Best Practices for Large WebSphere Application Server Topologies

Document Version: 2.0

Date: March 31st, 2011

© Copyright IBM Corporation 2007-2011 All rights reserved.

Contributors (in alphabetical order)

Tom Alcott

Rohith Ashok

Keys Botzum

Michael Cheng

Jason Durheim

Bill O'Donnell

Brandy Byrd Gantt

Alexandre Polozoff

Jim Stopyro

Lohitashwa Thyagaraj

Peter Van Sickel

© Copyright IBM Corporation 2007-2011 All rights reserved.

Intended audience

- WebSphere Application Server customers
- Product planning leaders
- Client IT architects
- IBM executives

Contents

1 Introduction

The purpose of this document is to address various topics related to the question "How large can I make my WebSphere Application Server cell?" The answer to that question is not simple. There are numerous factors that play a part in allowing or constraining the size of your environment. In fact, the answer is likely to be different for each customer.

There is no one number that is the simple outer limit for the cell size. Even before discussing the factors involved, you need to decide what measurement to use for "size" of a cell. You might consider the following measurements:

- Number of WebSphere process instances in the cell
- Number of WebSphere nodes
- Maximum number of applications that can be deployed into a cell
- Largest number of workload-balanced clusters of application servers

In fact, each of these measurements can contribute to the limits of cell size. Each of these measurements has relationships with, as well as effects on, the other measurements.

This paper examines large topologies and the best practices involved in configuring and managing them, from several perspectives. Different limiting criteria for cell size are explored and the considerations for each criterion are documented. The goal of this document is to provide the most accurate information about WebSphere Application Server behavior and scaling constraints so that you can develop the plans for your large topology based on the functional and operational priorities that are the most relevant to your organization.

WebSphere Application Server is a product that is composed of many components. Not all of these components affect cell size. This document only addresses the considerations that are related to the components that can affect cell size.

Numbers quoted or referenced within this document are approximations and may not be precisely reflected in your specific environment. This document provides recommendations and guidelines, but it is not intended to be a guarantee of product performance or imply any warranty for product behavior.

This document is a point-in-time document and any decisions that are made from information that is provided in this document should be re-evaluated with each subsequent release of the document. There might be significant changes introduced to this document at any time that might cause side-effects that can potentially affect large topologies.

This document is intended to be updated periodically with the latest information that is collectively understood by WebSphere brand development leaders and domain experts. This document is a work in progress that is extended on an ongoing basis through the activity of the WebSphere Large Topology task force, a dedicated workgroup within the WebSphere development organization.

1.1 What are large topologies?

The following topologies have been validated with customer groups for WebSphere Application Server.

Topology 1: A balanced mix of nodes and application servers per node. This is a production environment requiring few changes to the environment.

- One deployment manager exists on a dedicated machine.
- 30 nodes exist and each node exists on a dedicated machine.
- At least 40 servers exist on each node.
- Approximately 1200 application servers exist in the cell.

Topology 2: More nodes exist, but fewer application servers exist on each node

- One deployment manager exists on a dedicated machine.
- 60 nodes exist.
- At least 20 servers exist on each node.
- Approximately 1200 application servers exist in the cell.

Topology 3: Fewer nodes, but more application servers on each node. This is a test environment involving continuous deployment of applications.

- One deployment manager exists on a dedicated machine.
- Two nodes exist.
- Over 200 servers exist on each node.

1.2 Technical challenges for large cell size

Regardless of which topological style or usage pattern you choose, the primary thing that limits the size of the cell is the need to support shared information across all or a large set of Application Server processes. The breadth and currency requirements for shared information, which is something that must be known by all or many application server instances within the cell, present a challenge for any distributed computing system. Various components of WebSphere have made engineering trade-offs in the design of the mechanisms for communicating and coordinating shared information, with the end goal of proper function when all Application Server instances are running and serving application requests.

What runtime components can affect the size of a cell? In brief, all components have the potential to implement logic that can impede scaling up a cell. Also, you can host applications so that they restrict the cell size. The components described in this document represent the main runtime components that need to exchange shared information. Therefore, these components are affected as the size of the cell grows. In particular, these components include the High Availability Manager, as well as any component that uses it for shared information, systems administration, system integration bus, and security.

2 WebSphere scalability by component

The following sections of the document examine the implementation of some specific components within WebSphere and what issues those components currently pose to support large topologies.

2.1 High Availability Manager components

2.1.1 Introduction

The High Availability Manager was introduced in WebSphere Application Server Network Deployment Version V6. An instance of the High Availability Manager runs inside every process in a Network Deployment cell, including the deployment manager, node agents, application servers and proxy servers. The High Availability Manager provides a set of frameworks and facilities that other WebSphere services and components use to make themselves highly available. These frameworks include:

- A *highly available singleton* framework that is used to make singleton services highly available. The singleton service is configured to run in a cluster and the High Availability Manager ensures that exactly one instance of the service is available at all times. This framework is used to provide a high availability option for the default IBM JMS provider (messaging engine) and transaction logs. It is also used internally (often invisibly) by other WebSphere services.
- A *replication agent* framework that provides a reliable, high speed messaging construct across a group of application servers. The memory-to-memory option for Data Replication Services (DRS) is built using this abstraction.
- A *bulletin board* framework that makes state data both shareable across processes and highly available. The WebSphere Application Server Network Deployment routing components (workload management and on demand configuration) use this framework to make routing information highly available.

The High Availability Manager itself consists of three subcomponents or layers. These components are:

- The HA Manager, or HAM layer. This layer provides the abstractions that are directly used by other Network Deployment services and components. These abstractions are based on fine-grained group services and fine-grained group messaging capabilities.
- Distribution and Consistency Services, or DCS layer. This layer provides coarse-grained group services including reliable one-to-many messaging capabilities between processes.

- Reliable Multicast Messaging, or RMM layer. This layer provides network transport level functionality, including a multicast messaging abstraction over TCP connections.

The High Availability Manager relies on *core groups*. A core group is a collection of firmly coupled processes which collaborate to form a distributed group communication service. It is a requirement that all members of a core group must be in the same cell

Core group services are scoped to be available only between members of the same core group. As a result, the frameworks provided by the High Availability Manager are also scoped to the core group. For example, memory to memory replication domains and highly available singletons can only be configured across application servers that are in the same core group. Another aspect of this design is that bulletin board data is also directly shared only between members of the same core group.

As the size of a cell increases, it may be necessary to partition the cell into multiple core groups, because core groups do not scale to the same degree as other cell constructs. When a cell has been partitioned, it is often necessary to share routing information between core groups. For example, a web application located in core group 1 may call an enterprise bean application located in core group 2. There are also cases where it is necessary to share routing information across cells. A Core Group Bridge provides this capability to extend the High Availability Manager bulletin board beyond core group boundaries. Core groups that are connected with a core group bridge can share routing data.

The High Availability Manager has certain constraints based on its design and implementation. Components that exploit the High Availability Manager inherit these constraints and then add scalability constraints of their own. Sections 2.1.2, 2.1.3, and 2.1.4 discuss the fundamental constraints of the High Availability Manager itself, while section 2.1.5 covers considerations of the components that use the High Availability Manager. Section 2.1.6 discusses core group sizing and section 2.1.7 discusses High Availability Manager strategies for large cells.

2.1.2 Core groups

As noted previously, core groups do not scale to the same extent as cells. Cells may need to be partitioned into multiple core groups. If routing information needs to be shared between the core groups, a core group bridge is used to interconnect the core groups.

While there are no WebSphere-defined limits on the size of a core group, there are practical limits. The practical limits are primarily driven by available resources and stability. The amount of resource used by the High Availability Manager and core groups depends on a number of factors, including the core group size, core group configuration settings, the amount of routing data required to support the deployed applications, and quality of service settings. The stability of the core group depends on a number of factors including the fix pack or APARS installed, network quality, and application characteristics. For example, applications that habitually cause Out of Memory conditions can cause core group instability, as discussed in section 2.1.6.2.2.

The topics of core group size, configuration settings, and stability are discussed in more depth in section 2.1.6.

2.1.2.1 The network

All members of a core group must be located on machines that are connected by a high speed local area network (LAN). Do not locate members of the same core group on machines that are connected by a wide-area network (WAN). Do not place members of a cell across a firewall, as a firewall provides no meaningful security between members of WebSphere processes.

2.1.2.2 Application routing considerations

The Network Deployment routing components use the High Availability Manager to make routing data highly available. The routing components route a number of different protocols, including HTTP, enterprise bean, JMS, SIP, and web services client requests. A number of protocols, such as SIP, JMS and web services clients always use the routing information that is managed by the High Availability Manager. For HTTP and enterprise bean routing, there are several use cases in which routing information is not obtained from the bulletin board of the High Availability, but obtained in an alternate manner:

- Routing the HTTP protocol using the web server plug-in when memory to memory session replication has not been enabled.
- Routing the enterprise bean protocol when the bean is co-located with the requestor, or the bean is in a non-clustered application server.
- Routing to enterprise beans that are running on the z/OS platform.

The amount of routing data that is present in the cell depends on a number of factors, including the number of clusters and cluster members, the number of applications or JMS destinations, the protocols being routed and the composition of the applications (for example, the number of modules in the application). Routing data is long-lived, meaning that the routing data for a given server remains cached in memory while the server is running. The total size of all the routing data varies by topology. In some large topologies, the amount of heap consumed by the routing data can be in the tens of megabytes.

2.1.2.3 Quality of service considerations

There are two quality-of-service aspects that merit special consideration.

2.1.2.3.1 Active failure detection and heart beating

The High Availability Manager monitors the health of all running core group processes using both active and passive mechanisms. Whether a process is stopped normally or manually, socket closing events provide passive notification to other core group members that a process has exited. An active mechanism (periodic heart-beating) is used to detect other conditions such as network partitions or hung/unresponsive processes. Each process heartbeats the connections it has opened to all other running core group members. When a heartbeat timeout occurs, the non-responsive process is marked as failed and recovery is initiated. Any highly available singleton services that were running on the failed process are failed over to a healthy member. Routing data for the failed process is also removed from the routing tables managed by the High Availability Manager.

Socket closing events are typically propagated quickly, so failover due to socket closing events also happens quickly. For active heart-beating, the default configuration settings provide a 30 second heartbeat interval and a 180 second heartbeat timeout, meaning that failovers initiated by the active failure detection mechanism takes longer than failovers initiated by socket closing events. This default setting represents a compromise between failover time and background CPU usage. If faster failover is required, then the configured heartbeat timeout can be lowered, at the cost of additional background CPU usage.

2.1.2.3.2 Heartbeat and failure detection overhead

The amount of background CPU used by the HAManager for heart-beating and failure detection is affected by the heartbeat interval and core group size. Starting with a core group of 100 members as a baseline using the default heart beat interval of 30 seconds, approximately 20% of the background CPU used by a WebSphere product application server at idle is spent on heartbeat processing. Using this core group size as our reference point:

- Increasing the heartbeat period to infinity has limited effect on the background CPU usage. With no heart-beating, the background CPU usage would decrease by 20%.
- Decreasing the heart beat period to 3 seconds causes the background CPU usage to more than double.
- Increasing the core group size from 100 members to 200 members (with a default heart beat period) causes the background CPU usage to increase by 50%.

Observing a high background CPU at idle can be indicative of the core group (or groups) approaching the practical limit for your infrastructure and deployment. If you encounter high idle CPU, you should explore decreasing the number of members in existing core groups by moving processes to a new bridged core group to reduce the background CPU.

2.1.2.3.3 Memory to memory replication

When memory-to-memory replication is used, it is possible to configure the number of cluster members to which data is replicated. Increasing the number of replicas increases the cost of replication and resource usage by the High Availability Manager. For example, replicating data to all members of a cluster can result in significant load being placed on the network and CPU, if the amount of data being replicated is large.

2.1.3 The high availability coordinator

Every core group contains a *High Availability Coordinator*. By default, the coordinator functionality is assigned to a single process. The coordinator consumes extra memory and CPU cycles. If the coordinator workload becomes excessive, the coordinator functionality can be partitioned, or shared among multiple processes. For large core groups, consider multiple coordinators.

You can configure the one or more coordinators to run on a preferred process or processes. If the preferred processes are not configured, or if the preferred processes are not running, the High Availability Manager chooses the processes with the lowest lexical names from the set of currently running core group members to act as coordinators. If you choose not to configure preferred coordinators, then as processes start and stop, the coordinators might frequently move from one process to another. When a coordinator moves, the new coordinator must assemble the entire global state for the core group in its local memory, consuming extra CPU cycles and network bandwidth.

Therefore, it is a best practice to configure one or more preferred coordinator processes for each core group. This limits the movement of the coordinator and number of state rebuilds. Configuring preferred coordinators is a best practice. Ideally, assign processes that do not host applications and are located on machines with spare capacity as preferred coordinators.

In a topology that contains core group bridges, it is a best practice to create stand-alone application server processes that do not host applications to function as both bridge interfaces and preferred coordinators.

2.1.4 Core group bridge

The core group bridge, otherwise know as the bridge, is a service that can be enabled on any WebSphere Application Server process, such as the node agent, deployment manager, or application server. A bridge is used to expand bulletin board communication across core group boundaries. Bridges are typically required in a topology for one of the following reasons:

- A cell contains many processes and needs to be split into multiple core groups for scalability purposes.
- Multiple cells contain components that need to share bulletin board information across cell boundaries.

2.1.4.1 Failover and startup

Startup and bridge failover are the most resource-intensive times for a bridge. During startup, bridges handle subscriptions from every process in their local core group and any resulting bulletin board posts. Tests show that it took 20 - 30 minutes to start a 10-core-group topology with approximately 350 servers total. Startup time increased if a preferred coordinator was not assigned in each local core group.

During bridge failover, each remaining bridge receives state information from the other bridges in its access point group. The remaining bridges must process every bulletin board subscription that was previously handled by the failed bridge. In a large topology, it can take 5 minutes or more for state information to be recovered. This delay can be problematic for the proxy server for WebSphere Application Server and the on-demand router (ODR) for WebSphere Virtual Enterprise, which depend on the bulletin board for routing information.

2.1.4.2 Configuring core group bridge

To ensure high availability, each core group should have two bridges per core group. Although any WebSphere Application Server process can be a bridge, creating a stand-alone application server that does not host any production applications to operate as a bridge is likely best. This configuration helps to isolate the applications from the high CPU and memory requirements of a bridge in a large topology. If resource limitations prevent creating stand-alone servers as bridges, use node agents as bridges, particularly in small topologies. However, in large topologies, the large amounts of bulletin board traffic handled by a bridge makes it more susceptible to failure and results in out of memory errors or thread starvation. Therefore, it is best for the bridge to have a dedicated process.

Complete the following steps to bridge core groups in the same cell:

1. Identify a process to serve as the bridge process in each of the core groups. You can choose multiple processes, which are called *bridge interfaces*.
2. Collect the set of bridge interface processes, which are identified in the previous step, into an access point for each core group.
3. Collect the appropriate access points into an access point group.

There are a number of options that should be considered in the various steps, as discussed in the following sections.

2.1.4.3 Selecting bridge interfaces

There are a number of things to consider when selecting processes to serve as bridge interfaces. First, for high availability and load sharing, it is a best practice for each core group to have two bridge interfaces. If one bridge interface goes down, the other bridge interface takes over its workload. Second, bridge interfaces should be processes that are not frequently stopped. If all of the bridge interfaces in a given core group are stopped at the same time, applications cannot route bulletin board traffic until one of them is restarted. The third consideration is the usage pattern. When the cell is in a normal operating state, the bridge interfaces are fairly passive and perform little work. However, if a large portion of the cell is stopped or restarted at the same time, bridge interfaces are subject to large, sudden bursts of traffic.

Complete the following steps after you select bridge interfaces:

1. Create two stand-alone application servers in each core group to serve as dedicated bridge interfaces.
2. Create the bridge interfaces on different machines.
3. Configure the bridge interfaces as the first and second most preferred servers for the High Availability Manager Coordinator. This configuration isolates the High Availability load away from the normal application servers.
4. Initially, set the heap size for the bridge interfaces to a large value.
5. Enable verbose garbage collection on these processes, and monitor the heap usage. You can reduce the heap usage value as appropriate.

2.1.4.4 Bridge topologies

The bridge topology depends on how the access points are collected into access point groups. There are essentially two viable options.

2.1.4.4.1 Mesh topology

Collect all of the access points into a single access point group. This configuration is referred to as a mesh topology, as shown in Figure 1.

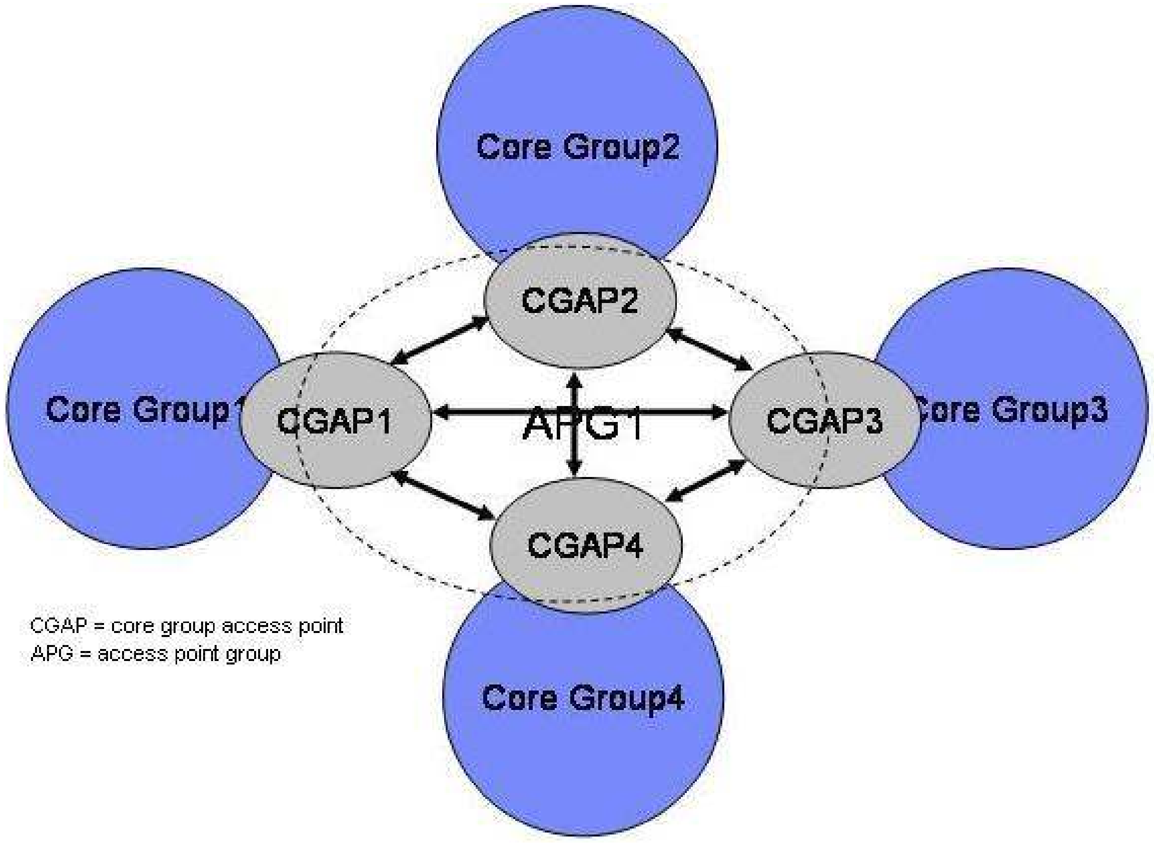

Figure 1 : Mesh topology

2.1.4.4.2 Doubly linked chain topology

Order the access points from 1 to N. Pair numerically adjacent access points into access point groups with two access points in each access point group. For example, pair 1 with 2, 2 with 3, N-1 with N and N with 1. This configuration is referred to as a doubly linked chain topology, as shown in Figure 2.

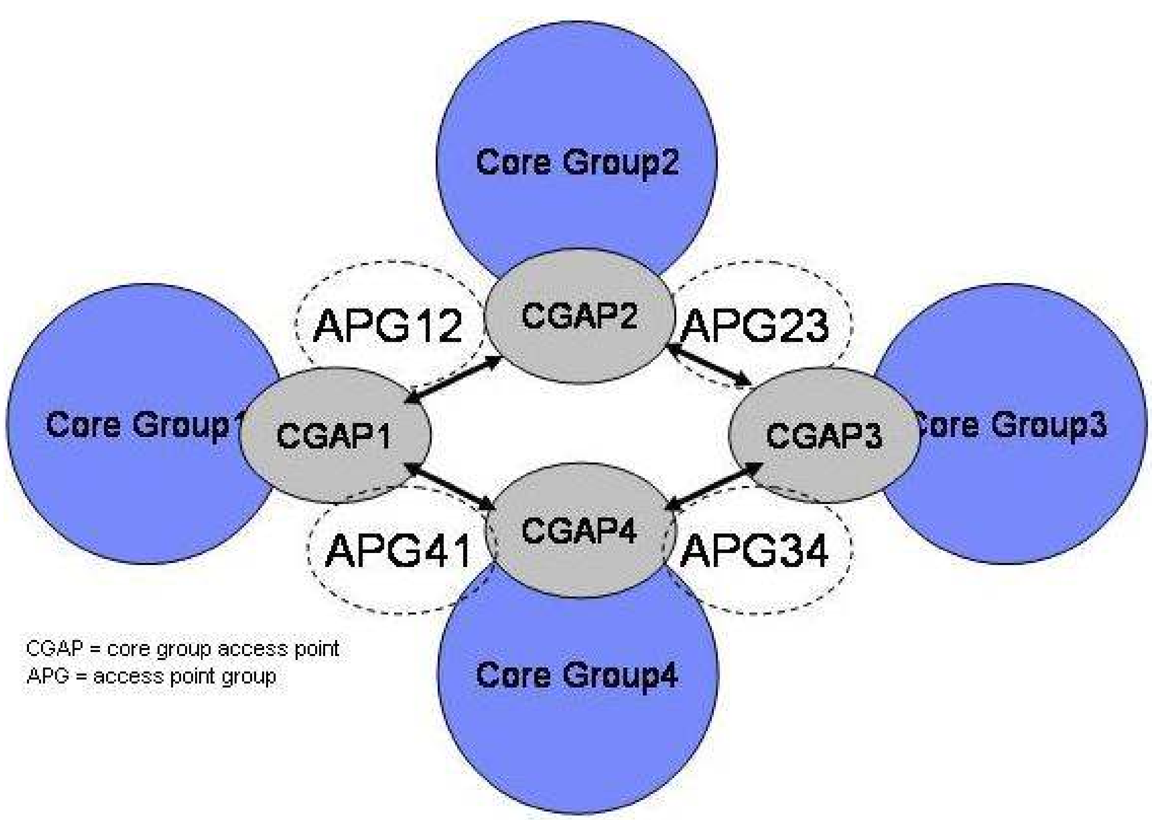

Figure 2: Doubly linked chain topology

Other topologies are possible, using a combination of mesh and chain. Each access point group is also another core group, subject to scalability limitations.

The advantage of a mesh topology is that all bridge interfaces can directly intercommunicate. Therefore, this topology offers the lowest propagation latency. The disadvantage of this topology is that it does not scale indefinitely. The number of bridge interfaces in a single access point group should not exceed 50.

The advantage of a doubly linked chain topology is that it scales indefinitely, as each link is a separate access point group, and therefore a separate core group. There are two major disadvantages to this topology. The first disadvantage is the variable and increased delay to propagate routing data from one core group to another. The second disadvantage is that if the bridge interfaces in two core groups go down, the bridge topology will partition.

Under normal conditions, the mesh topology is preferred.

2.1.5 Components that use High Availability Manager services

2.1.5.1 EJB workload management

Starting in WebSphere Application Serve Version 6, EJB workload management (WLM) depends critically on the High Availability (HA) manager to function properly. WLM used the HA manager bulletin board as a means of sending pieces of data from one process to another. If the HA Manager is not functioning properly, WLM may not be able to route requests to all members of the cluster. In previous releases WLM relied on the deployment manager to aggregate and propagate the run time cluster information to all members of cluster. In WebSphere Application Server V6, WLM uses the HA manager bulletin board to aggregate and propagate the run time cluster description information. Each cluster member posts data to the bulletin board about its cluster description information and all servers with access to the bulletin board information are informed of these posts. Recall that bulletin board information is shared within a core group but can also be bridged. Figure 3 and Figure 4 illustrate this process.

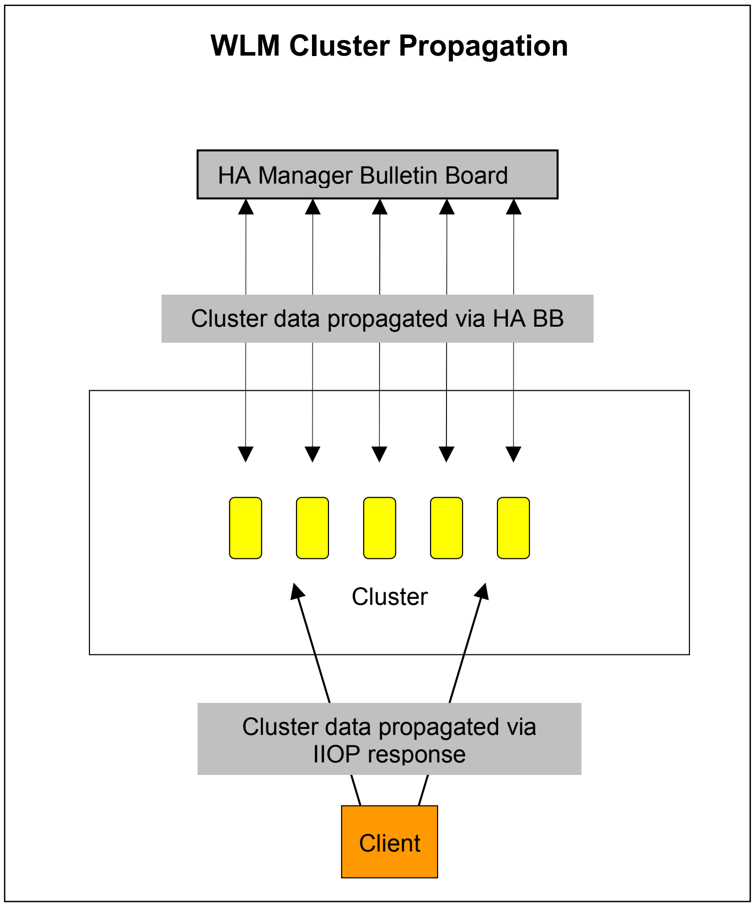
WLM Cluster Propagation
HA Manager Bulletin Board
Cluster data propagated via HA BB
Cluster
Cluster data propagated via IIOP response
Client

Figure 3: WLM cluster propagation for a client

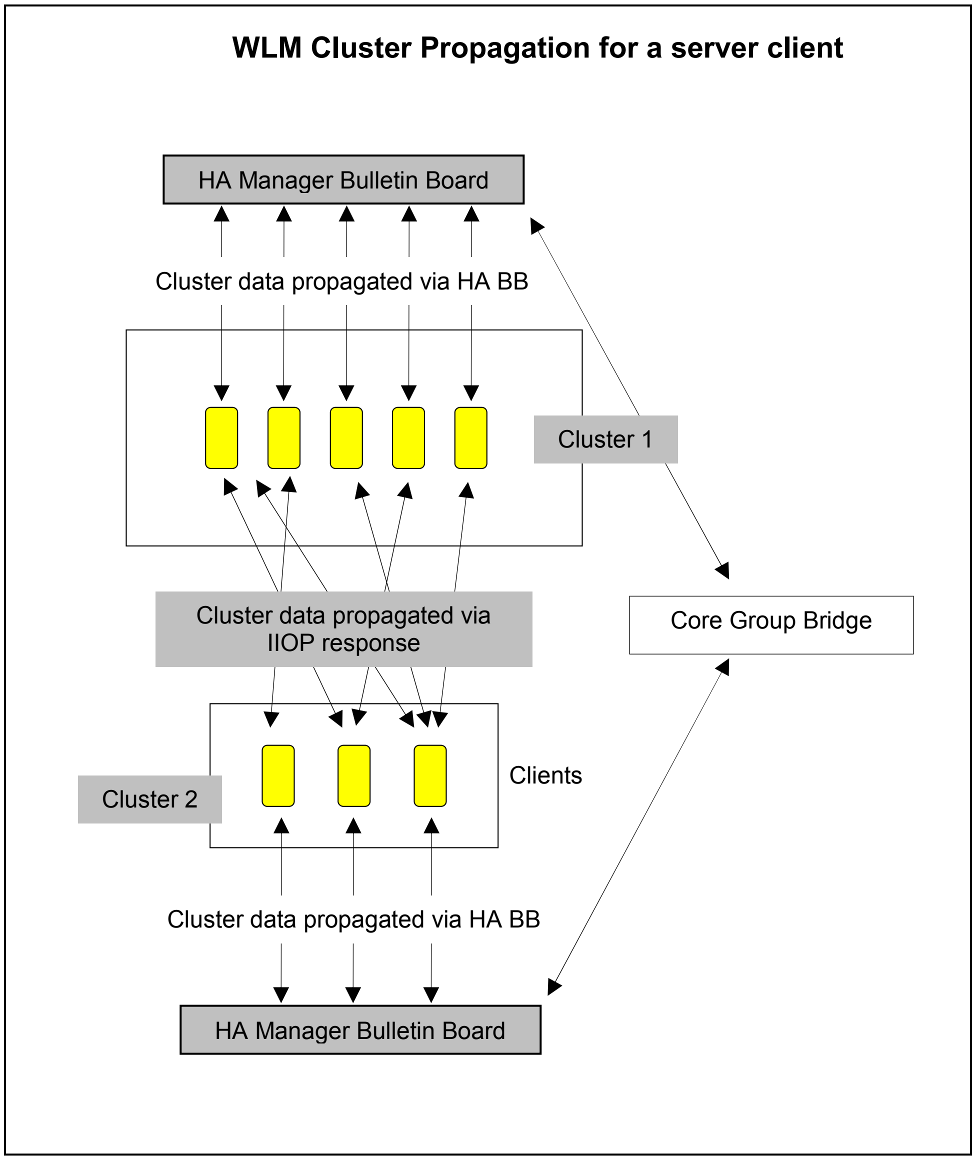
WLM Cluster Propagation for a server client
HA Manager Bulletin Board
Cluster data propagated via HA BB
Cluster 1
Cluster data propagated via IIOP response
Core Group Bridge
Cluster 2
Clients
Cluster data propagated via HA BB
HA Manager Bulletin Board

Figure 4: WLM cluster propagation for a server

As shown in Figure 3, enterprise bean stand-alone clients rely on WebSphere Application Server run time checking to see if the client has the most current WLM cluster information when a business method request occurs. If needed, the enterprise bean client ORB is updated with cluster information as part of the business method response.

As shown in Figure 4, when a client is a running in a WebSphere Application Server, the client not only has access to WLM cluster information in the same manner as a stand-alone client, but also has access to the same information from the HA manager bulletin board. The client running in a WebSphere Application Server uses the WLM cluster information that is the most current, whether it is from the bulletin board, or as part of the business method request/response flow.

The run time cluster description is a tree data structure that represents, in memory, the topology of cluster members, their weights, states (running, stopped, lost contact, and so on), endpoint data, such as IORs, and channel chains. This information is propagated by the bulletin board or received in a response from an EJB request. The cluster description also contains an epoch number that is updated anytime the cluster description changes. The client runtime honors the latest epoch number and update the cluster information accordingly. Under normal conditions, the cluster description information maintained in the bulletin board should be the most current, but the client runtime will use whatever cluster information is most current.

2.1.5.2 Data replication service

The Data Replication Service (DRS) is a component that depends on the High Availability Manager for some of its underlying capabilities. There are several other components of the product that build on the DRS for their functionality. Example components that depend on DRS (known as *DRS clients*) include:

- HTTP Session distribution when using memory to memory replication
- Dynamic cache distribution, when using memory to memory replication
- Stateful session bean failover, which relies on memory to memory replication
- Session Initiation Protocol (SIP)

2.1.6 Core group sizes

The limits on the size of a core group are practical, not programmatic. The most important considerations in determining core group sizes are resource usage and stability.

A discussion of resource usage by the High Availability Manager includes both resources used directly by the High Availability Manager and resources used by or on behalf of other services and components. In addition to the components in WebSphere Application Server Network Deployment, other WebSphere family products such as WebSphere Virtual Enterprise and WebSphere Portal add additional resource usage. The discussion that follows applies directly to WebSphere Application Server Network Deployment. Other WebSphere family products may introduce their own core group size recommendations or requirements

2.1.6.1 Resource usage by High Availability Manager

The High Availability Manager uses CPU, memory, and network resources. Generally speaking, memory is not a major factor in determining core group size. The amount of long-term heap memory required for routing data is determined by the topology and applications installed, not by the core group size. Splitting a cell into multiple core groups does not reduce the memory required for the routing data. Therefore, the size of the core group is determined almost exclusively based on the CPU required to establish and maintain the group communication service.

The High Availability Manager uses CPU to establish network connections and group communication protocols between running members of the core group. As processes are started, connections are opened to other core group members and the group membership and communication protocols are updated to include the newly started members in the group, or "View". This change is often referred to as a "View Change." As processes are stopped, connections are closed and the group membership and communication protocols are updated to exclude the stopped members.

Therefore, starting or stopping a process causes the High Availability Manager to use CPU to open or close connections and update the group communication service. This means that starting or stopping one process causes some CPU usage by all other running core group members. As the size of the core group grows, the number of connections and size of the group membership will grow, meaning that more CPU will be used for large core groups than for small ones. There is also some short-term usage of heap memory to send the network messages required to update the group communication service.

In general, it is more efficient to start or stop groups of processes at the same time, allowing the High Availability Manager to efficiently consolidate multiple group membership and communication protocol changes within a single view change.

When processes are not being actively started or stopped, the High Availability Manager operates passively, consuming minimal amounts of CPU and memory for monitoring activities, such as the active failure detection (heart beating) discussed in section 2.1.2.3.1. Starting, stopping, or restarting a process outside of a cell or node restart is expected to be a relatively rare event. Generally speaking, the High Availability Manager itself does not compete with applications for CPU time, since most of its CPU usage is limited to the points in time when processes are actively starting or stopping, often when the cell is being started, when applications are not yet running. However, High Availability Manager activities initiated by other components such as memory to memory replication can cause CPU usage at any time.

The High Availability Manager uses a relatively small amount of long-lived memory for its own internal data structures. It also uses short-lived heap memory for network messages, including internal messages and replication messages. Most of the long-term heap memory usage that is normally attributed to the High Availability Manager is for bulletin board data (routing information) that the High Availability Manager is managing on behalf of the routing components. This routing data is published to the bulletin board during process startup, but rarely changes at run time.

An additional factor to consider is the number of sockets that are consumed to create the connections between core group members. The members of a core group form a fully connected network mesh, meaning every member connect directly to every other member. The total number of sockets used to connect all members of a core group approaches n**2, where n is the number of core group members. Suppose for example that you tried to create a core group of 200 members on a single machine. The number of sockets required would be 200 x 199 or 39,800 sockets. The same 200 members split into 4 core groups of 50 members each would require 4 x 50 x 49 or 9800 sockets.

To summarize, there are three resource considerations for the High Availability Manager

1. The resources (CPU and short-lived memory allocations) that are required to establish and maintain the group communication protocols and membership.
2. The resources (long-lived memory allocations) that are required to establish and maintain the highly available bulletin board (routing) data.

3. The CPU resources that are required for background monitoring activities, such as active failure detection (heart beating).

2.1.6.2 Core group stability

A core group is a collection of firmly coupled, processes which collaborate to form a distributed (networked) group communication service. In unusual cases, problems on one core group member can affect the functioning of the entire core group. Typically these problems are simply a manifestation of another underlying issue such as network problems or heap memory issues. When a problem that affects the group communication service occurs, the High Availability Manager attempts to recover from the problem. Recovery continues forever, in the hope that the underlying problem is eventually resolved, at which point recovery succeeds. Core groups that are waiting in error recovery are said to be unstable. Unstable core groups cause increased CPU usage and other problems such as the inability to properly route applications.

2.1.6.2.1 Network issues

There are a number of network-related issues that can cause core groups to become unstable. The most commonly encountered issues are:

- Network Configuration: The most common examples of this are duplicate port assignments or duplicate IP address assignments. The High Availability Manager attempts to detect these issues at process startup. If a network configuration issue is detected, the appropriate HMGR message is logged.
- Unreliable Networks: There are many possibilities here, including improperly functioning network cards, network issues and software problems in the network components underneath the High Availability Manager (TCP/IP or Java Virtual Machine).
- Problems in the High Availability Manager connection management code that recover connection failures due to unusual network events. These problems have been remedied in APAR PK81240.

2.1.6.2.2 Memory issues

There are a number of memory-related issues that can cause core groups to become unstable. The most commonly encountered issues are:

- A core group member has gone Out of Memory. An Out of Memory process operates unpredictably. Typically, such a process can respond to heartbeats and can accept new incoming connections, but may be unable to receive or process or send messages related to updating the group communication protocols and membership. In a limited number of instances, configuring preferred coordinators for the core group allows the High Availability Manager to continue functioning properly when a core group member has gone Out of Memory. However, Out of Memory processes must be recycled as quickly as possible.
- A core group member is severely memory constrained and is thrashing. Typically, the process is running frequent, full garbage collection cycles, which prohibit normal threads in the process from running. In degenerate cases, physical memory

is overcommitted and garbage collection causes excessive paging. In such cases, normal threads may be unable to run for minutes at a time, causing heartbeat timeouts. The process is marked as failed by other core group members and removed from the group. Once the garbage collection cycle completes, the process appears to recover and is added back to the group. The next garbage collection cycle causes this sequence of events to repeat.

The presence of HMGR0152 CPU Starvation messages in the logs is a good indicator of long garbage collection. HMGR0152 messages with large thread delays are a good indicator of paging.

Installing APAR PK95297 enables the High Availability Manager to log advanced diagnostic messages when Out of Memory processes are interfering with normal operations. These messages allow the administrator to quickly identify the problematic process and recycle it. Therefore, installing this APAR can significantly improve problem determination.

2.1.6.3 Required core group tunings

2.1.6.3.1 Core group internal protocol improvements

In Versions 6.1 and 7.0, significant improvements have been made in the efficiency of the group communication protocols. These improvements must be enabled by setting the IBM_CS_WIRE_FORMAT_VERSION core group custom property to a value of 6.1.0. The following WebSphere Application Server Information Center link contains more details on how to configure this custom property.

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/crun_ha_protocol_ver.html

These improvements are available only in Version 6.1 and beyond.

2.1.6.3.2 Core group bridge improvements

Significant improvements have been made in the memory utilization and failover characteristics of core group bridges. These improvements are available in all releases. These improvements must be enabled by setting the IBM_CS_HAM_PROTOCOL_VERSION core group custom property to a value of 6.0.2.31. The following WebSphere Application Server Information Center link contains more details on how to configure this custom property.

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/crun_ha_protocol_ver.html

To enable these improvements, the service levels required for the various versions are 6.0.2.31, 6.1.0.19 and 7.0.0.1. The recommended service levels are 6.0.2.37 or later, 6.1.0.29 or later, and 7.0.0.7 or later.

2.1.6.3.3 Network connectivity improvements

Significant improvements have been made in the area of network connectivity and recovery from network problems. These improvements are available in APAR PK81240, which has been included in the normal maintenance releases (6.0.2.37, 6.1.0.21, 7.0.0.5). It is also available as a stand-alone fix. This APAR must be installed.

2.1.6.3.4 Transport memory settings

For Version 6.0 and 6.1, there are two memory or buffer size settings associated with the core group transport. The default values for these settings are sufficient for small core groups of 50 members or less. For core groups of over 50 members, these settings must be increased from the default values. Change both of these settings to "100". The following WebSphere Application Server Information Center link contains more details on the two settings and how to change them.

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/trun_ha_cfg_replication.html

For Version 7.0, the two memory configuration settings have been consolidated into a single setting that has a default value of 100. Therefore, no tuning is necessary for Version 7.0.

Increasing the value of these transport memory settings does not directly translate into more statically allocated memory or long-term heap memory usage by the High Availability Manager.

2.1.6.4 Core group sizes by release

The following recommendations assume that all the required tunings listed in section 2.1.6.3 have been performed. The following recommendations assume that WebSphere Application Server Network Deployment is being deployed on relatively modern hardware, CPU and memory are not constrained, applications are well behaved, and the network is stable. In determining a core group size, it is also important to factor in the information contained in section 2.1.6.2 regarding core group stability.

For mixed cell scenarios, follow the recommendations for the oldest release in the cell. For example, a cell containing both Version 6.0 and Version 7.0 nodes must follow the recommendations for Version 6.0.

2.1.6.4.1 Version 6.0

Version 6.0 is the least scalable release from a High Availability Manager perspective. For Version 6.0:

- Core groups of 50 members will work without issue.
- Core groups of up to 100 members should work without issue in many topologies. Exceeding a core group size of 100 members in this release is not recommended.

2.1.6.4.2 Version 6.1 and version 7.0

Versions 6.1 and 7.0 include improvements in the group communication protocols. For these releases, when the improvements have been enabled with the IBM_CS_WIRE_FORMAT_VERSION custom property:

- Core groups of up to 100 members will work without issue.

- Core groups containing more than 100 members should work without issue in many topologies. Exceeding a core group size of 200 members is not recommended.

Important: If you do not enable the improved group communication protocols, you must use the Version 6.0 recommendations.

2.1.6.5 Core group related messages

All the messages are described in the WebSphere Application Server Information Center. They are easy to find by searching the message code.

The messages of interest start with the following codes:

1. DCSV – Distribution and Consistency Service.
1. HMGR - High Availability Manager
2. CWRLS – Recovery Log Service (These messages are interesting because RLS issues warnings when it cannot do work due to HAM issues, for example, CWRLS0030W, which describes the need to tune the core group configuration.)

The messages with a *W* as the last character are warnings that may point to core group problems as noted in the table. (There are some exceptions where a Warning message can be treated as "normal", for example, DCSV1115W depending on circumstances, or HMGR0008W if the core group configuration has been tuned.) Usually, a message with an "I" as the last character is informational and is expected, particularly during startup and shutdown of application servers. (There are some exceptions where an Informational message is really a warning that there is a problem that needs to be addressed, for example, DCSV0008I.)

Message Code	**Description**	**Comments**
CWRLS0030W	Waiting for HAM to activate recovery processing.	This message is a sign of distress in the core group. The HAM is having difficulty initializing its view and thus cannot support the recovery service. You are likely to also see the DCSV0008I message.
DCSV0008I	Failed to form an initial view.	This message reports the number of tries as well as a list of all the members in the view, which will be a number less than the configured members of the core group. This message needs to be examined with other messages, such as DCSV1036W, which provide information about members to which a connection

		could not be made.
`DCSV1032I`	DCS received a new connection.	Application servers report this event when another member of the core group connects to them. This is normal and occurs frequently during startup.
`DCSV1033I`	All new view members have reported in with the new view identifier.	A normal informational message, particularly during server startup.
`DCSV1036W`	An unusual connectivity state occurred with a core group member.	This message is a warning that indicates an issue with establishing a connection between two JVM instances. The cause may be due to the target process being unresponsive (OOM or hung), network configuration issues, or problems on the underlying network.
`DCSV1051W`	DCS high severity congestion event for out-going messages.	If this warning is seen, then make sure that the DCS data stack and transport settings are set as described in section 2.1.6.3.4, or reduce the load on the application servers. Alternatively, the core group has too many members.
`DCSV1052W`	DCS medium severity congestion event for out-going messages.	Make sure that the DCS data stack and transport settings are set as described in section 2.1.6.3.4.
`DCSV1054W`	DCS medium severity congestion event for incoming messages.	Make sure that the DCS data stack and transport settings are set as described in section 2.1.6.3.4.
`DCSV1112W`	A connection was closed due to a heartbeat timeout.	This message indicates that a network partition may have occurred, or that the network is not delivering messages within a reasonable amount of time. It can also be an indication of a core group that is too large.
`DCSV1115W`	Another member closed its connection to the member reporting this event.	If the other member was stopped then this is a normal event. If the other member was not stopped, then this message indicates a symptom of core group being too large. Or there may be a network communication issue between

		the members.
DCSV1134W	Received an additional connection from another core group member.	Can occur due to a duplicate port (DCS_UNICAST_ADDRESS end point) in the configuration. If the other core group member also logs a DCSV1036W warning, there may be network issues, or overbooked CPUs.
DCSV2001W	Time out occurred during the synchronization procedure.	Can occur when the system is overly busy, or the network is slow in delivering messages.
DCSV2004I	DCS view synchronization completed successfully.	This message shows up as part of a normal application server startup.
DCSV8050I	A view change has completed.	This message indicates that members of the view are added, dropped out, or removed. The current number of members in the view is reported.
DCSV8104W	A warning that a member is being removed from the view by request from another member.	This message is normal if the member being removed is in fact stopped. However this message may also point to a communication problem between the core group member being removed and the other core group members. If the core group member being removed is still running, then the potential communication problem needs to be rectified.
DCSV9421W	A general networking problem occurred.	An “open connection” request was handed to the operating system and a timeout specified, but no response (succeed or fail) has been received. There is a problem in the underlying OS or network.
HMGR0008W	Warning about the number of members in a core group > 50.	If you have not performed the core group tuning described in section 2.1.6.3, perform the recommended tuning to avoid core group issues.

`HMGR0024W, HMGR0027W, HMGR0028W, HMGR0064W`	Various issues with network configuration (duplicate ports, duplicate IPs, no IP, no DNS mapping for an IP).	See the specific messages for more details.
`HMGR0152W`	CPU starvation	Typically due to long GC cycles preventing ready-to-run threads from being dispatched. If the starvation periods are very long (hundreds of seconds) then paging is probably involved. In rare cases, may be due to overbooked CPUs.
`HMGR0218I`	A new core group view has been installed.	Reports which core group the reporting process is part of, and some other specifics about the core group.
`HMGR0206I, HMGR0207I, HMGR0228I`	HA coordinator is/is not/is no longer the active coordinator.	Each JVM logs one of these messages on every view change to report whether it is or is not an active coordinator. In recent fix packs, the message also contains a list of the current active coordinators.

2.1.7 Configuring High Availability Manager in large cells

There are several strategies for handling large cells that may need to be partitioned into multiple core groups.

- Partition the cell into multiple core groups and connect all core groups with core group bridges.
- Determine which processes use functionality that requires the High Availability Manager. Move all processes that do not use High Availability Manager-related functionality to a separate core group and disable the High Availability Manager on those processes. Take the remaining processes, partition them into multiple core groups, as needed, and connect these core groups with core group bridges. Do not include the core group containing the processes where the High Availability Manager has been disabled in the bridging topology. See section 2.1.7.1 for the drawbacks of this strategy.
- Determine the routing relationships between the various applications to be deployed into the cell. Partition the cell into multiple core groups and deploy the

applications with routing interdependencies in the same core group. Account for the fact that enterprise bean routing uses the Location Services daemon located on the node agents, meaning that the node agents for the local node must be in the same core group as the servers hosting enterprise bean applications. Even though in this topology core group bridges are unnecessary, bridge all your core groups that are running with HAManager enabled. Even if there are no obviously dependencies between the core groups, bridging the core groups together will help you avoid possible issues in the future should the topology change.

- Another option is to partition the large cell into multiple cells. In many cases this partitioning makes good sense from an administrative perspective for reasons that have nothing to do with core group sizes. See section 2.2.1.2.1.

2.1.7.1 High Availability Manager not needed

The information center contains an article that details a set of services that depend on the High Availability Manager being enabled in order to function properly. This list is subject to change at any time, as more WebSphere services can change to use the High Availability Manager at any time. Moreover, since many internal components employ the HA Manager infrastructure or rely on internal services that employ the HA Manager, the services listed should not be considered as an all-inclusive list of services affected by disabling the HAManager.

Important: Disabling the HAManager might cause some critical functions to fail.

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/crun_ha_ham_required.html

For the reasons outlined previously, rather than disabling HAManager, either create multiple cells or partition the cell into multiple core groups and create bridges. Even if you do not currently use a component that requires HAManger, you may require one at a later time.

2.1.7.2 Partitioning the cell

When planning to partition a cell into multiple core groups there are a number of factors that need to be considered.

- All members of a given cluster must be assigned to the same core group. This rule is enforced by the WebSphere Application Server Network Deployment.
- For version 6.0 and 6.1 cells, each core group must contain one administrative process (a node agent or the deployment manager). This restriction is removed for Version 7, and for Version 6.1 when fix pack 6.1.0.27 (APAR 83664) is installed.
- Members of a cell should not be separated by a firewall.
- All core group members should be located on the same LAN. Core groups should not span WANs.
- Core groups need to be connected with a core group bridge if routing data needs to be shared between core groups.

2.2 Systems management components

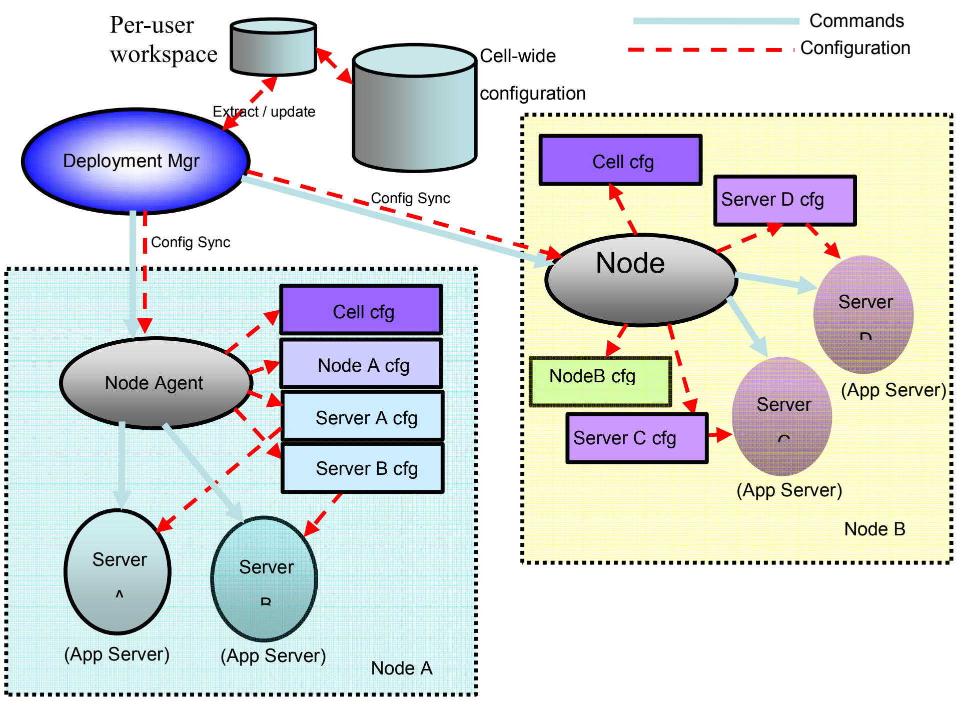

Figure 5: Systems management overview

The systems management components for WebSphere Application Server attempt to keep the configuration management and operational management functionality separate. As shown in Figure 5, configuration management is based on an internal model that is stored in XML and other files on the file system in the cell-wide configuration repository. The operational management model uses the Java Management Extensions (JMX) framework. Application deployment is composed of both configuration management to persist the deployment and bindings information, and operational management to start or restart the application.

One can consider the system management component as a special application that helps the user to manage the WebSphere environment. As with any application, it needs to be tuned for the environment where it is running for performance and scalability, especially for a large topology environment.

2.2.1 Configuration management

In a WebSphere Application Server Network Deployment cell, the deployment manager holds the cell-wide master configuration repository. As the master repository, the deployment manager holds the authoritative copy of every configuration document for every node in the cell. Each node does not replicate the entire configuration, but rather holds a sparse tree that contains only its own node and the serverindex.xml file for every other node. In a process called NodeSync, the node agent for each node contacts the deployment manager on a periodic basis to determine whether any files have changed. If files have changed, the updated files are downloaded and placed in the sparse tree for the node.

As the cell grows, more configuration data is stored in the master repository. This growth includes growth in applications, nodes, and application servers. This growth can affect users in terms of time needed to perform configuration operations, and the ability to support concurrent users. Configuration management is first described in more detail, followed by the best practices when making configuration changes.

2.2.1.1 Workspace sessions and configuration changes

When the user logs in to the administrative console or invokes scripts through wsadmin, a different session is used to track the configuration changes made by that user. Associated with each session is a distinct temporary workspace to store the changes made by that user. Files from the master repository are first extracted to the workspace where the changes are made. At the end of the configuration changes, the user must perform a save operation to save the changes back to the master repository.

If the changes made by different sessions modify different configuration files, all operations performed by concurrent sessions are expected to succeed. However, if the configuration files modified by different sessions overlap, only the first to save may succeed. Saving other concurrent sessions is rejected to prevent overriding changes already made by the first session. Those familiar with database concurrency control will recognize this behavior as a form of optimistic concurrency control that works well when there are few overlapping changes.

There is also special case handling to allow concurrent deployment of different applications from different sessions. Without special case handling, conflicts may arise when deploying applications to different application servers that reside on the same node. The reason is that application deployment changes node level serverindex.xml to track applications deployed to each node. The special case handling merges the changes made by the different sessions to ensure that there is no conflict when the user saves the changes.

2.2.1.2 Best practices for configuration changes

2.2.1.2.1 Determine the number of cells you need

Before you decide that you need a large cell, there are several considerations for creating more than one cell. In many cases working with several smaller sized cells are preferable. You do not have to tune a large cell. You do not have to partition your cell into core groups or create the bridge topology.

The factors that affect how many cells you should create include:

- Isolation:
    - Production and testing environments should be separate cells.

  - The back-up site should be its own cell distinct from the cell in the primary site.
  - Critical applications might warrant their own cells compared to sharing a cell among less critical applications.
- Lines of business: Each line of business may fund its own hardware or software. Creating a different cell for each line of business makes it much easier to implement changes as dictated by each funding area. These changes may include when to apply a temporary fix, when to apply a new service pack, and when to upgrade to a new major version of the application server.
- Geography: Do not create a cell that spans beyond a data center, due to latency for administration, restrictions on core group protocols, and risk of network failure partitioning your cell.
- Cell size: If after applying all the best practices you are still not meeting the manageability objectives, consider a new cell.

2.2.1.2.2 Set and track your performance goals

As the size of the topology increases, the time required to perform each additional configuration operation also increases. Even something as simple as creating a new cluster member may take less than 20 seconds when the topology is small and up to a couple of minutes when the topology is large.

As you increase the size of your topology, you should track the times required to perform your most common configuration operations. If after applying the other performance best practices you are still not meeting your performance goals, it is time to consider creating a new cell. Tracking the performance of your various operations also helps you identify the bottlenecks that are candidates to apply performance improvements.

2.2.1.2.3 Use scripting

When you use scripts to perform configuration changes:

- You get repeatable configuration functions that can be automated and applied to different WebSphere environments. It is less error prone compared to using the administrative console.
- You can also optimize it over time to get better performance.
- With properly constructed scripts, you can work around save conflicts. See section 2.2.1.2.9.3 for details.

2.2.1.2.4 Targeted queries

Often in a script you need to search for a specific configuration object, such as a specific node, server, or data source. The configuration service extracts what you are searching from the master repository to the workspace for you to make your changes. How you construct your query can greatly affect how many files are extracted. If you do not use a targeted query, you can potentially cause the entire repository to be extracted. For a large topology this is a very expensive operation. See the following examples:

This can cause the entire repository to be extracted:

```
wsadmin> AdminConfig.list(“JAASAuthData”)
```

This only extracts security.xml:

```
wsadmin> sec=AdminConfig.getid("/Cell:/Security:/")
wsadmin> AdminConfig.list("JAASAuthData", sec)
```

This extracts all servers from all nodes:

```
wsadmin>  AdminConfig.list(“Server”)
```

This only extracts servers on one node:

```
wsadmin> node=AdminConfig.getid("/Cell:/Node:<nodeName>")
wsadmin> AdminConfig.list("Server", node);
```

And this extracts a single server on one node:

```
wsadmin>
server=AdminConfig.getid("/Cell:/Node:<nodeName>/Server:<se
rverName>")
```

2.2.1.2.5 Caching query results

Since query operations can be expensive, cache your query results for subsequent use rather than querying again. Also consider helper functions that will query the topology and cache the results for later use if called again. For example, if you need a list of all application servers while inside a loop, then this scenario is expensive because it performs queries repeatedly:

```
for x in y:
    servers = AdminConfig.list("Server")
```

```
 #endFor
```

This scenario is much less expensive as it performs the query only once:

```
servers = AdminConfig.list("Server")
for x in y :
    …
#endFor
```

This scenario allows a list of servers to be used by all callers and is potentially even less expensive, with the drawback that you cannot use the code if you have created new servers:

```
cachedServers = ""
def getAllServers():
    global cachedServers
    global Adminconfig
    if  ( cachedServers == "" ):
        cachedServers = AdminConfig.list("Server")
    #endIf
    return cachedServers
#endDef
…
for x in y
    servers = getServers()
    …
#endFor
```

2.2.1.2.6 Save changes when configuration operations complete

Once you have built a library of scripts to perform configuration operations, you can aggregate them together to perform different configuration tasks. Sometimes the scripts are written in such a way that a configuration is saved at the end of each function, rather than once at the end of the process. If you are using the administrative console, you might be tempted to save after each change. Saving frequently lowers performance.

When configuration changes are saved, consistency checks are performed between the temporary workspace and the master repository to avoid save conflicts. This can be an expensive operation. In addition, the master repository is locked while performing consistency checks.

The advantages of waiting until the end of your configuration operations to perform a single save are:

- You get better performance by not spending time in multiple consistency checks. Some scripts have been shown to reduce run time by up to 75%.
- You give other concurrent users better performance as well.

2.2.1.2.7 Limit AdminConfig.validate

The AdminConfig.validate function allows you to perform consistency checks of your configurations. It is used to detect values that are valid in the XML configuration model, but not recognized by the runtime environment. If you do use this function, use it sparingly, as it may touch the entire configuration tree, which is expensive for a large topology environment. One possibility is to perform validation just before you save the configuration.

2.2.1.2.8 Avoid chaining wsadmin

Starting the wsadmin process may take 20 seconds or more, depending on hardware. Avoid breaking up your configuration operations into multiple wsadmin invocations. Do combine them into a single script that can be run within one wsadmin session. Consider structuring your scripts into multiple files, and import them from a front-end script.

2.2.1.2.9 Dealing with concurrent sessions

The memory requirement of the deployment manager increases as the size of the topology increases, and as the number of concurrent sessions increases. Since the deployment manager is just a single process, there is no mechanism to balance the load. Therefore, there is a limit to the number of concurrent users that can be supported on a single deployment manager.

2.2.1.2.9.1 Tuning the deployment manager heap size

Just as you would tune the application server heap size, you need to tune the deployment manager heap size to accommodate the number of concurrent users who access the deployment manager. Enable verbose garbage collection, and observe how the heap size increases with the increase in topology and in the number of users.

When you size the heap, remember:

- Java heap should be sized for 40%-70% average memory utilization.

- Garbage collection should occur no more frequently than 10 seconds apart. If garbage collection is occurring more than once every 10 seconds, or the heap utilization is running more than 70% of the heap, consider increasing the heap size.
- A garbage collection should last no more than one to two seconds. If you find that garbage collection is lasting more than one to two seconds, then that is a sign that the heap is too large. Or if you are using a generational garbage collection, it is an indication that the "nursery" is too large. Likewise, memory use of less than 40% is a sign that the heap is too large.
- Total time spent in garbage collection should be no more than 15% of the duration of the test or period of observation. Spending more than 15% in garbage collection during a test is usually the result of a combination of a couple of the other above noted conditions.

Adjust the maximum heap size as needed, and ensure that all the processes in the machine, including the deployment manager, can fit into physical memory to avoid thrashing due to excessive paging.

For a large environment the deployment manager runs the risk of running out of heap, or out of physical memory if there are too many concurrent sessions. In an environment with hundreds of applications and cluster members, it is possible for each session to take up 200 MB of heap. With seven concurrent user sessions, you need 1.4 GB of heap just for the deployment manager itself, not to mention the memory requirements of other processes on the same machine.

2.2.1.2.9.2 Limiting number of concurrent sessions

If too many concurrent sessions are overloading the deployment manager, you need to place a limit on concurrent access. For scripting, consider using the V7 job manager as a mechanism for users to submit wsadmin jobs. The jobs are run sequentially, and an email notification is sent to the user upon job completion. You can also construct a different front end to limit the number of concurrent wsadmin sessions.

For administrative console, there is no built-in WebSphere mechanism to limit concurrent access. You may limit the number of administrators given access to the environment, or you may ask the administrators to coordinate their usage to the environment.

2.2.1.2.9.3 Logout of administrative console

Make a habit of logging out of the administrative console when you are done. Every time you log out, the workspace in memory is cleared, making room for other users.

2.2.1.2.9.4 Reduce administrative console session timeout

If a user forgets to log out of the administrative console, the workspace session is cleared after the session times out. By default this time is 30 minutes. You can change the session timeout as follows:

- Stop the deployment manager.
- Edit the file, <PROFILE>/config/cells/<cellName>/applications/isclite.ear/deployments/isclite/deployment.xml, and change the following attribute to some other value:
    - invalidationTimeout="30"
- Start the deployment manager.

2.2.1.2.9.5 Recovering from save conflicts

As the number of concurrent sessions increase, the chances of save conflicts also increase. For example, you are using scripting, and you encounter a save conflict with the following error message:

```
WASX7015E: Exception running command: "AdminConfig.save()"; exception
information:
com.ibm.websphere.management.exception.ConfigServiceException
java.security.PrivilegedActionException:
java.security.PrivilegedActionException :
com.ibm.ws.sm.workspace.WorkSpaceException: RepositoryException
```

You can first discard your changes in the workspace session:

```
wsadmin>AdminConfig.reset()
```

You can then run your configuration operations again from the point of your last save. If each concurrent session modifies a few common files, then your operations are likely to succeed when you run them again. You can also consider constructing your script to retry a few times, in case the first attempt does not succeed.

2.2.2 Operational management

2.2.2.1 Narrow your JMX queries

WebSphere Application Server operations are based on the JMX framework. An application server cell is organized in a tree-like structure. A WebSphere Application Server Network Deployment cell has the deployment manager at the root and every node is a branch from the deployment manager. From each node agent, every application server that is hosted on the node is a branch from the node agent. As a result, a JMX request from the deployment manager to a single application server flows through the deployment manager to the node agent on the same node where the server resides, and finally to the application server itself. This design is intended for scalability. The deployment manager has to communicate with a node agent only, and each node agent has to communicate with its respective application servers only. If an invocation is made to all of the servers on a node, the deployment manager uses one invocation to the node agent and the node agent, in turn, broadcasts the invocation to every server on the node.

Consider the following JMX queries that are issued on the deployment manager. The following queries are shown to the reader using the Jython syntax of the wsadmin scripting utility.

Query 1**:**

```
AdminControl.queryNames("type=Application,* ")
```

In Query 1, the query searches for every MBean of the Application type in the cell. For this query to be fulfilled, every running server must be checked and returned.

Query 2:

```
AdminControl.queryNames("type=Application,node=SampleNode1,
*")
```

In Query 2, the query searches for every MBean of the Application type on a specific node within the cell. In this query, the specific node is the SampleNode1 node. Another node agent is not sent the query.

Query 3

```
AdminControl.queryNames("type=Application,node=SampleNode1,
process=server1,*")
```

In Query 3, the query searches for every MBean of the Application type on a specific node and server within the cell. In this query, the specific node is the SampleNode1 node and the specific server is the server1 server. The node agent on the SampleNode1 node forwards the query directly to server. Another server is not contacted on the SampleNode1 node. Also, neither an additional node agent nor a server on another node is contacted.

Query 4

```
AdminControl.queryNames("node=SampleNode1,process=server1,*
")
```

In Query 4, the query returns all of the MBeans on the SampleNode1 node and the server1 server, regardless of type. This query shows that you can use a wildcard for any attribute in the query.

Query 5

```
AdminControl.queryNames("*”)
```

To fulfill the Query 5, the deployment manager must query every server in the cell and return every MBean from them. This process is an expensive query that can take time to complete.

To fulfill a query, every server that matches the query must be checked. The JMX code is designed to walk the tree in a 'depth first' search pattern. Using this pattern, if a node matches the query string, all of its servers must be checked before the node can return control back to the deployment manager. The deployment manager waits for all processing to complete on one node before proceeding to query the next node in the tree. The problem occurs if a node or application server on a node accepts a query, but does not return an answer in a reasonable amount of time. A common cause of this problem is a server that is out of memory. However, the out-of-memory condition may not have stopped the server or caused the server to be stopped and restarted by the monitoring logic.

To avoid a scenario where queries get stuck, use narrow queries that target only the servers or nodes from which you really need information. Queries that touch every server can considerably consume cell resources.

This resource consumption can cause failures as cell sizes continue to grow. With more servers in a cell, there are more points and places a wide query can get stuck, which blocks not only that query but possibly any queries behind it.

2.2.2.2 Starting application servers concurrently

The following pseudo code illustrates the basic pattern for starting application servers concurrently:

```
# Start all servers without waiting for completion.
AdminControl.startServer("<server>", "<node>", 1);

# Query for Server MBeans from all servers:
mbeans = AdminControl.queryNames("*:*,type=Server,node=<node>");

# Check all Server MBean state = "STARTED"
for mb in mbeans: if AdminControl.getAttribute(mb, "state") ==
"STARTED": …

# Finally, ensure that the number of servers in STARTED state is what
you expect.
```

Here is an actual sample code for starting all application servers on a node:

```
# Set the number of servers to start.
set maxServer 50
set nodeName arcturus1

# In this loop, start all servers.
set servNum 2
while {$servNum <= $maxServer} {
    set serverName serverX[format "%3.3d" $servNum]
    $AdminControl startServer $serverName $nodeName 1
    incr servNum
}
# Start server1 also.
$AdminControl startServer server1 $nodeName 1
puts "Start request issued to all servers"

# WAIT for start to complete
# First wait for maxServer Application Server MBeans to be created
```

```
set event_flag 0
while {1} {
    after 10000 {incr event_flag} ;# wait 10 seconds to check MBean
creation
    vwait event_flag
    set serverMBeans [$AdminControl queryNames
type=Server,processType=ManagedProcess,node=$nodeName,*]
    if {[llength $serverMBeans] == $maxServer} {
        break
    }
}
puts "MBeans created for all servers"

# Then wait for each server to have a state of STARTED
puts "Waiting for each server's state to be STARTED"
set waitForStart 1
while {$waitForStart == 1} {
    set waitForStart 0
    for each server $serverMBeans {
        if {[string compare [$AdminControl invoke $server getState]
"STARTED"] != 0} {
            after 10000 {incr event_flag} ;# wait 10 seconds to check
MBean state
            vwait event_flag
            set waitForStart 1
        }
    }
}
```

2.2.3 Node synchronization

Node synchronization is the process by which the WebSphere configuration is transferred from the deployment manager to the node agent. The deployment manager and node agents compare MD5 hashes of the configuration files to determine whether the files are identical. In the cases of a node agent or deployment manager restart, the respective server must create all the MD5 hashes in memory for all the configuration documents in the node or cell. As the cell size and number of documents become larger, the start-up time also increases.

WebSphere Application Server has added support for "Hot Restart Sync." With this support, the node agent and deployment managers save the hashes in both memory as well as on the file system. When a restart is performed, the MD5 hashes do not need to be recomputed but rather can be loaded directly from disk.

To enable this support, add the following custom property to your deployment manager and node agent:

```
-DhotRestartSync=true
```

It can be added by going to the **Process Definition** > **Java Virtual Machine** panels on the administrative console.

Important: When you restart the application server, the deployment manager and node agent do not recompute hashes. Therefore, it is critical that no manual changes be made to the cell**.** If there is any doubt, issue a “Full Resynchronize” to recalculate the MD5 digests.

2.2.4 Application management

To understand how to improve application deployment performance, you must first understand the sequence of processing steps in the deployment process. First, consider the case of a wsadmin client running on a machine different from the deployment manager. When you run the wsadmin client, the following process occurs:

1. A temporary copy of the .ear file is created.
2. The .ear file is extracted.
3. Client bindings are collected.
4. The .ear file is saved.
5. The .ear file is uploaded to deployment manager’s machine.
6. The .ear file is saved to the workspace.
7. The .ear file is saved to the master repository.
8. During node sync, the .ear file is transferred to each node.
9. The .ear file is expanded.
10. The application is started for a new installation or restarted for updates.

Now consider the case of using the administrative console, with the application residing on a machine different from the machine of the deployment manager. The only difference in this deployment process is that the .ear file is first uploaded to the deployment manager machine, and then the .ear file is extracted.

In both cases, there are a number of copy operations that can be eliminated. When using wsadmin, you can eliminate step 5, the upload to the deployment manager, by ensuring that the application is accessible on the deployment manager machine, and starting wsadmin on the same machine as the deployment manager. For the administrative console, you can choose to browse for the application that is local to the machine of the deployment manager.

You can eliminate the synchronization of the application binary files on the nodes by using the –nodistributeapp option for wsadmin, eliminating step 8, or in the administrative console, by clearing the “Distribute application” check box. By enabling this option, you must take responsibility to ensure that the .ear file is available and extracted on each node. You can use the EARExpander utility to expand the .ear file on each node. In addition, you must complete the following steps to synchronize each node to ensure that the EARExpander is run on the node only when the application is stopped.

1. Stop the application on the node.
2. Synchronize the node.

3. Ensure a copy of the .ear file is available on the node, using either a shared file system, or an efficient remote copy mechanism.
4. Use EARExpander to expand the .ear file on the node.
5. Start the application on the node.

You can use a shared file system with the previous application deployment steps so that you only need to run EARExpander once, avoiding copying and expanding the .ear file to all the nodes. You can also specify the location of the expanded application to the shared file system. However, note the following tradeoffs:

- Your shared file system must be highly available, or you can lose application availability.
- Your shared file system must have sufficient bandwidth to accommodate all the nodes accessing the application.
- Due to all the nodes sharing a common set of expanded .ear files on the shared file system, you cannot preserve application availability during application update by controlling node synchronization one node at a time.

Complete the following steps to deploy applications and eliminate synchronizing application binary files on nodes. These steps assume that "/shared/testapp.ear" is the directory on the shared file system where the .ear file is expanded.

1. Stop the application if it has already been deployed.
2. Use EARExpander to expand the .ear file to the shared file system; for example:

   -
     ```
     EARExpander –ear testapp.ear  –operationDir
     /shared/testapp.ear -expansionFlags war -operation
     expand
     ```
3. Deploy the application:
   - For administrative console:
     - Specify "Directory to install application" as /shared.
     - Clear "distribute application."
   - For wsadmin new deployment:

     -
       ```
       AdminApp.install('<location of .ear>', '[ -
       installed.ear.destination /shared -
       nodistributeApp –appname testapp …]')
       ```
   - For wsadmin update:

     -
       ```
       AdminApp.update('<location of .ear', '[ -
       installed.ear.destination /shared -
       nodistributeApp –appname testapp …]')
       ```

    - Save and synchronize to each node.

4. Start the application.

You can also use the shared file system to bypass application updates altogether if you know with certainty that only the application logic has changed. Do not do this if you have any doubts about whether any deployment options, bindings, extensions, or annotations for Java EE 5 or later applications have changed. Complete these steps to bypass application updates:

1. Stop the application.
2. Expand the new version of the .ear file to the shared file system.
3. Start the application.

For wsadmin, you can further use the –zeroEarCopy deployment option to eliminate two additional copy operations in steps 6 and 7. Use the following example script:

```
AdminApp.install('/shared/testapp.ear', '[ -
installed.ear.destination /shared -zeroEarCopy -nodistributeApp -
appname testapp …]')
```

Attention:

- This option is not available through the administrative console.
- The location of the .ear file may be the same as the expanded .ear directory on the shared file system. This is required if you plan to edit the application, or apply partial updates, since the .ear file is not available in the master configuration repository.

Finally, another way to improve deployment performance is to reduce the size of the application. If the application is including JAR files that are already part of the WebSphere runtime environment, such as j2ee.jar, jsse.jar, and jaas.jar, then they must be removed from the application. If the application is using large utility JAR files, consider creating shared libraries for these JAR files. Avoid sharing these JAR files between applications, per discussion in the following paper:

WebSphere Contrarian: Options for Accelerating Application Deployment

2.3 Service integration bus

The service integration bus (SIBus) is the built-in messaging component of WebSphere Application Server to support Java Messaging Service (JMS). It is built on top of HAManager, and is designed to tolerate failures. The architecture of SIBus is shown in Figure 6, where a bus represents a logical channel through which message producers can send messages, and message consumers can receive messages. Producers and consumers are connected by destinations, which may be either topics or queues. Producers are configured to send messages to specific destinations, while consumers are configured to receive messages from the same destinations. A bus encompasses a set of bus members, each of which is either a WebSphere product application server, or a cluster. Within the bus members are message engines (MEs) responsible for sending and receiving messages. A producer may send a message through any ME in the bus, while a consumer may receive a message through any ME in the bus. The MEs are responsible for routing the messages through the bus from the producer to the consumer.

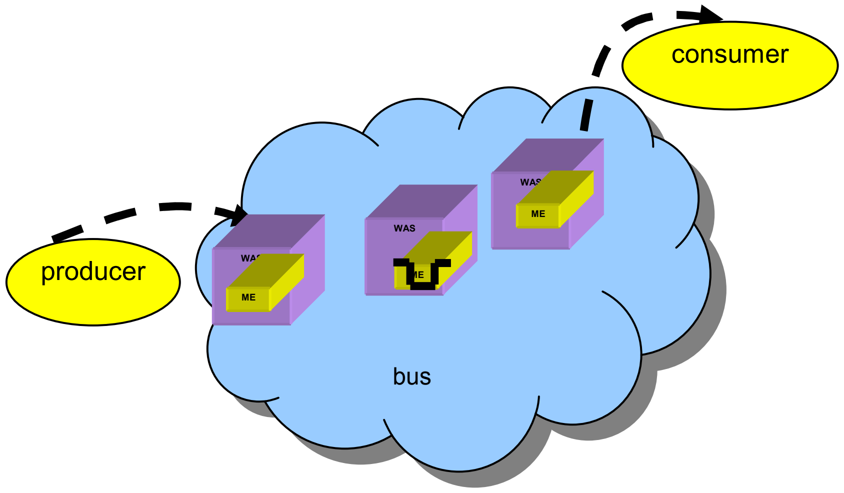

Figure 6: SIBus overview

Some additional considerations when configuring buses:

- A bus may only reside in one cell. However, buses in different cells may be linked through external bus definitions.
- Bus members that cross core groups must be bridged through core group bridges.
- To guarantee message delivery order, only one ME may be active within the members of a cluster.

2.3.1 Factors that affect SIBus performance

There are several factors that affect SIBus performance. To begin, the more destinations hosted on a message engine, the longer it takes for the message engine to start. Figure 7 shows how ME startup time is affected by the number of destinations.

© Copyright IBM Corporation 2007-2011 All rights reserved.

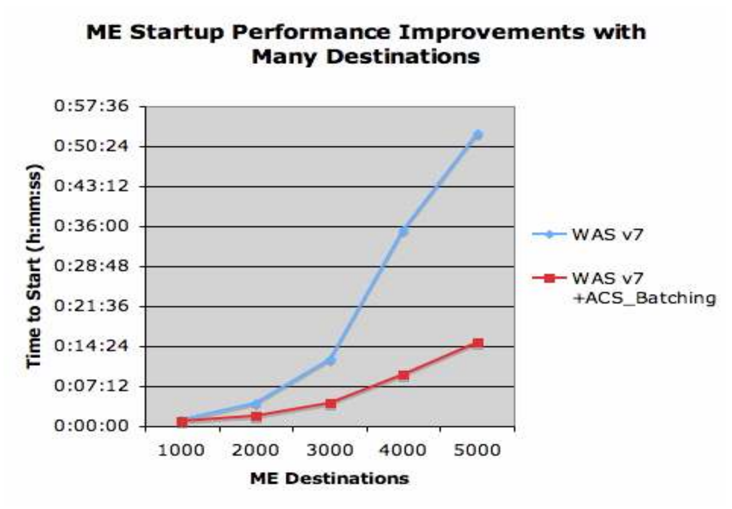

Figure 7: Startup time versus number of destinations

The ACS_BATCHING algorithm may be used in WebSphere Application Server V7 to improve ME startup time when there are many destinations on the bus. This property is documented in section 2.3.2.7.

If the same number of destinations is apportioned over more than one bus members, the startup time improves considerably. This is shown in Figure 8, comparing the startup time between one and two bus members. Not only do you get better ME startup time, you also get better throughput in message delivery, as each ME handles a fewer number of destinations. However, the drawback is that there are more network connections between the bus members, more overall resource usage, and that the configuration becomes more complex.

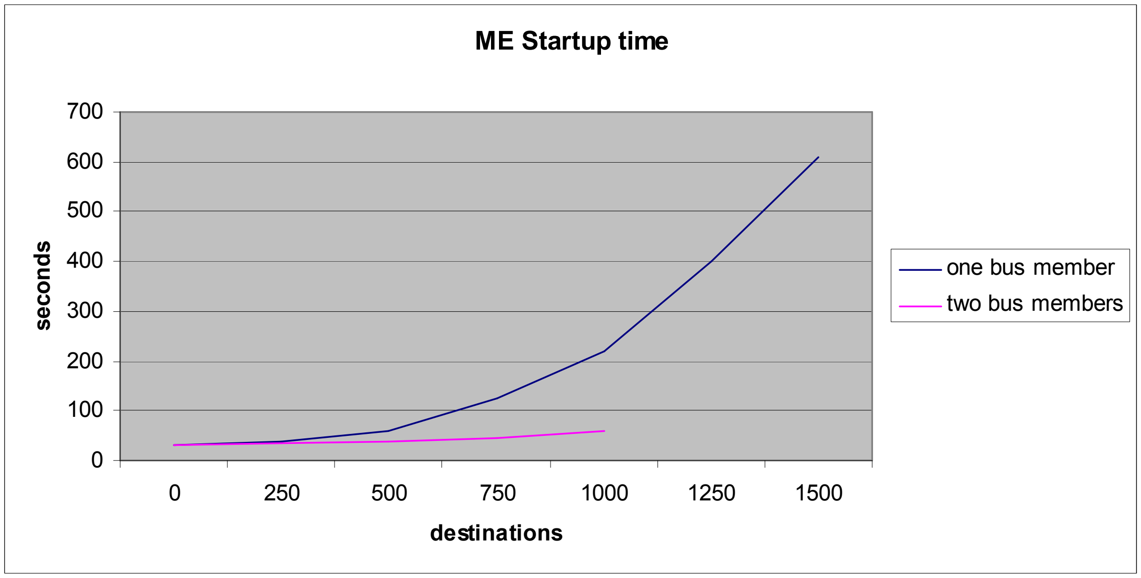

Figure 8: Startup time versus number of buses

The number of buses can also affect the performance of your SIBus infrastructure. Consider a bus with 50 MEs. Because the MEs all open network connections with each other, about 50*49= 2450 network connections are created. If there are two buses each with 25 MEs, the number of network connections becomes 25*24*2 = 1200 connections.

The rate of message production and consumption also affects bus performance. In general, the more producers there are, the more messages are generated that can overload a bus. Similarly, the faster the producers generate messages, or the slower the consumers process messages, the more messages are built up to overload the bus.

2.3.2 Best practices for SIBus

2.3.2.1 Consider creating more than one bus

If you have many disjoint destinations in your bus being used by different applications, consider creating different buses, which yields the following benefits:

- Isolation of traffic
- Isolation of management
- Higher performance

2.3.2.2 Minimize number of destinations

Minimizing the number of destinations can improve ME startup time, and messaging performance. Related applications do not all have to communicate by way of separate topics or queues. As an alternative, correlations ID or selectors may be used.

2.3.2.3 Find the right number of MEs per bus

There is no need to create an ME in every WebSphere product application server or cluster member. But you do need to tune the number of MEs so that you get acceptable start-up time and messaging throughput. As the message throughput requirements differ by application, you need to take measurements taking specific application requirements into account.

As you increase the number of destinations in your environment, you may find performance starts to degrade. You need to create additional MEs and reassign some of the existing destinations to the new ME.

2.3.2.4 Collocation of the producer, consumer, and ME

Locality matters for messaging performance. You can try to collocate the message producer, consumer, and ME in the application server or cluster member, giving you the lowest latency. However, collocation only works if the resource requirements of all components, such as CPU and memory, do not exceed the available resources. Otherwise, you need to assign each to a different application server or cluster.

2.3.2.5 Minimize number of outstanding messages

You must tune the environment so that messages are consumed at a rate slightly higher than the rate that they are produced. If the producer produces messages at a faster rate, the messages will overload the bus. An overloaded bus adversely affects ME startup performance, and decreases throughput by forcing the bus to queue the messages indefinitely. Consumers must consume messages at a slightly faster rate than the producer to handle those messages that pile up during consumer failover.

As a best practice:

- Do not increase the default maximum queue depth of 50,000 messages. Increasing the queue depth only makes the congestion worse. It is better to examine where the bottlenecks are, and apply the appropriate best practices to eliminate the bottlenecks.
- Take into account the failover time of the consumer, and ensure that messages that pile up during failover can be consumed at a reasonable rate after failover.
- Assign producers, consumers, and MEs to clusters for failover.
- Tune the number of buses and bus members.

2.3.2.6 Dealing with large numbers of clients

If there are many clients, whether producers or consumers, to the SIBus, the load placed on the bus members by the clients may adversely affect messaging throughput. To reduce the load on the existing bus members, a new cluster level bus member consisting of multiple MEs may be created. The cluster is a “concentrator tier” where the new MEs do not host destinations. They are used solely for load balancing among the different clients to offload the load placed on the existing MEs that host actual destinations.

2.3.2.7 Targetting Messaging Engines

When JMS applications connect to a bus, the minimum information required is the name of the bus. If the application is not running in a server, for example, if it is running in the client container, at least one provider endpoint is also needed to bootstrap to either a bus member, or a server with SIB service enabled.

The provider endpoint that you select does not automatically influence the type of ME used for the connection. It is used as a bootstrap. With just the name of the bus, a JMS connection is made to any ME in the bus, based on Workload Management (WLM) logic.

Hence, apart from just specifying the bus name to connect, it is also important to mention the Target, Target Type, and Target Significance along with the Provider end points to target a specific ME for higher performance

By targeting the connection, you can define varying levels of control over the actual connection to the bus.

Target type: the type of entity to be targeted. The following target types are valid:

Bus member: Any available ME in that bus member.

Messaging engine: A specific ME.

Custom messaging engine group: A manually configured set of MEs.

Target significance

Required: connect only to the specified target.

Preferred: Use the specified target if available, otherwise use any other available ME in the bus.

Target: The name of the target.

Connection proximity

This option provides an additional level of control, based on the physical location relative to the application or bootstrap server; for example, within the same server or cluster, or on the same host.

2.3.2.8 Store and forward

In a bus with multiple bus members, it is possible for an application to connect arbitrarily to any of the MEs in the bus. This is possible in case the Target, Target Type, and Target Significance attributes are not configured for the connection. When messages are sent, they are initially stored on the ME that the sending application is connected to. Since an application connects to only a single ME in the bus, the store and forward method is used to deliver the message:

- Messages are initially stored on a **Remote Queue Point.**
- The messages are then ***forwarded*** to an ME where a suitable queue point exists.
- Once received, the original copy of the message is deleted.

Store and forward can be beneficial because it allows message production to continue while a queue point’s ME is unavailable. However, this results in additional potential places where messages may become blocked for various reasons, complicating general problem determination due to message being stored on multiple MEs, and also leading to decreased performance.

2.3.2.9 Remote GET

When consuming from a queue point that is not located on the same ME that the application is connected to, a process similar to store and forward is employed, generally referred to as *remote get*:

- The connected ME (ME2) requests a message from the ME with the queue point (ME1).
- ME1 selects a message, persistently locks it for ME2, and sends a copy of the message to ME2.
- The consuming application receives the message, processes it, and commits the reception.
- ME2 persistently commits the message and asynchronously informs ME1 of the commit.
- Asynchronously, ME1 deletes the message, and ME2 removes the commit state.

Unlike store and forward, remote get requires the queue point's ME to be available for messages to be consumed. But the remote get can cause additional network traffic due to acknowledgment being passed between MEs, decreasing performance.

2.3.2.10 Other tuning parameters

If you have many destinations, and you are running 6.1.0.23 or 7.0.0.3 or later, consider setting the following cell level custom property to improve ME startup time:

- IBM_CLUSTER_ENABLE_ACS_DELAY_POSTING=true.

From the administrative console, go to **System Administration** > **Cells** > **Custom Properties**.

2.4 Security components

2.4.1 Performance fixes

Significant security performance and scalability improvements have been incorporated in WebSphere Application Server V6.0.2 Fix Pack17, and V6.1.0 Fix Pack 9. WebSphere Application Server Version V6.1.0 Fix Pack 9 was released in June 2007. APARs PK32086 (6.02 Fix Pack 17 and 6.1.0 Fix Pack 5) and PK43270 (6.1.0 Fix Pack 9) contain security performance and scalability improvements. You can upgrade to these service levels as soon as it is practical for you to obtain the benefits of these fixes for security. In addition, WebSphere Application Server for z/OS V6.1.0 Fix Pack10 includes additional performance fix for RACF Key Rings when using the JCECCARACFKS keystore.

2.4.2 Java 2 security

Java 2 security has a significant performance cost, and therefore, do not use Java 2 security unless your application really requires it. If you must use Java 2 security, there are tuning parameters introduced in V6.1.0 Fix Pack 9 with APAR PK43270, which might improve performance significantly. These parameters introduce a new concept called *Read-only Subject,* which introduces a new cache for J2C authentication subjects. If the J2C authentication subject is not modified after it is created, you can use these tuning parameters to improve Java 2 Security performance. You can add the following parameters through the custom properties panel in the administrative console for WebSphere Application Server:

- `com.ibm.websphere.security.auth.j2c.cacheReadOnlyAuthDataSubjects=true`
- `com.ibm.websphere.security.auth.j2c.readOnlyAuthDataSubjectCacheSize=50`

This property specifies the maximum number of subjects in the hash table of the cache. After the cache reaches this size, some of the entries are purged. For better performance, this size should be equal to or greater than the number of unique subjects or users.

To access the Custom properties panel in the administrative console, click **Security > Secure administration, applications, and infrastructure.** Then, click **Custom properties**.

2.4.3 Authentication

The biggest factor that affects a large topology from a security standpoint is how frequently the user registry is accessed during a login. You can reduce or eliminate registry accesses after the initial login by using security attribute propagation, authentication cache, hash table attribute assertion, and so on. Many of these issues are discussed in the IBM WebSphere Developer Technical Journal: Advanced authentication in WebSphere Application Server article, which provides a background on the authentication capabilities.

Another major issue is related to thread contention. Thread contention is not purely a security issue, but the issue is magnified when security is enabled. When too many threads run in a single JVM, create additional JVM configurations to more evenly spread the workload. This approach improves throughput tremendously from our performance analysis.

The following recommendations can likely improve throughput or reduce problems in a large topology. However, you must test these recommendations in your environment because different factors can cause differences in behaviors.

- When you use a proxy server to perform the primary authentication of users, make sure the associated Trust Association Interceptor (TAI) asserts the user to WebSphere Application Server using a hash table in the TAI Subject to prevent a double login for the same user request.

- Use downstream propagation to reduce registry overload. Propagation sends all of the Subject attributes to downstream servers for reuse without needing to access the registry at each server hop.
- Add front-end application servers to a common DRS replication domain to ensure that subjects get propagated to other front-end servers for reuse, which prevents additional user registry accesses. When you run stress tests with single sign-on (SSO) enabled, test throughput while both enabling and disabling the web inbound security attribute propagation option, which is horizontal propagation. To access this option in the administrative console, click **Security > Secure administration, applications, and infrastructure**. On this panel, expand web security and click **single sign-on (SSO)**. In some environments, horizontal propagation improves throughput by reducing registry access. However, in other cases, it decreases throughput due to the encryption and decryption costs.
- Increase the cache timeout if users tend to re-authenticate frequently. Make the timeout value larger than the 10 minute default, but not so large that the user population causes heap problems. For large login rates, reduce the cache timeout value to prevent significantly large cache growth, heap issues.
- Be careful not to assign users to too many registry groups. This process increases the size of each authenticated user Subject and causes more strain on the registry. Use them, but do not overuse them, when possible.
- Ensure that multiple Lightweight Directory Access Protocol (LDAP) replicas are in use either through an IP sprayer, through multiple LDAP end points for failover, or using the logical realm property to define multiple LDAP servers to handle requests across the topology. Optionally, create a custom registry that authenticates to multiple LDAP servers or remote databases that can spread the traffic around.
- Use the internal server ID feature to prevent administrative system outages if LDAP fails. This feature is available starting with WebSphere Application Server V6.1.
- Distribute the workload to multiple JVM configurations instead of a single JVM on a single machine. This process can improve security performance because there is less contention for authorization decisions.
- Improve the performance of security attribute propagation. There are tuning parameters that are introduced in WebSphere Application Server V6.1.0 Fix Pack 9 through APAR PK43270. By using these tuning parameters, which you can be set through custom properties from the security panel on administrative console, the extra overhead of security attribute propagation is almost eliminated. You can use the following custom properties and values:
    - `com.ibm.CSI.propagateFirstCallerOnly=true`
      This property logs the first caller in the PropagationToken that stays on the thread when Security Attribute Propagation is enabled. Without setting this property, all caller switches get logged, which affects performance. Typically, only the first caller is the most interesting.
    - `com.ibm.CSI.disablePropagationCallerList=true`
      This property completely disables adding a caller or host list in the

PropagationToken. This setting is beneficial when the caller or host list in the PropagationToken is not needed in the environment.

- **Attention**: Access to data in the caller and host list can be retrieved by way of the com.ibm.websphere.security.WSSecurityHelper API.

- Use a clock synchronization service to keep system clock values as close as possible. Security processing depends on time stamp validation and having clocks out of synchronization more than five minutes can affect performance due to unnecessary re-authentication and retry processing.

If WebSphere Application Server security is used only to protect administrative access, disable application security so that the collaborators do not perform actions that might affect throughput.

2.4.4 Authorization

Best practices for authorization security scaling include:

- Take advantage of the GROUP mappings to reduce the size of authorization tables. Associate GROUPS to ROLES rather than USERS to ROLES, whenever possible.
- Use a vendor-acquired authorization provider to simplify resource controls where large amounts of applications are managed.

2.4.5 Secure sockets layer (SSL)

Secure Sockets Layer (SSL) has major improvements in WebSphere Application Server V6.1. For more information, see IBM WebSphere Developer Technical Journal: SSL, certificate, key management enhancements for even stronger security in WebSphere Application Server V6.1.

Use certificates that are signed by a certificate authority (CA), preferably an internal CA for internal communications, whenever possible. This usage reduces the number of signers that are needed in a truststore and allows the replacement of a personal certificate without ramifications to clients.

You can use SSL offload devices to reduce the SSL overhead for internet and intranet facing applications. Using keepAlive, which is on by default, dramatically minimizes the SSL overhead, removing the SSL handshakes, which tends to be the largest overhead of SSL. While SSL has some overhead with keepAlive enabled, do not disable SSL in favor of performance due to the security implications. SSL adds message confidentiality, integrity, and prevents man-in-the-middle attacks when it is configured properly.

Another best practice is to manage file-based keystores in the deployment manager repository using the built-in administrative console or scripting commands in WebSphere Application Server V6.1. Also manage the plug-in keystore using the built-in administrative console or scripting commands.

3 High availability using WebSphere Application Server for z/OS

WebSphere Application Server for z/OS uses a number of availability and scaling features that are inherited in z/OS operating systems using a number of tightly coupled components. The z/OS operating system uses the self-healing attributes from the hardware, and extends them by adding functions such as recovery services, address space isolation, and storage key protections. Functions such as z/OS Workload Manager (zWLM), Resource Recovery Services (RRS), and Automatic Restart Manager (ARM) assure the availability of applications. The z/OS operating System can be configured into a Parallel Sysplex, which is the clustering Technology for z/OS providing a continuous availability without compromising client performance.

High availability requires at least two severs that provide the same services to their clients, so that failover recovery can occur, if a failure happens. Using z/OS Parallel Sysplex clustering technology added additional value in offering continuous availability. On z/OS, the zWLM advance workload technology provides balanced application workloads across the systems in the Parallel Sysplex. Workloads are not spread, but rather workloads are balanced based on current load, policy objectives, and the availabilities of the Systems or applications.

The z/OS operating system also offers the capability to use Geographically Dispersed Parallel Sysplex (GDPS) to further extend system availability for systems as far as 40 KM apart. GDPS is useful in the event of site-wide failures.

Using the z/OS Parallel Sysplex clustering architecture, you can achieve a near continuous availability of 99.999%, or five minutes of downtime a year. For more information refer to the Redbooks publication, “Architecting High Availability Using WebSphere V6 on z/OS.” (SG24-6850)

WebSphere Application Server for z/OS takes full advantage of the high availability features already built into the z/OS operating system, instead of using similar functionality in WebSphere Application Server HAManager. However, HAManager still brings value to the WebSphere Application Server for z/OS environment if you require any of the following HAManager services:

- Memory-to-Memory replication
- Singleton failover
- WebSphere Workload Management routing
- On-demand configuration routing.
- SIBus services

See the description about these services in section 2.

For WebSphere Application Server for z/OS best practices for high availability and scalability, use the following guidelines if you do not have HAManager enabled:

- Keep the number of nodes per local partition (LPAR) between one or two nodes with a maximum of four nodes per LPAR. Normally, multiple nodes on an LPAR are used to make the service rollout easier, such that one node can be updated while the other node is active.

- Keep the number of servers reasonable, as dictated by the amount of real memory present on the LPAR.
- Spread a cell or cluster over at least two LPARS. Using multiple LPARs ensures hardware redundancy as well, while still allowing the cluster to be upgraded on a per node basis.

In the following large topology, the customer's environment consisted of the following items:

- One deployment manager on a common LPAR (may be restarted on all LPARs on which the node resides).
- Two nodes exist across two LPARS.
- At least 300 servers exist on each node, for a total of 600 servers all clustered
- Approximately 300 applications exist in the cell.
- HAManager is not enabled in this cell.

If HAManager is needed within the cell, follow the HAManager best practices in Section 2.1. If HAManager services are needed, the cell growth is limited, as described earlier in terms of the number of processes within a core group. In addition, do not include the deployment manager in any core groups because it inhibits the deployment manager when it is started on a different system.

4 Conclusions

The WebSphere Application Server Network Deployment product is tuned for small to modest-sized cells in its default configuration. By understanding how the application server components are designed and behave, it is possible to tune the product so that large topologies, which contain hundreds of application servers, can be created and supported. The WebSphere development labs and many WebSphere customers have gone through this process of tuning the configuration to support a large cell. This document provided some of the best practices that have been discovered related to that tuning.

Every enterprise environment should thoroughly test their application for performance under realistic production load to determine whether the topology and configurations provide the best performance characteristics. Similarly, every enterprise environment should also test for administration performance and stability to ensure that the environment can be managed accordingly. While the numbers and configurations that are presented in this document are a good representation, it is not a guarantee that the same behavior is exhibited in your environment. If you decide not to thoroughly test your application for performance or your environment for manageability, you must understand the risks, including the possibility of a large-scale production outage.

5 References

Redbooks:

- WebSphere Application Server Network Deployment V6: High Availability Solutions
- Architecting High Availability Using WebSphere V6 on z/OS

Websites:

- IBM WebSphere Developer Technical Journal: Advanced Authentication in WebSphere Application Server
- IBM WebSphere Developer Technical Journal: SSL, certificate, and key management enhancements for even stronger security in WebSphere Application Server V6.1
- Recommended fixes for WebSphere Application Server
- J2EE Packaging and Common Code
- The WebSphere Contrarian: Options for Accelerating Application Deployment

Topics in the WebSphere Application Server Information Center:

- Core group Failure Detection Protocol
- When to use a high availability manager
- Fast singleton failover and active heartbeating
- Core group coordinator

- When to use a high availability manager
- Disabling or enabling a high availability manager
- Configuring a core group for replication
- Core group protocol versions

6 Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.