BEA WebLogic Portal™

Performance Tuning Guide

Version 10.2
Document Revised: April 2008

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

4. Tuning Your Portal Web Application

A. Performance Tuning Checklistst

CHAPTER 1

General Performance Tuning Guidelines

WebLogic Portal application performance is affected by many factors. This chapter discusses a few of the initial aspects that can affect performance and provides links to documentation resources that can assist you.

Understanding Performance Tuning and BEA WebLogic Portal

Performance tuning is a process which spans development, staging and deployment. During all phases, performance should be monitored and appropriate adjustments made. If you are new to performance testing, see Approaches to Performance Testing on BEA's dev2dev website.

BEA recommends that you establish an environment where you can performance test the installation for the following reasons:

- Testing under realistic load may uncover bugs not seen during development or QA.

- Testing your prototype under load will help you validate design decisions early in the development cycle that may significantly alter the performance of your application.
- Any configuration change can dramatically affect application performance (hardware, database, clustering environment, application tuning parameters, and so on). Load testing your application whenever design changes are made provides a way to narrow down performance problems to a particular area.
- Testing early and often increases the likelihood that your site implementation and deployment will perform well.

The recommended approach for performance testing is to start with the simplest aspect of the installation and then move into areas of increased complexity. If you observe slow behavior in any portion of this testing process, you should begin a more thorough investigation into its causes.

General Architecture

First, perform the following steps to identify performance issues with your network, database, or other software that is independent of WebLogic Portal.

1. Test your database (independent of any web components) to determine how well your schema and SQL work. Note any areas where the schema or SQL may not be optimized for performance. See the WebLogic Portal Database Administration Guide for more information about proper setup and performance tuning.
2. Test your network for sufficient bandwidth, and check that the TCP/IP parameters on the server's operating system can sufficiently handle the application load you expect. It is possible that the network is the slowest aspect of your deployment. Ensure that your IP Multicast for cluster deployment is configured correctly. See the WebLogic Server Troubleshooting Multicast Configuration Guide for more information about multicast configuration.
3. Test your web server, ensuring that it has sufficient capacity to serve static HTML pages when many concurrent threads are running.
4. Ensure that you have enough resources available to meet application requirements. Most large applications are clustered, but keep in mind that a clustered environment requires resources to perform load-balancing tasks. For more information, see Understanding Cluster Configuration and Application Deployment.
5. Test your servlet engine by running a load test against a trivial servlet such as a HelloWorld servlet. If this simple servlet does not perform and scale horizontally (meaning that as you add

Java Virtual Machines, performance increases accordingly), the performance problems you encounter may be related to an infrastructure or resource issue.

WebLogic Portal

After performing the steps in the previous section, General Architecture, perform the following steps to identify performance issues with WebLogic Portal:

1. Verify that your BEA WebLogic Portal database configuration is optimal. WebLogic Portal makes extensive use of the database. Check that your connection pool is large enough and verify that your database handles connection failures in an efficient manner. For example, the size of the JDBC connection pool should be set to handle the maximum number of concurrent users as possible, and it should be set on server startup rather than growing as connections are needed. This will increase the server startup time but will decrease the overhead creating those connections under server load. See the section "Performance Considerations" in the *WebLogic Portal Database Administration Guide* for more information.

2. Verify that each portlet is optimized for speed as follows:

   – If a portlet uses forms that update the data within the portlet this will cause the entire portal to refresh its data, which can be very time consuming. Therefore, portlets that have this behavior should have asynchronous rendering enabled via AJAX or iFrames so that the overall rendering of the portal is not affected. AJAX is supported at the portal desktop and individual portlet levels.

   Note: Individual portlets with asynchronous rendering methods have limitations such as not supporting inter portlet communication.

   – Place items that require heavy processing in an edit page or a maximized URL. If you do not, the portal must wait for the portlet to process, and this considerably slows down the eventual rendering of the portal. Process intensive portlets may benefit from parallel portlet rendering (also known as pre-render and render forking.) For more information about this see "Optimizing Portlet Performance" in the *Portlet Development Guide*.

   – Enable caching on portlets that do not rely on dynamic data.

3. Test your application's components, starting from the data access layer. Then proceed toward the GUI one step at a time. Pay attention to performance and scalability differences at each component and between each layer of your application. Finally, do end-to-end testing from a browser-based load-testing tool.

4. Test the behavior and performance of your application under simulated, real-world conditions. (Many tools are available to help you do this.) Be sure to use both anonymous and logged-in users simultaneously.

Tuning Your WebLogic Server

Because WebLogic Portal runs on WebLogic Server, factors impacting the performance of WebLogic Server will also impact the performance of WebLogic Portal.

For more information about tuning WebLogic Server, see the WebLogic Server Performance and Tuning Guide.

Table 1-1 lists the top ten tuning recommendations for WebLogic Server.

Table 1-1 Top Ten Tuning Recommendations

Tuning Question	For Information, See:
How big should the JDBC connection pool be?	Tune Pool Sizes
How to use JDBC caches?	Use the Prepared Statement Cache
What optimizations are there for transactional database applications?	Use Logging Last Resource Optimization
How many connections should WebLogic Server accept?	Tune Connection Backlog Buffering
What is the optimal size of the WebLogic Server network layer?	Tune the Chunk Size
What type of Entity Bean cache should be used?	Use Optimistic or Read-only Concurrency
How to avoid serialization when one EJB calls another?	Use Local Interfaces
How to load related beans using a single SQL statement?	Use eager-relationship-caching
How to tune session persistence?	Tune HTTP Sessions
What is the optimal JMS configuration?	Tune Messaging Applications

Tuning Your JVM

Your Java Virtual Machine is key to running your Portal efficiently. For more information about tuning WebLogic JRockit, see the *Diagnostics Guide* in the JRockit documentation.

Recommendations

When using JRockit, there are many different flags available. Depending on the application and the SLA, different parameters and garbage collection flags should be used. It is strongly recommended that before changing any parameters the application should be baselined so that the performance differences between subsequent tests can be measured. When using Sun Hotspot, adjust the `-XX:MaxPermSize` to be a minimum of 128MB.

Tuning Your Database

Keeping your database tuned is an important part of using WebLogic Portal. Portal uses the database to store content, rules, portal framework objects (streaming desktops, books, pages, and portlets), customizations, and user profile data.

Best practices for production deployment will vary between database vendors. See the database vendor documentation for these best practices. For WebLogic Portal specific tuning recommendations see, the WebLogic Portal Database Administration Guide.

Tuning Your Operating System

Tune your operating system according to your operating system documentation. BEA certifies WebLogic Platform on multiple operating systems, see the WebLogic Platform 10.2 Supported Configurations page for more details.

Following the WebLogic Server Performance and Tuning Guide is strongly recommended.

Other Resources

Remember that WebLogic Portal uses many components from WebLogic Server. See the following documentation for more information:

- WebLogic Server Performance and Tuning Guide
- WebLogic Portal Capacity Planning Guide
- dev2dev

CHAPTER 2

Tuning Your Portal Domain

Key aspects of portal performance are managed at the domain level. These include:

- Tuning Your Domain Configuration
- Removing Debugging Tools from Your Domain
- Tuning Log Levels

Tuning Your Domain Configuration

Optimally, when you deploy, you need to create a new domain that is configured for your production environment. However, if you have deployed a development domain and want to use it for production, you must change your domain environment settings to optimize performance.

Note: It is not recommended to use a development domain for production, see *Creating WebLogic Domains Using the Configuration Wizard* for more information.

The domain settings are managed by the `setDomainEnv.cmd` (or `setDomainEnv.sh`) script which is found in your domain directory. By default, the script is found in: *`WebLogic_Home`*`/user_projects/`*`domain_name`*`/bin/setDomainEnv.cmd/sh`.

To edit this file, open it in a text editor.

Table 2-1 lists the start script settings and their appropriate values for a production domain. Remember if you are using a domain that was created for production mode, you do not need to modify the configuration.

Table 2-1 setDomainEnv Settings

Flag Name	Production Mode Setting	Notes
`DOMAIN_PRODUCTION_MODE`	`true`	• Indicates whether you are in a production mode or a development mode. Default is false for domains created in development mode and true for domains created in production mode.
`iterativeDevFlag`	`false`	• Checks for updated files and if found, rebuilds and redeploys the application. Disable this option to prevent checking for changed WebLogic Workshop files. Default is true for domains created in development mode and false for domains created in production mode.
`debugFlag`	`false`	• Used in start scripts to set debugging options and indicate if the WebLogic Workshop Debugger should be started. When switched to false, you save the resource overhead used for debugging. • Default is `debugFlag=true` for domains created in development mode and `debugFlag=false` for domains created in production mode.
`testConsoleFlag`	`false`	• Enables the JWS test view. • Verify by checking the log for: `wlw. testConsole = false.` • Default is true for domains created in development mode and false for domains created in production mode.

Table 2-1 setDomainEnv Settings (Continued)

`logErrorsToConsoleFlag`	`false`	• Controls logging functionality. • Verify by checking the log for: `wlw.logErrorsToConsole = false` • Saves you additional logging. The trade-off is that you may see exceptions more easily when this is set to true (without checking the log). • Default is true for domains created in development mode and false for domains created in production mode.
`verboseLoggingFlag`	`false`	• If true, override the default `LOG4J_CONFIG_FILE` (`workshopLogCfg.xml`) with `workshopLogCfgVerbose.xml`. • Priority value in the default file is `warn`; in the verbose version it is `debug`. • Verify by checking the log for: `log4j.configuration = workshopLogCfg.xml` instead of `workshopLogCfgVerbose.xml` • You can also start in verbose mode using `startWebLogic.cmd verbose`. • Saves you debugging overhead. • Default is `false` for both domains created in development mode and in production mode.
`pointbaseFlag=`	`false`	• Indicates whether Pointbase should be started. • Verify by checking for a running Pointbase process. • Saves you the resource overhead of starting Pointbase when it is not needed. • Default is `true` for domains created with Pointbase as the database.

Removing Debugging Tools from Your Domain

When deploying a domain, you should remove the `debug.properties` file from the domain directory. Although this file is helpful during development, debugging should not be done in production environments.

Tuning Log Levels

WebLogic Server has several logging features available. When using the WebLogic logging infrastructure, make sure that the server logs at an appropriate level and to the correct location. For example, a production system logging at DEBUG or TRACE levels can produce gigabytes of log data fairly quickly when writing to a log file. A production system should have logging set to the INFO level or higher. This can be done from the command line, from MBeans, or from the console. See the WebLogic Server document Configuring Log Files and Filtering Log Messages for more detailed information on WebLogic Server logging.

Additionally, WebLogic server internally processes all log messages before writing these messages to the logging infrastructure. In a production system where the logging level has been set to INFO or NOTICE, having the server process all DEBUG messages, for example, can add significant overhead. It is a good idea to match the internal WebLogic Server log processing level to the logging framework level. Do this by specifying the `-Dweblogic.log.LoggerSeverity` flag to the server at startup.

CHAPTER 3

Tuning Your Portal Application

Key aspects of portal performance are managed at the portal application level. These include:

- Managing Caches
- Disabling Unused Services
- Tuning for Campaigns
- Tuning for Entitlements
- Tuning for Content Management
- Tuning for PageFlow Portlets
- Tuning for WSRP
- Tuning for Delegated Administration
- Using Work Managers

Managing Caches

WebLogic Portal provides a single framework for configuring, accessing, monitoring, and maintaining caches. If configured properly, the caches can vastly reduce the time needed to retrieve frequently used data. Keep in mind that caches are read-only and cluster-aware.

Many WebLogic Portal services use preconfigured caches that you can tune to meet your performance needs. Some services use internally configured caches that you cannot configure or

access. If you extend or create additional services you can use the cache framework to define and use your own set of caches.

The Cache Reference lists caches that might be used by your portal application. Use the list to assist you in your tuning and keep in mind the memory that is available to your system. When modifying the maximum cache sizes also monitor the system memory to determine the effects.

Using the Portal Administration Console to Configure Cache Settings

You can use the Service Administration tools within the WebLogic Portal Administration Console to configure statically-defined caches. For a list of configurable caches, see the Cache Reference.

When you configure a cache you modify its parameters to change its behavior or capability. Each cache has a Max Size setting and a Time To Live setting. For example, you can set up a cache to hold only the last 10,000 entries and set the time they can remain in the cache. You can also flush the cache so that all new requests for information come directly from the database.

For instructions on how to configure cache settings, see Adding a Cache from the Cache Reference.

Caching with JSP Tags

Some WebLogic Portal JSP tags support caching results at various scopes such as session or page. This allows for more control over the caching of individual content queries. Although this can be seen as an advantage, remember that when you control caches with coding, any cache change will require more maintenance, depending on the size (amount of code) of your application.

For example, the following content management-related JSP tags include cache-related attributes:

- `<cm:search>`
- `<cm:getNode>`
- `<cm:getProperty>`

For more information about these JSP tags and their attributes, see WebLogic Portal Javadoc.

Disabling Unused Services

When you create a new portal application WebLogic Portal enables most services, such as commerce services, event listening, and campaigns. If your portal application does not require these services, you can improve performance by turning them off.

You can disable behavior tracking or individual events. For more information on how to do this, see the Interaction Management Guide.

Tuning for Campaigns

Campaigns are powerful tools for personalization which allow the application to target users with specific web content, e-mails, and discounts based on fine-grained rules. The following tips allow you to tune your campaign settings to ensure better performance.

Referencing Events

Always make scenario rules dependent on a particular event. This allows optimizations based on the event types referenced in the scenario rules.

Avoiding Firing Extraneous Events

Whenever possible, avoid firing any extraneous events. The campaign services must listen to all events. Use events to signify important occurrences on the site.

Asynchronous Campaigns

Setting campaigns to asynchronous can result in better response times for the end user viewing those campaigns. This is done through the `AsynchronousEventListener` mechanism.

This optimizations is beneficial if the campaign results are not required within the same requests. If the campaign is executed prior to the next request which comes into the server (not necessarily from the user who made the original request) then setting the campaign to asynchronous will help improve the performance of campaigns. For example, if a user were to log in they wouldn't always see the campaign content placeholder on the screen immediately after the login form but the user would see it prior to their next page change or refresh. Due to the nature of multi-threaded applications the user might see the results on the next immediate screen, but that is not guaranteed.

Setting campaigns to asynchronous does not lower the overall load on the server, but it will lower the response time for the individual requests since the user won't be waiting for the campaigns to execute in the same thread.

There is a limitation to this however. If the campaign is required within the same request then setting the campaign to asynchronous is not recommended.

Using Goal Checking for Campaigns

If you are using campaigns that take advantage of goal checking set the goal checking appropriately. Goal checking is used to determine if a campaign's goals are met. When developers create campaigns they can set them to end on a specific date or use a set of goals (for example, number of views or clicks). You should set it according to the duration of your campaign. If a campaign's goal check mechanism is set too low it will affect portal performance. The default is 300000 milliseconds (five minutes).

You can adjust the goal check time for campaigns using the Administration Console.

For more information about how to adjust this setting, see Adjusting Goal Definitions in the *Interaction Management Guide*.

Using Ads During Campaigns

The Campaign service uses display counts to determine whether a campaign has met its end goals. Each time an ad placeholder finds an ad to display as a result of a scenario action the Campaign service updates the display count.

By default, the Campaign service does not update the display count in the database until an ad placeholder has found 10 ads to display as a result of one or more scenario actions. For performance tuning you can change this default to decrease the database traffic needed to support a campaign.

For sites with high traffic, increase this number to a range of 50 to 100.

To configure the Ad Service cache, use the Administration Console to perform the following steps:

1. From the Administration Console, choose **Service Administration**.
2. In the Application Configuration Settings Resource tree, select **Ad Service Group** under **Interaction Management**.

3. Edit the Ad Service and adjust the **Display Flush Size** to a number appropriate for your portal needs. The default is 10.
4. Click **Update**.

Tuning for Entitlements

If you want to cache entitlement information, you need to configure your application to recognize the cache settings. You can do this by editing the `netuix-config.xml` file.

The `netuix-config.xml` file resides in the portal web application in the `WEB-INF` directory.

After making any changes, you must redeploy your web application for the changes to take effect. For more information about modifying web descriptor files, see "Portal Web Application Deployment Descriptors" in the *Production Operations Guide*.

1. Edit the `netuix-config.xml` file to include the following text:

```
<entitlements control-resource-cache-size="200">
    <enable>true</enable>
</entitlements>
```

2. If your portal uses a large number of entitlements (more than 5000), review the WebLogic Server documentation, Best Practices: Configure Entitlements Caching When Using WebLogic Providers.
3. After completing the changes you will need to redeploy your portal application.

Using Role Caching When Using Entitlements

Role values are cached automatically. However, if you define roles using expressions that utilize dynamic attributes (such as session or request attributes), caching may have little or no value because these expressions are evaluated at runtime. In this case, turning off role caching may improve performance.

To disable role caching, you need to edit the `web.xml` file for the respective application.

Note: After making any changes, you must redeploy your web application for the changes to take effect. For more information about modifying web descriptor files, see "Portal Web Application Deployment Descriptors" in the *Production Operations Guide*.

1. Navigate to the respective `web.xml` file. It is located in the `WEB-INF` subdirectory of your portal application directory.

2. Open the `web.xml` file in a text editor.
3. Add the following lines

```
<env-entry>
   <env-entry-name>p13n.entitlements.disableRoleCache</env-entry-name>
   <env-entry-value>Y</env-entry-value>
   <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

4. Save the new `web.xml` file.
5. Redeploy your web application.

Tuning for Content Management

Reading and Searching Content

When using search capabilities with content management, it is possible to specify the search criteria used to find a node. Make sure to focus search queries to reduce the total number of nodes returned. This can be done by adding additional query criteria to the search request.

If you are not using the content search capabilities, then turn off this functionality by disabling both Autonomy and searching in the BEA repository configuration. To disable Autonomy, set the environment variable `WLP_SEARCH_OPTION=none` on server startup. To turn off search in the Content Repository see the Integrating Search Guide.

Pagination has been one focus of performance tuning efforts in the product. If doing pagination over result sets, use one of the objects provided by the Content API. Various options for paging can be found in the Content Management API Javadoc.

The larger the batch size (number of nodes) in a result set, the faster the overall performance will be. Where possible, increase the batch size for a returned result set. See the Capacity Planning Guide for more details on how batch size affects performance.

There are a couple of ways to get access to nodes in the database. The fastest way to retrieve a node is via the node ID. Whenever possible, use this method to retrieve the node. See the Capacity Planning Guide for more details on how different node access types affect performance.

Cache Settings

When you use a BEA repository for your content management system, you can tune the cache settings according to the needs of your portal application. Additional performance recommendations and benchmark data can be found on BEA's dev2dev website in the WebLogic Portal 9.2: Content Management Performance Optimization White Paper and in the Capacity Planning Guide.

You can adjust repository caches by editing Advanced Repository Properties in the *Content Management Guide*.

You can adjust cache settings for nodes or binaries according to how often your content is accessed and how much content you want to remain in the cache. Keep in mind that your server must have enough memory to handle the cache settings you assign. These settings are configured in the `content-config.xml` under the application's `META-INF` directory.

Table 3-1 Node Cache

Cache Setting	Usage Notes
Maximum Entries	Determines the maximum number of entries (folders) that can be cached.
Time To Live	Determines how long the entries will be cached.
Enable	Enables the cache. Mark this checkbox to enable this cache. To disable this cache, unmark the checkbox.

Table 3-2 Binary Cache

Cache Setting	Usage Notes
Maximum Entries	Determines the maximum number of entries (content items) that can be cached.
Time To Live	Determines how long the entries will be cached.

Table 3-2 Binary Cache

Cache Setting	Usage Notes
Cache Size/Item	Sets the maximum size of a single entry (content item) stored in the cache. The default is 1024 bytes (1K). If your content items average a larger size than this, you should consider changing this cache.
Enable	Enables the cache. Mark this checkbox to enable this cache. To disable this cache, unmark the checkbox.

Tuning for PageFlow Portlets

PageFlow portlets have the potential to significantly increase the memory usage on the server. This is caused by each portlet storing memory both locally and replicating it in the session. Each visible portlet consumes between 500 and 1000 bytes of data either in the local memory or in the session. This is true for each active session accessing the application. This can add up quickly if there are a high number of visible page flow portlets and a high number of active sessions on the server.

Setting the `requestAttrPersistence` setting on the portlet to `transient-session` can decrease the amount of data in the session. However, since this data is serialized it will still consume local memory resources. More information about this can be found in the Optimize Page Flow Session Footprint chapter under Designing Portals for Optimal Performance in the *Portal Development Guide*.

To work around this potential system limitation it is recommended that the number of page flow portlets visible in any given Portal be less than 100 and the memory on the system be increased to deal with the additional overhead.

Tuning for WSRP

For more information about performance guidelines for Web Services Remote Portlets, see Designing for Performance section in the *WebLogic Portal Federated Portals Guide*.

Clustering for WSRP

When tuning for WSRP, it is important to strike a balance between the number of producer machines and the number of consumer machines. In general WebLogic Portal is CPU bound, meaning that additional CPU resources (usually via clustering) can be used to eliminate

bottlenecks. Through performance testing the WSRP infrastructure it is possible to determine whether the producers or the consumer machines are the bottleneck, and then add additional resources as necessary. Depending on the configuration and application it might be necessary to cluster either the consumer or the producer. For more information regarding WSRP architecture refer to the WebLogic Portal Federated Portals Guide. For more information about clustering see the WebLogic Server document Using Weblogic Server Clusters.

Enabling Caches for WSRP

Cache can have an impact on performance, but the size of the portal (determined by the total number of portlets) has the most impact. If you are using WSRP portlets, adjust your caches accordingly. For specific information about WSRP caches, see the WSRP Caches section in the Cache Reference Guide.

Parallel Processing and WSRP

WebLogic Portal has the capability to render portlets in parallel. This is true for WSRP remote portlets as well. If a remote portlet is taking a long time to render, the overall portal may render faster by turning on parallel portlet processing. To enable parallel processing use the `forkPreRender` attribute of the portlet. See Understanding Portlet Development in the *Portlet Development Guide*.

Tuning for Delegated Administration

This section explains how to avoid performance problems with the Oracle WebLogic Portal Administration Console when you configure delegated administration roles.

Tip: For information on delegated administration, see the Oracle WebLogic Portal Security Guide.

Overview

For each delegated administration role that is configured on a portal resource, several security policies are created (CAN_VIEW, CAN_EDIT, and so on). It is important to note that for each policy that is created, information must be retrieved to perform the required evaluation. As the number of policies increases, the WebLogic Portal Administration Console will perform more and more poorly.

To avoid this poor performance, create as few delegated administration policies as possible.

Best Practice: Limit DA Policies on Resources

When a delegated administration (DA) role is added to a portal resource, such as a content management node or a book node, several security policies are created for each capability, such as CAN_VIEW, CAN_EDIT, and so on). The time taken to retrieve these policies from LDAP and the policy reference information from the database increases as the number of policies increases. Although caching of policy data might improve performance, it is not a secure solution. For instance, if an administrator changed the permissions for a user, the change would not take effect for the user's session. Other solutions such as flushing the cache and using the session to store information are either technically impractical or not secure.

To achieve the best possible performance, organize your portal resources so that as few as possible DA policies will be created. For example, consider the use case where there are 100 content nodes and you want a particular DA user to only see 10 of those nodes. The best practice is to create a parent node that contains these 10 content nodes and place the DA policies on the parent node only.

Using Work Managers

WebLogic Portal uses WebLogic Server's CommonJ WorkManager infrastructure for forked portlet pre-render and render. WorkManagers have similar but not identical configuration parameters, behavior, and deployment options. When you upgrade an 8.1.4+ application, any existing customizations to the portalRenderQueue thread pool will not be automatically applied to the default WorkManager used for forking.

To tune this WorkManager, configure a WorkManager and associate it with the name wm/portalRenderQueueWorkManager. For more information about WorkManagers and thread usage in WebLogic Server 10.2, refer to Using Work Managers to Optimize Scheduled Work in the WebLogic Server documentation.

CHAPTER 4

Tuning Your Portal Web Application

One of the key things you can do to ensure good performance for your web application is to design appropriately, see Designing Portals for Optimal Performance for more information about designing portals.

This chapter covers a few configuration settings and key areas that can be optimized according to your needs and includes the following sections:

- Optimizing Your Portal Control Tree
- Modifying Your Portal Web Application Parameters

Optimizing Your Portal Control Tree

Portal web applications use a control tree to cache and access different functionality. For example, portals use controls to access desktops, books, pages, portlets, and menus. When you create complex portals that require a large number of controls, tree optimization is the easiest way to ensure optimal portal performance. Controls that are not active in the current portal instance are not built, saving considerable time and overhead. However, the use of multilevel menus negates much of the performance benefit that control tree optimizations provide. This is due to the menu traversing the control tree in order to build up the multilevel menu.

For more information about when to optimize your control tree, see the Designing Portals for Optimal Performance chapter of the *WebLogic Portal Development Guide*.

Modifying Your Portal Web Application Parameters

Your portal application uses configuration files to store application settings. Some default settings may not be applicable to your particular portal application.

Each portal application uses unique configuration files to customize parameters that can affect performance. Four configuration files that are key to portal performance include:

- `netuix.config.xml` (portal framework)
- `web.xml` (web application settings)
- `weblogic.xml` (server settings)
- `p13n-cache-config.xml` (portal cache settings)

For most settings you can adjust them using either the WebLogic Server Console or the WebLogic Portal Administration Console. However, many of the settings discussed in this section must be manually entered in the configuration file.

Modifying Portal Framework Settings

The `netuix-config.xml` file resides in the portal web application directory under `WEB-INF`.

After making any changes, you must redeploy your web application for the changes to take effect. For more information about modifying web descriptor files, see Configuration Files in the *Production Operations Guide*.

Table 4-1 lists key performance tuning elements within the `netuix-config.xml` file.

Table 4-1 netuix-config.xml

Element	Usage Notes
`<customization>`	A switch to indicate if a portal is customizable or not. If a portal is served from a `.portal` file (rather than from a database) and users are not allowed to customize it then customization can be disabled by setting `enable` element's value to `false`. If a portal supports customizations then customization should be enabled but keep in mind that there will be an impact on the performance of the system with the use of this feature.
`<pageflow>`	A switch to enable or disable page flows usage in a portal. Disable it if a portal is not using any page flows.

Table 4-1 netuix-config.xml

Element	Usage Notes
`<validation>`	A switch for validating portal related files such as `.pinc`, `.portlet`, and `.portal` files. Disable validation when running portal server in production.
`<entitlements>`	A switch to indicate that a portal is setup to use entitlement policies (users able to view portal resources such as desktop, books, pages, portlets, and so on). Disable entitlements if a portal is not using any security policies. If a portal is using security policies, enable it and set the value for `<control-resource-cache-size>` attribute using number of desktops + number of books + number of pages + number of portlets + number of buttons (max, min, help, edit) used in a portal. The default value could be used if memory is a concern. For more information, see "Tuning for Entitlements" on page 3-5. Using entitlements will result in additional overhead for WebLogic Portal.
`<localization>`	A switch to indicate that a portal supports multiple locales. This should be disabled if a portal supports only one locale.

Modifying Web Application Settings

The `web.xml` file configures your web application. After making any changes you must redeploy your web application for the changes to take effect. For more information about modifying web descriptor files, see Portal Web Application Deployment Descriptors in the *Production Operations Guide*.

The `web.xml` file is located in the `WEB-INF` subdirectory of your portal web application directory.

Table 4-2 lists key elements of the `web.xml` file.

Table 4-2 web.xml

Parameter	Usage Notes
`<createAnonymousProfile>`	Set this to false if your portal does not store or use user profile information.

Table 4-2 web.xml (Continued)

`<enableTrackedAnonymous>`	Set this to false unless you are tracking anonymous users. When this is set to false, only users who login to the portal are tracked.
`<fireSessionLoginEvent>`	Set this to false unless using campaigns or behavior tracking. If this is set to true, session login events are generated.
`<trackedAnonymousVisitDuration>`	This setting allows you to determine when to start tracking anonymous users and is ignored unless you are tracking anonymous users. The longer you wait during a session to start tracking anonymous users, the less performance overhead there will be on the server.
`<skipRequestPattern>`	Set to determine which request patterns to skip. Each page displayed in a web application may have many separate requests, several of which are irrelevant. For example, the tutorial portal sends requests for images, JavaScript, and CSS files. Ignoring these requests for PortalServletFilter processing increases performance and guarantees that tracking anonymous users will behave as expected.

Modifying WebLogic Server Settings

You can modify the `weblogic.xml` file via the WebLogic Server Console. For more information on how to modify the descriptor elements see the "weblogic.xml Deployment Descriptor Elements" chapter in the Developing Web Applications, Servlets, and JSP's for WebLogic Server Guide.

The following parameters can be adjusted for performance. Table 4-3 lists key performance tuning elements in the `weblogic.xml` file.

Table 4-3 weblogic.xml

Parameter	Usage Notes
`<jspPageCheckSeconds>`	Sets the interval, in seconds, at which WebLogic Server checks to see if JSP files have changed and need recompiling. Dependencies are also checked and recursively reloaded if changed. If set to 0, pages are checked on every request. This default is preset for a development environment. If set to –1, page checking and recompiling is disabled. In a production environment where changes to a JSP are rare, change the value of `pageCheckSeconds` to –1 to disable page checking and recompiling.
`<servletReloadCheckSecs>`	Sets the interval, in seconds, at which WebLogic Server checks to see if servlet files have changed and need recompiling. Dependencies are also checked and recursively reloaded if changed. If set to 0, servlets are checked on every request. This default is preset for a development environment. If set to –1, servlet checking and recompiling is disabled. In a production environment where changes to a servlet are rare, change the value of `servletReloadCheckSecs` to –1 to disable servlet checking and recompiling.

Table 4-3 weblogic.xml

`<PersistentStoreType>`	Must be edited manually. Sets the persistent store method to one of the following options: • `memory` – Disables persistent session storage. • `file` – Uses file-based persistence. • `jdbc` – Uses a database to store persistent sessions. • `replicated` – Same as memory, but session data is replicated across the clustered servers. • `cookie` – All session data is stored in a cookie in the user's browser. • `replicated_if_clustered` – If the web application is deployed on a clustered server, the in-effect PersistentStoreType will be replicated. Otherwise, memory is the default. **Note:** In a clustered production environment, it is important that you configure the `PersistentStoreType` property in `weblogic.xml` to enable session replication to take place across the cluster. To do this, set the element to the `replicated_if_clustered` value. Without this setting, you will not have failover of a user's state information if a server in the cluster is stopped. By default if `persistent-store-type` is not set, it defaults to disabling persistent session storage. Also note that there will be increased memory utilization and additional overhead on the system with this feature enabled.
`<Timeout Secs>`	Sets the time, in seconds, that WebLogic Server waits before timing out a session, where *x* is the number of seconds between a session's activity. Minimum value is 1, default is 3600, and maximum value is integer `MAX_VALUE`. On busy sites, you can tune your application by adjusting the timeout of sessions. While you want to give a browser client every opportunity to finish a session, you do not want to tie up the server needlessly if the user has left the site or otherwise abandoned the session. This attribute can be overridden by the session-timeout element (defined in minutes) in `web.xml`.
`<debug>`	Turn off debugging by setting debug property to false.

Table 4-3 weblogic.xml

`<precompile>`	Precompile the JSPs in the web application to reduce the time needed to display pages on their first invocation by setting precompile to true.
`<precompile-continue>`	Also set `<precompile-continue>` to true, because if any JSPs do not compile, deployment of the web application stops. **Note:** Alternatively, you can use weblogic.appc to precompile JSPs. See the WebLogic Server documentation for more information.

WebLogic Portal Cache Settings

You can modify the `p13n-cache-config.xml` file via the WebLogic Portal Administration Console. For more information on how to modify the cache and a comprehensive list of WebLogic Portal Caches see the Weblogic Portal Cache Reference chapter in the *Cache Reference Guide*.

Caching Portlet Categories

Portlet category information is automatically cached, which enhances performance. If for any reason you do not want to cache portlet categories, you can turn off this cache by setting the following system property: `-enable.portlet.category.caches=false`

APPENDIX A

Performance Tuning Checklistst

This appendix provides checklists and tips for the following components of WebLogic Portal:

- Portal Framework Guidelines
- Portal Administration Console Guidelines

Portal Framework Guidelines

Table A-1 Portal Framework Guidelines

Guideline Question	How to Verify
Is the WebLogic Server well tuned?	See "Tuning Your WebLogic Server" on page 1-4
Is the JVM properly tuned?	See "Tuning Your JVM" on page 1-5
Is the WebLogic Portal database tuned?	See "Tuning Your Database" on page 1-5
Is the Domain running in Production mode?	See"Tuning Your Domain Configuration" on page 2-1
If you do not need to support multiple locales, is localization disabled?	See "Disabling Unused Services" on page 3-3.

Table A-1 Portal Framework Guidelines

Guideline Question	How to Verify
Are Campaigns tuned properly?	See "Tuning for Campaigns" on page 3-3
Are Entitlements enabled? If yes, is `control-resource-cache` size is set correctly?	See "Tuning for Entitlements" on page 3-5
Is the Content Management System optimal?	See "Tuning for Content Management" on page 3-6
How many visible PageFlow portlets are in the portal?	See "Tuning for PageFlow Portlets" on page 3-8
Is the `portalControlTreeCache MaxSize` set to the correct size for your portal?	See "Optimizing Your Portal Control Tree" on page 4-1
Is validation turned off?	See "Modifying Portal Framework Settings" on page 4-2.
Is `jspPageCheckSecs` in `weblogic.xml` is set to -1?	See "Modifying WebLogic Server Settings" on page 4-4.
Is `servletReloadCheckSecs` in `weblogic.xml` is set to -1?	See "Modifying WebLogic Server Settings" on page 4-4.
Are sessions replicated? If so what `persistent-store-type` is used?	See "Modifying WebLogic Server Settings" on page 4-4
Has the application been performance tested?	See Approaches to Performance Testing on dev2dev

Portal Administration Console Guidelines

You can improve the performance of the Portal Administration Console. Specifically, you can decrease the time it takes to work with desktops and to browse portal resources.

This section includes the following topics:

Creating Desktops

When you create a new desktop in the Administration Console, a list of `.portal` files is used to populate the templates drop-down list. If all `.portal` files reside under the same directory under the web application directory, this drop-down list can be created quickly.

To take advantage of higher performance in building the drop-down list, you must define the `portalFileDirectory` in the web application's `web.xml` file.

Note: After making any changes, you must redeploy your web application for the changes to take effect. For more information about modifying web descriptor files, see "Portal Web Application Deployment Descriptors" in the *Production Operations Guide*.

1. Navigate to the respective `web.xml` file. It is located in the `WEB-INF` subdirectory of your portal application directory.
2. Open the `web.xml` file in a text editor.
3. Add the following lines

```
<context-param>
   <param-name>portalFileDirectory</param-name>
   <param-value>/</param-value>
</context-param>
```

4. Save the new `web.xml` file.
5. Redeploy your web application.