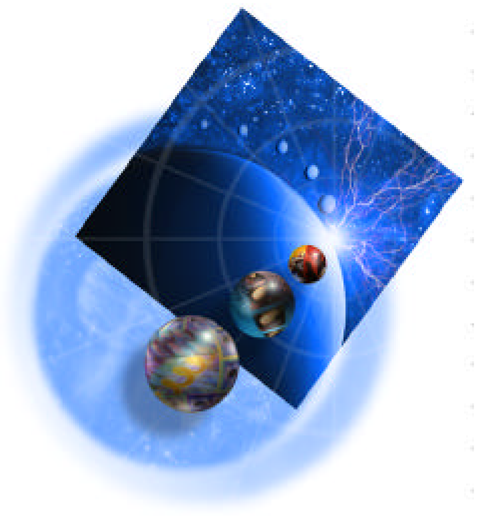
Whitepaper

IBM WebSphere Application Server Standard and Advanced Editions

Best Practices for Admin Performance and Scalability



By Gennaro (Jerry) Cuomo
Hiroyuki Tarusawa

IBM WebSphere

Document Draft Version 1.0

12/12/2000

Intended audience

This paper is intended for IT Specialists or Application Administrators who are planning to deploy production eBusiness solutions with IBM Websphere Application Server Advanced or Standard Edition. This paper assumes the reader is familiar with the WebSphere product, as well as the basic concepts of web applications, Enterprise Java Beans and Java.

Acknowledgments

The following paper was written with help from several members of the IBM WebSphere team as well as several WebSphere customers. In particular, I would like to thank Hiroyuki Tarusawa, Arihiro Iwamoto and Toshimasa Shimizu from IBM Japan and Michael Fraenkel, Erik Daughtrey, Daniel Julin, Brian K. Martin, Nataraj Nagaratnam, Carolyn Norton, Stan J. Cox, Tom Alcott and Ruth Willenborg from the WebSphere Development Team.

Related Documents and Tools

The following documents provide additional information related to WebSphere performance and tuning:

- *IBM WebSphere Application Server Standard and Advanced Editions, Version 3.0 Performance Report*
  http://www-4.ibm.com/software/webservers/appserv/whitepapers.html

- *WebSphere V3 Performance Tuning Guide (SG24-5657-00)*
  http://www.redbooks.ibm.com/abstracts/sg245657.html

- WebSphere 3.5 Resource Analyzer -
  http://www-4.ibm.com/software/webservers/appserv/download_ra.html

- WebSphere Application Server Development Best Practices for Performance and Scalability-
  http://www-4.ibm.com/software/webservers/appserv/ws_bestpractices.pdf

- WebSphere Application Server - A Methodology for Production Performance Tuning
  http://www-4.ibm.com/software/webservers/appserv

Overview

The following document provides insight into how to maximize the performance and scalability of the Administrative System of WebSphere. Although most of these concepts apply to WebSphere Advanced Edition, some concepts can be applied to Standard Edition as well.

First, a general overview of Systems Management is presented including the main design point, functions and implementation. Next, an overview of System startup is described using an example application. A detailed profile of where time is spent during startup is illustrated for the Trade 2 Application. An analysis of the factors influencing Admin startup performance and scalability is detailed. This includes factors involving the EJB Container, Servlet Container, Datasource and Clones. Lastly, this paper presents a summary of the best practices for administering a high performance and scalable Admin System in WebSphere Advanced Edition. **Customers who have applied these best practices to their installation of WebSphere Advanced Edition have seen up to a 3X improvement in startup performance and scalability.**

Systems Management Overview

Before describing methods by which we can improve the total effectiveness of the WebSphere Admin System, it is important to understand some of the key design points of this system. The following section provides an overview of WebSphere Systems Management, focusing on three concepts:

1. **Design-** *A Single Logical Image*
2. **Implementation-** *Built Using WebSphere Technology*
3. **Function-** *The Swiss Army(tm) Knife of WebSphere*

A Single Logical Image

WebSphere Advanced Edition supports a design notion of a ***Single Logical Image***. The value proposition of this design is very powerful. It ultimately allows a system administrator to define and manage complex clusters of WebSphere Application Servers from a single logical management point. The Single Logical Image is housed in the WebSphere Admin Repository, which is a standard JDBC compliant database. Within this repository the entire Administrative Domain is represented. The Domain is the root of the admin hierarchy. Under the Domain, there are System nodes, which are the physical server machines (i.e., hardware) in your cluster. Nodes contain Servers, each of which represent an individual process running a Java Virtual Machine (JVM). The Servers are the Application Server instances that run your WebSphere applications. Servers hold containers. The two primary types of containers are EJB Containers and Servlet Containers.[1]

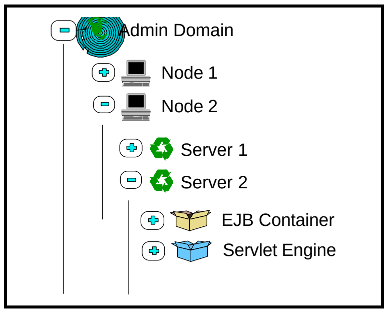


Figure. Admin Hierarchy

From this Single Logical Image, servers can be started and stopped and configurations across servers can be customized and copied. With the WebSphere Admin model, configurations can be easily replicated to other nodes or within the same node. This is done using a technique called Modeling and Cloning. The idea of models/clones is very much like cut/paste. For example, modeling is similar to the action of selecting an object and copying it to a clipboard. Cloning would be the pasting of that object from the clipboard to create a new but identical configuration.

Built using WebSphere Technology

The WebSphere Admin Server is built using WebSphere Technology. In particular, the Admin Server is a special case of an ordinary Managed Server. A Managed Server, in architectural terms, is a Java Virtual Machine that runs Java 2 Enterprise Edition (J2EE) applications. In WebSphere Standard/Advanced Editions, the Admin Server as well as instances of Application Servers all share a common, Managed Server, code base. The Admin Server is an EJB application that uses Session Bean and Bean Managed Entity Beans

[1] In the WebSphere Admin Console the Servlet Container is actually referred to as a ServletEngine.

(BMP Entities). It is configured to run a set of EJBs, called Repository Beans, that mediate requests between Admin clients and the Admin Repository Database. The Repository Beans handle configuration and operation requests for all WebSphere components (i.e., nodes, application servers, servlet engines, EJB containers, etc.) running in the WebSphere Admin Domain. Configuration requests include setting and clearing parameters to the components, and operation requests on components include starting, stopping, creating, and deleting.

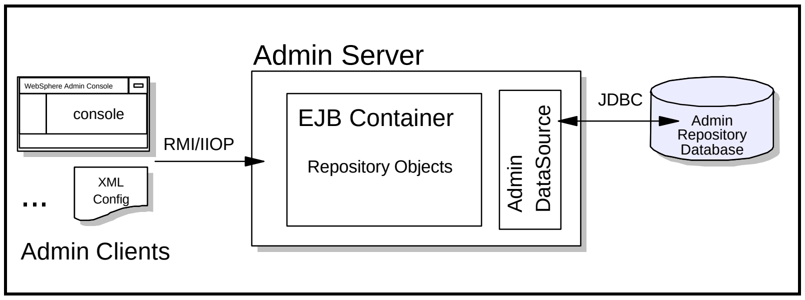


Figure. Admin Server - Built using WebSphere Technology

WebSphere Standards/Advanced Editions have several styles of Admin clients:

- **Admin Console-** The Admin Console provides a graphical user interface to the Admin Server(s) in a WebSphere Admin Domain. Property sheets and wizards are used to change configuration of a WebSphere component.
- **Web-based Admin-** A web-based version of the Admin Console that provides a subset of the functionality of the stand-alone Java client. Most of the functionality of the web-based Admin is for creating and modifying web applications.
- **XML Import/Export-** An alternative to GUI-based consoles is the XMLConfig utility that began shipping with WebSphere Application Server 3.02. The utility provides a way to use XML files to import or export WebSphere configuration data. XML files have the advantage of automating the configuration of multiple WebSphere components. For example, an application that consists of multiple servlets, JSPs, and EJBs can be configured with one XML file.
- **Scripting -** Yet another alternative to the GUI-based administrative client is the WebSphere scripting processor. Originally called *ejscp* in WebSphere 3.02, *wscp* operates in two modes: *script file processing* mode, and *interactive mode.*

One of the Admin best practices recommended later in this document is to use the Admin Console when developing applications or during pre-production tests. Prior to a production implementation the now stabilize configuration should be exported into an XML file utilizing the XMLConfig tool . Once in production, it is common for customers to solely use XMLConfig or WSCP to administer their systems.

The Swiss Army(tm) Knife of WebSphere

The Admin Server's role in systems management goes beyond that of configuration management. The Admin Server also manages critical runtime components of WebSphere. In addition to configuration management, the Admin Server provides the following services:

- **Name and Location Service -** A centralized naming service that exports Datasources and EJB Homes, as defined in the deployment descriptor, into a global name space. The Name Service uses the Java Naming and Directory Interface (JNDI) to implement a naming service**.**
- **Security Service-** A security service that handles authentication and authorization for principals that need to access WebSphere resources.

- **Nanny Service-** A Nanny Service starts and monitors the Admin Server. In turn, the Admin Server monitors the health of Application Server Instances.
- **Tracing and Monitoring-** A service to control when and how data is gathered for tracing or performance monitoring.

Server Startup

The following section examines the processes of starting up a WebSphere Application Server instance. In order to understand how to improve performance and scaling of the WebSphere Admin System, it is important to understand what happens when a Server starts and the sub-processes the occur.

Process of Starting the Trade 2 Server

The following section describes the process of starting an Application Server. An Application Server configured with a representative application called Trade 2 is used as an example. Trade 2 is a standard performance application benchmark used by the WebSphere development team. It contains a collection of EJBs, Servlets, Java Server Pages and a Datasource definition.

There are two major phases of activity that occur when an Application Server is started. Phase 1, as illustrated in the figure below, is initiated by the WebSphere Admin Console. When the console requests that a server be started, a new JVM process is spawned on the specified server node. The Admin Server passes arguments pertaining to the Trade 2 Server via the command line.

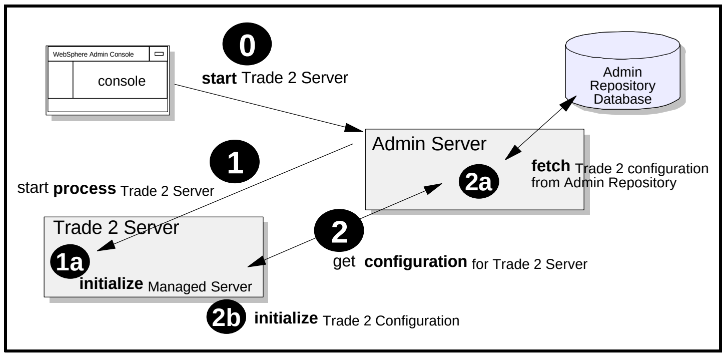

WebSphere Admin Console
console
0
start Trade 2 Server
Admin Repository Database
Admin Server
fetch Trade 2 configuration from Admin Repository
2a
1
start process Trade 2 Server
Trade 2 Server
1a
initialize Managed Server
2
get configuration for Trade 2 Server
2b
initialize Trade 2 Configuration

Figure. Process of Starting the Trade 2 Server

The majority of Phase 1, labeled 1a above, is spent initializing the basic services of a Managed Server. For example, during initialization the ORB, Logging, Security and Transaction Services are initialized. Phase 1 concludes with the newly initialized Trade 2 server notifying the Admin Server that it is ready to start Phase 2.

Phase 2 involves configuring the Trade 2 Server with its specified application resources. During the first part of Phase 2, labeled 2a above, the Admin Server gathers the Trade 2 Server configuration from the Admin Repository Database. It then packages the configuration and passes it in one single message back to the Trade 2 server. Phase 2b involves the Trade 2 Server parsing through its configuration and initializing the relevant run time components. For example, during this Phase application Jars are loaded, EJBs are registered, connections are made to the Web Server plug-in and the Datasources make contact with their respective databases. Once Phase 2 concludes, the server is "open for e-business".

Startup Breakdown

This section breaks down the time spent during the 2 startup phases of the Trade 2 Server.[2]

Phase 1. The following pie chart illustrates the individual aspects of Phase 1 startup. The chart gives some perspective of how time is spent during Phase 1 and the percentage of time spent during these phases.

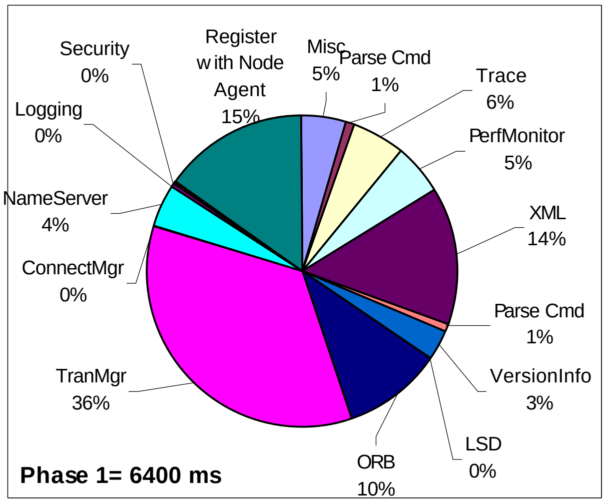
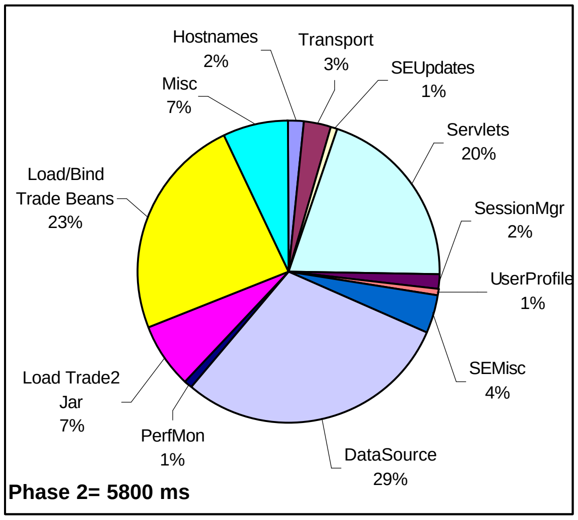


Figure. Startup Phase 1- Server Initialization

The most significant operation performed during this Phase is the initiating of the Transaction Manager (36%), which establishes the Transaction Log that is used for EJB transaction recovery. The initialization of the XML parser and the ORB combine for 25% of this Phase. The remaining times are well distributed over a variety of activities. Lastly, 15% of the overall time is spent requesting and retrieving the Trade 2 Server configuration.

Phase 2. The following pie chart characterizes startup performance of Phase 2, which is the configuration of the Trade 2 Server.



Startup Phase 2- Server Configuration

There are three major activities that make up the Phase 2. They are the loading of Servlets (20%), the loading and registering of the Trade EJBs (23%) and the loading of the Trade Datasource (29%). The remainder of the time is distributed across a variety of other activities.

[2] These measurements were taken on Microsoft Windows 2000, Intel PIII 1-way 500mhz, 512meg Ram. WSAE 3.5.2 was used with "best practices items" 1, 2, 3, 4, 5, 7 and 10 applied.

Factors Influencing Admin Performance and Scale

The following section provides a cost analysis of managing EJBs, Servlets, DataSources and Server/Clones. Cost is measured from the perspective of factors that influence performance and scalability of Managed Server startup. Examining Managed Servers has the dual benefit of providing insight into the performance and scalability of both Application Server Instances as well as the Admin Server itself. This is because both are derived from a common Managed Server code base.

As we will see there are factors that influence startup time which are purely time based. Some of these factors, however, cause “side-effects” that will also have an impact on scalability. For example, loading classes or Jars is a factor that purely effects performance. Class loading is a localized operation and only effects the performance of the local machine performing the action. Loading an EJB, however, is both a local and remote operation. EJB initialization involves interactions with the Admin Server and the Admin Repository, which can effect the overall Admin System’s scalability.

EJB Container

This section examines three factors related to EJBs that influence the performance and scalability of the WebSphere Admin System. They are the **Number of EJBs, Packaging of EJBs and Bean Naming**.

Number of EJBs

Each EJB deployed within a container has an overall influence over startup and scalability. As you increase the number of EJBs, you increase the overall startup time of your system. At a high level, there are two phases involved in initializing an EJB in a container. Phase 1 involves loading the bean and parsing its deployment descriptor. Phase 2 involves registering the EJB with the Global Name Space. Phase 1 is covered in more detail in the next section, *Packaging EJBs.* Since Phase 1 is a local operation with no side effects, it can be considered a constant activity. Hence, it will not vary based on the conditions of the entire WebSphere Admin System. Phase 2 is very different because it does influence the entire WebSphere Admin System. An EJB registers with the Global Name Space by calling remote EJB methods on the WebSphere Name Service, which is built into the Admin Server. Under normal conditions this operation is quite constant. However, performance begins to vary as the Admin Server and Admin Database become busy servicing other requests.

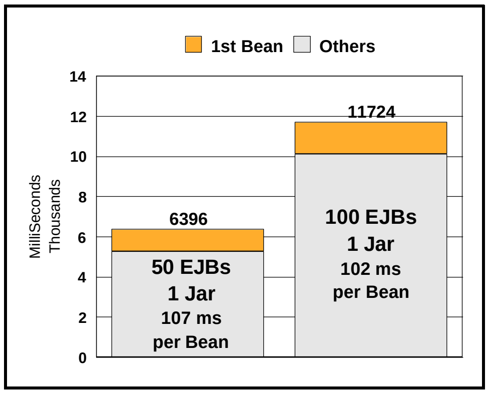


Figure. Container with 50 EJBs versus 100 EJBs

The following figure illustrates the results of an experiment that measured the startup time of a container with 50 EJBs versus 100 EJBs.[3] The startup time of the container increased **1.83x** from 50 to 100 EJBs. In both cases, Phase 1 and Phase 2 of initialization of a single EJB took place in under **110ms**. Extrapolating from this data, we can estimate that it would take under 1 ½ minutes to start a Container containing 500 EJBs.

WebSphere Enterprise Edition has support for “lazy” initialization of business objects. If this option is specified, a remote object is not initialized until a client calls a method on it. As you can imagine, this option will enable a server with many thousands of objects to be initialized very quickly.

[3] Experiment was conducted using WebSphere Advanced Edition 3.5.2 on Windows 2000 on an 8-way Intel PIII 550mhz per processor with 2 Gig of Ram. The Admin Repository Database was DB/2 6.1 and was running on the same machine.

Packaging of EJBs

The manner in which an EJB is packaged can influence the overall startup time of the EJB Container.

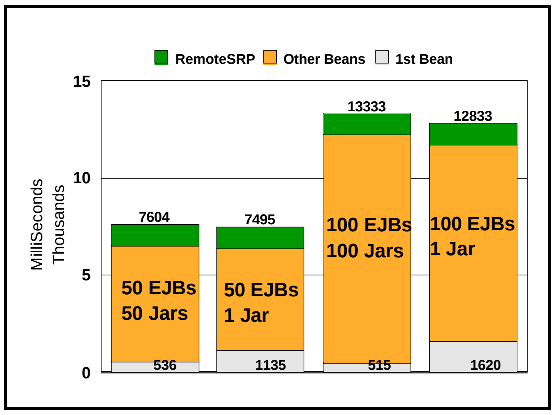


Figure. Beans per Jar. The following figure illustrates that packaging all related EJBs in a single Jar file is slightly more efficient than deploying each EJB in a separate Jar. In the case of 50 EJBs, the packaging difference was less than **2%**. In the 100 EJB case, the difference was less than **4%**. This experiment also illustrates, as one would expect, the time to load the first EJB takes longer if all beans are in the same Jar. In normal circumstances, where the Jar exclusively holds EJBs, this is not a problem. However, this experiment also illustrates a case where loading a single EJB has a significant impact if it is in housed in a large Jar file. In this example, the Remote Servlet Request Processor Bean (Remote SRP Bean) takes over 1 second to load. This is because this bean is housed in the 2.8MB ibmwebas.jar. This is a multipurpose Jar that contains many artifacts related to the WebSphere Servlet Container. When the SRP bean is loaded, a majority of the time is spent loading and parsing this large Jar.

Bean Naming

When a WebSphere Container is started it binds each of its EJBs into the Global Name Space. This process can be exacerbated if the JNDI Home Names of these beans are deeply nested.

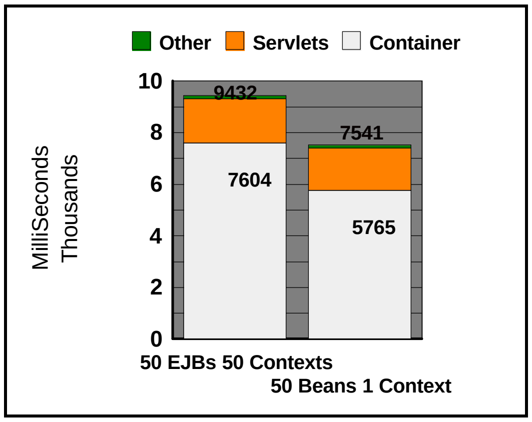


Figure. Many contexts versus one context. The following figure shows deploying beans to the same naming context is more efficient then deploying to different contexts. In the “50 Contexts" case, fifty EJBs are deployed to fifty different JNDI naming contexts. (For example, \App01\Ejb01, \App02\Ejb02, \App03\Ejb03, etc.) In the “Single Context” case, fifty beans are all deployed to the same context. (For example, \myApp\myEjb01, \myApp\myEjb02, \myApp\myEjb03, etc.). This study shows that the Single Context case is **20%** faster than the Multiple Context case.

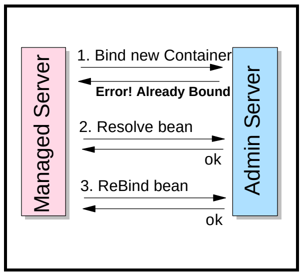


Figure. Bean Registration. The following figure describes the messages sent between the Managed Server and the Admin Server during the registration of a bean. In the "Multiple Context" case Steps 1-3 are followed for each bean. In the "Single Context" case, Step 1-3 is followed for only the first bean. All other beans that are part of the same context simply repeat Step 3, which is the *rebind* phase. This explains why the Single Context case is more efficient then the multiple context case.

Servlet Container

This section examines three factors related to the Servlet Container that influence the performance and scalability of the WebSphere Admin System. The three factors are **Autoloaded Servlets, Plug-in Configuration Regeneration and Servlet Re-director / Remote SRP Bean.** The following figure illustrates the overall impact of these factors as realized when starting up the Trade 2 Server.

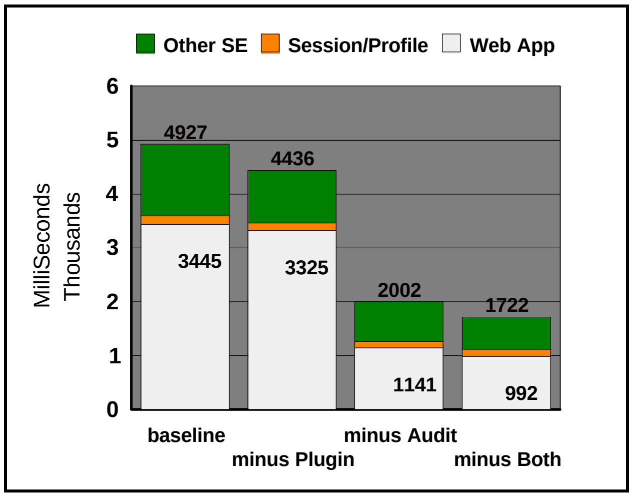


Figure. Factor related to Servlet Container

Autoloaded Servlets

Servlets can be defined to automatically load during the startup sequence of an Application Server. Otherwise, the Servlet will be initialized upon its first request for service. Autoloading Servlets will have an impact on Server startup. There are several considerations that need to be made when using Autoloaded Servlets. When Servlets are loaded, their **init()** methods are called. Servlet engineers sometimes use this method to initialize services, like connection management to backend systems or initializing large pools of static data. These heavy weight init() processes can dominate startup time. Therefore, one must be mindful of the cost of a Servlet's initialization and treat it accordingly.

When a Servlet is loaded, it generates several *serious event messages*, called Audit messages, which are logged backed to the Admin Repository, by way of the Admin Server. When Autoloading Servlets, these messages can congest the Admin Server. An e-Fix exists for WebSphere Advanced Edition 3.0.2 and 3.5 that will allow these messages to be simply written to Standard Output. For example, the above figure shows the impact of using this e-Fix on the Trade 2 Server. With this fix, the Trade 2 Server starts **better than twice as fast!**

Plug-in Configuration Regeneration

When Servlets are added to a Web Application, the configuration for these Servlets needs to be “pushed” out to the WebSphere Web Server Plug-in. This is done in order for these new Servlets to be recognized by the Web server. Plug-in Configuration Regeneration, *Regen*, is performed by the Admin Server. The Admin Server generates three properties files that are referenced periodically by the Plug-in. The Default behavior of WebSphere Admin is that a Regen occurs each time a new Application Server is started. An e-Fix exists for WebSphere Advanced Edition 3.0.2 and 3.5 that will allow this behavior to be relaxed. With this fix, the only time Plug-in Configuration is regenerated is when the Regen button is pressed via the WebSphere Admin Console. The above figure illustrates the impact of Regen during startup. Although a Regen only seems to add an additional **10%** to startup cost, Regen has an even greater impact as you start servers concurrently within your Admin System. This is because the Regen action involves interactions between the Application Server and the Admin Server. **Customers have witnessed up to 3X improvement when disabling Regen within Admin Systems with dozens of servers defined**.

Servlet Re-director and Remote SRP Bean

The Servlet Re-director and Remote SRP Bean are typically used in Firewall configurations to forward a Servlet request from a Servlet Container to a remote Servlet Container. If your configuration does not require this function, there is no need to configure the SRP Bean because it takes considerable time to load and register. This is illustrated in the *Packaging of EJB* section above. In this example, the SRP Bean was taking 1 second, out of the 7 seconds it took to load the 50 Bean server. Hence, if the Servlet Re-director is not being used, delete the Remote SRP bean from your EJB Container.

DataSources

There are two aspects of Datasources that will influence the startup of the Managed Servers that use them. The first aspect is the time it takes to lookup the Datasource in the Global Name Space. The other is the time it takes the Datasource to create the initial set of connections to its database. For example, the lookup, creation and connection initialization of the Trade Datasource takes **1.3 seconds out of the 7 seconds** it took to load the entire Trade 2 Server. Hence to improve startup time, one should consider setting the Minimum Connection Pool Size setting, on the Advanced Tab of the Datasource property sheet, to a small number.

Servers and Clones

This section examines the effects of starting multiple Application Servers.

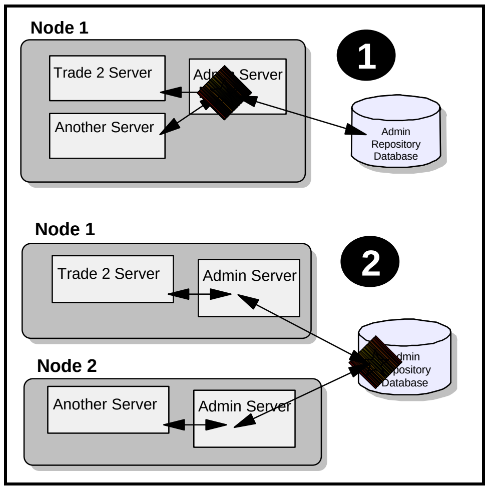


Figure- Admin Points of Contention. The following figure illustrates two Admin scenarios. **Scenario 1**, is a vertical configuration where two Application Servers are defined to run on the same machine. **Scenario 2**, is a horizontal configuration with one Application Server, Admin Server pair running on each machine. As described in the *Server Startup* section of this document, there are two phases to Application Server startup. Phase 1 is predominately composed of local operations. Hence, in Scenarios 1 or 2, provided the hardware is sufficiently supplied with memory and CPU power, two Application Servers can perform Phase 1 startup without contention. It is Phase 2 that has the potential to cause contention. In Scenario 1, there is potential for congestion at the Admin Server for performing operations like configuration retrieval or EJB registration. This congestion tends to only impact performance when many Application Servers are started at once. In Scenario 2, there is less contention at the Admin Server, however, congestion can form at the Admin Repository Database. Again, this only tends to impact performance when many Application Servers are started concurrently.

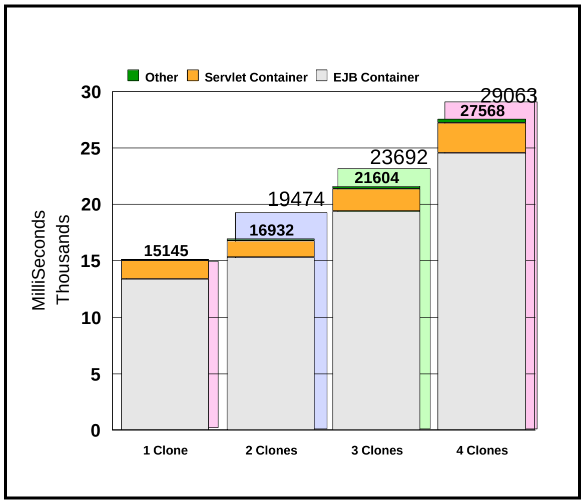


Figure. Starting 4 clones concurrently. The following figure shows the results of a series of experiments where starting Clones of an Application Server containing 101 EJBs and 25 Servlets. First, one Clone was started, then two, three and four. The 4 bars in the foreground represent the average time to start each Application Server. The bars in the back ground are the time it took to start each of the Clones.

The most important observation from this figure is that Applications Server startup only increases **1.9x** from 1 to 4 Servers. The figure also illustrates that as we increase the number of concurrent Clones, the EJB Container startup seems to proportionately increase to **1.8x** from 1 Clone to 4 Clones. This implies that the additional startup time can be attributed to contention in registering the 400 EJBs (100 per Server).

Admin Repository Configurations

An option exists in the WebSphere Admin System to specify a specific schema for the Admin Repository data. By doing this a single database "instance" can be configured with multiple schemas, each representing a unique instance of an Application Server. This configuration is a "contention free" configuration and is highly optimized for scale and performance. However, the downside to this configuration is that it breaks the Single Logical Image design point. Although the WebSphere Admin Console will still provide a single point of management for this cluster, key functions like Model/Clone are not supported.

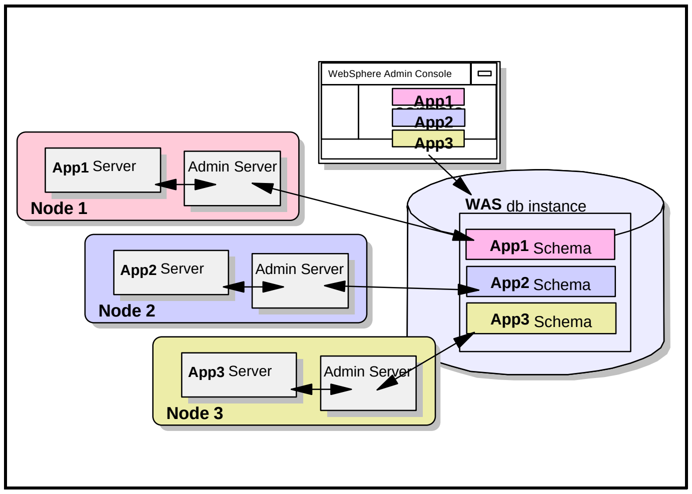


Figure. Single database instance, multiple schema. The following figure illustrates 3 Application Server instances, App1, App2 and App3, all running against the same database instance, called WAS. Each Server instance has its own schema, named App1, App2 and App3. In this configuration there is no contention between Nodes, Admin Servers or the Admin Repository.

On each node a property must be set in websphere\appserver\bin\admin.config For example, the following property was added to support the APP1 Schema. "Com.ibm.ejs.sm.adminServer.dbSchema=APP1".

Best Practices for Admin Performance and Scalability

The following section is a summary of the best practices for administering a high performance and scalable Admin System in WebSphere Standard and Advanced Editions. Each best practice includes a brief summary as well as a rating of its overall impact. A ***High*** rating has the greatest impact on Admin performance and scalability. A ***Low*** rating has a marginal, but measurable impact. Where applicable, the instructions for applying the best practice, usually in the form of setting a property in a property file, is specified. Most of these practices are derived from analysis in the previous section, *Factors Influencing Admin Performance and Scale*.

1	**WebSphere Standard/Advanced Edition 3.5.2**	High

WebSphere Standard/Advanced Edition 3.5.2 (or later) has significant improvements, over past versions, in the area of Admin startup performance and scale. One of the major reasons for 3.5.2's improvement is that it contains JNDI reference caching that significantly improves startup times of all Managed Server (Admin or Application) as well as start up time of any Client including the Admin Console and XML Config.

In general, using WAS 3.5.2 in conjunction with the other best practices described in this section, can lead to **improved Admin behavior and startup improvements of up to 300%.**

2	**Java Heap Settings**	Med

Currently, the Admin Server does not set a minimum starting heap size (-Xms) parameter. Since a "vanilla" Managed Server takes about 30 megabytes to operate, setting the -Xms to at least 48M will help start up time. Depending on the size of the application(s) running in this Managed Server, this value can be set even higher. **By modifying this setting, startup time of the Admin Server can be improved by up to 20%.** (Note: The ***adminServerJmArgs*** property supports a series of values. Do not disturb the other settings, simply append these new values to the end of this setting.)

Property Name	Location
For 3.0.2 set: **com.ibm.ejs.sm.util.process.Nanny.adminServerJvmArgs=** **-ms48m** ***For 3.5.x set:*** **com.ibm.ejs.sm.util.process.Nanny.adminServerJvmArgs=** **-Xms48m**	**/websphere/appserver/bin/admin.config**

3	**Admin Table Creation**	Med

By default, the Admin Server attempts to create approximately 35 tables in the Admin Repository, regardless if they exist or not. (Using JDBC there is currently no elegant way to test if a table exists.) If the table already exists, which will be the case in all but the first time the Admin Server is run, the *create* operation will obviously fail with an "Already Exists" exception. This exception occurs after the *create* operation times out. In some configurations this can impede startup performance, especially when the Admin database is configured remotely. For example, customers have measured **as high as 2 seconds per table** for the table request to fail in a remote DB2 config.

WebSphere has a property that will disable the creation of tables on Admin Server startup. The property is:

Property Name	Location
com.ibm.ejs.sm.adminServer.dbInitialized=true	/websphere/appserver/bin/admin.config

4 Admin Server EJB Method Calling Model — High

As described in the *Systems Management Overview* section of this paper, the Admin Server is, itself, a Managed Server running an EJB application. Like any Application Server, the Admin Server can be run using the Call-by-Reference method calling convention instead of the default, Call-by-Value. Doing this will increase the performance of local EJB interactions. When the Admin Server is run in this mode, performance of many operations including: **Admin and Application Server Startup times, Admin Client Startup, XMLConfig and WSCP all improve up to 40%**. Any time the calling model of an EJB container is changed to Call-by-Reference, there is a risk because the local/remote transparency of EJBs is violated. This can lead to unexpected side effects because the EJB programmer may be counting on parameter copying behavior that Call-by-Value provides. However, since this application is written by the WebSphere development team, we can control its behavior and ensure that we obey the proper calling guidelines.

To configure the WebSphere Admin Server to run using the Call-By-Reference model, set the following property: (Note: The ***adminServerJmArgs*** property supports a series of values. Do not disturb the other settings, simply append these new values to the end of this setting.)

Property Name	Location
com.ibm.ejs.sm.util.process.Nanny.adminServerJvmArgs= -Djavax.rmi.CORBA.UtilClass=com.ibm.CORBA.iiop.Util -Dcom.ibm.CORBA.iiop.noLocalCopies=true	/websphere/appserver/bin/admin.config

5 Plug-in Configuration Regeneration — High

During the initialization of an Application Server instance, the Web server plug-in configuration is regenerated during a server startup. This action is performed whether it needs to be or not. This process is described under *Servlet Container*, in the *Factors Influencing Admin Performance and Scale* section of this paper. An option was described that disables the automatic regeneration behavior and will only allow plug-in configuration to be generated on demand. Configuring this option can lead to Admin startup time **improvements ranging from 10% all the way up to 100% depending on the number of server starting concurrently.**

To configure this option create a file called ***global.properties*** in the /websphere/appserver/properties directory. Add the following property to the file:

Property Name	Location
com.ibm.servlet.engine.disableAutoPluginCfg=true	/websphere/appserver/properties/global.properties

An e-fix is required in order to apply this best practice. See the *E-Fixes referenced within this Paper* section for details on how to get this fix.

6	DNS Configuration	Med

WAS works best on network systems that use static IP addresses. Using Dynamic Host Configuration Protocol, DHCP, can sometimes lead to admin operations taking a long time to execute. The most common form of this problem has to do with the fact that the WebSphere Admin Server sometimes stores IP addresses in the Admin Repository. Since with DHCP an IP address can change dynamically, an address that was valid when the Admin Server was last started, might not be valid if the Admin is recycled. Hence, when the Admin Server tries to contact that address, it will pause until the attempt DNS name resolution times out. This can take as long as 5 seconds for each failed contact. Hence, it is recommended that WAS is not run using DHCP. If DHCP must be used, be aware of operations that seem to “stall” (i.e.,CPU idle time nears 100%) before the operation finishes.

7	EJB Naming	Med

When an instance of a WAS server is starting containing an EJB Container, the time it takes to bind the EJBs into the JNDI name space can be exacerbated by naming the EJB Homes with names that are deeply nested. This is described under *EJB Container,* in the *Factors Influencing Admin Performance and Scale* section of this paper. **Keeping EJB Home names simple can improve the startup time of your EJB Container up to 20%.**

9	Auto Loading Servlets	Med

Using the Servlet Autoloader option can influence the startup times of your WAS server. This is described under *Servlet Container*, in the *Factors Influencing Admin Performance and Scale* section of this paper.

10	Disabling Servlet Audit Logging	High

Audit logging of Servlet information is a very expensive operation. This is described under *Servlet Container*, in the *Factors Influencing Admin Performance and Scale* section of this paper. For example, WAS issues several audit messages for each servlet loaded into the system. Given a system with many servlets, running across many nodes, message congestion can become a real issue. Disabling Audit Logging can **improve the startup time of you're WAS server by as much as 2X**, depending on the number of Servlets defined as AutoLoaded in your system. To configure this option create a file called ***global.properties*** in the /websphere/appserver/properties directory. Add the following property to the file:

Property Name	Location
com.ibm.servlet.engine.disableServletAuditLogging=true	/websphere/appserver/properties/global.properties

An e-fix is required in order to apply this best practice. See the *E-Fixes referenced within this Paper* section for details on how to get this fix.

11	Datasource Connections	Low

When a Datasource is initialized within a WAS server it creates an inital set of connections to the database as specified in the Minimum Connection Pool Size setting in the Advanced Tab of the Datasource property sheet. If this setting is too large (e.g., greater than 50), it can impact the startup time of the WAS server. Keeping this setting small (e.g., 10 connections or less) will minimize the impact on Server startup.

12 Admin Datasource Tuning — Low

In WAS 3.0.2, the Datasource, which is used to connect the Admin Server to the Admin Repository Database, is not optimally tuned. This is described under *Datasource*, in the *Factors Influencing Admin Performance and Scale* section of this paper. In particular, in 3.0.2, there are too many connections defined in the connection pool of the Admin Server's Datasource. This leads to several problems. One problem is that the Prepared Statement Cache, which typically aids in the performance of database interactions, is not being utilized. To improve the hits on this cache, one can increase the size of the case.

Property Name	Location
com.ibm.ejs.sm.util.process.Nanny.adminServerJvmArgs= -Dcom.ibm.ejs.dbm.PrepStmtCacheSize=200	/websphere/appserver/properties/admin.config

13 Class Loading and Jar File Arrangements — Med

An estimated 40% of overall Managed Server startup time is spent in loading classes, Jars, and resource bundles. Here are the "rules of thumb" for ensuring that class loading is optimally performed:

1. **System Classpath.** It is often common practice for Servlets or EJBs to share common Java class libraries containing either third party Java packages (e.g., class libraries for managing XML or connectivity to proprietary services). These classes are typically placed in large Jars and/or class files. It is typically more efficient to place these classes or Jars on the WebSphere System Classpath rather than the Web Application Classpath. This is done by adding an entry to the *com.ibm.ejs.sm.adminserver.classpath* property in the Admin.Config file.

Property Name	Location
com.ibm.ejs.sm.adminserver.classpath=	/websphere/appserver/bin/admin.config

2. **Packaging of EJB in Jars.** This is described under *EJB Container- Packaging of EJBs*, in the *Factors Influencing Admin Performance and Scale* section of this document. The conclusion presented in this section is that it is slightly more efficient to group EJB into single Jars, rather than packaging each EJB in its own Jar.

3. **Class loading e-Fix.** There is an e-Fix for WAS 3.0.2 and 3.5 that provide a general optimizations for class loading. See the *E-Fixes referenced within this Paper* section for details on how to get this fix.

14 Servlet Redirector and Remote SRP Bean — Med

This is described under *Servlet Container- Servlet Redirector and Remote SRP Bean*, in the *Factors Influencing Admin Performance and Scale* section of this document. Removing Servlet Redirector and Remote SRP Bean configuration can help startup performance. If the Servlet Redirector service is not being used, simply right click on the Servlet Redirector entry in the left pane of the WebSphere Admin Console and select Delete. You will also have to go into each EJB Container and delete the Remote SRP Bean as well. If your application is not using EJBs you can delete the EJB container in which the Remote SRP Bean was contained.

15	**Start Production Servers using XML Config**	Med

It is much more efficient, from a performance and scalability perspective, to start WebSphere Application Server instances and/or Clones using either XMLConfig or WSCP. Starting servers using the WebSphere Admin Console will cause more interactions between the Admin Server and the Application Server instance. These interactions are allow the Admin Console to display the correct property sheet information or state information in the iconic view of the resources displayed in the Admin Tree. When using either XMLConfig or WSCP to start servers, there is no extraneous communication, hence the startup process can be significantly enhanced. This is especially true, if there are many resources being loaded within a given server (e.g., many Servlets and/or EJBs) For this reason, it is recommend that the Admin Console be used during developing and during pre-production tests. Prior to a production implementation the now stabilize configuration should be exported into an XML file utilizing the XMLConfig tool. Once in production, it is common for customers to solely use XMLConfig or WSCP to administer their systems. Lastly, if the Admin Console must be used, collapsing all containers views under a Server definition, will maximize the startup process.

16	**Serious Event Table**	Med

WebSphere Admin infrastructure allows for auditing of *serious events.* Examples of such events are a critical application exception or messages indicating a server has been started or stopped. When a serious event is generated it is logged backed to the Admin Repository. By default, WebSphere does not set bounds on this log. As the log grows, the time it takes to update this log will increase. This will eventually start to impact the overall performance and scalability of the WebSphere Admin System. For example, it will impact the time to start the Admin server or an instance of an Application Server. An option is available which will prevent this log from growing past a specified number of entries. This is done by adding the following property in the *admin.config* file.

Property Name	Location
com.ibm.ejs.sm.adminServer.seriousEventLogSize=1024	/websphere/appserver/properties/admin.config

17	**One Admin Server per Node**	Med

WebSphere Advanced Edition allows a single Admin Server to service multiple Server Nodes. However, for availability and performance reasons it is best to run a WebSphere Admin Server on each Node in the WebSphere Admin Domain.

Summary

WebSphere Application Server has a sophisticated Administrative system that provides the powerful ability of a single point of management for a cluster of servers. This document providee a series of insights into how to maximize the performance and scalability of the WebSphere Administrative System. This paper presents a summary of the best practices administering a high performance and scalable Admin System in WebSphere Advanced Edition. **Customers who have applied these best practices to their installation of WebSphere Advanced Edition have seen up to a 3X improvement in performance and salability.**

Glossary of Terms

Admin. Short term of Administration. A typical reference to some component of Administering WebSphere

Admin Repository - Database containing the WebSphere configuration

Application Server. An Application Server or Application Server instance, refers to the Java Virtual Machine that is running your Servlets, Java Server Page, and/or Enterprise Java Bean based applications.

Domain. An Admin Domain is the root of the WebSphere Admin hierarchy. The Admin Domain contains all associated Nodes (i.e., machines) and Servers (Application Server Instances) as well as a variety of other definitions including Datasources, Models and Clones.

EJB Container. A container for EJBs. Manages transactions between EJBs and the persistence of EJBs.

Managed Server. A Managed Server is an abstract term that applies to either an Application Server Instance or an Instance of the Admin Server. Since both the Application Server and the Admin Server are derived from the same code base, certain behaviors apply equally to both servers.

ServletEngine/ServletContainer. A ServletEngine is a container for Web Applications. A Web Application can contain a combination of Servlet and JSP definitions. A ServletEngine is synonymous with ServletContainer.

System/Server Nodes. A machine that is part of the WebSphere Admin Domain.

WAS. Abbreviation for the WebSphere Application Server. If an Edition is not specified, the reference typically applies to both Standard and Advanced Edition.

E-Fixes referenced within this Paper

Several of the best practices, referenced in this paper, requires electronic fixes to either WebSphere 3.5 or WebSphere 3.0.2.   The following table lists the aforementioned fixes:

Best Practice	Fix versions and description
Plug-in Configuration Regeneration **Best Practice #5**	*3.0.2.2 -* *3.5.2 -*
Disabling Servlet Audit Logging **Best Practice #10**	*3.0.2.2 -* *3.5.2 -*
Class Loading and Jar File Arrangements **Best Practice #13**	*3.0.2.2 - **PQ41070** — Improve performance of Jsp tag and Servlet engine classloader.* *3.5.2 - PQ42952 — Jarfileclassprovider does not cache contents for efficient class loading*