IBM WebSphere Portal software family
Your world. Your way.

WebSphere. software

IBM

IBM WebSphere Portal 6.1.X

Performance Tuning Guide

IBM WPLC Performance Team
March 2009

Document version 2.1

WebSphere Portal

Contents

PERFORMANCE TUNING OVERVIEW 2

Environment Considerations 3

32-bit and 64-bit Considerations 3

Hardware Multithreading (Hyper-Threading) 3

BASE PORTAL TUNING 4

Application Server Tuning 5

JVM Initial and Maximum Heap Size 5

JVM Heap Large Page 7

JVM Heap New Area Size 8

Additional SUN JVM Arguments 8

Session Timeout 9

Web Container Thread Pool Size 9

Security Attribute Propagation 10

VMM Context Pooling 11

ORB Service Tuning For z/OS 11

WebSphere Portal Services 12

Navigator Service 12

Registry Service 13

Cache Manager Service 14

Database Tuning 15

Datasource Tuning For DB2 15

DB2 Database Server Tuning 15

Oracle Database Server Tuning 19

Other Database Considerations 21

Directory Server Tuning 22

Web Server Tuning 23

Operating System Tuning 25

AIX 25

Linux 26

Windows 2003 26

Solaris 27

Z/OS 29

Required Fixes 29

WEB 2.0 THEME TUNING 30

JVM Initial and Maximum Heap Size 30

Navigator Service Properties 30

Internet Explorer Support of Vary Header 31

Caching Proxy Tuning 31

Web Server Tuning 32
Portlet Caching 33
MANY PAGES TUNING 34
DB2 Database Tuning 34
Cache Manager Service 35
Required Fixes 35
WEB CONTENT MANAGEMENT TUNING 36
Application Server Tuning 36
WebSphere Portal Service Properties 37
Cache Manager Service 37
Navigation Service 38
WCM Object Cache 38
WCM Configuration Service 39
JCR Text Search 39
DB2 Tuning (Authoring Environment) 40
Multiplatform (LUW) 40
Z/OS 41
COMPOSITE APPLICATIONS TUNING 43
Cache Manager Service Properties 43
Composite Applications Best Practices 44
CLUSTER TUNING 46
Application Server Tuning 46
Dynacache Custom Properties 46
z/OS Dynacache Custom Property 46
Thread Pools 47
Transport Buffer Size 47
WMM Context Pooling 47
Web Server Tuning 48
Session Persistence To Database Tuning 49
Vertical Cluster Tuning 50
Required Fixes 51
OTHER PERFORMANCE TUNING OPTIONS 52
Improving Portal Startup Performance 52
Managing the Retrieval of User Attributes 53
Identifying a Full Fetch of User Attributes 54
Minimum Attribute Set 55
Use of Dynamic Content Features 55
Real-World Network Considerations 56
Compress Content on the HTTP Server 56
Enabling Client-Side Caching 57

WEBSPHERE PORTAL CACHES 58
General Information 58
Cache Configuration Properties 58
Cache Usage Patterns 61
Cache Instances 62
Access Control 62
Portal User Management 67
Datastore 68
Model 69
URL Mappings 74
Virtual Portals 74
WSRP 75
Dynamic Assembly / Process Integration 77
Policy 78
Collaboration Services 78
Miscellaneous 79
Example Scenarios 82
General Comments 82
Small Number of Pages and Small Number of Users 83
Small Number of Pages and Large Number of Users 83
Portals with Long Session Timeouts 84
Portals with Many Pages 84
WEB CONTENT MANAGEMENT CACHES 86
WCM Cache Instances 86
WCM Item caching 86
WCM Summary 86
WCM Basic Caching 87
Advanced and Resources 87
Session Cache 87
Menu 88
Navigator 88
Absolute path 88
Missed Items 88
Library 88
Library Parent 89
Draft Summary 89
User cache 89
Appendix A. References 90
Appendix B. Credits 91

Figures

Figure 1 Portal Access Control Cache Hierarchy 63
Figure 2 Portal Model Cache Hierarchy 70

Tables

Table 1: Additional Sun JVM Settings 8
Table 2: WebSphere Security Attribute Propagation Settings 10
Table 3: VMM Context Pool Setting 11
Table 4: Navigation Service Settings 12
Table 5: Registry Service Settings 13
Table 6: Cache Manager Service Settings 14
Table 7: DB2 Database Domains 15
Table 8: Oracle Database Tuning 20
Table 9: IDS Tuning 22
Table 10: Web Server Tuning 23
Table 11: AIX Network Settings 25
Table 12: Linux Network Settings 26
Table 13: Windows Network Settings 26
Table 14: Solaris Network Settings 27
Table 15: z/OS System Tuning 29
Table 16: Navigation Service Settings for Web 2.0 Theme 30
Table 17: Reverse Proxy Settings 31
Table 18: DB2 Database Settings for Many Pages 34
Table 19: Cache Manager Service Settings for Many Pages 35
Table 20: Cache Manager Service Settings for WCM 37
Table 21: Navigation Service Settings for WCM 38
Table 22: WCM Object Cache Settings 38
Table 23: DB2 z/OS Bufferpool Settings 41
Table 24: DB2 z/OS Default Bufferpool Settings 42
Table 25: Cache Manager Serivce Properties for Application Infrastructure 43
Table 26: Web Server Tuning for Clusters 48
Table 27: WebSphere Session Persistence Tuning 49
Table 28: IDS Tuning in Vertical Cluster 51

ABOUT THIS DOCUMENT

This white paper provides a basis for parameter and application tuning for IBM WebSphere Portal for Multiplatform V6.1. Remember that both tuning and capacity are affected by many factors, including the workload scenario and the performance measurement environment. For tuning, the objective of this paper is not to recommend that you use the values we used when measuring our scenarios, but to make you aware of those parameters used in our configuration. When tuning your individual systems, it is important to begin with a baseline, monitor the performance metrics to determine if any parameters should be changed and, when a change is made, monitor the performance metrics to determine the effectiveness of the change.

WebSphere Portal

1

PERFORMANCE TUNING OVERVIEW

Tuning a WebSphere Portal environment involves tuning and configuring the various systems and components of the environment. This chapter discusses some general concepts and details the specifics of the configuration used in our measurement environments. These specifics entail:

- Configuring the application server and the resources defined for that application server
- Tuning the database(s) and database server
- Tuning the directory server and its database
- Tuning the web server and/or proxy server
- Tuning the operating system and network
- Tuning the WebSphere Portal services

When tuning your individual systems, it is important to begin with a baseline, monitor the performance metrics to determine if any parameters should be changed and, when a change is made, monitor the performance metrics to determine the effectiveness of the change.

In addition to the tuning changes we made in our measurement environments, there are some additional tuning options available which can improve performance in certain circumstances; these will be discussed in a separate section.

Environment Considerations

Before beginning your install of WebSphere Portal you should consider how to use the environment in order to achieve ideal performance. Topics to consider include:

- Choosing between 32-bit and 64-bit JVMs
- Use of hardware multithreading, also known as Simultaneous Multithreading or Hyper-Threading.

32-BIT AND 64-BIT CONSIDERATIONS

The choice of a 32-bit or 64-bit JVM involves some trade-offs. The key advantage of a 64-bit JVM is its vastly larger address space. Heap sizes of 2.5GB or larger can be practical on modern server systems. This can be a significant benefit for applications with high memory demands.

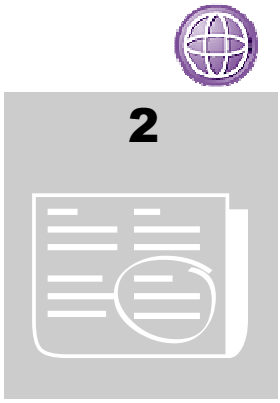
A 64-bit JVM does have disadvantages as well. Machine instructions and memory references in a 64-bit JVM are larger than in a 32-bit JVM. This means that Java objects, which typically contain multiple memory references, are larger in a 64-bit JVM than compared to a 32-bit JVM. Therefore a 64-bit JVM will need a larger heap than a 32-bit JVM for the same population of objects.

The increased size of instructions and memory references imposes a second performance penalty. They increase the demand on the memory subsystem of the system, causing more cache misses and a higher demand for memory bandwidth. As a result, executing a set of operations in a 64-bit JVM can be slower than executing the same operations in a 32-bit JVM.

When considering a deployment of WebSphere Portal 6.1, consider the memory demands your applications will have. If you expect a high demand for memory, the best performance will probably come from a 64-bit JVM. On the other hand, if the memory demand is lower, a 32-bit JVM is likely to give superior performance.

HARDWARE MULTITHREADING (HYPER-THREADING)

Many modern processor architectures support hardware multithreading. For example, this is known as Hyper-Threading (HT) on Intel processors and Simultaneous Multithreading (SMT) on Power-series processors. Our experience is that using hardware multithreading provides an improvement in capacity in all of the scenarios and platforms reported in this report, so we would recommend its use on platforms where this is an option.



BASE PORTAL TUNING

The Base Portal Scenario covers user login, page navigation, and interaction with simple portlets. Users can see a small set of pages, some of which are visible to all authenticated users, with access to others based on their group membership.

We have also benchmarked a number of other scenarios, which focus on different functions or use cases for WebSphere Portal. For example, there are scenarios which make use of Web Content Management (WCM), and a scenario where users have access to thousands of pages. While we have used different tuning to optimize performance for some of those scenarios, the tuning is all based on the tuning done in the Base Portal Scenario.

In all of our measurement environments, we use a separate database server and directory server, in addition to the WebSphere Portal server. We run these servers on separate systems to avoid resource contention on the system running the WebSphere Portal server. This helps improve the maximum capacity achievable.

Application Server Tuning

There are many aspects to configuring and tuning an application server in WebSphere Application Server. We found that those aspects presented here were critical to a correctly functioning and optimally performing WebSphere Portal in our laboratory environment.

For more details on tuning a WebSphere Application Server, see the Tuning Section of the information center located at:

http://www-01.ibm.com/software/webservers/appserv/was/library/

How to get to Admin Console

There are two methods to get to WebSphere Administrative Console.

- Start Server1 and use port 10001
  1. In <WAS_root>/profiles/wp_profile/bin
  2. ./startServer.sh server1
  3. http://yourhost:10001/admin

- Start Portal and use port 10027
  1. In <WAS_Root>/profile/wp_profile/bin
  2. ./startServer.sh WebSphere_Portal
  3. http://yourhost:10027/ibm/console

Customer ports can differ from the ports 10001 or 10027 mentioned on this page. To find out the ports in use for your installation, look for 'adminhost' in <wp_profile root>/config/cells/<cell_name>/nodes/<node_name>/serverindex.xml.

The following are settings based on our experience with the Base Portal workloads described above:

JVM INITIAL AND MAXIMUM HEAP SIZE

Java Virtual Machine heap size: The value of the JVM Heap size is directly related to the amount of physical memory on the system. Never set the JVM heap size larger than the physical memory on the system.

How-To Set: In the WebSphere Administrative Console: Servers → Application Servers → WebSphere Portal → Server Infrastructure: Java and Process Management→Process Definition → Java Virtual Machine
- Initial Heap Size
- Maximum Heap Size

See JVM Max Heap Size Limits for further discussion.

See instruction on How to get to Admin Console

JVM MAXIMUM HEAP SIZE LIMITS

When setting the heap size for an application server, keep the following in mind:

- Make sure that the system has enough physical memory for all of the processes to fit into physical memory, plus enough for the operating system. When more memory is allocated than the physical memory in the system, paging will occur, and this can result in very poor performance.
- We set the minimum and maximum heap sizes to the same values since we're using the gencon garbage collection policy available in 1.5 IBM JDK which avoids heap fragmentation, this may not be the best choice if you plan to use a different garbage collection. In our measurement runs, the system is under load for a relatively short time (around 3 hours), and it is running with portlets which do not have large memory requirements. When using portlets which have larger memory requirements, or for continuous operation, it may be possible to reduce heap fragmentation by setting the initial heap size to 320 megabytes.
- After doing any tuning of heap sizes, monitor the system to make sure that paging is not occurring. As mentioned above, paging can cause poor performance.
- 32-bit operating systems have an address space limit of 4GBytes, regardless of the amount of physical memory in the system. This space limits the maximum size of each individual process in the system. In addition, some operating systems restrict the size of processes to be even less than this limit. Many versions of Windows limit processes to 2GBytes in size; you can find more information at http://support.microsoft.com/default.aspx?scid=kb;en-us;555223.
- The address space limit further restricts the size of the JVM process. If the process grows larger than the limit imposed by the operating system, it may terminate unexpectedly.

Due to the demands on native memory by WebSphere Portal V6.1 and its underlying components, we chose a maximum heap size of 1408MB in our Windows environments. There is a balance between JVM heap and native memory, all of which must fit within the 2GB restriction in 32-bit Windows. 1408MB was the largest value we could use to successfully measure all of our Windows configurations and workloads. If your application has additional native memory requirements then you may need to choose a smaller maximum heap size. For more information, see the WebSphere Application Server information center.

On Solaris and zLinux, we use 3.5GB heap size in 64-bit environment.

Parameter	*AIX POWER5*	*Linux*	*Solaris*	*Windows 2003*	*z/Linux*	*z/OS*
Initial and Maximum heap size (Mbytes)	1792	2048	3584	1408	3584	2048

JVM HEAP LARGE PAGE

Large pages can reduce the CPU overhead needed to keep track of heap. With this setting we have seen 10% throughput improvement in our measurements.

This setting does improve performance on Windows, we did not set it for our measurements because Portal doesn't start reliably when –Xlp is set, sometimes it requires a system reboot to get the jvm to start.

How-to Set: In the WebSphere Administrative Console: Servers -> Application Servers -> WebSphere Portal -> Server Infrastructure: Java and Process Management -> Process Definition -> Java Virtual Machine -> Generic JVM Argument. Add **–Xlp**.

Large pages are supported by systems running Linux kernels V2.6 or higher. See JVM Large Page Tuning for AIX Operation System.

JVM LARGE PAGE TUNING ON AIX OPERATING SYSTEM

To use JVM Large Page, AIX operating system must be configured to support large pages.

How-To Set:

1. We use the following steps to allocate 4GB of RAM as large pages (16MB) . We chose this amount based on having 8GB of physical memory in these systems. These values may need to be adjusted on systems with different amounts of physical memory.
   vmo -r -o lgpg_regions=256 -o lgpg_size=16777216
   bosboot -ad /dev/ipldevice
   reboot -q
   vmo -p -o v_pinshm=1
   chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE $USER
2. Add: -Xlp command-line option as described above.
3. In the WebSphere Administrative Console: Servers → Application Servers → WebSphere Portal → Server Infrastructure: Java and Process Management→Process Definition-> Environment Entries → New → EXTSHM=OFF (note: When EXTSHM is on it prevents use of large page).
4. Restart Portal Server. To verify if large pages are being used, run the AIX command vmstat -l 1 5 and check the "alp" column which is the active large page used. It should be a non-zero value if large pages are being used.

Parameter	*AIX POWER5*	*Linux*	*Solaris*	*Windows 2003*	*z/Linux*	*z/OS*
JVM Heap Large page	-Xlp	-Xlp	Not Applicable	Not Applicable	Not Applicable	Not Applicable

JVM HEAP NEW AREA SIZE

The Generational Garbage Collector introduced in Java 5.0 is efficient to Portal application JVM memory management, and it is set as default by installation with the –Xgcpolicy:gencon command-line option. Use –Xmn to further fine tune the Java heap new area (Nursery).
The –Xgcpolicy:gencon option does not apply to Solaris.

How To Set: In the WebSphere Administrative Console: Servers → Application Servers → WebSphere Portal → Server Infrastructure: Java and Process Management→Process Definition → Java Virtual Machine -> Generic JVM Arguments:–Xmn256m

Parameter	*AIX POWER5*	*Linux*	*Solaris*	*Windows 2003*	*z/Linux*	*z/OS*
New Area Size	-Xmn320m	-Xmn256m	-Xmn768m	-Xmn256m	-Xmn1024m	-Xmn320m

ADDITIONAL SUN JVM ARGUMENTS

On the Solaris platform, we use the following Java HotSpot parameters to achieve optimum performance.

Table 1: Additional Sun JVM Settings

Parameter	*Value*	*Additional Information*
-server		Offers higher throughput than the "client" mode.
-XX:MaxPermSize	768m	
-XX:+UseConcMarkSweepGC		Use concurrent mark-sweep collection for the tenured generation. The application is paused for short periods during the collection; we found this collector works best in Portal.
-XX:SurvivorRatio	6	
-XX:+UseParNewGC		By default concurrent low pause collector uses the default, single threaded young generation copying collector. Set this parameter to use parallel young generation collector for new area.
-XX:ParallelGCThreads	5	Reduces the number of garbage threads. On the Chip multithreading processor based system, we set the threads no higher than one quarter of the hardware threads. We also distribute the threads for 6 JVMs. Our system has 128 virtual processors, we set a total of (128/4)=32 GC threads across all the JVMs. So 5 or 6 GC threads per JVM.
-XX:+PrintGCDetails		Print more details at garbage collection. This does not improve performance, but it provides additional information related to garbage collection activity, which is useful in tuning garbage collection.
-XX:+PrintGCTimeStamps		Print timestamps at garbage collection. See above.

SESSION TIMEOUT

Session timeout: The default value of Session Timeout is 30 minutes. Reducing this value to a lower number can help reduce memory consumption requirements, allowing a higher user load to be sustained for longer periods of time. Reducing the value too low can interfere with the user experience.

For Solaris, on a T5240 hardware, we used a much lower think time, 5 seconds, than was used for other platform hardware measurement of 12 seconds. With a lower thinktime, fewer vusers will result in a heavier load on the system. The reason we lowered the thinktime was specifically to decrease the number of vusers required for this measurement. Our pool of LoadRunner vuser licenses was inadequate to generate enough load with the higher think time. With a shorter think time than is used in the other measurements, the duration of each virtual user's interaction with the site is shorter by approximately 2 minutes. To compensate for this, and keep the sessions live on the server for the same period of time, we increased the session timeout by 2 minutes, to 12 minutes.

How To Set: In the WebSphere Administrative Console: Servers → Application Servers → WebSphere Portal → Container Settings: Web Container Settings → Session Management → Session Timeout -> Set Timeout

Parameter	*AIX POWER5*	*Linux*	*Solaris*	*Windows 2003*	*z/Linux*	*z/OS*
Session timeout	10 minutes	10 minutes	12 minutes	10 minutes	10 minutes	10 minutes

WEB CONTAINER THREAD POOL SIZE

Servlet engine thread pool size: Set this value and monitor the results. Increase this value if all the servlet threads are busy most of the time.

How To Set: In the WebSphere Administrative Console: Servers → Application Servers → WebSphere Portal→ Additional Properties: Thread Pools→ Web Container → Thread Pool - Minimum size threads - Maximum size threads

Parameter	*AIX POWER5*	*Linux*	*Solaris*	*Windows 2003*	*z/Linux*	*z/OS*
Web Container Thread pool size	50	50	50	50	50	50

To reduce the Security Attribute Propagation (SAP) overhead, please use a custom property 'disable Callerlist'. If SAP is not used, you can disable that, to remove the extra overhead to improve the login performance.

If Subject has not been customized, then there is no need to enable Security Attribute Propagation. Security Attribute Propagation can add extra overhead due to some extra processing that is required. However, there are certain configurations where performance might be better with security propagation enabled due to reduction of remote registry calls. See the WebSphere 6.1 InfoCenter (search for 'security attribute propagation') for a discussion of when propagating security attributes is desirable. If you want to enable SAP for functional reasons, you can improve the performance with CallerList tuning mentioned below.

These settings apply to all platforms.

How to Set: In the WebSphere Administrative Console: Security->Secure Administration, Applications, and Infrastructure -> Custom properties ->

Table 2: WebSphere Security Attribute Propagation Settings

Security Attribute Propagation	***Name***	***Value***
	com.ibm.CSI.disablePropagationCallerList	true
	com.ibm.CSI.rmiOutboundPropagationEnabled	false
	com.ibm.CSI.rmiInboundPropagationEnabled	false
	com.ibm.ws.security.webInboundPropagationEnabled	false

For **com.ibm.CSI.disablePropagationCallerList** create a new property, for the other 3 properties, modify their value to “false”.

Note to WAS 7:

In our WAS 7 environment, we add com.ibm.CSI.disablePropagationCallerList = true, and use the other 3 default true attributes. For was7, this field is accessed through: Security->Global Security ->CustomProperties->New.

VMM CONTEXT POOLING

Tune VMM Context Pooling to improve the performance of concurrent access to an LDAP server.

We changed the following Context Pooling settings line in:
<wp_profile_root>/config/cells/<cellname>/wim/config/wimconfig.xml

```
<config:contextPool enabled="true" initPoolSize="10" maxPoolSize="0"
poolTimeOut="0" poolWaitTime="3000" prefPoolSize="30"/>
```

You can also set them via the administrative console as described in
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/uwim_ldapperfsettings.html

Table 3: VMM Context Pool Setting

Context Pool Setting	Default Value	Value
initPoolSize	1	10
prefPoolSize	3	30 Number of open connections to maintain to LDAP server.
maxPoolSize	20	0. A value of 0 allows the pool to grow as large as needed. If access to the LDAP server is shared by many systems, this setting may allow an excessive number of connections to the LDAP server; in such a case, set the maximum pool size to a value appropriate to your environment.

ORB SERVICE TUNING FOR Z/OS

In the WAS Admin Console, set the ORB Service to be "**pass by reference**" instead of "**pass by value**" (default) for both server1 and WebSphere_Portal

How to Set:
- Servers→ Application Servers→ server1→ Orb Service

  - check box for "Pass by Reference"

- Servers→ Application Servers →WebSphere_Portal → Orb Service

  - check box for "Pass by Reference"

WebSphere Portal Services

WebSphere Portal has a number of configurable "services"; each service has several parameters available to it. This section describes which services we tuned, the tuning values used, and the rationale for those changes.

How to Set:

1. Edit <wp_profile_root>/PortalServer/config/properties/xxxService.properties
2. uncomment the line, then change the size.
3. run <wp_profile_root>/ConfigEngine/ConfigEngine.sh update-properties

The changes should appear on WAS Console -> Resource Environment Providers -> WP_xxxService -> Custom properties

NAVIGATOR SERVICE

The navigator service manages the content model for unauthenticated users, which controls the pages those users are able to see. This content model is periodically reloaded by WebSphere Portal; new pages which are visible to unauthenticated users will not be available until the next reload occurs. Our environment assumes a low rate of change for pages, so we set this reload to only occur once per hour. In a production environment where new pages for unauthenticated users are rarely created, setting this reload time to an hour or more will give better performance. In a test or staging environment where updates to unauthenticated pages need to be seen more often, a lower reload time is more appropriate.

This service also controls the HTTP cache-control headers which will be sent on unauthenticated pages. While our environment did not exploit HTTP page caching, increasing these cache lifetimes in a production environment can reduce load on the portal. For more discussion of the use of HTTP cache-control headers with WebSphere Portal, refer to the "Caching" section of the "Tuning" topic in the WebSphere Portal V6.1 InfoCenter.

Table 4: Navigation Service Settings

NavigatorService.properties			
Parameter	***Default Value***	***Value Used***	***Definition***
public.expires (seconds)	60	3600	Determines cache expiration time for caches outside of WebSphere Portal and for unauthenticated portal pages only. If the setting remote.cache.expiration is also set to a value greater than or equal to 0, the smaller one of the two values is used.
public.reload (seconds)	60	3600	Determines cache expiration time for the portal internal cache for unauthenticated pages
remote.cache.expiration (seconds)	60	28800	Determines cache expiration for caches outside of portal server for authenticated as well as for unauthenticated pages

REGISTRY SERVICE

WebSphere Portal maintains information about many resource types in its databases. Some of these resources are replicated into memory for faster access; this is provided by the registry service. This replicated information will be periodically reloaded from the database, thus picking up any changes which may have been made on a peer node in a clustered environment.

The registry service allows configuring a reload time, in seconds, for each type of data which it is managing. In a production environment, we expect this type of information changes very infrequently, so we used very long reload times for the registry service. A full list of the types of information managed by the registry service is in table 4.

Table 5: Registry Service Settings

RegistryService.properties			
Parameter	***Default Value***	***Value Used***	***Definition***
default.interval	1800	28800	Reload frequency for any object types not explicitly specified in the file.
bucket.application.interval	600	28800	Reload frequency for application definitions
bucket.portlet.interval	600	28800	Reload frequency for portlet definitions
bucket.theme.interval	3000	28800	Reload frequency for theme definitions
bucket.skin.interval	3500	28800	Reload frequency for skin definitions
bucket.client.interval	19000	28800	Reload frequency for client definitions
bucket.markup.interval	20000	28800	Reload frequency for markup definitions
bucket.transformation application.interval	600	28800	Reload frequency for transformation application definitions
bucket.transformation.interval	600	28800	Reload frequency for transformation definitions

The cache manager service in WebSphere Portal is used to cache a wide variety of types of information in memory. These caches are somewhat similar to the registries maintained by the registry service, as each type of information gets its own cache. The key differences are:

- The information stored in the cache manager service's caches tends to be more dynamic than the information stored in the registry service's registries.
- The caches used by the cache manager service are limited in size, and entries will be discarded when the caches become full. The registries used by the registry service are not size-limited; they contain all entries of the specific data type.
- Expiry times are managed individually for each entry in the cache, managed by the cache manager service. In contrast, when the reload time is reached for a registry, the entire contents of that registry are reloaded.

Each cache has several configurable options. A full discussion of these options, along with a list of the caches in WebSphere Portal V6.1, is given in chapter 2. Table 5 lists the changes which we made to the cache manager service configuration file. Size values are specified in "number of objects" and lifetime values are specified in "seconds".

Table 6: Cache Manager Service Settings

CacheManagerService.properties		
Cache Name	***Default Value***	***Value Used***
com.ibm.wps.model.factory.ContentModelCache.live.size	1000	2500
com.ibm.wps.ac.ExplicitEntitlements Cache.USER_GROUP.size	1000	2000
com.ibm.wps.model.factory.Navigation SelectionModelCache.live.size	1000	2500
com.ibm.wps.ac.OwnedResourcesCache.enabled	true	false
com.ibm.wps.ac.ProtectedResourceCache.lifetime	5000	14400
com.ibm.wps.datastore.services.Identification.SerializedOidString Cache.size	2500	5000
com.ibm.wps.puma.DN_OID_Cache.size	500	5000
com.ibm.wps.puma.DN_User_Cache.size	500	3000
com.ibm.wps.puma.DN_Group_Cache.size	500	1500
com.ibm.wps.puma.OID_DN_Cache.size	1500	3000
com.ibm.wps.puma.OID_User_Cache.size	1500	3000
com.ibm.wps.puma.OID_Group_Cache.size	1500	5000
com.ibm.wps.ac.groupmanagement.NestedGroupCache.enabled	true	False
com.ibm.wps.ac.RolesCache.enabled	true	False
com.ibm.wps.ac.ChildResourcesCache.lifetime	7200	28800
com.ibm.wps.policy.services.PolicyCacheManager.lifetime	7780	43200

Database Tuning

DATASOURCE TUNING FOR DB2

Multiple databases are used to hold information in WebSphere Portal V6.1. We used six separate DB2 databases, each representing a separate database domain and having their own datasources. These are:

Table 7: DB2 Database Domains

Database	Database name	Datasource name
Release	`release`	`reldbDS`
Community	`community`	`commdbDS`
Customization	`custom`	`cusdbDS`
Feedback	`fdbkdb`	`fdbkdbDS`
Likeminds	`Lmdb`	`lmdbDS`
JCR	`jcrdb`	`jcrdbDS`

All datasources are configured in a similar manner by logging on to the WebSphere Application Server administrative console. For the prepared statement cache size, the path is Resources → JDBC Providers → provider name → Data Sources → datasource name. The provider name and datasource name are based on the names selected for that database during the database transfer step. Look for the parameter Statement cache size.

For the connection pool settings, the path in the WebSphere Application Server administrative console is Resources → JDBC Providers → Provider name → Data Sources → Datasource name → Connection Pools. The settings are Minimum connections and Maximum connections.

The default settings were used for the prepared statement cache size, and connection pool minimum and maximum sizes.

DB2 DATABASE SERVER TUNING

WebSphere Portal V6.1 uses database servers for core functionality. In our measurement environment, we used DB2 database server for the Portal application. The LDAP server, IBM Tivoli Directory Server also included a DB2 database as a repository, but it is largely unseen and was operated as an out of box configuration.

We recommend using a remote database server for the largest capacity. For our measurements we used IBM DB2 Enterprise Edition V9.1 fixpack 5 as our database server. WebSphere Portal V6.1 uses the concept of Database domains to designate either groups of tables belonging to one domain, or even entirely separate databases to store the data specific to each domain.

We built six separate databases within one database server to house the tables and data needed to support each domain. For the Base Portal and Many Pages measurements, the Release domain is the primary database being exercised.

The databases and related domains supported by Portal V6.1 are:

1. Release (release domain). This is the primary database domain used by the Base Portal and Many Pages Scenarios.

2. Customization (customization domain). This database receives some light traffic in our scenarios.

3. Community (community domain). This database receives some light traffic in our scenarios.

4. JCR (JCR domain). JCR database is used heavily in WCM (Web Content Management) Scenario. This database receives light traffic in all other scenarios measured in our Benchmark report.

5. Likeminds database, used for Likeminds enabled systems. This database is not used in the scenarios measured in our Benchmark report.

6. Feedback database, used by the feedback subsystem. This database is not used in the scenarios measured in this report.

DB2 ON AIX SETUP

We configure our DB2 database on AIX using the following setup,

- Set the filesystem which will hold the Portal databases to be a Enhanced Journal File System (JFS2) because a large file system is limited to 64GB.

- Turn on concurrent I/O (CIO) for Enhanced Journal File System as this improves performance.

  To enable CIO, use the following command to mount the database fileset.

  Mount –o cio /portaldb

- Increase AIX maximum number of processes per user to 4096.

  The default 500 processes per user is too low for database server, we increase it to 4096 in our AIX environment. To increase it,

  chdev –l sys0 –a maxuproc=’4096’

While the Portal databases are configured for high capacity performance, various tuning adjustments may be necessary from time to time. Typically these tuning needs are based on the volume of database traffic and the size of table populations.

Our database tuning settings is documented in the Portal Info Center under 'Creating Remote Database' section.

DB2 ON Z/OS SETUP

After transferring the database tables, first Identify what tables need to be reorganized.

Perform a re-org check to improve performance.

- Run the EJPSDBTC job after database transfer. This job contains the DB2 check and RUNSTATS utility for the JCR, Likemind and Feedback database.
- For details on re-org DB2 database, visit WebSphere Portal Info Center.

Create a Re-org job to re-org all table spaces in WPSDBJCR database.

RECOMMENDED DATABASE MAINTENANCE FOR DB2 LUW

Two of the database attributes, which DB2 relies upon to perform optimally, are the database catalog statistics and the physical organization of the data in the tables. Catalog statistics should be recomputed periodically during the life of the database, particularly after periods of heavy data modifications (inserts, updates, and deletes) such as a population phase. Due to the heavy contention of computing these statistics, we recommend performing this maintenance during off hours, periods of low demand, or when the portal is off-line. The DB2 runstats command is used to count and record the statistical details about tables, indexes and columns. We have used two techniques in our environment to recompute these statistics. The form we recommend is:

```
db2 runstats on table tableschema.tablename on all columns with
distribution on all columns and sampled detailed indexes all
allow write access
```

These options allow the optimizer to determine optimal access plans for complex SQL.

A simpler, more convenient technique for recomputing catalog statistics is:

```
db2 reorgchk update statistics on table all
```

Not only does this command count and record some of the same catalog statistics, it also produces a report that can be reviewed to identify table organization issues. However, we have found instances where this produces insufficient information for the

optimizer to select an efficient access plan for complex SQL, particularly for queries of the JCR database.

We have determined a technique that has the same convenience of the reorgchk command and provides the detailed statistics preferred by the optimizer.

```
db2 -x -r "runstats.db2" "select rtrim(concat('runstats on table
',concat(rtrim(tabSchema),concat('.',concat(rtrim(tabname),' on all
columns with distribution on all columns and sampled detailed indexes
all allow write access'))))) from syscat.tables where type='T'"
db2 -v -f "runstats.db2"
```

The first command is used to create a file, runstats.db2, which contains all of the runstats commands for all of the tables. The second command uses the db2 command processor to run these commands.

To determine which tables might benefit from reorganization, we use the command:

```
db2 reorgchk current statistics on table all  > "reorgchk.txt"
```

For those tables which require reorganization, we use the command:

```
db2 reorg table tableschema.tablename
```

to reorganize the table based upon its primary key.

You should also ensure that your database servers have adequate numbers of disks. Multiple disks allow for better throughput by the database engine. Throughput is also improved by separating the database logs onto separate physical devices from the database.

You should ensure that the database parameter *MaxAppls* is greater than the total number of connections for both the datasource and the session manager for each WebSphere Portal application server instance. If MaxAppls is not large enough, you will see exceptions in your connection pools.

You should use System Managed Storage (SMS) for temporary table spaces to benefit complex SQL which require temporary tables to compute their result sets. This saves time in buffer writes and improves disk utilization.

Database performance is very important for obtaining good performance from WebSphere Portal. The maintenance tasks and practices mentioned here were found to be critical to the performance and correct operation of WebSphere Portal in our lab environment. Additional database maintenance and tuning may be needed in your production environments. For further information on DB2 administration and tuning, refer to the DB2 Information Center.

ORACLE DATABASE SERVER TUNING

WebSphere Portal V6.1 uses database servers for core functionality. In this measurement environment, we used Oracle database server for the Portal application. The LDAP server, IBM Tivoli Directory Server included a DB2 database as a repository.

PLANNING FOR ORACLE ENTERPRISE EDITION DATABASE

For our Solaris platform measurements we also used Oracle 10*g* R2 as our database server. WebSphere Portal V6.1 uses the concept of Database domains to designate either groups of tables belonging to one domain, or even entirely separate databases to store the data specific to each domain.

On Oracle, we built a single database and create Oracle users to own the tables and data needed to support each domain. The domains are listed in PortalDatabaseDomain, above. For the Base Portal measurements, the Release domain is the primary database being exercised.

A well designed database can save a lot of trouble later down the road, and improve database performance. We recommend that you refer to the Oracle Administrator's Guide to help you make informed database design decisions. Here are the key choices we have implemented in our Oracle database.

- To avoid I/O contention and allow for better throughput, you should ensure your database server have adequate number of disks. Our database is on seven stripped disks.
- For better management and performance of storage structures, Oracle-Managed Files are used for database, as well as redo logs, and control files.
- Database block size: 8k
- The following tablespace sizing was required to support roughly a medium sized Portal, with 100,000 authenticated users, approximately 180 installed portlets and 220 pages, which the load generally consisting of database read operations. We recommend monitoring your tablespace sizing and growth on a regular basis. We used DBCA to create database with the following Tablespace size:
    - SYSAUX: 800MB
    - SYSTEM: 800MB
    - TEMP: 800MB
    - UNDOTBS: 1024MB
    - USERS: 2048MB
- Redo log groups: 500MB each.

ORACLE ON AIX SETUP

We configure our Oracle database on AIX using the following setup,

- Set the filesystem which will hold the Portal databases to be a Enhanced Journal File System (JFS2).
- Turn on concurrent I/O (CIO) for database filesystem as this improves performance. Do not enable CIO for Oracle product filesystem, ie, /u01, as Oracle could fail to start.

  To enable CIO, use the following command to mount the database fileset.

  ```
  Mount –o cio /u02
  ```

- Increase AIX maximum number of processes per user to 4096.

  The default 500 processes per user is too low for database server, we increase it to 4096 in our AIX environment. To increase it,

  ```
  chdev –l sys0 –a maxuproc='4096'
  ```

- Enable AIX async I/O, and increase MinServer to 5.

  ```
  smitty aio → Change/Show Characteristics of Async I/O →
  MinServers = 5
  ```

- We also set in oracle user's profile as Oracle Installation Guide for AIX recommends,

  AIXTHREAD_SCOPE=S

ORACLE ENTERPRISE EDITION DATABASE PARAMETER TUNING

Database performance is very important for obtaining good performance from WebSphere Portal. Below is a list of tuning applied on our Oracle database server with the alter system command. Additional database tuning maybe needed in your production environments. For further information on Oracle database tuning, refer to Oracle Performance Tuning Guide at http://www.oracle.com/technology/documentation/database10g.html.

Command used:

Alter system set <parameter> scope=spfile;

Table 8: Oracle Database Tuning

Parameter	*Value*
sessions	900
sga_target	1813M
pga_aggregate_target	604M
processes	750
open_cursors	1500
db_files	1024

RECOMMENDED ORACLE DATABASE MAINTENANCE

Optimizer statistics are a collection of data about the database and the objects in the database, these statistics are used by the query optimizer to choose the best execution plan for each SQL statement. Because the objects in a database can be constantly changing, statistics must be regularly updated so that they accurately describe these database objects, particularly after periods of heavy data modifications (inserts, updates, and deletes) such as a population phase. We have used the following commands in our environment to recompute these statistics:

```
execute
dbms_stats.gather_database_stats(dbms_stats.auto_sample_size,
method_opt=>'FOR ALL INDEXED COLUMNS SIZE AUTO',cascade=>TRUE);
```

OTHER DATABASE CONSIDERATIONS

WebSphere Portal maintains some information about users in its database tables, which grow when a user first logs in. We were interested in the steady-state performance of WebSphere Portal, not the performance of a user's first login to the site. Therefore our performance evaluates after all users logged in at least one time.

One of the most important database tuning factors is bufferpool sizing. It is important to evaluate the database's physical versus logical reads and size the bufferpools to achieve as much as a 95% logical read rate if possible.

Directory Server Tuning

Our measurements used IBM Tivoli Directory Server versions 6.0 as the directory server. These products use a database for storing user information; DB2 Enterprise Server was used for this database in our environment. This database is typically located on the same system as the directory server. If your workload involves creating, updating, or deleting users, then database maintenance described above may be needed on this database.

The following table shows the tuning values used for the directory servers in our Solaris Base Portal Scenario measurements

How-to-Set: These values are in the file /opt/IBM/ldap/V6.0/etc/SchemaV6.0/ibmslapd.conf. You must restart the LDAP server after changing these values.

Table 9: IDS Tuning

Parameter	Value
Ibm-slapdACLCacheSize	250000
Ibm-slapdEntryCacheSize	250000
Ibm-slapdFilterCacheSize	250000
Ibm-slapdFilterCacheBypassLimit	7500

The IBM Tivoli Directory Server uses IBM DB2 as the database server. The database instance and alias are named IDSLDAP. We applied the following tuning to this database:

```
db2 “update db config for idsldap using dbheap 4800”
db2 “update db config for idsldap using num_ioservers 10”
db2 “update db config for idsldap using num_iocleaners 5”
db2 alter bufferpool LDAPBP size 3690
db2 alter bufferpool IBMDEFAULTBP size 88500
```

Web Server Tuning

We used IBM HTTP Server 6.1 in our measurement environment. The cluster configuration and the Solaris configuration has a remote web server, find the tuning in Web Server Tuning in Cluster Tuning section. All other configurations have the web server running on the same system as the WebSphere Portal application server. If, during your monitoring, you notice insufficient processor capacity on the system when running the web server and the portal application server on a single system, consider separating the servers onto different systems. We used the following tuning on our web servers:

Table 10: Web Server Tuning

Parameter	AIX POWER5	Linux	Windows 2003	z/Linux	Additional Information
KeepAliveTimeout	5	5	5	5	This value is less than the think time defined in our scripts to ensure that testing is conservative. Each user is assumed to open a new TCP connection for each page view. However, in a live environment, it can be helpful to increase the KeepAlive Timeout. A higher timeout value can increase contention for HTTP server processes, if you are running out of HTTP processes, decrease this value.
ThreadsPerChild	25	25	2000	25	The higher number of threads per child on Windows is due to a different process model for IHS on Windows.
MaxKeepAliveRequests	0	0	0	0	Selecting 0 lets an unlimited number of requests on a single TCP connection.
MaxRequestsPerChild	0	0	0	0	
StartServers	2	2	N/A	2	
Access logging	off	off	off	off	This was turned off by commenting out the following configuration line: CustomLog /usr/HTTPServer/logs/access_log common
ThreadLimit	25	25	2000	25	
ServerLimit	150	120	N/A	180	Set it

					MaxClient/ThreadsPerChild.
MinSpareThreads	25	25	N/A	25	
MaxSpareThreads	3750	4500	N/A	4500	Set it same as MaxClients.
MaxClients	3750	4500	N/A	4500	

We also enabled the `server-status` module so that we could monitor the number of running and available Web server processes. This enables appropriate tuning of the `MaxClients` and `ThreadsPerChild` parameters.

We did additional Web Server tuning in Web 2.0 Scenario. See Web 2.0 section for details.

Note: For z/OS, no Web Server was configured.

Operating System Tuning

In any high-load environment, the network must be closely monitored to ensure that its performance is acceptable and consistent. Note that, the following is not to suggest that all network parameters are set to these values, but merely make the reader aware that the network is also an entity in the performance environment and bottleneck resolution process.

AIX

NETWORK TUNING

Use smitty->Performance and Resource Scheduling->Tuning Kernel and Network Parameters->Tuning Network Option Parameters->Change/Show Current Parameters to change. These will take effect immediately, improving the network layer performance in high volume environments.

Then remember to 'Save current parameters for Next Boot'.

Table 11: AIX Network Settings

Parameter	*Value*
tcp_sendspace	131072
tcp_recvspace	131072
udp_sendspace	65536
udp_recvspace	655360
somaxconn	10000
tcp_nodelayack	1
rfc1323	1

LINUX

NETWORK TUNING

For Red Hat Linux and z/Linux (Suse Linux on zOS), we add the following settings to file /etc/sysctl.conf, then run the command: sysctl -p

To inspect current TCP parameters, run the command: sysctl -a | fgrep tcp

Table 12: Linux Network Settings

Parameter	Value
net.ipv4.ip_forward	0
net.ipv4.conf.default.rp_filter	1
net.ipv4.conf.default.accept_source_route	0
net.core.rmem_max	16777216
net.core.wmem_max	16777216
net.ipv4.tcp_rmem	4096 87380 16777216
net.ipv4.tcp_wmem	4096 65536 16777216
net.ipv4.tcp_fin_timeout	30
net.core.netdev_max_backlog	3000
net.core.somaxconn	10000
net.ipv4.tcp_keepalive_intvl	15
net.ipv4.tcp_keepalive_probes	5

WINDOWS 2003

NETWORK TUNING

Use the `regedit` command, the following registry settings were made in the section `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters`. Create a new `REG_DWORD` named below.

Table 13: Windows Network Settings

Parameter	Value
MaxFreeTcbs	dword:00011940
MaxHashTableSize	dword:0000ffff
MaxUserPort	dword:0000fffe
TcpTimedWaitDelay	dword:0000001e
TcpWindowSize	dword:0000ffff (65535)

NETWORK TUNING

For Solaris, use the ndd command to set the following TCP layer parameters. These will take effect immediately, improving the network layer performance in high-volume environments. We use the following settings in Portal server running Solaris 10:

How-to-Set: `ndd -set /dev/tcp` **<PARAMETER> <VALUE>**

Table 14: Solaris Network Settings

Parameter	*Value*
tcp_time_wait_interval	60000
tcp_keepalive_interval	15000
tcp_fin_wait_2_flush_interval	67500
tcp_conn_req_max_q	16384
tcp_conn_req_max_q0	16384
tcp_xmit_hiwat	400000
tcp_recv_hiwat	400000
tcp_cwnd_max	2097152
tcp_ip_abort_interval	60000
tcp_rexmit_interval_initial	4000
tcp_rexmit_interval_max	10000
tcp_rexmit_interval_min	3000
tcp_max_buf	4194304

KERNEL TUNING

Our Portal Server is running on Solaris 10. In Solaris 10, we use the following '**projmod**' commands to set system parameters. After making the changes, we must logout then login to take these changes into effect. To examine your current settings, do 'cat /etc/project'.

```
projmod -s -K 'project.max-shm-memory=(privileged,4294967296,deny)' user.root

projmod -s -K 'project.max-shm-ids=(privileged,1024,deny)' user.root

projmod -s -K 'project.max-sem-ids=(privileged,1024,deny)' user.root

projmod -s -K 'process.max-sem-nsems=(privileged,4098,deny)' user.root

projmod -s -K 'process.max-sem-ops=(privileged,16384,deny)' user.root

projmod -s -K 'process.max-file-descriptor=(privileged,16384,deny)' user.root
```

SOLARIS CONTAINER

Use Solaris Containers to better utilize your modern, powerful T2 server with hundreds of virtual processors. In our lab, we use Processor Sets to partition virtual processors. We create a vertical cluster with six Portal members, then bind each member to a Solaris Processor Set, this configuration gives the optimum performance result.

The commands we use to setup,

1."pooladm –e" to enable pool facility

2."pooladm –s" to create a static configuration file that matches the current dynamic configuration

3."poolcfg –c 'create wp_pset1 (unit pset.min=20; unit pset.amx =21)'"

Create a processor set, named wp_pset1 or your choice, with between 20 and 21 processors. Create one per processor set.

4."poolcfg –c 'create pool wp_pool1'"

Create resource pool named wp_pool1 or your choice.

Create one per pool.

5."poolcfg –c 'associate pool wp_pool1(pset wp_pset1)'"

Join the pool and the processor set with an association.

Do this for each Processor set.

6."pooladm –c"

Commit the configuration at /etc/pooladm.conf.

7."poolbind –p wp_pool1 <PortalPID>"

Bind the resource pool to a Portal process.

Refer to IBM Redbook "IBM WebSphere Application Server V6.1 on the Solaris 10 Operating System", sg247584.

SYSTEM TUNING

In the PARMLIB member BPXPRMxx check the values of the following parameters:

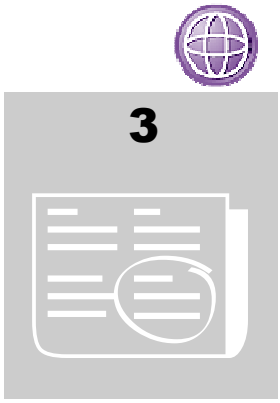
Table 15: z/OS System Tuning

Parameter	Value	Additional Information
MAXPROCSYS	15000	System will allow at most 15000 processes to be active concurrently.
MAXPROCUSER	15000	Allow each user (same UID) to have at most 15000 concurrent processes active.
MAXUIDS	200	Allow at most 200 z/OS UNIX users to be active concurrently.
MAXFILEPROC	65535	Allow at 65535 open files per user.
MAXPTYS	800	Allow up to 800 pseudo-terminal sessions
MAXTHREADTASKS	5000	System will allow at most 5000 threads tasks to be active concurrently in a single process
MAXTHREADS	10000	System will allow at most 10000 threads to be active concurrently in a single process.
MAXMMAPAREA	40960	System will allow at most 40960 pages to be used for memory mapping.
MAXFILESIZE	NOLIMIT	Unlimited file size.
MAXCORESIZE	4194304	
MAXASSIZE	2147483647	This size is same as the region size for TSO. By default USS ids get some pre-defined minimum which is usually not enough for WPS kind of stuff. To avoid problems instantiating java processes this size should be set to 214783647.
MAXCPUTIME	2147483647	To improve the CPU process time.
MAXSHAREPAGES	32768000	System will allow at most 32768000 pages of shared storage to be concurrently in use.

Required Fixes

The following fix is required to apply in WebSphere Portal Version 6.1 Solaris environment.

PK73368: Cache synchronization issue, deadlocks can occur in the CacheOnRequest class. This fix is included in WebSphere Portal 6.1.0.1 and later.



WEB 2.0 THEME TUNING

In the Web 2.0 theme environment a reverse proxy was used to cache content outboard of WebSphere Portal. The reverse proxy was set up to take advantage of the fact that portlet fragments are fetchable and cacheable. This avoids having to refetch the entire portal page in many cases. This allowed some content to be fetched without going to the web server or the portal server. Performance can be further improved by having the reverse proxy set up to gzip much of the content.

In general, the same tuning that was used for the Base Portal Scenario described in previous section was used for the Web 2.0 Scenario. The differences in tuning are mentioned below.

JVM Initial and Maximum Heap Size

JVM's Initial and Maximum heap size (ms and mx) were set to 1280. With higher values the system ran out of native memory under high Vuser load. This can be alleviated by using the – Xalwaysclassgc jvm parameter along with setting -Xmx=1408. However the throughput was better with -Xmx=1280 than when using – Xalwaysclassgc and –Xmx=1408.

Navigator Service Properties

The following values were specified in **NavigatorService.properties** in addition to the parameters changed in the Base Portal tuning.

Table 16: Navigation Service Settings for Web 2.0 Theme

Parameter	*Setting Used*
remote.cache.expiration.feed.cm	600
remote.cache.expiration.feed.nm	600
remote.cache.expiration.feed.lm	600
remote.cache.expiration.feed.pm	600

Internet Explorer Support of Vary Header

When Internet Explorer 7 is sent a 'vary' http header, it is unable to cache that reply effectively. To configure WebSphere portal to not send the vary header to IE 7, log in as portal administrator and navigate to Administration -> Portal Settings -> Supported Clients. Then select IE 7 as the browser and remove support for the 'vary' header.

Caching Proxy Tuning

The following are the settings and tunings specified in the reverse proxy's ibmproxy.conf file for the Web 2.0 performance test.

Table 17: Reverse Proxy Settings

Parameter	*Setting Used*	*Additional Information*
Proxy /wps/* http://{server-name}/wps/*		Proxy for /wps
Proxy /wps_semanticTag* http://{server-name}/wps_semanticTag* :80		Proxy for /wps_semanticTag
Proxy /searchfeed* http://{server-name}/searchfeed* :80		Proxy for /searchfeed
ConnThreads	15	
ServerConnPool	on	
MaxSocketPerServer	20	
CacheTimeMargin	5 seconds	
CacheFileSizeLimit	2 M	
flexibleSocks	off	
LimitRequestFieldSize	16384	
CompressionFilterEnable	C:\PROGRA~1\IBM\edge\cp\Bin\mod_z.dll	
CompressionFilter AddContentType	Image/bitmap,text/css,text/xml,application/xml	Compresses everything except text/html, application/atom+xml, text/plain, application/x-javascript. Portal compresses those types. Experiments were done where reverse proxy gzipped those files as well which caused the reverse proxy CPU to become a bottleneck. If a more powerful reverse proxy server was available, it might make sense to do all gzipping on the reverse proxy. Note that fixes for PMR 43866,499 were applied to Edge Server v6.02 to get proper gzipping behavior.

Web Server Tuning

Http server tuning for cacheability:

 # uncommented these to enable statics to be cached

```
LoadModule expires_module modules/mod_expires.so
LoadModule headers_module modules/mod_headers.so
```

 # from http://www.contentwithstyle.co.uk/blog/147 avoid gzip bug in IE 6

```
BrowserMatch ^Mozilla/4\.[0678] no-gzip
BrowserMatch \bMSIE\s7 !no-gzip !gzip-only-text/html
```

 # added this for caching of dojo javascript and the theme's xsl files, max-age = 1 day

```
<Location /wps/themes/dojo>
Header set Cache-Control public;max-age=86400
</Location>
<Location /wps/themes/html/PortalWeb2/xsl>
Header set Cache-Control public;max-age=86400
</Location>
# info for these directives at http://httpd.apache.org/docs/2.0/mod/mod_expires.html
# http://httpd.apache.org/docs/2.0/mod/mod_headers.html
```

 # set cache-control public for various static content

```
<FilesMatch "\.(gif|jpeg|jpg|png|ico|css|js|swf)$">
Header set cache-control "public"
</FilesMatch>
```

 # expire images after a month in the client's cache. Note that one month expiration worked fine for a performance evaluation in a test lab. It should be set appropriately for your environment where images might be updated more frequently than once a month.

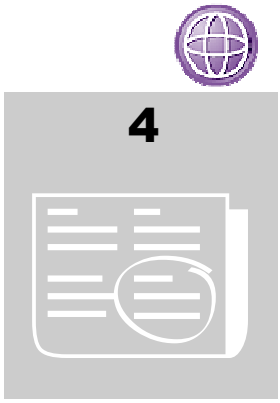
```
ExpiresActive On

ExpiresByType image/gif A2592000

ExpiresByType image/jpg A2592000

ExpiresByType image/jpeg A2592000

ExpiresByType image/png A2592000

ExpiresByType application/x-javascript "access plus 1 week"

ExpiresByType text/javascript "access plus 1 week"

ExpiresByType text/css "access plus 1 week"

ExpiresByType application/xml "access plus 1 week"

ExpiresByType application/vnd.mozilla.xul+xml "access plus 1 week"

ExpiresByType application/x-www-form-urlencoded "access plus 1 week"

ExpiresByType text/html "access plus 1 week"

ExpiresByType text/xml "access plus 1 week"
```

Portlet Caching

portlet.xml is part of a portlet's war file. It is located in the portlet's WEB-INF directory. To make portlet fragments publicly cacheable set:

a. `<expiration-cache>28800</expiration-cache>`

b. `<cache-scope>public</cache-scope>`



MANY PAGES TUNING

The “Many Pages Scenario”, derived from the Base Portal Scenario, measures the effects of having larger numbers of pages visible to the users.

Since it is derived from the Base Portal Scenario, the same tuning that was used for the Base Portal Scenario applied for the Many Pages Scenario. The differences in tuning are mentioned below.

DB2 Database Tuning

We applied the following tunings to our Release database.

Table 18: DB2 Database Settings for Many Pages

Release DB	
Parameter	***Setting Used***
dbheap	4800
applheapsz	4096
logbufsz	256
num_IOServers	8
num_IOCleaners	8

How-To Set: In the DB2 server run the following commands:

```
db2 “update db cfg for release using dbheap 4800”
db2 “update db cfg for release using applheapsz 4096”
db2 “update db cfg for release using logbufsz 256”
db2 “update db cfg for release using app_ctl_heap_sz 4096”
db2 “update db cfg for release using num_IOServers 8”
db2 “update db cfg for release using num_IOCleaners 8”
```

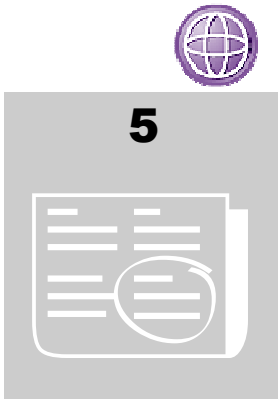
Cache Manager Service

Table 19: Cache Manager Service Settings for Many Pages

Parameter	Setting Used
com.ibm.wps.datastore.pageinstance.OIDCache.size	10000
com.ibm.wps.datastore.pageinstance.OIDCache.lifetime	28800
com.ibm.wps.datastore.pageinstance.DerivationCache.size	10000
com.ibm.wps.datastore.pageinstance.DerivationCache.lifetime	28800

Required Fixes

On WebSphere Portal 6.1, PK70946 is required in Many Pages Scenario. This fix is included in WebSphere Portal 6.1.0.1 and later.



WEB CONTENT MANAGEMENT TUNING

In general, the same tuning that was used for the Base Portal Scenario was used for the WCM authoring, rendering and API Scenario. The main differences are to the cache tuning settings: WCM increases demands on the portal access control component which requires a different set of cache tunings to accommodate and WCM has an internal set of object caches that can be tuned as well. On top of cache tunings, WCM can require more Web Container threads and JCR data source connections than the Base Portal Scenario, especially for heavy authoring workloads. The differences in tuning are mentioned below.

Application Server Tuning

- **Web Container Thread Pool** – we used 60 threads for both the minimum and maximum value
- **Data Source Connection Pool** – We used the following values:

Data Source	*Rendering Value (min/max)*	*Authoring/API Value (min/max)*
JCRDB	10/150	10/150

WebSphere Portal Service Properties

CACHE MANAGER SERVICE

Portal Caches sizes – Ignore the Base Portal values and set the following in CacheManagerService.properties:

Table 20: Cache Manager Service Settings for WCM

CacheManagerService.properties File	
Cache Name	**Value Used**
cacheinstance.com.ibm.workplace.searchmenu.helper.SearchMenuCacheHelper.size	5000
cacheinstance.com.ibm.wps.ac.ContainedRolesCache.size	500
cacheinstance.com.ibm.wps.ac.AccessControlUserContextCache.size	5000
cacheinstance.com.ibm.wps.ac.ApplicationRolesForPrincipalCache.size	12500
cacheinstance.com.ibm.wps.ac.AccessControlUserContextCache.lifetime	10800
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.size	500
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.ICM_CONTENT.size	12000
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.VIRTUAL.size	500
cacheinstance.com.ibm.wps.ac.ProtectedResourceCache.size	12500
cacheinstance.com.ibm.wps.ac.ExternalOIDCache.size	12000
cacheinstance.com.ibm.wps.ac.AccessControlUserContextCache.lifetime	10800
cacheinstance.com.ibm.wps.ac.RolesCache.size	7500
cacheinstance.com.ibm.wps.ac.groupmanagement.NestedGroupCache.size	1200
cacheinstance.com.ibm.wps.datastore.pageinstance.DerivationCache.size	250
cacheinstance.com.ibm.wps.datastore.pageinstance.OIDCache.size	250
cacheinstance.com.ibm.wps.datastore.services.Identification.SerializedOidString.cache.size	500
cacheinstance.com.ibm.wps.model.content.impl.ResourceCache.size	500
cacheinstance.com.ibm.wps.model.content.impl.TopologyCache.size	500
cacheinstance.com.ibm.wps.pe.portletentity.size	250
cacheinstance.com.ibm.wps.services.cache.cachedstate.CachedStateServiceSession Bound.cache.size	250
cacheinstance.com.ibm.wps.ac.ApplicationRoleChildrenCache.size	500
cacheinstance.com.ibm.wps.ac.ApplicationRoleDescriptorCache.size	500
cacheinstance.com.ibm.wps.ac.ApplicationRoleOIDCache.size	500
cacheinstance.com.ibm.wps.ac.ChildEntitlementsCache.size	500
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.APPLICATION_ROLE.size	500
cacheinstance.com.ibm.wps.policy.services.PolicyCacheManager.lifetime	28800
cacheinstance.com.ibm.wps.model.content.impl.ResourceCache.lifetime	14400

Portal Navigator Service – In addition to the settings mentioned for Base Portal we set the following property to allow public sessions required for rendering portlets on anonymous pages:

Table 21: Navigation Service Settings for WCM

NavigatorService.properties File			
Parameter	**Default Value**	**Value Used**	**Definition**
public.session	false	true	Controls whether anonymous users have sessions

WCM Object Cache

Table 22: WCM Object Cache Settings

WCM Object Caches	
Cache Name	***Value Used***
abspath	8000
abspathreverse	8000

processing	10000
session	6000
menu	500
nav	500
strategy	8000
global	100
module	100

How-To Set: Login to the WAS Administration Console → Resources → Cache instances → Object cache instances.

WCM Configuration Service

Enable the user cache

Find the WCMConfigService.properties file under:
`<wp_profile>/PortalServer/wcm/shared/app/config/wcmservices`

Set `user.cache=true`

JCR Text Search

icm.properties – Disable jcr textsearch

During our measurements, we have disabled text indexing. Text indexing is done periodically, adding new content to the text index. However, the indexing interval is not synchronized with our load plateaus. As a result, if we let text indexing run during our performance measurements, it would likely reduce the reliability and repeatability of our measurements.

We do not recommend disabling text indexing in production environments, as doing so would mean that new content will not be added to the text index, and therefore would not appear in search results.

If you wish to disable text indexing, this can be done in the file `icm.properties`, by setting the property `jcr.textsearch` to the value false. This file is found in the directory `<wp_profile>/PortalServer/jcr/lib/com/ibm/icm`.

DB2 Tuning (Authoring Environment)

MULTIPLATFORM (LUW)

On top of the DB2 tunings for the base portal scenario, during our testing we found that the following tunings to the JCR database below significantly decreased load on the CPU and disk i/o of the DB2 server in our environment.

In our authoring scenario we found that it was necessary to initially size the IBMDEFAULTP and ICMLSMAINBP32 bufferpools. This was because DB2 was unable to autosize them fast enough during our user ramp ups and it was therefore causing inconsistent results during the early stages of the scenario. We also noticed a large amount of database file handles being opened and closed during our runs stressing the disk i/o prompting us to increase the maximum number of file handles that can be opened for the JCR database. Finally, three indexes were added to eliminate some troublesome queries that were table scanning.

```
db2 connect to jcrdb
db2 alter bufferpool IBMDEFAULTBP IMMEDIATE size 26000
db2 alter bufferpool ICMLSMAINBP32 IMMEDIATE size 24000
db2set DB2_ASYNC_IO_MAXFILOP=512
db2 update db cfg for jcrdb using MAXFILOP 512
db2 create index taw_ut01590_idx6 on jcr.icmut01590001 (attr0000001334, itemid, versionid )
db2 reorgchk update statistics on table jcr.icmut01590001
db2 create index taw_entry_idx2 on jcr.ev_entry (parentid)
db2 reorgchk update statistics on table jcr.ev_entry
db2 create index taw_ICMSTJCRWSX_2 on jcr.icmstjcrws (basewsid, wstype)
db2 reorgchk update statistics on table jcr.icmstjcrws

db2stop force
db2start
```

The following section details the tunings that we made in our DB2 9 for z/OS backend database during our testing. To start here are a few general recommendations:

- When the DB2 z/OS server is on a different server to the Portal/WCM installation, the use of the Universal Driver type 4 database driver is recommended
- For data sharing groups, we additionally recommend enabling Sysplex Distributor to enhance high availability and exploit workload balancing.

In our environment we created nine databases to support WCM on Portal. The release, customization, community, likeminds, and feedback are required for Portal. For WCM, a minimum of two JCR databases are required for scalability and in our environment we used four.

TABLESPACES

Following DB2 best practices, it is recommended to create all tables into individual tablespaces. This will avoid device contention and provides better monitoring possibilities. Furthermore, most DB2 utilities such as REORG operate with tablespaces rather than tables.

BUFFERPOOLS

It is also beneficial to create separate bufferpools for use by Portal to avoid contention. When creating your database, ensure that each tablespace/indexspace has a specific bufferpool specified by the BUFFERPOOL/INDEXBP attributes rather than using the DB2 system defaults. It is recommended that a set of bufferpools separate from the Portal databases gets created for the JCR databases. The following table shows the settings for our configuration.

Table 23: DB2 z/OS Bufferpool Settings

Bufferpool settings				
Database Domain	**wkplc_comp.properties**	**DB2 BP**	**BP Pagesize (KB)**	**BP Size**
RELEASE CUSTOMIZATION COMMUNITY LIKEMINDS FEEDBACK	<domain>.Db4KBufferPoolName	BP2	4	40000
	<domain>.DbIndex4KBufferPoolName	BP3	4	5000
	<domain>.Db32KBufferPoolName	BP32K	32	1000
JCR	jcr.Db4KBufferPoolName	BP4	4	80000
	jcr.DbIndex4KBufferPoolName	BP5	4	40000
	jcr.Db32KBufferPoolName	BP32K1	32	20000

Note: When running Portal, DB2 objects like tablespaces will be created dynamically. It is important to keep your default bufferpools well defined to avoid causing contention due to an overloaded bufferpool. This is especially true for LOB and 4-KB tablespaces as they default

to BP0. In DB2 9 for z/OS, ZPARM's can be set to specifiy default bufferpools. In our environment we used the following values.

Table 24: DB2 z/OS Default Bufferpool Settings

Default Bufferpool Settings			
ZPARM	**Value Used**	**BP Size**	**Description**
TBSBPLOB	BP16K0	10000	Specifies the default buffer pool to use for LOB table spaces that are created implicitly and for LOB table spaces that are created explicitly without the BUFFERPOOL clause.
TBSBPOOL	BP4	80000	Specifies the default buffer pool to use for 4-KB page size table spaces that are created implicitly and for 4-KB page size table spaces that are created explicitly without the BUFFERPOOL clause.
TBSBP8K	BP8K0	35000	Specifies the default buffer pool to use for 8-KB page size table spaces that are created implicitly and for 8-KB page size table spaces that are created explicitly without the BUFFERPOOL clause.
TBSBP16K	BP16K0	10000	Specifies the default buffer pool to use for 16-KB page size table spaces that are created implicitly and for 16-KB page size table spaces that are created explicitly without the BUFFERPOOL clause.
TBSBP32K	BP32K	1000	Specifies the default buffer pool to use for 32-KB page size table spaces that are created implicitly and for 32-KB page size table spaces that are created explicitly without the BUFFERPOOL clause.
IDXBPOOL	BP5	40000	DB2 uses the IDXBPOOL value if you do not specify a value for INDEXBP on the CREATE DATABASE statement. DB2 does not use this value for a CREATE INDEX statement without the BUFFERPOOL option. In that case, DB2 uses the default index buffer pool for the database.

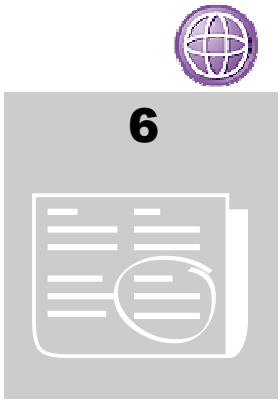
DB2 FOR Z/OS V8 FIXES

The following fixes are required for running WCM on WPS 610x on DB2 for z/OS v8.

- PK62728 - DB2 improves the performance of selected queries (UK36555)
- PK67292 - DB2 was fixed to transform the subquery into a join (UK37970)
- PK68259 - Addresses CORRELATED SUBQUERY TO JOIN TRANSFORMATION ENHANCEMENT FOR MULTI-TABLE JOINS IN ANY QUERY BLOCK (UK37971)

ADDITIONAL RESOURCES FOR DB2 Z/OS

- WCM/JCR database table usage information for WebSphere Portal v6
  http://www-01.ibm.com/support/docview.wss?uid=swg21255445
- Performance Improvement Possible For Remote TCP Access to z/OS
  http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/FLASH10621
- DB2 9 for z/OS Performance Topics
  http://www.redbooks.ibm.com/redbooks/pdfs/sg247473.pdf



COMPOSITE APPLICATIONS TUNING

For the Composite Application Infrastructure scenario, we started with the tuning given in the Base Portal Scenario above. However, the Composite Application Infrastructure scenario accesses a large number of applications, and therefore a large number of pages and portlets. Therefore there is higher demand on some of the caches in WebSphere Portal; to improve performance in this scenario, we modified the sizes of some of the caches in WebSphere Portal.

Cache Manager Service Properties

The following values were specified in CacheManagerService.properties in addition to the parameters changed in the Base Portal tuning. We recommend that you monitor the caches in your environment to determine which caches might need tuning.

Table 25: Cache Manager Serivce Properties for Application Infrastructure

Parameter	*Value*
cacheinstance.com.ibm.wps.resolver.friendly.cache.size	8000
cacheinstance.com.ibm.wps.ac.ProtectedResourceCache.size	10000
cacheinstance.com.ibm.wps.ac.PermissionCollectionCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.AccessControlUserContextCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.ProtectedResourceCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.OwnedResourcesCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.RolesCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.ParentResourceRoleMappingCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.ResourceRoleMappingCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.ChildEntitlementsCache.lifetime	28800

cacheinstance.com.ibm.wps.ac.SingleEntitlementsCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.CONTENT_NODE.lifetime	28800
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.WEB_MODULE.lifetime	28800
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.APPLICATION_ROLE.lifetime	28800
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.PORTLET_APPLICATION_DEFINITION!PORTLET_DEFINITION.lifetime	28800
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.USER_GROUP.lifetime	28800
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.VIRTUAL.lifetime	28800
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.WSRP_PRODUCER.lifetime	28800
cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.ICM_CONTENT.lifetime	28800
cacheinstance.com.ibm.wps.ac.ExternalOIDCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.ApplicationRoleOIDCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.ApplicationRoleDescriptorCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.ApplicationRolesForPrincipalCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.ApplicationRoleChildrenCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.ContainedRolesCache.lifetime	28800
cacheinstance.com.ibm.wps.ac.AuthLevelConfigurationCache.lifetime	28800
cacheinstance.com.ibm.wps.ai.rt.impl.service.ActiveApplicationOIDCache.lifetime	28800

Composite Applications Best Practices

As with other components of WebSphere Portal, the way in which teamspaces are used will influence the performance of the site. Based on our experience and analysis, the following tips can help you achieve better performance with teamspaces:

- As the measurements above show, the number of teamspaces on the system influences the capacity of the system. However, the key determining factor is not the total number of teamspaces, but rather the number of teamspaces visible to each user. For example, a site with 1,000 teamspaces visible to all users could have a capacity lower than a site with 5,000 teamspaces where the typical user can see no more than 50 of those.

- One way to exploit the observation above is to not make teamspaces public by default. Public teamspaces are visible to all users, so a large number of public teamspaces will increase the average number of teamspaces visible to each user.

- To paraphrase Albert Einstein, “keep teamspaces as simple as possible, but no simpler”. In implementing this, consider both the number of pages as well as the number of portlets on each page. Adding additional pages or portlets to a teamspace increases the overhead associated with that teamspace. While this is not a great overhead when considering an individual teamspace, when aggregated across thousands of teamspaces, the overhead can become significant. In addition, the more portlets are on each page, the more work will required to render that page.

- Another area to consider regarding teamspace complexity is the number of application roles. For many teamspaces, two roles (manager and user) are adequate. Don’t create additional roles unless they are really needed.

- When assigning membership to a teamspace, the best performance will be seen when membership is assigned by groups rather than by individual users. For example, WebSphere Portal will cache permissions based on the way permissions are assigned, giving the chance for more cache hits if permissions are assigned by groups.

- Memory demand increases with the number of teamspaces on the system. Therefore, if you expect to use large numbers of teamspaces, a 64-bit JVM will probably provide better capacity than a 32-bit JVM.

See also the section regarding “Use of Dynamic Content Features”. Experiments in our lab showed a reduced demand for memory, and thus an improvement in capacity, in the application infrastructure scenario when disabling dynamic content support.

CLUSTER TUNING

The Base Portal Scenario is measured in a three-node horizontal cluster environment, with or without session persistence, and six-members vertical cluster environment. In general, the same tuning that was used for the Base Portal Scenario was used for cluster. The additional settings are mentioned below:

Application Server Tuning

DYNACACHE CUSTOM PROPERTIES

There are several properties which can be set to reduce the number and size of Dynacache messages sent between nodes. This will improve scalability and reduce resource consumption in a clustered Portal environment. To set these properties, do the following:

1) Open and log in to the Administrative Console.
2) Click Application servers -> WebSphere_Portal -> Java and Process Management -> Process Definition -> Java Virtual Machine -> Custom properties -> New
3) Under General Properties, add the following information:
   Name: com.ibm.ws.cache.CacheConfig.ignoreValueInInvalidationEvent
   Value: true
   Name: com.ibm.ws.cache.CacheConfig.filterTimeOutInvalidation
   Value: true
   Name: com.ibm.ws.cache.CacheConfig.filterLRUInvalidation
   Value: true

Z/OS DYNACACHE CUSTOM PROPERTY

A custom property has been defined: `com.ibm.ws.cache.CacheConfig.propogateInvalidationsNotShared`, which when set to true leads to invalidations being sent for cache entry insertions and updates for a NOT_SHARED cache instance. This property should be removed on z/OS configurations as it has a major impact on performance.

THREAD POOLS

Increase Default Thread Pool size to help DRS traffic.

How-To Set: Portal Server->Thread Pools ->DefaultPool=150/150 (default=5/20)

TRANSPORT BUFFER SIZE

Default Transport Buffer size is insufficient. We increase to 200MB.

How-To Set:
Portal Server-> Core Group Service ->Transport buffer=200mb (default=10MB)

Must also configure the same size for Node Agent & DM.

System Administration -> Deployment Manager-> Core Group Service ->Transport buffer size=200

System Administration -> Node agents-> node_agent_name ->under Additional properties->Core Group Service ->Transport buffer size=200

WMM CONTEXT POOLING

Tuning Cluster for WMM Context Pooling must be done in Deployment Manager, then do full resync to each node from Administrative Console. See VMM Context Pooling on how to set.

Web Server Tuning

Table 26: Web Server Tuning for Clusters

Parameter	*Setting Used*
ThreadLimit	25
ServerLimit	180
StartServers	2
MaxClients	4500
MinSpareThreads	25
MaxSpareThreads	4500
ThreadsPerChild	25
MaxRequestsPerChild	0

Sample configuration:

```
<IfModule worker.c>
        ThreadLimit         25
        ServerLimit         180
        StartServers         2
        MaxClients          4500
        MinSpareThreads     25
        MaxSpareThreads     4500
        ThreadsPerChild     25
        MaxRequestsPerChild  0
</IfModule>
```

Session Persistence To Database Tuning

To enable Session Persistence to Database, a data source with non-XA JDBC driver must be created. We also configured DB2 Session Database with 32K page size to optimize performance for writing large amounts of data to the database. For details on configuring tablespace and page size for DB2 session database visit WebSphere Application Server Info Center. Additional tunings are applied below:

Table 27: WebSphere Session Persistence Tuning

Parameter	Setting	Additional Details
Session in memory count	***2000***	The default value of Session in memory count is 1000. For Session database persistence enabled load, we set session in memory count to 2000. **How-To Set Parameter:** In the WebSphere Administrative Console: Servers → Application Servers → WebSphere Portal → Container Settings: Session Management → Max in memory - Set Max in memory
Allow overflow	***disable***	The default value of session allow overflow is checked. For Session database persistence enabled load, we unchecked it. **How-To Set Parameter:** In the WebSphere Administrative Console: Servers → Application Servers → WebSphere Portal → Container Settings: Session Management -> Allow overflow -> uncheck it.
Session Distributed Environment	***Enable with database 32K page tablespace***	**How-To Set Parameter:** In the WebSphere Administrative Console: Servers → Application Servers → WebSphere Portal → Container Settings: → Session Management DistributedEnvironment Settings ->database Jndi name: jdbc/sessions (set it according to your Session datasource) Userid/password: set it according to your session db DB2 Row size: ROW_SIZE_32KB Tablespace name=sess_user32k (set it according to your db tablespace) Multiple row schema: uncheck it
ConnectionPool size for Session Data Source	***Min=10*** ***Max=33***	Refer to Datasource Tuning For DB2 on how-to set parameter.
Prepared Statement Cache size for Session Data Source	***15***	Refer to Datasource Tuning For DB2 on how-to set parameter.

SESSION DATABASE TUNING

In addition to creating bufferpool and tablespace to support 32K page size for Session database, we applied the following tunings to our dedicated session database server,

```
db2set DB2_USE_ALTERNATE_PAGE_CLEANING=ON
db2set DB2_RR_TO_RS=YES
db2set DB2_PARALLEL_IO=*

# disable session tablespace FILE SYSTEM CACHING
db2 alter tablespace sess_user32k NO FILE SYSTEM CACHING
db2 alter tablespace sess_temp32k NO FILE SYSTEM CACHING

db2 “update db cfg for <sess61> using locklist 5120”
db2 “update db cfg for <sess61> using maxlocks 80”
db2 “update db cfg for <sess61> using dbheap 4800”
db2 “update db cfg for <sess61> using num_iocleaners 20”
db2 “update db cfg for <sess61> using num_ioservers 20”
db2 “update db cfg for <sess61> using logbufsz 256”
db2 “update db cfg for <sess61> using logfilsiz 12288”
db2 “update db cfg for <sess61> using logprimary 40”
```

Vertical Cluster Tuning

We set the following in our vertical cluster environment,

- See Dynacache Custom Properties in Cluster Tuning section to reduce the number and size of Dynacache messages sent between JVMs. Additional DynaCache properties for Vertical Cluster:

  Name: com.ibm.ws.cache.CacheConfig.cacheEntryWindow **Value:** 10

  Name: com.ibm.ws.cache.CacheConfig.cacheInvalidateEntryWindow **Value**: 10

  Name:com.ibm.ws.cache.CacheConfig.propogateInvalidationNotShared**Value**: false

  Name: com.ibm.ws.cache.CacheConfig.useServerClassLoader **Value**: true

- See Transport Buffer Size in Cluster Tuning section to increase transfer buffer size.
- Increase Dynamic cache size to 3500.

  How to set: Portal Server -> Container Services -> Dynamic Cache Service -> Cache size = 3500

- See WMM Context Pooling on how to improve the performance of concurrent access to an LDAP server.
- Use the following command to increase DBHEAP for Release database.

  ```
  db2 “update db cfg for <release> using dbheap 4800”
  ```

IBM Tivoli Directory Server Tuning

The following table shows the tuning values used for the directory servers.

How-to-Set: These values are in the file /opt/IBM/ldap/V6.0/etc/SchemaV6.0/ibmslapd.conf. You must restart the LDAP server after changing these values.

Table 28: IDS Tuning in Vertical Cluster

Parameter	Setting Used
Ibm-slapdACLCacheSize	250000
Ibm-slapdEntryCacheSize	250000
Ibm-slapdFilterCacheSize	250000
Ibm-slapdFilterCacheBypassLimit	7500

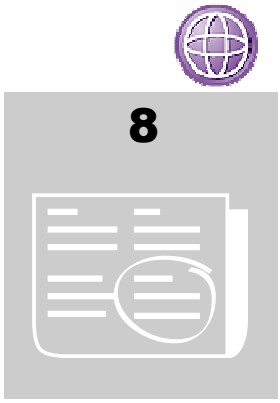
The IBM Tivoli Directory Server uses IBM DB2 as the database server. The database instance and alias are named IDSLDAP. We applied the following tuning to this database:

```
db2 “update db config for idsldap using dbheap 4800”
db2 “update db config for idsldap using num_iocleaners 5”
db2 “update db config for idsldap using num_ioservers 10”
db2 alter bufferpool IBMDEFAULTBP size 88500
db2 alter bufferpool LDAPBP size 3690
```

Required Fixes

- The following fixes are required to apply in 6.1 Cluster environment,
  - PK67324 (for Windows is PK67800)
  - WAS DynaCache PK67942
  - WAS DynaCache PK59026 (include PK62850 and prereq PK67942), to eliminate renounce messages
  - PK70944: to eliminate GroupCache invalidation
  - PK70890: Friendly URL fix
- The following fix is required to apply in 6.1.0.1 Cluster environment

  PK76988: Cluster performance Degradation by large amount of DRS invalidation messages



OTHER PERFORMANCE TUNING OPTIONS

In addition to the scenarios discussed above, WebSphere Portal has some other tuning options which may be useful in specific scenarios. These options include:

- Improving portal startup performance
- Managing the retrieval of user attributes
- Use of dynamic content features

Improving Portal Startup Performance

WebSphere Portal 6.1 introduced a “development mode” that greatly improves startup performance, so that WebSphere Portal will start more quickly. This can be very useful for development environments where WebSphere Portal must be stopped and started frequently.

However, it’s important to note that this mode is only meant to be used for development or test environments, not production or performance benchmark environments. Development mode turns on lazy-start for almost all applications in WebSphere Portal, and this can cause a performance impact the first time an application is accessed under load. Development mode also changes the way the JVM is started to give better startup speed at the cost of reducing capacity under load.

To switch to development mode, run the `enable-develop-mode-startup-performance` configuration task to complete the configuration and optimize the portal startup. The changes can be reverted to the original values using the `disable-develop-mode-startup-performance` configuration task.

For more information, please visit the following URL:
http://publib.boulder.ibm.com/infocenter/wpdoc/v6r1m0/index.jsp?topic=/com.ibm.wp.ent.doc/install/inst_opt.html

Managing the Retrieval of User Attributes

A user directory doesn't just contain a user's ID and password; it also contains a number of other pieces of information – attributes – about the user. A directory server can contain a lot of attributes for each user, so if every reference to a user required retrieving all of these attributes, this would impose a performance penalty on both the Portal server node(s) and the directory server node(s).

Therefore WebSphere Portal attempts to optimize the loading of these attributes. Two sets of user attributes are defined: the *base* set of attributes, and the *minimum* set of attributes. Depending on what action caused the user to be retrieved from the directory, either the base or the minimum set of attributes will be retrieved. Typically, the *base* set of attributes will be loaded when the user is retrieved; for example, this is what occurs when a user logs in. If the user was looked up when searching for users, then the *minimum* set of attributes will be loaded. For example, this can occur when searching for users to assign access to a page.

By default, WebSphere Portal defines the user attribute sets as follows:

- Base set: the following attributes are in the base set:
  - `uid`
  - `cn`
  - `sn`
  - `preferredLanguage`
  - `ibm-primaryEmail`
  - `givenName`
  - `displayName`
- Minimum set:
  - `uid`
  - `cn`

What happens if additional attributes are needed? For example, consider a portlet which requires the user attribute `countryName`. Assume that the user in question was looked up on login, so the base set of attributes was retrieved. The attribute `countryName` isn't in the base set, so the full user record – with all of its attributes – will be retrieved from the directory server at that point. This will require a second request to the directory server. Also, since all user attributes are retrieved on the second request, this can end up consuming more memory on the WebSphere Portal server.

This provides an important performance tuning point to both improve response times and reduce load on the directory server. If a user attribute will commonly be needed, then it should be included in the base set of attributes so that it's retrieved on the initial lookup, eliminating the need for a second request. However, if an attribute is only needed infrequently, consider leaving it out of the base set of attributes, so that it's not retrieved for all users.

IDENTIFYING A FULL FETCH OF USER ATTRIBUTES

How can you identify a second request is made to the directory server to retrieve the full set of user attributes? This is best done in a test or staging environment, rather than a live production environment, as it requires turning on tracing in the portal server, and this can impose a significant performance overhead. There are two traces to enable to look for this condition. The first one will show if the all the needed user attributes have been retrieved. If this is false, then a full fetch of the user information will occur. The second trace shows which attributes are being requested, so you can tell which ones should be added to the base set.

The two trace strings are:

```
com.ibm.wps.um.PrincipalImpl=all=enabled
com.ibm.wps.um.PumaProfileImpl=all=enabled
```

Enable those traces, and then execute the use case you wish to test. Then, look for this message in the trace.log:

```
PrincipalImpl 3 com.ibm.wps.um.PrincipalImpl isCompletelyLoaded false
```

This message may be output multiple times for the same user, so check all occurrences of it. If the value after `isCompletelyLoaded` is always `true`, then all the needed attributes have already been loaded, and no changes are needed. In this example, the value after `isCompletelyLoaded` is `false`, showing that the needed user attributes haven't all been loaded. This will result in reloading all the user information from the user directory.

In that case, the trace will then typically show a call to reload the information for that user; this will tell the full distinguished name of the user whose information is being loaded from the user directory:

```
PrincipalImpl > com.ibm.wps.um.PrincipalImpl reload ENTRY id: cn=Yin Yin_000_992,
cn=users,l=SharedLDAP,c=US,ou=Lotus,o=Software Group,dc=ibm,dc=com
```

Next, search above that in the trace for the getAttributes call, which will show the attributes the user has requested. It will look like this:

```
PumaProfileIm > com.ibm.wps.um.PumaProfileImpl getAttributes ENTRY id: cn=Yin
Yin_000_992, cn=users,l=SharedLDAP,c=US,ou=Lotus,o=Software Group,dc=ibm,dc=com

…more user details follow…
      isExternal: false[preferredLanguage, ibm-primaryEmail, countryName,
displayName, givenName, cn, sn, uid]
```

The last line of the log entry shows the attributes being requested. In this case, the attributes being requested are `[preferredLanguage, ibm-primaryEmail, countryName, displayName, givenName, cn, sn, uid]`. Comparing this against the list of base user attributes, we can see that `countryName` is not in the base user attributes. Depending on whether the action being executed is a common one or not, consider adding this to the base set of attributes.

Generally, the minimum set of attributes does not need to be modified from the default provided by WebSphere Portal, as that attribute set is satisfactory for the user management applications provided with WebSphere Portal. However, if your site contains a custom application for managing users and groups, and it uses attributes other than those in the minimum set, then you should consider expanding the minimum attribute set.

Use of Dynamic Content Features

WebSphere Portal contains dynamic content support infrastructure which supports two dynamic content features: dynamic user interfaces and attribute based administration. If neither of these features is being used, the dynamic content support can be disabled. Note that attribute based administration is only one use of the Personalization capabilities in WebSphere Portal; other uses of Personalization, such as placing content spots within a portlet, do not require the dynamic content features.

Disabling the dynamic content features will provide a modest performance benefit. It will provide a reduction in the memory needed for each user, and also reduce the processing time for generating pages in WebSphere Portal. For example, in one measurement with our Base Portal scenario, capacity improved about 5% when disabling the dynamic content support.

Disabling dynamic content support is done by adding a custom property to the `ConfigService.properties` resource provider. The property is:

```
content.topology.dynamic=false
```

See “Overview of configuration services” in the WebSphere Portal information center for more information on how to update configuration properties.

Real-World Network Considerations

In our lab environment, we had the luxury of our clients and servers being on the same LAN segment, so that they could take advantage of a high-bandwidth, low-latency network connection. However, this is typically not the case for real clients. Over a wide-area network, latencies can be significant, and bandwidth limited. In this case, the time to transfer the page content from the server to the client can become a significant contributor to overall page response time.

Here are some steps which can help alleviate this situation:

- Compress content on the HTTP server
- Allow client-side caching of images, Javascript files, and stylesheets,

Details on these steps will be given below.

COMPRESS CONTENT ON THE HTTP SERVER

Much of the content served by a WebSphere Portal site can be compressed to reduce transmission time and save network bandwidth. Typically, images should not be compressed (as they are usually stored in a compressed format), but other types of content can show a significant size reduction from compression.

IBM HTTP Server supports Deflate compression through the `mod_deflate` module. When it is enabled, the HTTP server checks the `Accept-Encoding:` header sent by the browser to see if it can accept a compressed version of the content. If so, the HTTP server will compress the content before sending it to the browser.

In one measurement, we observed an average of 65% network traffic reduction when Deflate compression is enabled. However, the compression operation does not come free as we also observed approximately a 10% processor utilization increase on the HTTP server.

To enable deflate compression in IBM HTTP Server, add the following lines in `httpd.conf`:

```
# compress everything but images
LoadModule deflate_module modules/mod_deflate.so
DeflateFilterNote Input instream
DeflateFilterNote Output outstream
DeflateFilterNote Ratio ratio

# Insert filter
SetOutputFilter DEFLATE
# Netscape 4.x has some problems...
```

```
BrowserMatch ^Mozilla/4 gzip-only-text/html
# Netscape 4.06-4.08 have some more problems
BrowserMatch ^Mozilla/4\.0[678] no-gzip
# MSIE masquerades as Netscape, but it is fine
BrowserMatch \bMSIE !no-gzip !gzip-only-text/html

# Don't compress images
SetEnvIfNoCase Request_URI \
\.(?:gif|jpe?g|png|exe)$ no-gzip dont-vary
```

ENABLING CLIENT-SIDE CACHING

The HTTP protocol allows the server to tell clients how long they can cache responses. When the client has the content in their cache, they do not need to request it again, saving the round-trip time to the server to retrieve the content.

This is done by adding Cache-Control: headers to the content which we wish to make cacheable. By default, WebSphere Portal will include these headers in the stylesheets it uses, making that content cacheable at a client for 5 days (432,000 seconds). It is possible to use `mod_headers` in IBM HTTP Server to add the same headers to images and JavaScript files by adding the following lines to `httpd.conf`:

```
LoadModule headers_module modules/mod_headers.so

<Location ~ "\.(js|gif|jpg|jpeg|png)$">
       Header add Cache-Control "public, max-age=432000, post-
check=172000"
</Location>
```

WebSphere. Portal

9

WEBSPHERE PORTAL CACHES

In the preceding chapter we described the specific values we modified for the WebSphere Portal caches in our environments. This chapter describes the WebSphere Portal caches, the general parameters for those caches, which cache instances WebSphere Portal v6.1 provides, and, finally, some sample portal usage patterns along with suggestions on portal cache properties.

General Information

With WebSphere Portal V6.1, portal configuration properties, including cache configuration properties, are managed via the WebSphere Application Server administrative console. In previous WebSphere Portal releases these configuration properties were maintained in properties files. More information on how to modify portal configuration properties can be found in the Setting configuration properties section of the WebSphere Portal Version 6.1 information center.

CACHE CONFIGURATION PROPERTIES

The cache configuration properties are organized in two groups: global configuration properties and cache instance specific properties. Global properties have the prefix `cacheglobal` and apply to all caches unless they are specifically overridden with a cache instance specific property. Cache instance specific properties have the prefix `cacheinstance` and then contain the name of the cache instance and the name of the property, for example:

cacheinstance.com.ibm.wps.ac.ExplicitEntitlementsCache.USER_GROUP.size

All entries of a cache are governed by a single set of properties.

The cache configuration properties that are safe to modify are: `enabled`, `lifetime`, `size`, `shared`, `replacement`, and `admit-threshold`. The `replacement` and

admit-threshold properties do not apply to all cache implementations. In general, only caches that are not shared will use these properties. There are other properties that should not be modified unless specifically instructed to do so by IBM WebSphere Portal support.

enabled: The enabled property determines whether a cache is used or not. If a cache is not enabled, the property has a value of false, then no values are held by the cache and every cache lookup will return a null value. This property should only be modified for testing purposes, never in a production environment. The supported values are true and false and the global default value is true.

lifetime: The lifetime property determines the number of seconds an entry will exist in a cache. A cache no longer returns an entry once the entry has existed longer than the lifetime property. Cache entries can also be invalidated prior to reaching their lifetime due to explicit invalidation of the entry or Least Recently Used (LRU) eviction from the cache.

A value of -1 indicates an infinite lifetime. This value should be used with caution since cache entries will only be invalidated programmatically. Infinite lifetimes are particularly discouraged with access control caches because:

- In a cluster there can be rare occurrences when not all cache invalidation messages are processed on every node due to race conditions in the application server's dynacache code. While the probability of this occurring is low, it can not be completely avoided with the current code base. Finite lifetimes allow these entries to be invalidated.
- Finite lifetimes allow modifications made to roles, which have been externalized to an External Security Manager, to be reflected in role caches.

If updates to production environments are restricted to a well-defined staging process using XML Access, it is usually safe to use infinite lifetimes.

size: The maximum number of entries in a cache is limited by the size property. If this size limit is reached, entries are removed from the cache by an algorithm which usually includes 1) remove invalidated entries and entries which have exceeded their lifetime and 2) apply a LRU algorithm to the valid entries.

Any positive integer is allowed. Cache sizes have a direct impact on the memory requirements of your portal, specifically the demands on the Java heap. You should monitor and record the Java heap metrics and any performance impact when modifying the size of a cache.

shared: Cluster-aware caches are *shared* across the nodes of a cluster. These caches propagate invalidations of cache entries by using the WebSphere Application Server *DistributedMap* interface.

Supported values are `true` and `false`. The default values shipped in WebSphere Portal V6.1 should apply to most configurations. If you do not have a cluster there may be a small performance benefit to setting this property to false since a different cache implementation is used. We did not modify the defaults in our single node measurement environments.

If this parameter is false in a cluster, it can ultimately lead to data inconsistencies between the cluster members.

`replacement`: The cache replacement algorithm used by these caches works on the frequency of recent access to cache entries; entries that have been used frequently are less likely to be discarded than entries that have not been used frequently. This parameter controls how long the access history will be kept. A setting of `aggressive` means those only recently accessed entries will be considered, which causes stale entries to be discarded more quickly. The opposite setting, `conservative`, will consider a longer access history. The intermediate setting of `moderate` is appropriate for most caches.

`admit-threshold:` Caches that have a very high insert rate may cause useful entries to be discarded prematurely. An admittance threshold restricts the rate at which entries are allowed into the cache by only allowing them to enter after an attempt has been made to insert the same entry into the cache multiple times. The default value of 0 means “no admittance threshold”, which will allow entries into the cache on the first insert attempt. This is appropriate for most caches. A higher value indicates that a cache entry will not be allowed into the cache until that many attempts have been made to insert the same key. For example, a value of 2 means that the first two attempts to insert a cache entry will be ignored, and the third attempt will insert the value into the cache. We did not modify the `admit-threshold` for any cache in our measurement environments.

Cache Usage Patterns

Most WebSphere Portal caches follow the simple paradigm: if an entry already exists use it, otherwise add the entry. However, there are caches that behave differently. Each cache follows one of the following four patterns:

- Pattern: *regular*

  The *regular* pattern, described earlier, is the most common cache pattern:

  ```
  value = cache.get(key);
  if (value == null) {
     value = calculateNewValue();
     cache.put(key, value);
  }
  ```

- Pattern: *invalidation checking*

  Invalidating cache entries in a clustered environment is rather expensive. Therefore, portal caches often check whether the entry to be invalidated actually exists in the local cache.

  ```
  test = cache.get(key);
  if (test != null) {
     cache.invalidate(key);
  }
  ```

  Caches following this pattern follow the *regular* pattern for all but invalidation actions.

- Pattern: *multiple object types*

  Most caches hold only a single object type. When caches can hold multiple types, they follow the *regular* pattern for each of those types.

- Pattern: *cascading object types*

  This pattern is a special case of the 'multiple object types' pattern in that two or more object types that are queried in a certain order are stored in a single cache. There may be one cache hit along with a cache miss on a regular basis.

  ```
  value = cache.get(keyA);
  if (value == null) {
     value = cache.get(keyB);
     if (value == null) {
        value = calculateNewValue();
        cache.put(keyA || keyB, value);   // either key could be used
     }
  }
  ```

Cache Instances

This section describes the caches in WebSphere Portal V6.1 along with hints to best configure those caches. As you saw in the modifications we made in our measurement environments, the `size` and `lifetime` properties are the most commonly modified properties when tuning portal caches. You may wish to increase the size of a cache if many values are used on a regular basis and there is sufficient space available in the Java heap. You may wish to increase the lifetime of the entries of a cache if the cached data rarely, if ever, changes and it is not critical to your business to reflect changes immediately in your portal.

Each cache description includes the following attributes:

- Default `size`, default `lifetime` and *cache usage pattern*
- Cache content and scaling factor (i.e. what causes the cache to grow)
- Information on the read and write access to the cache
- Approximate costs for re-creating cache entries and relative size of cached objects. Small objects range from 16 to 300 bytes and the largest cache entries are not larger than a few thousand bytes. One known exception are access control caches in systems with many resources per user can hold entries that are very large, beyond 50,000 bytes, to reflect all the resources which a user can access.
- Some cache descriptions include a sample scenario with suggested property values.

ACCESS CONTROL[1]

This section describes each of the access control caches. It is critical for proper operation of a portal that the access control information be current. Hence it is vital that these caches be shared within a cluster so that the information is propagated to all members of the cluster. Different lifetime values should be chosen to avoid concurrent reload of information from multiple caches. This pattern of rather random lifetime and invalidation intervals could also be applied to other caches.

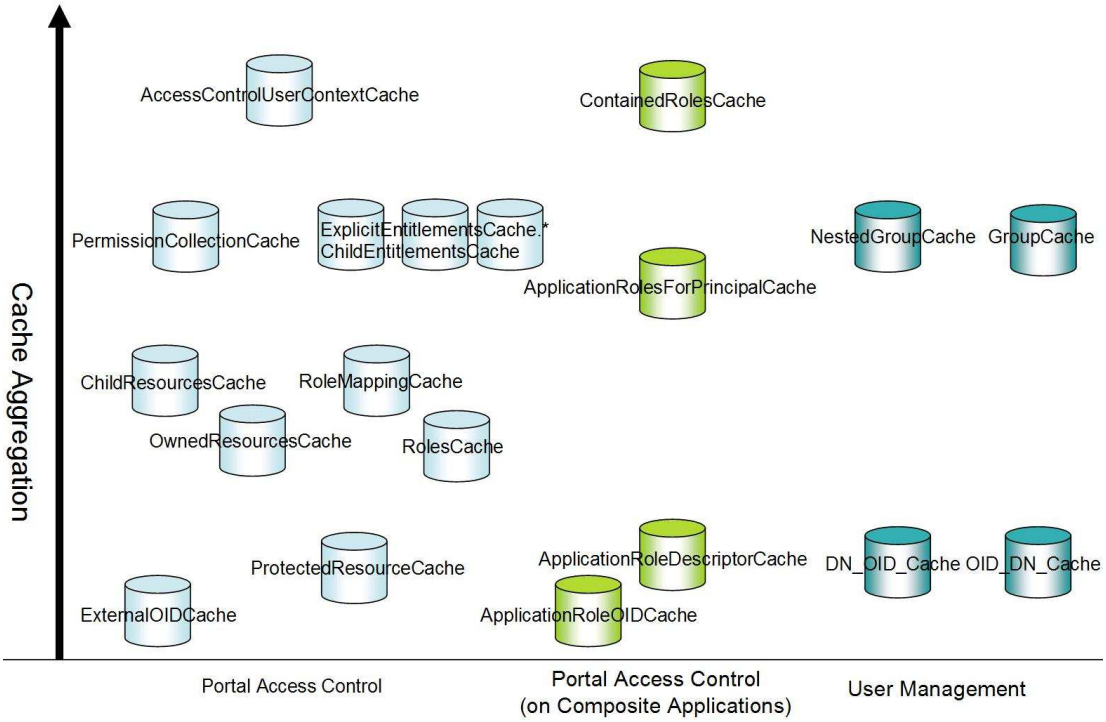
The access control caches are divided into two groups: those caches (the first caches in the list) used during all access control operations in all portal setups and those caches (starting with the cache `com.ibm.wps.ac.ApplicationRoleOIDCache`) used for the WebSphere Portal Composite Application Infrastructure.

[1] This section is partially taken from another whitepaper: Portal Access Control Performance Tuning: http://www-128.ibm.com/developerworks/websphere/library/techarticles/0508_buehler/0508_buehler.html

Figure 1 shows the relationships among the various caches. The small cylinders represent cache instances. The green caches are caches of the portal user management (PUMA) component that are closely related to the caches of the portal access control component. The PUMA caches contain information originating from the user registry. Portal access control uses these caches for user identification and group membership retrieval.

The vertical axis represents the cache aggregation direction. The cache instances in *layer N* leverage cache instances of lower layers to compute their values. For example, when computing effective permissions (entitlements) for a user (cached in the ExplicitEntitlementsCache), the portal access control component leverages existing cache values from the ChildResourcesCache and RoleMappingCache.

Figure 1 Portal Access Control Cache Hierarchy



com.ibm.wps.ac.PermissionCollectionCache

Default size: 2000, default lifetime: 10240, usage pattern: regular (admit-threshold).

This cache contains permission collections that can be used for permission checks. It scales with the number of permissions in the system, i.e. the number of portal resources and permissions assigned on those. Entries in the cache typically are requested very frequently during permission checks. An admit-threshold is used to avoid caching rarely used permissions. You may wish to try different admit-threshold settings to tune this cache. Entries are never invalidated from the cache. Creating a cache entry is very fast since all required information is in-memory. A cache entry is small.

com.ibm.wps.ac.AccessControlUserContextCache

Default size: 8000, default lifetime: 1200, usage pattern: regular.

This cache contains the access control user context objects, a local cache for permissions assigned to a specific user. If possible all requests against access control are answered using this information so that access control methods can return very quickly. This cache scales with the number of active users. For fast portal operation, you should make sure that the entries for all actively working users fit into the cache, especially if a user has access to many portal resources. Entries are invalidated from the cache upon any portal administrative action. Creating a cache entry typically is rather cheap because most information is in-memory, but can take a while if the required information cannot be found in other caches. An entry in the cache can be become very large, depending on the number of resources the user can access.

com.ibm.wps.ac.ProtectedResourceCache

Default size: 5000, default lifetime: 10143, usage pattern: regular.

This cache contains the resources protected by portal access control. The size of this cache scales with the number of protected resources accessed by the active users in the system. Entries are read from the cache during every permission call or entitlements call against access control. Entries are invalidated from this cache during resource deletion, resource relocation, modification of the resource state (private/shared), modification of the resource owner, externalization, internalization, and role block change. Creating a cache entry requires a single-row lookup in the portal database. An entry in the cache is relatively small.

com.ibm.wps.ac.OwnedResourcesCache

Default size: 5000, default lifetime: 10043, usage pattern: invalidation checking.

This cache maps resource owners (user groups or individual users) to the resources they own. This cache scales with the number of active users/groups multiplied with the different ResourceTypes they access. There is one entry in the cache per principal per resource type per WebSphere Portal domain. Data is read from this cache during many portal access control requests, if the corresponding entitlements are not already cached in an entitlements cache. Entries are invalidated from this cache during resource deletion, modification of the resource owner, externalization, and logout of the user. Creating a cache entry means executing a multi-row query against the portal database. An entry in the cache is relatively small.

com.ibm.wps.ac.RolesCache

Default size: 10000, default lifetime: 3630, usage pattern: invalidation checking.

This cache contains the role instances. The size of this cache scales with the number of active users/groups multiplied by the different ResourceTypes they access. There is one entry per role instance per principal per resource type per WebSphere Portal domain. Data is read from the cache during many portal access control requests, if the

corresponding entitlements are not already cached. Entries are invalidated from this cache during role mapping creation, role mapping deletion, resource deletion, externalization, internalization, and logout of the user. Creating a cache entry means executing at least one, but potentially multiple database queries. An entry in the cache is relatively small.

com.ibm.wps.ac.ExplicitEntitlementsCache.* and com.ibm.wps.ac.ChildEntitlementsCache

Default size: 10000, default lifetime: varying (around 10000), usage pattern: invalidation checking.

These caches contain the permissions of a user or group on a number of resources of the same ResourceType. There are dedicated caches for the different ResourceTypes. For example, the cache for pages is called `com.ibm.wps.ac.ExplicitEntitlementsCache.CONTENT_NODE`. All ResourceTypes that are not specified explicitly will be cached in the default cache. The size of this cache scales with the number of active users/groups multiplied by the different ResourceTypes valid for this cache and accessed by the users and groups, either by 'using' the resource during navigating the portal or by portal administration. There is one entry per set of permissions per WebSphere Portal domain. Entries are read during 'regular' access control requests, during page rendering and, especially, during portal administration. If a certain resource type is not used, you will see only misses and no other activity on the corresponding cache. Entries are invalidated from this cache during all access control modifications and logins. Creating an entry in one of these caches typically can be done from in-memory information in the lower-level caches. If the required information is not available multiple database requests might be required to create a cache entry. An entry into the cache is rather small, but built of multiple objects typically stored in other caches.

com.ibm.wps.ac.ExternalOIDCache

Default size: 10000, default lifetime: 8640, usage pattern: regular.

This cache contains the mapping between the external ObjectIDs of individual protected resources, for example page or portlet IDs, and the portal access control specific ObjectIDs stored in the database table PROT_RES. Entries are read from the cache during many portal access control requests. The size of this cache scales with the number of protected resources accessed by the active users in the system. Since this mapping is immutable, this cache is never explicitly invalidated. Creating a cache entry requires a single row database query. An entry in the cache is fairly small.

com.ibm.wps.ac.groupmanagement.NestedGroupCache / com.ibm.wps.ac.groupmanagement.GroupCache

Default size: 1000, default lifetime: 3600, usage pattern: regular.

Only one of these two caches is used in a WebSphere Portal installation depending on your 'nested groups' setting. If nested groups are supported, the NestedGroupCache cache will be used, otherwise the GroupCache is used. The caches contain the nested

or direct groups to which a user belongs. The size of this cache scales with the number of active users and the number of virtual portals they access. The cache is accessed during login into portal, but typically not during regular portal navigation. Its main use case is during administration of users and user groups. Entries are invalidated from this cache during login of the user and after user and group administrative changes. Creating a new cache entry requires queries against the WMM component and then typically against the user repository. An entry in the cache is medium-sized.

com.ibm.wps.ac.ChildResourcesCache

Default size: 1000, default lifetime: 7200, usage pattern: regular.

This cache contains the resource hierarchy within portal access control. The size of this cache scales with the number of protected resources accessed by the active users in the system, like the protected resources cache. This cache does not contain leaf objects in the access control tree, so the number of entries typically is smaller. The cache is accessed during most portal access control requests. Entries are invalidated from this cache during resource deletion, parent change of the resource, modification of the resource owner, externalization, internalization, and role block change. Creating a cache entry includes a multi-row query against the portal database. An entry in the cache is fairly small.

com.ibm.wps.ac.ApplicationRoleOIDCache

Default size: 5000, default lifetime: 7650, usage pattern: regular.

This cache maps application role names to the corresponding object IDs. It scales with the number of application roles defined in the system. Data is read from the cache frequently when accessing or administering composite applications. In all other situations the cache is basically not used at all. Entries are invalidated when application roles are deleted. There is one entry in the cache per application role per WebSphere Portal domain, except for the customization domain. Creating a cache entry means reading a single row of data from the portal database. A cache entry is fairly small.

com.ibm.wps.ac.ApplicationRoleDescriptorCache

Default size: 5000, default lifetime: 8450, usage pattern: regular.

This cache maps the object ID of an application role to its corresponding descriptor object, which contains the application name, parent application and other information. The cache scales with the number of application roles defined in the system. Data is read from the cache frequently when accessing or administering composite applications. In all other situations the cache is basically not used at all. Creating a cache entry means reading a single row of data from the portal database. A cache entry is medium-sized.

com.ibm.wps.ac.ApplicationRolesForPrincipalCache

Default size: 5000, default lifetime: 8760, usage pattern: regular.

This cache maps the available application roles to a portal user. It scales with the number of active users in the system. Data is read from the cache frequently when accessing or administering composite applications. In addition this cache is also used as a lookup for application role information even if no application roles are used. Hence you will see frequent read access on this cache under all circumstances. Creating a cache entry is rather expensive. It involves three multi-row queries against three WebSphere Portal domains. A cache entry is medium-sized.

com.ibm.wps.ac.ContainedRolesCache

Default size: 5000, default lifetime: 8650, usage pattern: regular.

This cache contains the mappings between application roles and the 'regular' roles defined in them. The cache scales with the number of application roles in the system. There is one entry for every WebSphere Portal domain. Data is read from the cache frequently when accessing or administering composite applications. In all other situations the cache is basically not used at all. Creating a cache entry is rather expensive. It involves two multi-row queries. A cache entry is medium-sized.

com.ibm.wps.ac.ApplicationRoleChildrenCache

Default size: 5000, default lifetime: 8760, usage pattern: regular.

This cache is not used in WebSphere Portal V6.0.

com.ibm.wps.ac.ParentResourceRoleMappingCache and com.ibm.wps.ac.ResourceRoleMappingCache

Default size: 1000, default lifetime: infinite, usage pattern: regular.

These two caches are used for special access control scenarios and typically are not accessed during portal processing. Settings of these caches should not be modified

PORTAL USER MANAGEMENT

The following caches are used by the portal user management component (PUMA). In so far they are closely related to the access control caches and caching within WMM.

com.ibm.wps.puma.DN_OID_Cache / com.ibm.wps.puma.OID_DN_Cache

Default size: 1500, default lifetime: infinite, usage pattern: regular.

These two caches contain the mapping between the distinguished name of users and groups and their internal ObjectID identifier. The size of these caches scales with the number of active users and groups or users and groups that are used for delegation. Entries are invalidated from this cache during deletion of a user or group. Creating an entry requires one database lookup. An entry into the caches is fairly small.

The datastore caches contain data read from the portal database. It is not the goal of these caches to be a complete image of the DB content, but to have frequently-accessed but raw information available for all other portal components to use.

com.ibm.wps.datastore.services.Identification.OidAndUniqueName.cache

Default size: 5000, default lifetime: infinite, usage pattern: regular.

This cache stores unique names. It is used quite frequently during page rendering and especially administration of unique names. Page and portlet unique names make up the biggest part of the cache content. The cache should be large enough to hold entries for the most frequently used pages and portlets having a unique name associated with them. Note that not all resources have a unique name associated with them. To eliminate database lookups the cache size could correspond to the database table UNIQUE_NAME multiplied by two, to allow for mapping in two directions. Creating a cache entry involves reading one entry from the portal database. An entry object into the cache is fairly small.

com.ibm.wps.datastore.PortalIdCache.vpPerLpid.cache

Default size: 1000, default lifetime: infinite, usage pattern: regular.

This cache maps long Virtual Portal object IDs to the corresponding portal internal short ID. It scales with the number of virtual portals in the system, plus one additional entry. It is used heavily only if more than one virtual portal exists in the system. Data is read from the cache during every rendering request then. For optimal caching the size should be set to the number of Virtual Portals defined in the system. Creating a cache entry involves one single-row database lookup. An entry object into the cache is fairly small.

com.ibm.wps.datastore.PortalIdCache.explicitLpidPerVP

Default size: 100, default lifetime: infinite, usage pattern: regular.

This cache maps the short object ID for a virtual portal to the corresponding long ID. In comparison to cache com.ibm.wps.datastore.PortalIdCache.vpPerLpid.cache it stores the reverse mappings. Hence all other descriptions given above also apply here.

com.ibm.wps.datastore.pageinstance.OIDCache

Default size: 3000, default lifetime: infinite, usage pattern: regular.

This cache stores information on portal pages for fast retrieval during login or page navigation. It scales with the number of page instances in the system. It is one of the most frequently used caches and should be large enough to hold all pages that are frequently accessed by users. Pages are loaded and put into the cache by direct navigation, creating a link to another page or by working with the page during portal administration (always including all higher derivation levels). Creating a cache entry includes one single-row database lookup. An entry to the cache is medium sized. To

achieve best performance, in terms of cache hit rate, the size should be set to a value so that all pages defined in the system fit into the cache. This corresponds to the row count of the database table PAGE_INST.

com.ibm.wps.datastore.pageinstance.DerivationCache

Default size: 3000, default lifetime: infinite, usage pattern: regular.

This cache stores the mappings between pages and their derivation children, or empty mappings if no such children exist. Like the pageinstance.OIDCache cache this one also is accessed very frequently during page rendering and administration. Creating a cache entry involves one multi-row database query. This cache also scales with the number of pages in the system. Hence, you can use the same sizes for the previous cache and this one. In most portal usage scenarios the actual size of this cache will be somewhat lower than with the page instance cache. An average entry in the cache is rather small. Only if all your pages have long lists of derivation children will the entries become larger. To achieve best performance, in terms of cache hit rate, the size should be set to a value so that all pages defined in the system fit into the cache. This corresponds to the row count of the database table PAGE_INST.

com.ibm.wps.datastore.pageinstance.DynamicNodeCache

Default size: 5, default lifetime: infinite, usage pattern: regular.

This cache stores one list per domain. These lists contain all pages in the corresponding domain that are flagged as dynamic nodes, i.e. dynamic assembly content nodes can be added below these pages. Since the number of domains does not grow, the size as well as other properties of this cache should not be modified. The size of one entry into the cache ranges from small in a portal with very few dynamic nodes up to medium with many dynamic nodes in the system.

com.ibm.wps.datastore.services.Identification.SerializedOidString.cache

Default size: 2500, default lifetime: infinite, usage pattern: cascading object types.

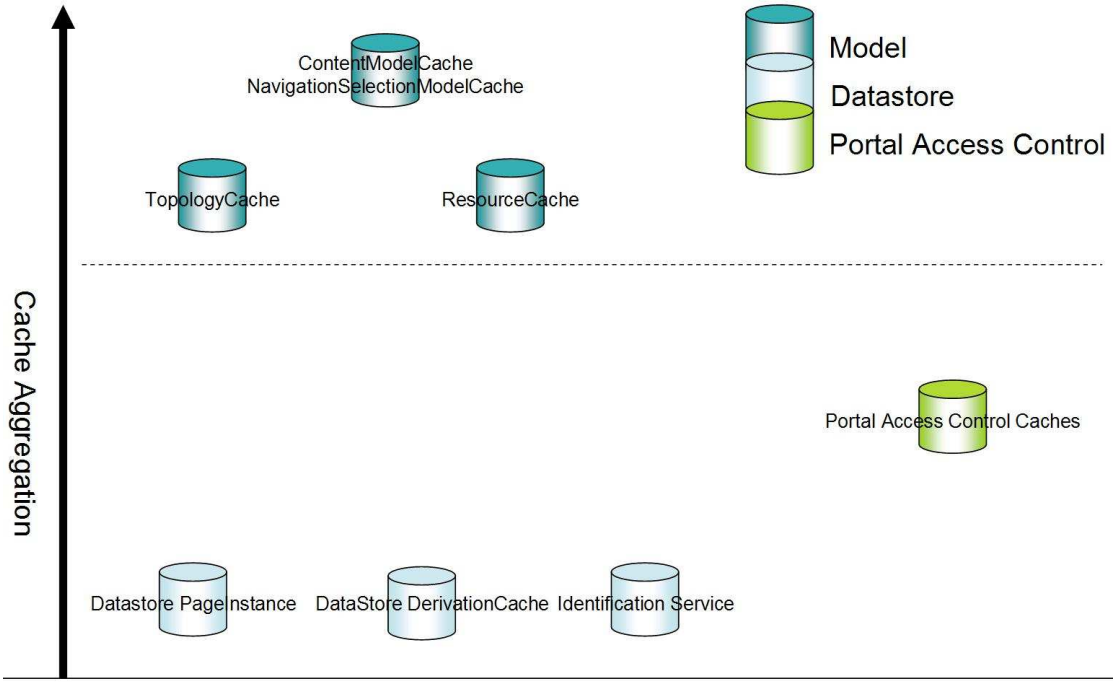
This cache stores serialized ObjectIDs used in request parameters or XML Access files. It contains a subset of all the loaded ObjectIDs in memory. In so far it scales with the number of ObjectIDs in the system, but not for all of these IDs the serialized version is requested, hence the actual size is impossible to predict. The cache is used during every request. Creating a cache entry is rather cheap. Typically all information can be retrieved in memory, database lookups are scarcely necessary. A cache entry is fairly small.

MODEL

The model caches can be categorized into two groups: One group of caches is accessed during every portal request during page rendering. The second group of caches is especially important for administrative actions. Hence those caches are especially important in those environments where content and portal administration is done. Most run-time caches have the name suffix live; the administrative caches have the suffix isolated.

Figure 29 describes the hierarchy of caches in the model component and depending portal components. The structure of the picture is identical to figure 28: The vertical axis shows caches with increasing aggregation of data. The model component only caches data at a rather high aggregation level. All data cached here hence is rather valuable, reloads can be expensive if the corresponding data is not available in the lower-level caches. Model caches are dependent upon the datastore and portal access control caches. The figure only features the most important caches.

Figure 2 Portal Model Cache Hierarchy



com.ibm.wps.model.factory.SimpleCacheKey

Default size: 2500, default lifetime: infinite, usage pattern: regular.

This cache is a helper cache for other model caches used by the portal model factory. It contains a small number of entries based on the model types available in portal. In addition there can be one entry per active user session. The size of this cache might be adapted to the number of active sessions in one portal JVM. Re-creating a cache entry is a rather cheap operation since it usually can be accomplished in memory. A cache entry is a small object.

com.ibm.wps.model.content.impl.ResourceCache

Default size: 5000, default lifetime: 5600, usage pattern: regular.

This cache contains aggregated pages. In contrast to the data store page instance cache this cache contains the complete models of pages and their content, i.e. the portlets and containers on them. The page instance cache rather holds the raw page data. This cache scales with the number of pages defined in your portal and the different

sets of access control rights on these pages. This cache contains very 'valuable' information; it utilizes several other caches, for example, page instance and access control caches, to build its data. Hence creating a cache entry usually only requires in-memory information, but can also lead to many database queries. The size of an entry in the cache depends on the complexity of the pages, but typically the objects are medium-sized, since they are usually made of references to other cached data. The cache should be large enough to hold the most frequently accessed pages multiplied with the number of different access control settings on these pages. Increasing the cache lifetime can be useful if page definitions do not change often in your environment.

Example: A portal has 500 pages and all users have the same permissions on these. In addition there are another 50 pages; two groups of users have different access rights on these pages. In this case a maximum of 600 entries would be in the cache.

com.ibm.wsp.mode.content.impl.TopologyCache

Default size: 10000, default lifetime: 5700, usage pattern: regular.

This cache contains portal topology information, i.e. portal navigation elements being composed of navigation nodes and their sorted, access control-filtered children. Topology elements undergo several processing steps from first loading from the database until finally being added to the cache. This cache only contains the completely processed topology entities. This cache is explicitly used during login and whenever a user navigates to a part of the portal where he has not been before during the same session. If a cache entry is not found, a private copy is created that is then further processed. Once the private copy is completely processed -that does not happen for all navigation nodes- it is added to the cache. If a user finds an entry in the cache a reference is copied into his private topology model and additional cache accesses are no longer necessary. Hence there is only one cache hit (or miss) per user and navigation node. The cache scales with the number of navigation nodes and the number of different sets of permissions on these and, possibly, the derivation chain (children and parents) a page belongs to. Entries in this cache are expensive to create; they rely on other cached information, like the access control caches and the page instance cache. The entries in the cache are medium-sized, being mainly some lists of references to other cached data. The cache should be sized in a way such the most important pages multiplied with all the different sets of permissions that exist on theses page can be stored.

com.ibm.wps.model.factory.ContentModelCache.live

Default size: 1000, default lifetime: infinite, usage pattern: regular.

This run-time cache contains the content models for portal users. There is one entry per active portal user. The cache should be large enough to hold all models for these users. An entry in the cache has the maximum lifetime of the corresponding user session, i.e. entries are removed at the end of the session. Creating a cache entry can be very expensive. Typically all required information is in memory, but accessing the database, also many times, might be necessary if underlying information is also no longer cached. Furthermore the number of pages summarized in the model can be very large which

also adds to the time it takes to rebuild a cache entry. Building the content model is done incrementally as required for the current request; the model is not built at once. Depending on the size of the model also the memory requirements vary. The more pages a user can access and has accessed already during the current session the larger the cache entry, ranging from medium to very large. A cache entry typically is composed of references to other cached and shared objects. Hence an entry size is not made up by the number of page and all subordinate objects but only contains references to these.

com.ibm.wps.model.factory.ContentModelCache.isolated

Default size: 1000, default lifetime: infinite, usage pattern: regular.

This cache contains the administrative content models. There is one entry for every user doing administrative work at a certain point in time. In so far the number of entries in this cache typically is much lower than in the other cache. But for this cache you should make sure that no cache entries of active users are evicted. Compare with the content model run-time cache for all other information.

com.ibm.wps.model.factory.NavigationSelectionModelCache.live

Default size: 1000, default lifetime: infinite, usage pattern: regular.

This run-time cache contains the navigation selection models used by portal users. There is one entry per user session. The cache should be large enough to hold all these models for the active users. An entry in the cache has the maximum lifetime of the corresponding user session, i.e. entries are removed at the end of the session. Creating a cache entry is less expensive than creating a content model cache entry. Typically all required information is in memory, but accessing the database might be necessary. In comparison to the content model cache creating an entry for the navigation selection model cache is much cheaper. In addition also the in-memory size of elements in this cache is much smaller since this type of model references fewer objects.

com.ibm.wps.model.factory.NavigationSelectionModelCache.isolated

Default size: 1000, default lifetime: infinite, usage pattern: regular.

This cache contains navigation selection models used by administrative users. The details given for the administrative content model cache also apply here.

com.ibm.wps.model.factory.URLMappingCache.live

Default size: 50, default lifetime: infinite, usage pattern: regular.

This cache is the run-time model cache for the URL mappings defined in your portal installation. It should be large enough to hold all URL mappings defined in your system. Creating an entry to the cache involves reading one entry from the portal database. A cache entry is fairly small in size.

com.ibm.wps.model.factory.URLMappingCache.isolated

Default size: 50, default lifetime: infinite, usage pattern: regular.

This cache is the administration cache for URL mappings. The details given for the other isolated caches also apply here.

com.ibm.wps.model.factory.MultiModelCache.live

Default size: 50, default lifetime: infinite, usage pattern: regular.

This cache contains run-time models for several different resource types in WebSphere Portal, for example clients, supported markups and languages. One entry, for example, is a list containing all supported markups. Those resources typically remain stable for a long time, hence you should mostly experience read accesses to this cache. Creating a cache entry involves reading the corresponding data from the database. An entry can be fairly large, but the number is very low so that the total size of this cache is negligible.

com.ibm.wps.model.factory.MultiModelCache.isolated

Default size: 1000, default lifetime: infinite, usage pattern: regular.

This cache contains the administrative models for several portal resource types. Typically this cache is empty and not used, because administration on those resource types is a rare event. There is one entry for every user doing administration on any of the resource types that are stored in the cache. The creation behavior and size are similar to the run-time cache.

com.ibm.wps.model.factory.NavigationModelCache.live

Default size: 2, default lifetime: infinite.

This cache is not used in WebSphere Portal V6.0 and hence disabled. Changing any of its properties does not have any effect.

com.ibm.wps.model.factory.NavigationModelCache.isolated

Default size: 2, default lifetime: infinite.

This cache is not used WebSphere Portal V6.0 and hence disabled. Changing any of its properties does not have any effect.

com.ibm.wps.model.content.impl.DynamicLoadCache

Default size: 4, default lifetime: 600.

This cache is not used WebSphere Portal V6.0 and hence disabled. Changing any of its properties does not have any effect.

com.ibm.wps.model.impl.RuntimeClientMap.userAgent2client

Default size: 1000, default lifetime: infinite, usage pattern: regular.

This cache maps user agent strings, i.e. the identification strings sent by browsers in the HTTP header, to client profiles. These profiles basically correspond to CC/PP profiles. Hence the cache scales with the number of browser identification strings. Data from this cache is accessed during every request. Creating a cache entry is very cheap since the profile information is in memory already. An entry in the cache hence is fairly small since already existing data is referenced.

URL MAPPINGS

The following caches contain data on portal URL mappings. Be sure to size the caches in a way such that these are large enough to hold all defined URL mappings in your system.

wps.mappingurl.ContextsCache

Default size: 500, default lifetime: infinite, usage pattern: regular.

This cache contains URL mapping contexts. It scales with the number of mapping contexts defined in the system. This cache is used if a URL mapping cannot be resolved using the lookup cache. Creating an entry involves reading a mapping entry from the database. An entry in the cache is medium-sized.

wps.mappingurl.LookupCache

Default size: 600, default lifetime: infinite, usage pattern: regular.

This cache is used as a final lookup cache for the computed mappings between (a hierarchy of) URL mappings and a WebSphere Portal resource. It is accessed during every request when analyzing the incoming URL for being a URL mapping. The size of this cache should be the number of all mappings. Creating a cache entry typically is cheap because the information often s in memory. An entry in the cache is rather small.

VIRTUAL PORTALS

The following group of caches is only relevant if you have defined additional virtual portals in your system. In all other situations it is safe to set the size of these caches to one and the lifetime to infinite.

com.ibm.wps.services.vpmapping.VirtualPortalIDToRealmCache

Default size: 120, default lifetime: 3600, usage pattern: regular.

This cache stores the realm information for virtual portals. One realm can contain several virtual portals, but one virtual portal can only be part of a single realm. As a consequence, the optimum size of this cache is the number of virtual portals defined in your environment. You may increase the lifetime for better performance if your setup of virtual portals changes infrequently. If you only use the default portal and no additional

virtual portal, you will see one entry in the cache and only little traffic on the cache. Creating a new cache entry requires one database query. An entry into the cache is fairly small.

com.ibm.wps.services.vpmapping.VirtualPortalIDToURLCache

Default size: 120, default lifetime: 3600, usage pattern: regular.

This cache maps virtual portal IDs to their respective LPID. The LPID usually is used to create URLs for a specific virtual portal. Since the number of LPIDs is equal to the number of virtual portal IDs, the optimum size of this cache is the number of Virtual Portals defined in your environment. You may increase the life time for better performance if your setup of virtual portals changes infrequently. If you only use the default portal and no additional virtual portal, you will see one entry in the cache and only little traffic on the cache.

com.ibm.wps.services.vpmapping.URLToVirtualPortalIDCache

Default size: 120, default lifetime: 3600, usage pattern: regular.

This cache maps LPID values to virtual portal IDs. LPIDs are encoded in a URL that points to a certain virtual portal. Therefore the number of LPIDs is equal to the number of virtual portal IDs. Accordingly the optimum size of this cache is the number of virtual portals defined in your environment. You may increase the lifetime for better performance if your setup of virtual portals changes infrequently. If you only use the default portal and no additional virtual portal, you will see one entry in the cache and only little traffic on the cache.

W S R P

All WSRP caches are only accessed if the portal is used as either a WSRP consumer or producer. Each of the caches is used on either side of the WSRP communication, but not on both sides. Most of the WSRP caches are used and read during every WSRP request, either displaying a page with a provided portlet on it, or administering WSRP properties. Exceptions to this general rule are noted below.

wsrp.cache.portletdescription

Default size: 500, default lifetime: 3600, usage pattern: regular.

This cache contains the portlet descriptions delivered by producers. These descriptions could be considered meta information on the provided portlets, like languages and descriptions. It is used on the producer side. The cache scales with the number of remote portlets provided by the producer. Increasing the default lifetime can improve performance if portlet descriptions of the provided portlets change infrequently. Rebuilding cache entries is rather expensive. It includes loading data from the database with several calls. The cached entries are rather expensive in terms of memory usage.

wsrp.cache.servicedescription

Default size: 150, default lifetime: infinite, usage pattern: regular

This cache contains service descriptions of WSRP producers. It is used on the consumer side. It scales with the number of WSRP producers integrated into the consuming portals; there is exactly one description per producer. The service description is generated using all the portlet descriptions from the producer portal plus some additional data. Hence a service description can be large in terms of memory requirements. Rebuilding the description requires several roundtrips and is an expensive operation. Cache entries are rebuilt if a user clicks the 'Browse' button in the WSRP administration portlets. This leads to a refresh of all service descriptions of all producers. This cache is only used during WSRP administration.

wsrp.cache.portlet.instance

Default size: 2500, default lifetime: 3600, usage pattern: regular.

This cache contains the proxy portlet instances on the WSRP consumer side and is only used there. It scales with the number of integrated remote portlets multiplied with the number of users having their own customizations of portlet preferences for these remote portlets (portlet settings for legacy portlets respectively). Creating an entry for the cache involves one multi-line database query. The size of a cached entry depends on the number of parameters associated with the portlet. Hence the size ranges from small to fairly large.

wsrp.cache.producer.user

Default size: 5000, default lifetime: 3600, usage pattern: multiple object types.

This cache contains the descriptor of the producer and context information between users and producers. It is used on the consumer side. It scales with the total number of active users accessing remote portlets of these producers, i.e. as a maximum the number of producers multiplied with the number of active users accessing them plus the number of producers. Recreating cache entry is fairly expensive. It involves some DB queries and in-memory operations. Therefore the session timeout should not be higher than the lifetime of entries in the cache. Cache entries are explicitly invalidated during user session destruction.

wsrp.cache.portlet.window

Default size: 2500, default lifetime: infinite, usage pattern: regular.

This cache contains a WSRP specific wrapper on a WebSphere Portal portlet entity object. It is used on the producer side. It scales with the number of provided portlets and the number of occurrences of these portlets on consumer pages. Recreating cache entries is rather cheap and typically only includes in-memory operations. An entry into this cache is fairly small. This cache is accessed very during a request.

wsrp.producer.portletpool.pops

Default size: 1000, default lifetime: infinite, usage pattern: cascading object types.

This cache stores the Producer Offered Portlets and hence scales with their number. The number of entries in this cache is identical to the number of entries in the portletdescription cache. The WSRP object model data is stored in here, though. Offered portlets are first looked up in this cache and, if the lookup is not successful, the in the ccps cache (see below). Reloading cache entries involves one query against the database. Cached entries are rather small.

wsrp.producer.portletpool.ccps

Default size: 1000, default lifetime: infinite, usage pattern: regular.

This cache stores the client configure portlets. It is used on the producer side. It scales with the number of provided portlets and the number of remote users having personalized those (Consumer Configured Portlets); hence the maximum would be the number of provided portlets multiplied by the number of remote users accessing the producer. Reloading cache entries involves one query against the database. Cached entries are rather small.

DYNAMIC ASSEMBLY / PROCESS INTEGRATION

The following caches are used when dynamic UI functionality, often together with WebSphere Process Server integration are used.

processintegration.PendingTasksCache

Default size: 2500, default lifetime: infinite, usage pattern: regular.

This cache contains the pending process tasks in the scope of a user. The size of this cache scales with the number of users concurrently using process integration functionality. Each cache entry consists of a complete set of pending process tasks for a given user and therefore can be fairly large in memory. Reloading a cache entry involves accessing the Human Task Manager via an EJB call. The cache is always accessed when the PendingTasksTag is used in a portlet JSP.

You should also configure the setting processintegration.pendingtasks.lifetime in ConfigServices.properties which defaults to a value of 30 seconds. This setting describes the interval at which a process engine is queried for pending tasks of a user and the cache entries are updated.

wp.te.transformationAssociationCache

Default size: 500, default lifetime: infinite, usage pattern: regular.

This cache contains transformation extension nodes. So typically there are only few entries in the cache. There is typically one access to the cache per request. Building an

entry to the cache involves one database query. One entry is fairly small. Typically there is no need to modify the settings for this cache.

POLICY

The WebSphere Portal policy manager uses the following caches.

com.ibm.wps.policy.services.PolicyCacheManager

Default size: 1000, default lifetime: 7780, usage pattern: regular.

This cache stores the policies. Out of the box portal comes with twelve theme policies and one mail policy, each of them being one entry into the cache. Hence the maximum number of cache entries depends on your system and the number of custom policies. This cache is accessed fairly often, if you use policies at all. The WebSphere Portal V6.0 default theme uses policies and query this cache during every request, but it is possible to create themes that do not use policies at all. Furthermore when opening mails the cache is accessed. Creating a cache entry involves reading data from a database. An entry into the cache is fairly small.

com.ibm.wps.policy.services.UserPolicyNodeCacheManager

Default size: 2500, default lifetime: 600, usage pattern: regular.

This cache stores connections between a policy and a policy target, for example a user distinguished name. Theme policies do not use targets, hence there is no cache entry based on these policies. The out-of-the-box mail policy uses the user as target. Hence there is at least one entry for every user accessing the CPP mail portlet. The size of a cache entry depends on the size of the target object. For a distinguished name a cache entry is fairly small.

COLLABORATION SERVICES

All of the following caches are used by the DEPP portlets and some services around these portlets. In so far the caches are not used if the DEPP portlets are not utilized in the portal system. These caches store credential information needed for the backend servers, server information for these servers and user information that would otherwise require LDAP lookups.

com.lotus.cs.services.directory.ldap.BasicLDAPDirectoryService.server

Default size: 50, default lifetime: infinite, usage pattern: regular.

This cache stores mail server information. In so far it scales with the number of different mail servers used in the environment. It is accessed whenever a mail server is accessed. Creating a cache entry requires one LDAP search. An entry in the cache is fairly small.

com.lotus.cs.services.directory.ldap.BasicLDAPDirectoryService.user

Default size: 2000, default lifetime: 10780, usage pattern: regular.

This cache stores user-specific information read from the LDAP. It scales with the number of users working with DEPP portlets. The cache is accessed during rendering a DEPP portlet, whenever those need user information. This could be multiple times per page reload. In addition the cache is accessed whenever a mail server is accessed. Creating a cache entry is fairly expensive and can involve multiple LDAP lookups. An entry into the cache is medium-sized.

com.lotus.cs.services.directory.wmm.WMMDirectoryService

Default size: 4000, default lifetime: 10980, usage pattern: regular.

This cache stores user-specific information read from the LDAP and WMM. Entries in this cache represent a more complete set of data stored in the LDAP than is available in other parts of WebSphere Portal. The cache scales with the number of users working with DEPP portlets. The cache is accessed during rendering a DEPP portlet, whenever those need user information. This could be multiple times per page reload. In addition the cache is accessed whenever a mail server is accessed. Creating a cache entry is fairly expensive and can involve multiple LDAP lookups. An entry into the cache is medium-sized.

com.lotus.cs.services.UserEnvironment

Default size: 2000, default lifetime: 10880, usage pattern: regular.

This cache stores user-specific information. Entries represent a compilation of credential information for one user to different LDAP directories and details which data on the given user can be found in which directory. For example, the general info may be stored in one directory, but the mail server and file may be in another. The cache scales with the number of users working with DEPP portlets. The cache is accessed whenever a DEPP portlet is accessed. Creating a cache entry can be fairly expensive since multiple resources might be queried. An entry to the cache is medium-sized.

com.lotus.cs.services.domino.DominoService

Default size: 2000, default lifetime: 11080, usage pattern: regular.

This cache stores user-specific Domino information. It is used for awareness functions. It scales with the number of users working with the corresponding function. The cache is accessed whenever awareness functions are requested during page rendering. Creating a cache entry is cheap and simply involves creating a new Domino session. An entry to the cache is medium-sized.

MISCELLANEOUS

This group of caches does not fit in any of the other categories.

com.ibm.wps.pe.portletentity

Default size: 10000, default lifetime: 5800, usage pattern: regular.

This cache contains configuration for portlets on pages (portlet instances, shared and per-user). It scales with the number of pages defined in your portal, the number of portlets on the pages and the number of portlet instances that have been personalized by users. The cache is accessed many times during portal page rendering. In so far it is important that the most relevant portlet entities are cached. Creating a cache entry involves a single database lookup. An entry into the cache is fairly small.

Example: In a portal with 500 pages and on average three portlets per page, the optimal cache size would be 1500 to store all possible portlet entity data in the cache, if users are not allowed to personalize the portlets. If the portal has 100 users that are logged in concurrently and these users have personalized 50 portlets on average, the required cache size to cache all data would be 6500. These numbers give the maximum number of entries to the cache. Typically for this cache it is not required to have all portlet entities in memory, because most users will not view all pages so that not all personalized data must be loaded. The most frequently accessed un-personalized portlet entities should fit into the cache, though.

com.ibm.wps.services.cache.cachedstate.CachedStateServiceCache.cache

Default size: 50000, default lifetime: 7200, usage pattern: typically regular.

This cache stores session-scoped data in the portal context and is used by various portal components. This cache scales linearly with the number of active sessions in the system and the number of portal components using this cache for data retrieval. The usage pattern, access times, entry creation costs and entry memory sizes depend on the portal component using this cache and cannot be stated in general.

wp.xml.configitems

Default size: 1000, default lifetime: infinite, usage pattern: regular.

This cache stores XML Access configuration items. It is used only during XML Access processing. The entries resemble references between nodes in the XML Access document. Especially when working with complex XML files, usually used for imports or Release Builder processes, it can be beneficial to increase the cache size. The cache will be cleared after XML processing is completed. Reloading a cache entry involves one database query. Entries in the cache are medium-sized.

PortletMenuCache

Default size: 200, default lifetime: infinite, usage pattern: regular.

This cache contains portlet menu trees for all portlets that define their portal menu as global, meaning identical for all users. The portal functionality that utilizes this cache is deprecated with WebSphere Portal Version V6.0.

RegistryService

Default size: 32, default lifetime: infinite, usage pattern: regular.

This cache is used in a cluster for portals to notify the other cluster members when one of the registries needs to be reloaded due to administrative action. It should never be disabled or set to `shared=false`.

com.ibm.workplace.searchmenu.helper.SearchMenuCacheHelper

Default size: 2500, default lifetime: 3730, usage pattern: regular.

This cache stores a variety of information having to do with the search scopes menu, located at the top of the theme, left to the search box. There are six rather small cache entries per user. Hence the cache scales directly with the number of users. There are no invalidations from the cache, but after login a user will always get fresh data from the cache via a coupling between the cache and the user session. The cache will be accessed on every subsequent user request for building the search bar. If the search bar is not used, the cache will not be used, either. Rebuilding the cache is fairly inexpensive, but it does require some calls to the search engine backend to get the required data.

Example Scenarios

This section describes some example usage scenarios along with descriptions of possible cache settings and suggested cache sizes. This discussion may be useful as starting point for the caches in your environment.

GENERAL COMMENTS

Most portal caches fall into one of four groups:

1. Caches where the number of entries scales with the number of active users. For example, the access control user context cache (`com.ibm.wps.ac.AccessControlUserContextCache`) falls into this category.

2. Caches where the number of entries scales with the number of users using a specific function. For example, the cache `com.lotus.cs.services.directory.ldap.BasicLDAPDirectoryService.user` falls into this category.

3. Caches which scale with the number of pages being visited. The resource cache (`com.ibm.wps.model.content.impl.ResourceCache`) is an example of this type.

4. Caches which scale based on the growth of some other resource, such as URL mappings, which are stored in the cache `com.ibm.wps.model.factory.URLMappingCache.live`.

Those that scale on portal resources should have lifetimes and sizes based on the number of portal resources in the system and how frequently users access these resources. The other caches depend upon usage scenarios such as those described in this section.

Most caches have a lifetime associated with them because the cached content might change over time. For example, access control information could be changed via user administration in the administrative portlets, XML Access or the WebSphere Portal scripting interface. All code that uses caches within WebSphere Portal is implemented in a way such that cache entries that are no longer valid are removed from the cache if the corresponding information has been changed or deleted. The lifetime therefore is used for three reasons:

- Expired cache entries can be removed to free up memory.
- There are rare race conditions in cluster setups so that invalidation events are processed correctly but the cache still reflects wrong data.
- Updates within external systems, like an external access control system, will never become visible.

If there is no or very little administration on your system and you have free memory in the Java heap available, it is safe to increase the lifetime of cache content to save the additional workload for reloading cached data.

Now we shall consider some recommendations for specific scenarios.

SMALL NUMBER OF PAGES AND SMALL NUMBER OF USERS

In this scenario a portal only has a limited number of pages and users accessing them. For example, there might be 200 pages in the system and up to a few hundred users working with the portal simultaneously. You will find portals of this kind often during development and testing or in smaller portal production systems.

For portals of this size and usage the default cache values typically are good. Hence only small modifications to the defaults given above should be required. Nevertheless you should be careful not to translate those cache settings directly into production for larger user communities.

SMALL NUMBER OF PAGES AND LARGE NUMBER OF USERS

In this scenario a portal only offers a rather small number of pages to the user. Overall there might be again only a few hundred pages, maybe with different access rights on them so that users might see only subsets of the pages. But in this scenario there are thousands of users accessing the system at the same time. In other words, thousands of users have active sessions.

Properties of caches that store information on pages typically do not need to be modified in this scenario. But all caches that store user-dependant information might be a problem. Assume you have 2000 active users in your system. Per-user caches being sized to only 1000 entries will operate at their upper limit nearly all of the time and constant re-calculating or re-loading of data from the portal database will the consequence. You should size the user-dependent caches in a way such that enough entries for the number of currently active users can remain in memory. We define the number of ‘currently active users’ as those who have a session and still issue requests against WebSphere Portal. By contrast there are passive users who still have a session, but no longer issue requests and have forgotten to log out or simply went away from the screen and let the session time out.

We increased the sizes of the following nine caches in our measurement environments in such a way that the data of all concurrent users fits into the caches.

- `com.ibm.wps.model.factory.ContentModelCache.live`
- `com.ibm.wps.ac.ExplicitEntitlements Cache.USER_GROUP`
- `com.ibm.wps.model.factory.NavigationSelectionModelCache.live`
- `com.ibm.wps.datastore.services.Identification. SerializedOidString.cache`

- com.ibm.wps.puma.OID_User_Cache
- com.ibm.wps.puma.DN_User_Cache
- com.ibm.wps.puma.OID_DN_Cache
- com.ibm.wps.puma.DN_Group_Cache
- com.ibm.wps.puma.OID_Group_Cache

We increased the lifetimes of all caches to at least one hour.

PORTALS WITH LONG SESSION TIMEOUTS

If the session timeout has a high value, it is likely that there will be a large number of users who still have sessions with the portal, but who have not interacted with the site for a significant period of time. These users are known as passive users, and they can cause some special performance challenges.

In this situation the number of sessions can be much larger. But typically many of these sessions are 'passive'. It is typically not necessary to have all information in memory for all these users when they leave their desk but not the portal, for example during lunch. To find proper sizes for the portal caches you need a good understanding on the behavior of your users. Users who have not worked with the portal for more than an hour typically will accept response times of two or three seconds for their first request after such a long break, whereas users who work with the portal rather constantly do not want to see their data being evicted from caches.

For this scenario it is hard to give precise cache size recommendations. The values simply depend too much on your portal usage scenario. You have to monitor your system and users closely to determine good values.

PORTALS WITH MANY PAGES

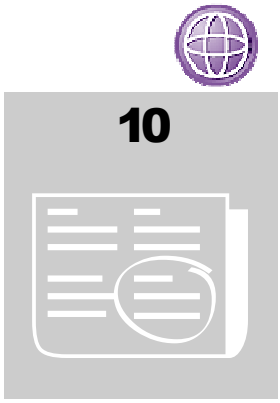
Portals in this group have several thousand pages that are available for larger groups of users and therefore are potentially accessed quite frequently. We do not count installations with many customized pages (sometimes known as 'implicit derivations') to this group because these are private resources and are loaded for the current user only. Private data is not added to the shared portal caches.

For example, your portal could have 5000 pages in total. Half of those are available to all users; on the other half there are several user groups who have view rights, other have manage right on those pages. In this case you typically do not want to have all pages and all corresponding information in memory at all times. But you want to make sure that all frequently accessed data is in memory. Typically not all portal pages are accessed equally frequently. The better your page view statistics are, the easier it is for you to tune the portal caches.

We increased the sizes of the following caches in our measurement environments so that all frequently-accessed pages, which depend on our scenario, can be cached.

- `com.ibm.wps.datastore.pageinstance.OIDCache`
- `com.ibm.wps.datastore.pageinstance.DerivationCache`
- `com.ibm.wps.model.factory.ContentModelCache`
- `com.ibm.wps.model.factory.NavigationSelectionModelCache`
- `com.ibm.wps.ac.PermissionCollectionCache`
- `com.ibm.wps.ac.ProtectedResourceCache`
- `com.ibm.wps.ac.ExplicitEntitlementsCache.USER_GROUP`
- `com.ibm.wps.datastore.services.Identification.SerializedOidString`

We increased the lifetimes of all caches to at least one hour.



WEB CONTENT MANAGEMENT CACHES

In the preceding chapter we described the specific values we modified for the Web Content Management (WCM) caches in our environments. This chapter describes the Web Content Management caches and the general parameters for those caches.

WCM Cache Instances

With WebSphere Portal V6.1, the WCM caches are managed via the WebSphere Application Server administrative console under Resources > Cache instances > Object cache instances.

WCM ITEM CACHING

services/cache/iwk/strategy – WCM Item caching

Default size: 2000, default lifetime: infinite, usage pattern: regular.

This cache stores internal WCM items. Any WCM item read from the database will first check this cache. WCM items cover Content, Workflow, Workflow Stages, Workflow actions, Taxonomies, Categories, Authoring Templates, Presentation Templates, Sites, Siteareas, and all Library Components. The cache entry will be updated or cleared when its corresponding WCM Item is updated or deleted.

WCM SUMMARY

services/cache/iwk/objectsummary – WCM Summary

Default size: 2000, default lifetime: infinite, usage pattern: regular.

This cache stores summaries of WCM Items. The summaries are used to display in lists in the authoring portlet or used internally in the WCM API to calculate WCM Item Document IDs used for Iterators. The cache entry will be cleared when a WCM Item is updated that will affect this summary.

services/cache/iwk/module

Default size: 2000, default lifetime: infinite, usage pattern: regular.

This cache is used for WCM Basic caching. See the InfoCenter on setting up Basic caching. The Basic cache stores the entire response. The key is based only on the URL so all users will see the same response.

ADVANCED AND RESOURCES

services/cache/iwk/processing – Advanced and Resources

Default size: 2000, default lifetime: 1 month (configurable), usage pattern: regular.

This cache stores the binary MIME for file and image resources in WCM. The maximum size of resources to store is set in the WCMConfigService.properties file as the property resourceserver.maxCacheObjectSize (in kb). Resources over this size are not cached and are streamed directly to the response. The expiry is set in the same file as: resourceserver.cacheExpiryDate. The cache entry will be cleared when that resource is updated.

This cache also stores page data if WCM Advanced caching is enabled. See the InfoCenter for enabling WCM Advanced caching. The processing cache stores advanced caches for the following types:

- **Site:** Similar to “Basic” Caching except that “Connect Tags” are processed each time.
- **User:** Stores a copy of an item in the cache for each user
- **Secured:** Users that belong to the same groups will access the same cached items
- **Personalized:** Users who have selected the same personalization categories and keywords, and who belong to the same Group, will access the same cached items

NOTE that the ‘session’ option for Advanced caching is not stored in the processing cache, but the ‘session’ cache.

SESSION CACHE

services/cache/iwk/session - Session

Default size: 2000, default lifetime: infinite, usage pattern: regular.

This cache stores the page data for when session advanced caching is enabled. See the InfoCenter for enabling WCM Advanced caching.

MENU

services/cache/iwk/menu - Menu

Default size: 2000, default lifetime: infinite, usage pattern: regular.

This cache stores WCM Menu entries. An entry comprises of the Content IDs associated with a particular menu. The entries are retrieved and cached without applying security. Whenever a user needs that menu's results, their specific security will then be applied to the cached results. A dynamic menu, which is one that is affected by the current user's context (e.g. based on categories in a users profile) will store a separate cache entry for each different context. The cache entry will be cleared when a WCM Item is updated that will affect this menu.

NAVIGATOR

services/cache/iwk/nav Navigator

Default size: 2000, default lifetime: infinite, usage pattern: regular.

This cache stores parent to child relationships that comprise a WCM navigator. A complex navigator might have multiple parent to child relationships (e.g. if siblings are included). The navigator entry is made up of the IDs of the parent and children. This cache will be cleared upon any WCM Item update in the system.

ABSOLUTE PATH

services/cache/iwk/abspath – Absolute path

Default size: 5000, default lifetime: infinite, usage pattern: regular.

This cache stores JCR path to id relationships. The cache entry will be cleared when a WCM Item is updated that will affect it.

MISSED ITEMS

services/cache/iwk/missed – Missed Items

Default size: 5000, default lifetime: infinite, usage pattern: regular.

This cache stores JCR paths that does not exist. This is used primarily for multi locale solutions to determine if items of other locales exist or not. The cache entry will be cleared when a WCM Item is updated that will affect it.

LIBRARY

services/cache/iwk/global - Library

Default size: 2000, default lifetime: infinite, usage pattern: regular.

This cache contains a lookup for library id, name and path to the library object. This is pre-populated up to the cache size at Portal startup.

LIBRARY PARENT

services/cache/iwk/libparent – Library Parent

Default size: 2000, default lifetime: infinite, usage pattern: regular.

This cache stores a list of all children library ids to a given parent id. Introduced for Quickr to group libraries within a teamspace together.

DRAFT SUMMARY

Services/cache/iwk/draftSummary – Draft Summary

Default size: 2000, default lifetime: infinite, usage pattern: regular.

This cache stores the identity of the draft summary to the identity of the draft WCM Item.

USER CACHE

User cache

Size is fixed to 2000. default is disabled.

This cache operates using a Least Recently Used algorithm. It is not shared across nodes in the cluster and it does not use dynacache. It does not update when LDAP changes. It is disabled by default but can be enabled through setting:

```
user.cache.enabled=true
```

in WCMConfigService.properties. Need to run a module called MemberCacheManager or restart server. To enable the module, add to WCMConfigService.properties

```
connect.businesslogic.module.template.class=com.presence.connect.wmmcomms
connect.businesslogic.module.template.remoteaccess=true
connect.businesslogic.module.template.autoload=false
```

Appendix A. References

WebSphere Portal Information Center:

http://publib.boulder.ibm.com/infocenter/wpdoc/v6r1m0/index.jsp

The Tuning section of the WebSphere Application Server Information Center located at:

http://www.ibm.com/software/webservers/appserv/was/library/

Redbook "WebSphere Application Server V6.1 on the Solaris 10 Operation System" located at:

http://www.redbooks.ibm.com/

WebSphere Portal Benchmark Results:

Contact WPLC Performance team.

DB2 Information Center:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

Oracle Information Center:

http://www.oracle.com/technology/documentation/database10g.html

The ROLTP factors for IBM pSeries Servers™ can be found at

http://www.ibm.com/servers/eserver/pseries/hardware/system_perf.html

For additional performance-related information, consult the following resources:

- WebSphere Application Server Performance information:

  http://www.ibm.com/software/webservers/appserv/was/performance.html

- *Recommended reading list: J2EE and WebSphere Application Server*

  http://www.ibm.com/developerworks/websphere/library/techarticles/0305_issw/recommendedreading.html

- *WebSphere Application Server Development Best Practices for Performance and Scalability:*

  http://www.ibm.com/software/webservers/appserv/ws_bestpractices.pdf

- *Diagnosing Performance Problems for WebSphere Portal 5.1* (though this document was written for WebSphere Portal 5.1, the lessons apply to WebSphere Portal 6.1 as well):

  http://www.ibm.com/support/docview.wss?uid=swg27007059

Appendix B. Credits

Thanks to the following team members of the WebSphere Portal Performance Team for contributing to this document.

Mark Alkins, *Manager*

Lee Backstrom, *Document Coordinator*

Andrew Citron

Nathan Cook

Sabine Forkel

Uwe Haller

Shibi John

Klaus Nossek

Kyung Lee

Denny Pichardo, *Technical Lead*

Martin Presler-Marshall

Terence Walker

Laura Yen, *Document Coordinator*

Sonja Zwissler

Kenny Sabir

Maria Sueli Almeida

Alesio Pfeifer

Joerg Huehne

David De Vos

Mike Coletta

IBM United States of America

Produced in the United States of America