



# Idle WebSphere Tuning Considerations



**WebSphere** software

**Beena Hotchandani, IBM**  
Performance Analyst

**Stephen Kinder, IBM**  
Senior Technical Staff, WebSphere Virtualization Architect

## Table of Contents

Overview: .....	3
Introduction: .....	3
Testing Methodology and Configuration .....	3
WebSphere Idle Server Tuning Considerations .....	4
Application Tuning Options.....	5
Application Server Tuning Options .....	6
Node Agent Tuning Options.....	8
High Availability Manager Tuning Options.....	10
Conclusion: .....	12
Acknowledgements:.....	12
Document History: .....	12

## Overview:

Over the years, WebSphere Application Server has been engineered to keep itself ready for incoming work and do housekeeping when possible in the background. When WebSphere Application Server is deployed on a single server, this is effective use of “whitespace” or unused capacity on the box to ensure that real requests are processed optimally. More recently, server consolidation projects attempt to leverage this same “spare” capacity to run other workloads. Server consolidation scenarios typically use virtualization to *over-commit* resources such as CPU and memory – a condition where the amount of virtual resources defined exceeds the actual physical resources available. In scenarios where WebSphere is sharing resources with other workloads trade-offs may need to be made between optimal “WebSphere” configuration and optimal usage of the overall system resource.

This paper provides a list of WebSphere configurable options that can be used to reduce background processing within the application server and other components in the Network Deployment (ND) infrastructure (i.e. Nodeagent, DMGR, etc.) that consume CPU cycles when WebSphere would typically be considered “idle”. The paper details some of the tuning options, their implications and some recommendations which may help customer conserve resources during idle periods. Also provided are tuning considerations to minimize server startup times as CPU consumption during startup can have effects on other guests.

## Introduction:

WebSphere provides a highly resilient application server infrastructure, complete with fail-over, high availability, and distributed cache management capabilities. This resiliency and robustness comes with a price. WebSphere schedules timed events to check for system updates and the availability of other servers; keeping the entire infrastructure up-to-date and running flawlessly. There are several configurable timers and options which affect the amount of background processing during idle windows. In production environments like those seen in server consolidation plays, careful examination, understanding, and configuration of these features can help the underlying infrastructure manage the shared resources more effectively.

## Testing Methodology and Configuration

Performance Measurements for tuning the idle server were done in a virtual environment, using z/VM and running WebSphere Application Server v7 on System z Linux (SLES 10.2). Even though these tests were performed under a z/VM virtual environment, the tuning recommendations apply to WebSphere Application Server under any shared environment, native operating system or any other virtualized platform including PowerVM, VMWare, KVM, etc.

The test configuration consisted of one Deployment Manager, two Node Agents, and three application servers with the DayTrader 1.2 Benchmark installed. DayTrader is a Java Enterprise Edition (Java EE) benchmark modeling an online stock brokerage application.

## WebSphere Idle Server Tuning Considerations

The following table describes the configuration options that were tested. Choosing an appropriate setting for each option may involve a functional trade-off. Consequently, each option should be carefully considered in your environment to ensure that required capabilities are not fully or partially disabled. Furthermore, each option should be thoroughly tested with your application and environment to ensure that both functional and performance expectations are met.

	<b><i>Our Test Results</i></b>	<b><i>Pros</i></b>	<b><i>Cons</i></b>	<b><i>Detail page</i></b>
<b>Application Tuning</b>				
Class loading and update detection	Measureable reduction of CPU consumption during idle periods.	Idle CPU reduction.	Manual update required to reflect application changes	Page 5
JSP and JSF options	Although our application did not have a JSP, we have seen measureable improvements in customer environments.	Idle CPU reduction.	Manual update required to reflect application change.	Page 5
<b>Application Server Tuning</b>				
Start components as needed	Measureable reduction of CPU during startup.	Startup CPU Reduction.	Some component level startup processing is deferred, until first usage	Page 6
Dynamic cache service background processing	Not measurable	Reduces frequency of batched updates and reduces scan time for unused objects.	Possibly some impact (see below)	Page 6
<b>Node Agent Tuning</b>				
Automatic File Synchronization	Measureable Idle CPU Reduction	Eliminates node sync polling for nodes which are out of sync.	Manual synchronization required.	Page 8
<b>High Availability Manager Tuning</b>				
Core group service	Measureable Idle CPU Reduction	Reduces Overall CPU consumption.	Possible functional risk, if HA features are required, depending on your overall infrastructure topology.	Page 10

## Application Tuning Options

This section discusses 2 WebSphere Application tuning properties that can be used to reduce CPU consumption during idle periods that have an effect on application behavior. They effect how often the WebSphere Application server will check for updates to EJBs, web modules, and JSPs. In a production environment where changes to these components are made infrequently, this level of monitoring may not be necessary and can be reduced or eliminated.

### Class loading and update detection

**Description:** WebSphere Application Server provides the ability to control how often application classes in EJB and web modules are polled for updates. For large applications, this scanning can consume considerable CPU cycles on a frequent basis if left unchanged.

**Consideration(s):** This feature is primarily needed in development environments (i.e. Rational Application Developer) where application updates are expected frequently. In production environments where individual class file updates are very infrequent, update checking can safely be disabled. This can be done by setting the class reloading setting for Web and EJB modules to true and by setting the polling interval to zero as described in the WebSphere for Application Server InfoCenter at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/urun\\_rapp\\_classload.html](http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/urun_rapp_classload.html)

**Trade off(s):** If this feature is disabled, the application must be stopped and restarted manually to reflect changes made in the updated classes.

### Java Server Pages (JSP) Reloading

**Description:** WebSphere Application Server provides the ability to translate and compile JavaServer Pages (JSPs) at runtime when the JSP or its dependencies are modified. This is known as JSP reloading. This feature can be disabled by setting the reloadEnabled JSP engine parameter to false in the WEB-INF/ibm-web-ext.xmi or WEB-INF/ibm-web-ext.xml file, as described in the WebSphere Application Server InfoCenter at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/rweb\\_jspreloading.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/rweb_jspreloading.html)

**Consideration(s):** This feature is primarily intended for development environments (i.e. RAD) where frequent application updates are expected. In production environments where JSP updates are much less frequent, update checking can be safely disabled.

**Trade off(s):** If this feature is disabled, the application will have to be stopped and restarted manually to reflect changes made to JSP(s)

## Application Server Tuning Options

WebSphere Application Server provides a variety of settings that can be used to configure caching and startup configuration of an application server, cluster as well as the Java Virtual Machine which can effect how well WAS performs in a virtualized environment. This section discusses these settings and their effects on idle CPU consumption.

### Start components as needed

**Description:** This application server property controls how components and services are started within the application server. If enabled, components are started as they are needed by the application. Otherwise, all components are started during the server startup phase. Details on how to select this property can be found at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/urun\\_rappsvr.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/urun_rappsvr.html)

**Consideration(s):** Enabling this feature allows server components and services to start as needed. For instance if the applications installed on a server do not leverage EJBs or SIP, the EJB and SIP components will not startup. Selecting this option can improve startup time and reduce the memory footprint of the server, because fewer components are started during the startup process. With this feature, it is important to group similar apps together on a server based on the components they will use (and therefore start).

**Trade off(s):** Customers may want to consider “warming up” an application to avoid any delay that might occur when the first application unit of work is processed.

### Dynamic cache service background processing

**Description:** The dynamic cache service is used to provide generalized caching services for servlets, portlet fragments, web services, application data, etc. Several background processing tasks are needed to manage these caches effectively. Since the dynamic cache service cannot be disabled in Version 7, these background tasks are always active even if the dynamic cache service is not being used. Following is the list of custom properties that can be used to reduce the overhead of these processes.

**com.ibm.ws.cache.CacheConfig.batchUpdateMilliseconds:** This option controls the frequency of batched cached updates between distributed caches in a cluster via DRS (Data Replication Service). The default is 1000 ms.

**com.ibm.ws.cache.CacheConfig.timeGranularityInSeconds:**

**com.ibm.ws.cache.CacheConfig.lruToDiskTriggerTime:** These two settings control how frequently the caches are checked for expiring entries. The lruToDiskTriggerTime should always be a multiple of the timeGranularityInSeconds setting. The default setting for both is 5 seconds.

**com.ibm.ws.cache.CacheConfig.timeHoldingInvalidations:** This option controls how long cache invalidations are held in memory before being committed to the cache. The default is 300,000 ms.

Detailed instructions on how to set these properties can be found on the InfoCenter at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multipatform.doc/info/ae/ae/rdyn\\_tunediskcache.html](http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multipatform.doc/info/ae/ae/rdyn_tunediskcache.html)

**Consideration(s):** For Deployment Manager and Application Server instances where the dynamic caching service is not specifically being used, these parameters can be configured to higher values to reduce the number of associated background operations.

Before making permanent modifications to these custom properties you should:

1. Verify the dynamic cache service is not in use by enabling the dynamic cache service trace and monitoring the trace logs for activity or by installing and monitoring the Cache Monitor application in a test environment.
2. Thoroughly test the application and environment from a functional and performance perspective to ensure no impact prior to making these configuration changes in production.

**Trade off(s):** If the dynamic caching service is actually being used by your application, increasing these values could have negative impacts on performance and/or cache behavior and you may experience out of memory conditions.

## EJB cache and pool background processing

**Description:** The WebSphere EJB Container maintains a number of caches and pools for performance reasons. Much like the dynamic cache service, these caches and pools are monitored and cleaned frequently by background tasks. The following properties can be modified to reduce or eliminate the overhead of these processes.

**Cleanup interval:** Default 3000 (ms)

This is the interval at which EJB entity caches are scanned for unused objects that can be removed from the cache. Details on how to set this property can be found on the InfoCenter at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multipatform.doc/info/ae/ae/uejb\\_recnt.html](http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multipatform.doc/info/ae/ae/uejb_recnt.html)

**Inactive pool cleanup interval:** Default 30000 (ms)

This is the interval at which the EJB Stateless Session Bean pools are scanned for inactive instances that can be removed from the pool. Details on how to set this property can be found at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/uejb\\_rcash.html](http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/uejb_rcash.html)

**com.ibm.websphere.bean.delete.sleep.time:** Default 4200 (sec). *z/OS-Only*.

This is the interval at which Stateful Session Bean pool is scanned for timed out beans instances. Note that this setting can only be modified on z/OS. Details on how to set this property can be found at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/xrun\\_jvm.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/xrun_jvm.html)

**Considerations(s):** For DMGR and App Server instances where the EJB container is not specifically being used, these parameters can be configured to higher values to prevent the associated background operations from occurring frequently (as dictated by the defaults). Increasing these values to extremely high levels will essentially disable this behavior.

Before making permanent modifications to these properties you should:

1. Verify that the EJB container is not in use by enabling the EJB container trace and monitoring the trace logs for activity in a test environment.
2. Thoroughly test the application and environment from a functional and performance perspective to ensure no impact.

**Trade off(s):** If the EJB container service is being used by your application, increasing these values could have negative impacts on performance and/or container behavior which may cause out of memory conditions.

## Node Agent Tuning Options

This section briefly describes the Automatic File Synchronization function and how to modify property settings that can reduce the overhead of this function.

### Automatic File Synchronization

**Description:** This function controls whether or not synchronization is performed automatically between the Node Agents and DMGR and how often that synchronization is performed.

**Considerations(s):**

#### 1. *Security*

The security run time depends on node synchronization to propagate updated certificates during automated replacement processes. The security runtime also depends on node synchronization for Lightweight Third Party Authentication (LTPA) key changes.

If you are *not* using LTPA key regeneration, then you may consider turning off or increasing the file synchronization interval. Automatic LTPA key regeneration requires a very short node sync interval at the time the keys are regenerated, or you could cause a production outage. LTPA keys are not like certificates, where there can be multiple concurrent certs. It's possible for one process to present a new LTPA key that another process knows nothing about. So it's crucial that all the keys are in sync. Since any number of things can go wrong when keys are regenerated, we recommend that you disable the auto-regeneration feature if you increased the node sync interval.

## 2. *File synchronization for application deployment*

If node synchronization is disabled, you need to sync to all nodes after making a configuration change, either manually through the admin console, or programmatically through scripting. The main reason is that Serverindex.xml is changed whenever you create new servers, cluster members, or deploy applications. If it's not propagated everywhere, it can prevent HAManager from stabilizing, especially when you add new cluster members. So you should have a best practice to sync to all nodes after a configuration change, or rely on short node sync interval.

It may be beneficial to *temporarily* disable node sync during application update in order to control when the nodes receive

Details on how to modify this setting can be found at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/uagt\\_rsynchservice.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/uagt_rsynchservice.html)

**Recommendation:** If you change the node synchronization interval to something other than the default, tune to an interval of 10 minutes maximum.

### **Trade off(s):**

- You *cannot* use automatic LTPA key exchange if you permanently disable automatic file synchronization.
- If automatic synchronization is disabled (*not recommended*), forced synchronizations must be performed via the admin console or wsadmin scripting once config changes have been made. Otherwise changes will not be reflected on the relevant nodes.
- As the WebSphere Application Server runtime evolves other components may rely on node synchronization, and disabling node synchronization could result in a breakage and a loss of service.

The WebSphere Application Server Information Center article cited above has this important notice about node synchronization intervals:

**Important:** Do **not** routinely disable the synchronization process as various portions of the application server run time depend on synchronization. For example, the security run time depends on node synchronization to propagate updated certificates during automated replacement processes. Also, the security runtime also depends on it for Lightweight Third Party Authentication (LTPA) key changes. Other portions of the run time are dependent on synchronization. However, a complete list is not available. If you permanently disable synchronization, nodes might not be synchronized and an outage will result. The only exception is the configuration save operation. To avoid synchronizing the configuration when the configuration repository is being updated by a save operation, it might be beneficial to temporarily disable the synchronization process, save the configuration, and then re-enable the synchronization process. This process ensures that changes are fully committed to the configuration repository before the node synchronization.

## High Availability Manager Tuning Options

**Description:** A High Availability Manager instance runs on every application server, proxy server, node agent and deployment manager in a cell. A cell can be divided into multiple high availability domains known as core groups. Each high availability manager instance establishes network connectivity with all other high availability manager instances in the same core group, using a specialized, dedicated, and configurable transport channel. The transport channel provides mechanisms which allow the high availability manager instance to detect when other members of the core group start, stop, or fail. The amount of CPU and memory resources consumed by the HA manager and associated components increases as the size of a core group increases. The procedure used to disable the HA Manager can be found at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/trun\\_ha\\_ham\\_enable.html](http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/trun_ha_ham_enable.html)

Also refer to Configuring the Core Group Bridge Service at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/trun\\_ha\\_coregroupbridge.html](http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/trun_ha_coregroupbridge.html)

**Consideration(s):** If none of the HA manager related services are being leveraged by your application or environment, the service can be disabled to eliminate the associated overhead. Depending on the topologies and services being used, the service can be disabled selectively among the DMGR, node agents and app server processes.

If you are considering disabling the HA Manager, it is strongly suggested that you review the WebSphere documentation at the link below carefully and perform adequate functional, behavioral, and performance testing to ensure that the disabled services do not negatively impact your application.

[http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/crun\\_ha\\_ham\\_required.html](http://publib.boulder.ibm.com/infocenter/wasinfo/fep/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/crun_ha_ham_required.html)

**Trade off:** Because of the number of WebSphere services and features linked to the HA manager, disabling this component may involve a certain degree of risk. Furthermore, some extended WebSphere technologies or stack products may also have dependencies on the HAManager. For instance, WVE v6.1.1.0 and releases prior require the HAManager features to be enabled.

## Conclusion:

As larger shared environments become common place, it is important to review your WebSphere configuration to ensure that you've adequately evaluated its effect primarily when WebSphere is idle. It was our experience that the biggest hitters were Class Loading and update detection at the application level, automatic file synchronization, and disabling HA Manager, while others used in combination provided additional savings. Turning off unneeded features and dialing back background process intervals can be beneficial to the overall resource consumption of your entire configuration and help WebSphere participate better as part of your infrastructure.

## Acknowledgements:

The following people are gratefully acknowledged for their contributions to this paper.

Thomas Alcott, IBM Worldwide WebSphere Application Infrastructure  
 Chris Blythe  
 Jim Cunningham, IBM WebSphere Performance on z platform  
 Tom Seelbach, IBM WebSphere SCA Development  
 Steve Wehr, IBM systems & Technology Group, System z Linux Infrastructure  
 Bruce Hayden, IBM Advance Technical Skills – z/VM System z Linux  
 Romney White, IBM System z Virtualization Technology  
 Andrew Low, IBM JVM development  
 Juergen Doelle IBM, System z Linux performance

## Document History:

April 6, 2011	Original document.
<i>May 6, 2011</i>	Updated node synchronization section to further clarify the importance of not disabling the node synchronization function. A tuning option is to extend the synchronization interval perhaps. But never disable entirely.

**End of Document**