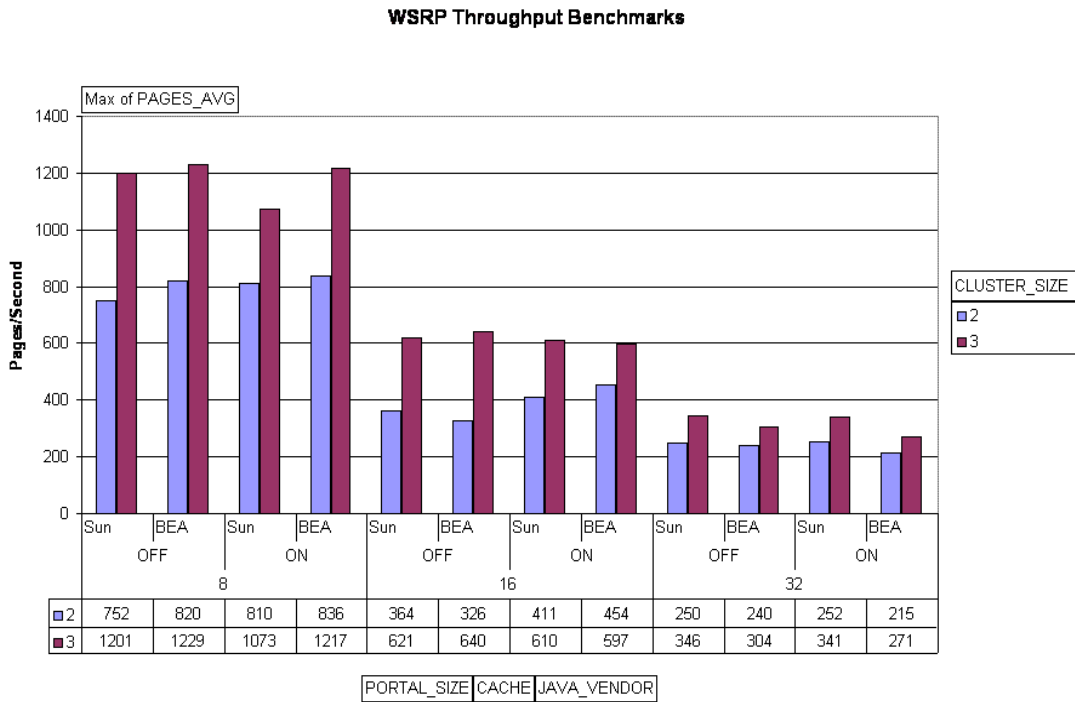
For information on performance tuning WSRP application see the [“Tuning for WSRP”](#) section of the Performance Tuning Guide.

Table 6 WSRP - Throughput in Pages Per Second

Number of Portlets	Caching	JVM	4 CPU's	6 CPU's
8	OFF	Jrocket	820	1229
		Hotspot	752	1201
	ON	Jrocket	836	1217
		Hotspot	810	1189
16	OFF	Jrocket	326	640
		Hotspot	364	621
	ON	Jrocket	454	597
		Hotspot	411	610
32	OFF	Jrocket	240	304
		Hotspot	250	346
	ON	Jrocket	215	271
		Hotspot	252	341

This data is represented graphically in Figure 2. The measured throughput data is represented by the vertical bars, one for each unique configuration. Across the X-axis, the data is partitioned according to the configuration that was run. The lowest set of numbers (8, 16, 32) corresponds to the left-most block "PORTAL_SIZE" - this is the number of portlets in the test configuration. The middle variable in the configuration is related to whether or not caching is on. This corresponds to the second column on the chart "CACHE". The upper variable on the graph corresponds to the "JAVA_VENDOR" column on the chart, and is set to either BEA (Jrocket), or Sun (Hotspot). The data is further divided into 2 bars representing a "CLUSTER_SIZE" of either 2 Node (4 CPU) or 3 Node (6 CPU). At the very bottom of the graph the raw numbers are given for each configuration.

Figure 2 Graphical View of WSRP Benchmark Data



Content Management Benchmark Results

Content Management Benchmark tests are designed to show the effects that different types of load have on the Content Management API and infrastructure. These are “Benchmark” style tests that seek to load the system up to maximum capacity and test under various configurations. Where as other test results in this document use a throughput calculation to determine performance statistics, the content tests use response time as a measure. The response time is the amount of time that it takes for a single request from the client to be processed by the server and the response handed back to the client. Other tests use rendered HTML as part of the performance measure. Since the content tests are primarily for driving native APIs (rather than rendering

HTML) the measure isn't on how much HTML can be rendered, but rather, how fast the API responds.

Content Management Benchmark tests run on a single machine in a standalone, non-clustered configuration. While the database schema for content can be located in any location, for these tests, the content schema was co-located with the WebLogic Portal schema.

These tests do not follow the LoadRunner ramp-up/duration model that the rest of the tests in this document follow. Instead, these tests have all users running concurrently from the start of the test, and they repeat API requests for a fixed number of iterations. These iterations roughly correspond to the number of nodes in the database. The duration of a test is controlled by periodically flushing all the caches when all content from the database has been viewed once, and then repeating the cycle. In all cases, zero "think time" seconds is used between requests. This creates the maximum amount of load on the server in the shortest period of time.

All Content tests were run with the JRockit JVM.

For information on improving Content Management performance, see the ["Tuning for Content Management" section of the Performance Tuning Guide](#).

Content Creation Results

These tests measure the response time required to create a node within the content management system, via the content API. These tests include numbers for importing binary data into a binary property. This binary data is located in a file inside the webapp. The first time it is requested, it is loaded into a binary object in memory and then read out of memory for each successive request.

Two different types of content were used, simple and complex. These are defined in the section: ["Content Management Application" on page 14](#).

Table 7 - Content Creation Results

Nodes	Users	Binary Size (KB)	Average Response Time (Milliseconds)	
			Simple	Complex
10000	1	0	13	43
		10	42	108
	10	0	44	381
		10	254	766
100000	1	0	13	52
		10	48	119
	10	0	74	629
		10	346	987

Content Read Results

These tests are designed to measure how much time it takes to retrieve a random node from the content repository. Each test uses one of the following methods to retrieve random nodes from the content repository:

- By node “ID” (calling `INodeManager.getNodeByUUID(ContentContext, ID)`).
- By node “PATH” (calling `INodeManager.getNode(ContentContext, String)`).

Each node is retrieved only once to make sure that caching is defeated.

Two different types of content were used, simple and complex. These are defined in the section: [“Content Management Application” on page 14.](#)

Table 8 - Content Read Results

Users	Read Method	Average Response Time (Milliseconds)	
		Simple	Complex
10	ID	5	5
	PATH	121	116
100	ID	25	26
	PATH	1179	1180

Content Pagination Results

These tests are used to determine how different pagination methods affect performance. The tests vary over different levels of concurrency and pagination batch size (the number of content nodes returned per page). These tests replicate how a “real world” application might read data from the repository and then display it with pagination. These tests compare the following pagination methods available in the Content API:

- IPagedList (referred to as “LIST”).
- ICMPagedResult (referred to as “RESULT”).

Two different types of content were used, simple and complex. These are defined in the section: [“Content Management Application” on page 14.](#)

Table 9 Content Pagination Results

Nodes	Users	Pagination Method	Pagination Batch Size	Average Response Time (Milliseconds)	
				Simple	Complex
1000	1	LIST	10	397	528
			100	361	481
		RESULT	10	197	259
			100	203	267
	10	LIST	10	701	889
			100	1343	2182
		RESULT	10	244	310
			100	795	1013
5000	1	LIST	10	1991	266
			100	1958	2621
		RESULT	10	1003	1332
			100	1021	1345
	10	LIST	10	4072	7595
			100	3350	6111
		RESULT	10	2064	3585
			100	2750	4570

Content Security Results

Content Security tests are designed to measure overhead implicit in the security mechanisms in the Content Management System. These tests are run with a user who does not have administrator permissions and who views paginated content. Depending on the test configuration, different overall percentages of content repository nodes will be visible to the user. This is managed via node entitlement, and varied over at 50% and 100% node entitlement. The [“Content Pagination Results” on page 26](#) demonstrate that increasing the pagination batch size will lessen the average response time slightly. A ten-fold increase in batch size translates to a small reduction in the average response time. These tests take that into account, and fix the pagination batch size at 100.

Two different types of content were used, simple and complex. These are defined in the section: [“Content Management Application” on page 14.](#)

Table 10 Content Security Results

Nodes	% Entitled	Pagination Method	Average Response Time (Milliseconds)	
			Simple	Complex
1000	50	LIST	304	390
		RESULT	303	389
	100	LIST	395	514
		RESULT	228	296
5000	50	LIST	2411	3050
		RESULT	1426	1767
	100	LIST	2496	3164
		RESULT	1311	1657

Content Concurrent Read/Write Results

Content Concurrent tests measure the impact of multiple users concurrently creating and reading data from the content repository. These tests are modelled on a realistic use case of the Content Management System. The tests are run against a repository that already contains 100,000 nodes. The tests then add an additional 5000 or 10000 nodes while simultaneously reading nodes out of the database. The users that are performing the read operations do so by calling into the content API method: `INodeManager.getNodeByUUID(ContentContext, ID)`.

Two different types of content were used, simple and complex. These are defined in the section: [“Content Management Application” on page 14.](#)

Table 11 Content Concurrent Read/Write Results

Nodes	Create Users	Read Users	Average Create Response Time (Milliseconds)		Average Read Response Time (Milliseconds)	
			Simple	Complex	Simple	Complex
25000	5	10	122	356	37	65
		15	154	394	51	91
50000	10	10	175	507	51	87
		15	203	505	64	116

Other Resources

Remember that WebLogic Portal uses many components from WebLogic Platform. See the following documentation for more information about tuning WebLogic Portal.

- [Designing Portals for Optimal Performance](#)
- [WebLogic Server Performance and Tuning Guide](#)
- [WebLogic Server Capacity Planning Guide](#)
- [JRockit Diagnostics Guide](#)

Capacity Planning for WebLogic Portal

- BEA's [dev2dev](#) Website