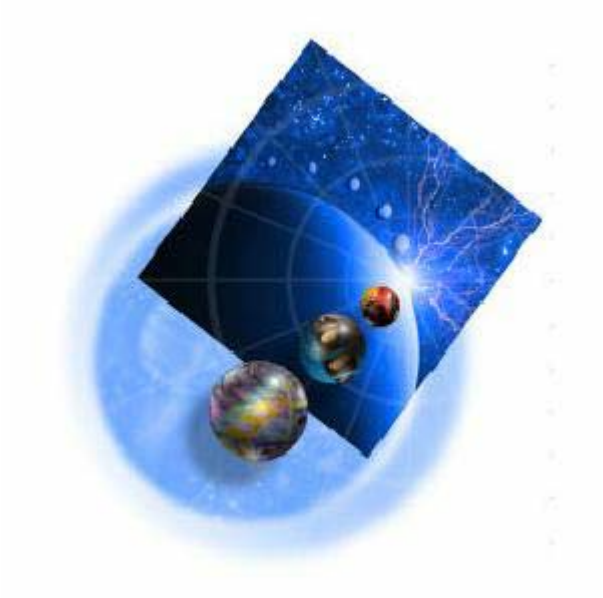
Server Clusters For High Availability in WebSphere Application Server Network Deployment Edition 5.0



By:
Hao Wang
Mark Bransford

Preface

This document lays out the various tradeoffs involved with, and the options available for, building "highly available" production e-business solutions utilizing IBM's WebSphere Network Deployment V5.0 (WAS-ND) product. This document is directed towards WebSphere architects and advanced WebSphere planners. There are three key concepts that one should be comfortable when proceeding to use this document: high availability, failover, and clusters. These concepts , along with a road map on how best to read this document, are outlined in this preface.

I. High availability (HA). An application or service that is provided in a near-continuous fashion. In order to accomplish this requirement, a solution must include the concepts of

    II. Process availability - A process is considered available when there is a server (or cluster member) available to service a request. WebSphere implements process availability as follows:

        III. Clusters. The ability to service any given request is supported by a number of similar processes. (see fuller description below)

        IV. Failover. When a process becomes unavailable, requests are directed away from the unavailable process and toward the other, available, processes in the cluster.

        V. When there are no available processes to service a request, a programming model exists that allows clients to perform recovery, if possible, and to resubmit requests, if necessary.

This path may lead to loss of data and to the system being down for some time, depending on the nature and seriousness of the failure.

VI. Data availability - occurs when data is preserved across process failures and is available for processes that continue to be available. Data availability requires:

VII. Failover to redundant copies of data, when the primary source of data for a process is no longer available

VIII. Procedures for recovery of data, when the catastrophic errors occur in system (e.g. fire)

IX. Clusters. A cluster is a collection of servers working together as a single system to ensure that mission-critical applications and resources remain available to clients. A cluster can be thought of consisting of several logical components:

X. Application servers (processes or "clones") that service requests

XI. Administrative servers (processes) that that administer the application servers

XII. Data repositories (databases or native operating system files) that store information used by the Application and Administrative servers

How should this document be read? Chapter 1 provides an in-depth introduction to this guide, expanding upon the above terms, and outlining the WAS-ND solution for HA, and where it can be enhanced through use of other IBM and non-IBM vendor products. Chapters 2-6 present the architected WAS-ND solution for high availability. Chapters 7-12 speak to other IBM and third party software solutions, as well as how to configure hardware to enhance HA using WebSphere. Therefore, in order to understand how to use WebSphere for HA in the WAS-ND release, read Chapters 1-6. For those interested in expanding upon the solution provided by WAS-ND, read Chapters 7 -12, as needed, depending upon environment and HA requirements

Chapter 1 - Introduction

1.1 High Availability

When we speak of high availability, the builder of highly available e-business solutions needs to look at the problem from the perspective of low unavailability. We refer to the periods of unavailability as outages. We classify outages as being either scheduled or unscheduled. A scheduled outage consists of the time when the solution is not available due to maintenance activities. An unscheduled outage consists of the time when the solution is not available due to some failure. The combination of the number and duration of these outages are what define the total unavailability of the solution.

In addition to the amount of time that a solution is unavailable the developer of a highly available solution must also consider other impacts that an outage has on the solution. Some outages may result in loss of data, some outages may result loss of some business transactions. In turn these outages may result in the lost business opportunities.

There is no easy answer to the question of "how do I build a highly available solution?" There are, instead, fundamental tradeoffs that each business must make to develop a solution that meets both the availability objectives of the business as well as the financial objectives of the business. As a general rule more availability and less impact with each outage increases the cost of the solution. Examples of various business needs include some businesses that need only be available during normal business hours 5 to 6 days a week, while some businesses are *ideally* expected to be available 7 days a week, 24 hours a day. The goal of continuous availability (0% down time), while laudable, is in practice is never achieved. High availability in the context of this document does not imply continuous availability.

For purposes of discussion in this paper we'll adopt a definition of high availability as providing service that satisfies a defined service level. A service level means that a system will provide availability as appropriate for the business requirements. This includes provisions for both planned and unplanned outages, but does not strive to exceed the business needs. By way of example, a business requires a service level with 0.05% unscheduled downtime and has allowed for 4 hours of scheduled downtime per week. As a result their system is expected to be available Monday through Saturday, 24 hours a day and on Sunday until 8:00 p.m. At 8:00 p.m., 4 hours are allotted for system maintenance. The requirement of 99.95% availability meant that the system is required to be servicing requests for 163.92 hours out of the 164 hours specified. Therefore, all unplanned outages can total no more than 5 minutes a week. This example probably represents a reasonable and cost-effective goal for most enterprises. Some enterprises will have a higher availability requirement which will incur increasingly higher costs for the hardware and system infrastructure, while other enterprises can operate effectively with a lower high availability requirement. The cost of a high availability infrastructure is typically the deciding factor in what's appropriate for a buiness, since the incremental cost of providing a highly available architecture increases rapidly once an organization reaches availability of 99.9% or more.

When looking at measurements of system availability, it is important to consider the context of that measurement. In order to achieve a highly-available system, and accurately measure that availability, one must consider the entire system and all of the components, with business constraints and goals

folded into the decision of what service level is acceptable. Issues to consider when designing for high availability are:

- Disk failure
- CPU or machine failure
- Process crash/failure
- Network failure
- Catastrophic failure, e.g., the loss of an entire data center
- Human error
- Planned hardware and software maintenance and upgrades
- Power grid failure
- Localized disasters (fire, flood, etc. In a single room or building.)
- Regionalized disasters (fire, flood, etc. In a large geographic area)

To summarize the job of creating a highly available e-business solution involves identifying potential outages for which the solution is expected to be available then including in the design those procedures and protections necessary to realize the expectations. The business may decide that some outages (or impacts to outages) are acceptable because of the cost required to eliminate them.

1.1.1 Types of Availability

There are two types of availability: process and data. Process availability is simply the state where processes exist that can process requests. Data availability occurs when data is preserved across process failures and is available for processes that continue to be available. In many systems, data availability is crucial. For example, it is of little value for a banking system to remain available if it can not access account data.

Data availability can be further broken down by the general types of data:

- static data – binaries, install images, etc.
- rarely changing data – configuration information, new document versions, passwords, etc.
- active data.- data that is rapidly changing. This type of data usually represents the essence of the system. For a banking system this would include account data.

Each of these data types require different types of actions to maintain their availability. For example, static data can usually just be installed at initial application load and perhaps copied to replica machines. Rarely changing data can often be manually updated on each replica. Of course, in either case, manual processes can be automated to reduce human error, but the key point is that data rarely changes, so maintaining availability is not difficult. For active data, the problem is much harder.

Significant efforts must be undertaken to maintain the availability of active data. Common techniques include fault tolerant disk arrays and automated replication.

1.2 Clustering in WebSphere Network Deployment V5.0

Clustering technology is used extensively in the high availability solutions involving WebSphere V5.0. A cluster consists of multiple copies of the same thing with the expectation that at least one of the copies will be available to service a request. In general the cluster works as a unit where there is some collaboration among the individual copies to ensure that requests can be directed toward a copy that is capable of servicing a request. The designers of a highly available solution participates in establishing a service level as they determine the number and placement of individual members of clusters. The WebSphere product provides the management for some of the clusters needed to create the desired service level. Greater service levels of availability can be obtained as the WebSphere clusters are supplemented with additional clustering technologies.

WebSphere Network Deployment (ND) is architected to provide for clusters of application servers. It is the concept of server clusters, that allows ND customers to implement a high availability solution, given their service level needs. A high level depiction of a cluster in ND is shown in Figure 1.1.

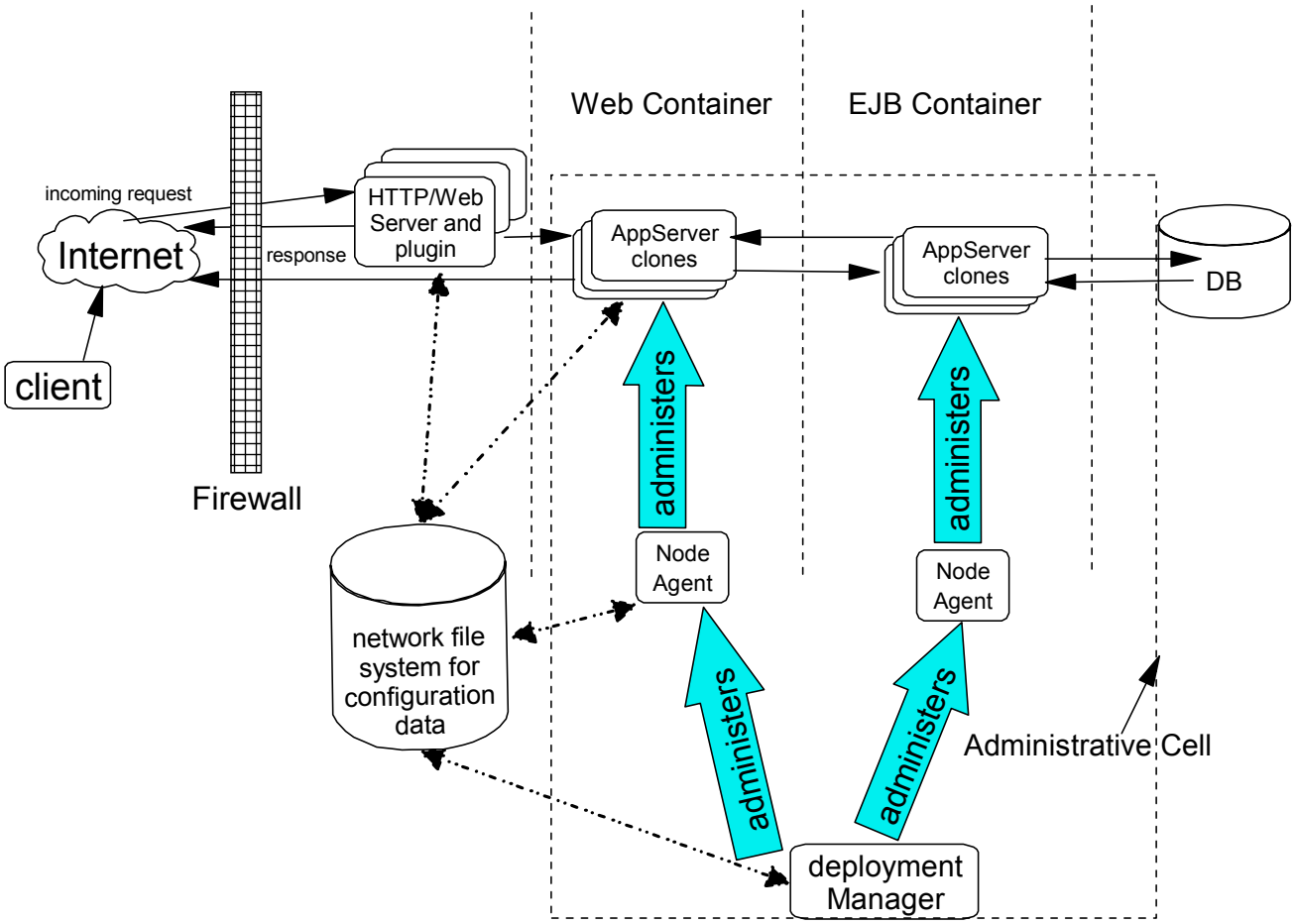


Figure 1.1: WebSphere5.0 Cluster

Inspection of Figure 1.1 reveals there are several distinct entities employed to provide a highly available production environment.

1.2.1 HTTP server and plug-in

The WebSphere Application Server works with an HTTP server (or "Web server"), to handle requests for dynamic content, such as servlets, from Web applications. The HTTP server and application server communicate using the WebSphere HTTP plug-in for the HTTP server. The HTTP plug-in uses an easy-to-read XML configuration file to determine whether a request should be handled by the Web server or the application server. It uses the standard HTTP protocol to communicate with the application server. It can also be configured to use secure HTTPS, if required. The HTTP plug-in is available for popular Web servers. These servers may be clustered, but do not fall directly within WebSphere Network Deployments administrative domain (or Cell - see 1.1.7 below).

1.2.2 Cells

Cells are arbitrary, logical groupings of one or more nodes in a WebSphere Application Server distributed network.

A cell is a configuration concept, a way for administrators to logically associate nodes with one another. Administrators define the nodes that make up a cell according to whatever criteria make sense in their organizational environments.

Administrative configuration data is stored in XML files, as shown in Figure 1.1. A cell retains master configuration files for each server in each node in the cell. Each node and server also have their own local configuration files. Changes to a local node or server configuration file are temporary, if the server belongs to the cell. While in effect, local changes override cell configurations. Changes at the cell level to server and node configuration files are permanent. Synchronization occurs at designated events, such as when a server starts.

1.2.3 Application servers

A collection of application servers working together as a single system collaborates with the Web server to return customized response to a client's request. Application code including servlets, JSPs, EJBs and their supporting classes run in an application server. In keeping with the J2EE component architecture, servlets and JSPs run in a Web container, and EJBs run in an EJB container. One can define multiple application servers, each running in its own Java Virtual Machine (JVM), thus producing an application server cluster.

1.2.4 Web containers

The WebSphere Web container provides the J2EE programming environment in which servlets and JavaServer Pages (JSP) files run. Servlets and JSP files are server-side components used to process requests from HTTP clients, such as Web browsers. They handle presentation and control of the user interaction with the underlying application data and business logic. They can also generate formatted data, such as XML, for use by other application components.

When handling servlets, the Web container creates request objects, response objects and invokes the servlet service method. The Web container invokes the servlet's destroy method when appropriate and unloads the servlet, after which the JVM performs garbage collection. The Web container also selects the appropriate Java Server Page (JSP) to run when requests are directed to the Application server. Selecting this JSP involves the use of the PageListServlet for mapping to Uniform Resource Identifiers (URI) to a JSP file that is located in the Web module of the application.

1.2.5 EJB containers

The EJB container provides all the runtime services needed to deploy and manage Enterprise Java Beans (EJBs). It is a server process that handles requests for both session and entity beans.

The enterprise beans (inside EJB modules) installed in an application server do not communicate directly with the server; instead, an EJB container provides an interface between the EJBs and the server. Together, the container and the server provide the bean runtime environment.

The container provides many low-level services, including threading and transaction support. From an administrative viewpoint, the container manages data storage and retrieval for the contained beans. A single container can hold more than one EJB JAR file.

1.2.6 Nodes

A node is a logical grouping of the managed servers on a physical computer machine.

A node usually corresponds to a physical computer system with a distinct IP host address. Node names usually are identical to the host name for the computer.

A node agent manages all WebSphere Application Servers on a node. The node agent represents the node in the management cell.

1.2.7 Deployment Manager

Deployment managers are administrative agents that provide a centralized management view for all nodes in a cell. The management of clusters and the management of workload balancing of application servers across one or several nodes is accomplished in this Deployment Manager.

A deployment manager hosts the administrative console. A deployment manager provides a single, central point of administrative control for all elements of the entire WebSphere Application Server distributed cell.

1.2.8 Vertical and Horizontal Clusters

Server clusters come in two main varieties: vertical and horizontal. These are sometimes referred to as vertical scalability and horizontal scalability.

- Vertical clustering refers to the practice of defining multiple clones of an application server on the same physical machine. In some cases a single application server, which is implemented by a single JVM process, cannot always fully utilize the CPU power of a large machine and drive CPU load up to 100%. Vertical clusters provides a straightforward mechanism to create multiple JVM processes, that together can fully utilize all the processing power available as well as providing process level failover.

- Horizontal clustering refers to the more traditional practice of defining clones of an application server on multiple physical machines, thereby allowing a single WebSphere application to span several machines while presenting a single system image. Horizontal cloning can provide both increased throughput and failover.

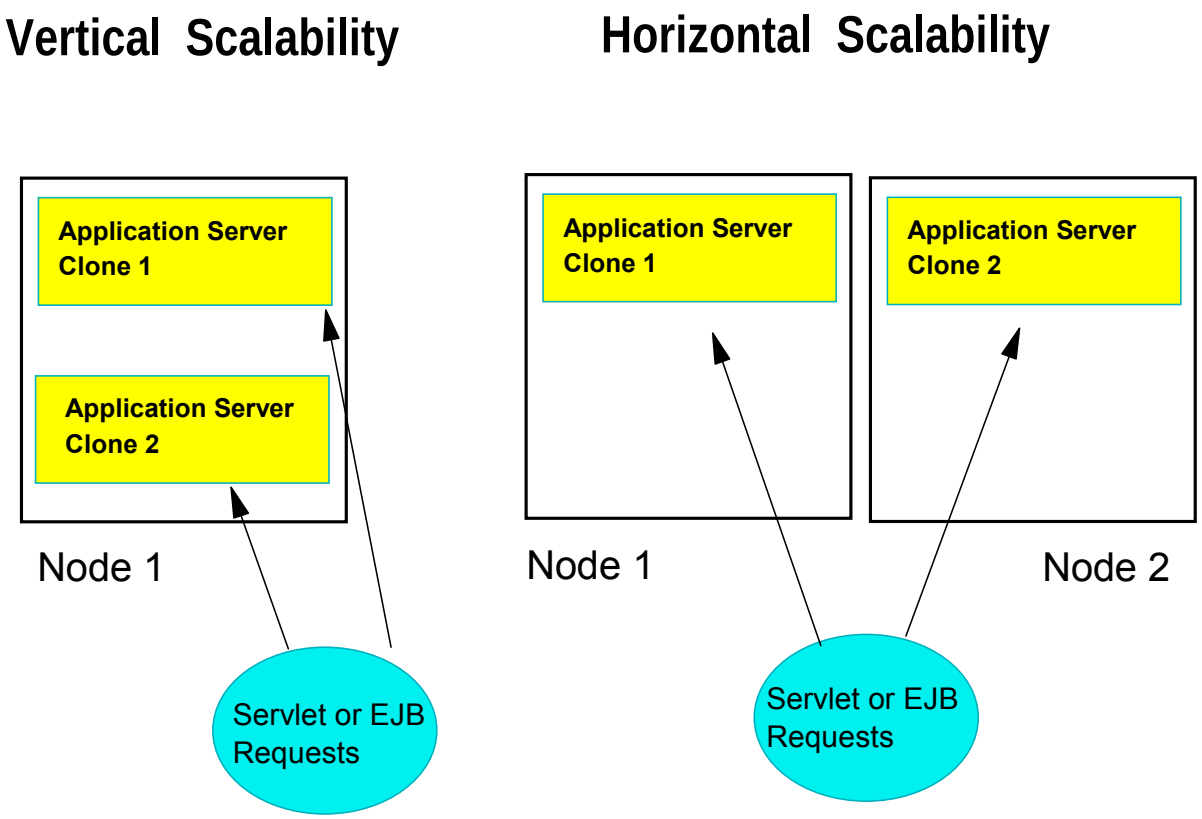


Figure 1.2: Vertical and Horizontal Scalability

1.3 Clustering of Servers in WebSphere Help Provide High Availability

By providing both vertical and horizontal scalability the WebSphere Application Server runtime architecture eliminates a given application server process as a single point of failure. Horizontal scalability eliminates a single node as a single point of failure. Vertical scalability can be added to improve process availability, or in some cases may provide better utilization of resources on single node.

1.3.1 What WebSphere V5.0 Network Deloyment provides

1.3.1.1 Routing of Servlet Requests

The routing for requests directed to a cluster of WebSphere Application Servers is provided with the HTTP Server plug-ins. An instance of the plug-in is associated with each HTTP server. These plug-ins are dependent upon the definition of the cluster contained in the config plug-in file and dispatch servlet requests from the HTTP server to one of the application servers in the cluster. In addition to providing for distribution of requests, the plug-in also provides for failover of requests once

an application server is determined to be unavailable or not responding to requests. The specifics of how this determination occurs are discussed in chapter 3 - **Clustering Application Servers within the Web Container**

1.3.1.2 Routing of EJB Requests

In a similar vein, WAS-ND also provides for distribution of requests for Enterprise Java Beans. These requests can come from a standalone Java client, another EJB, or from a servlet (or JSP) as is typically the case in web client architecture. Again the key point for purposes of this discussion is to recognize that requests can be dispatched among 1 to N Application Servers residing on multiple physical machines, eliminating any single point of failure. These "replica processes", on a set of server clones, provide high process availability. Data availability is an application domain issue, and is discussed below. The specifics of the implementation of EJB request distribution, and failover, are discussed further in chapter **4 - Clustering Application Servers in the EJB Container**

1.3.1.3 Deployment Manager and Node Agents

In WAS-ND, the deployment manager is the central controller of administration and configuration for the administrative domain (or cell) The deployment manager works with node agents to do all administrative and configuration tasks. The deployment manager doesn't participate directly in the distribution of requests to cluster members , however, the runtime of the application servers depend upon the data distributed by the deployment manager. The unavailability of a deployment manager impacts both the ability to make configuration changes and for the changes to be propagated to the application servers (including the stopping and starting of application servers). Making the deployment manager of the WAS-ND environment highly available is covered in chapter **6 - Techniques for Clustering the Deployment Manager.**

The deployment manager controls the master repository (configuration data) for the deployment cell and replicates relevant portions of this repository to all of the nodes. This replication, along with several runtime services such as naming and security having some implementation in every server process allows these services to run even when the deployment manager or node agents are not available.

In accordance with the CORBA architecture IIOP requests utilize a Location Service Daemon (LSD). WebSphere V5 provides a LSD in the Deployment Manager, Node Agent and Application Server processes. While each LSD can be used , the quality of service from an HA perspective varies depending on the process chosen.. Details on routing of requests through the LSD and why one would choose to bootstrap to a Deployment Manager, Node Agent or Application Server to are given in Chapter 4 - **Clustering Application Servers within the EJBContainer**. We will further discuss deployment and configuration of deployment managers and node agents, to enhance HA in ND, in chapter 5 - **Deployment Manager and Node Agent High Availability Best Practices**.

1.3.1.4 Routing of requests to Proxy Servers

The proxy server is the first server within the enterprise where application components may be hosted. These proxy servers are clustered using the tools available in the WebSphere DMZ. Routing request to them must be taken care of with an IP Sprayer (like the WebSphere Edge Load Balancing Dispatcher). The individual members of the cluster are configured to this IP sprayer.

1.3.1.5 Routing of requests to Web Servers

The routing of requests directed to web servers is dependent upon where the cluster of web servers is placed in the application environment. When the web servers receive web requests directly from the web then an IP sprayer is used to route the requests. When the web servers receive requests not handled by the proxy servers then it is the proxy servers that route the requests to the web servers.

1.3.2 Additional products that enhance the WAS-ND solution

While the WebSphere Network Deployment V5.0 provides a robust failover and request distribution capability for web servers and application servers , there are other components that are included in a production web application that affect the availability of that application in a production environment . It is important to consider the entire production environment, not just the WebSphere Application and Administration Servers. These include dispatchers (if something besides theWebSphere Edge Server Load Balancing Dispatcher is employeed for incoming HTTP requests to clustered web servers), firewalls, network file systems, LDAP servers, MQ servers, and JMS servers.

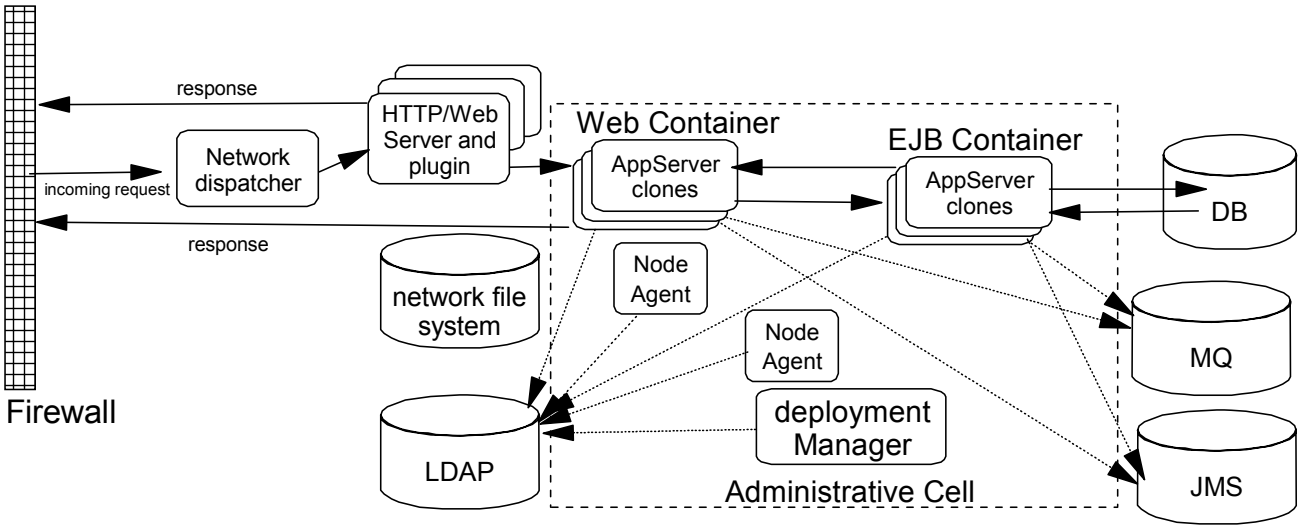


Figure 1.3: MQ, LDAP and JMS additions to cluster topology

1.3.2.1 Distribution of HTTP Requests

A mechanism of providing request distribution/failover, for incoming HTTP requests, is required. Without this mechanism, the HTTP server becomes a single point of failure, and scalability is limited

by the size the hardware that can be used to host the HTTP server. A likely solution would be to employ an IP Sprayer, such as WebSphere Edge Server Load Balancing Dispatcher. Like WAS-ND, the WES architecture provides for cluster support, eliminating the need for third party hardware to provide server clusters. The subjects of HTTP request failover are discussed further in chapters 2 and Appendix A.

1.3.2.2 LDAP Server

WAS-ND can use an LDAP server for its security and directory service, and greatly enhance these areas for the WAS-ND solution. However, a single LDAP server can be a single point of failure in the system, and we will discuss how to build high availability and scalability WebSphere LDAP-enabled system based on hardware clustering, referral, replication, and IP sprayer in Chapter 11.

1.3.2.3 JMS and MQ server

In order to support Message-Driven Beans (MDB), an embedded JMS server is intergrated in WAS V5.0 . The embedded JMS server, packaged in WAS-ND, provides all functions to support J2EE spec 1.3 messaging applications. The embedded JMS server doesn't support clusters without additional setup and configuration, and therefore is another single point of failure. We will discuss hardware-based clustering to remove this failure point, thus providing JMS server high availability, in chapter 7.

In chapter 7, we also discuss the use of IBM MQSeries queue clustering in the WAS-ND environment for support of JMS, and how to use combined hardware-based clustering and MQSeries queue clustering to enhance the MQ availability and automatic fault detection.

1.3.3 Discussion of weak points, within the WAS-ND solution, for high availability

1.3.3.1 Database Servers

Without highly available application data, web applications cannot provide dynamic and personalized data for client requests. Without consistently available databases, order processing and transactions are not possible, and a production site could not serve dynamic content. Therefore highly available data is critical to a highly available e-business. Database availability, and details for handling database failover within the WAS-ND environment (and within applications for application recovery) will be discussed in Chapter 6.

1.3.3.2 Firewalls, Networks, and Network File Systems

Firewalls, the network (LAN or WAN), and network file systems remain a single point of failure. Depending on how much down time is acceptable for a customers enterprise, there are solutions to help with outages in these areas. Detailed discussion of these points occur in Chapter 12.

1.4 Where next?

Now that we have outlined clusters in WebSphere Network Deployment V5.0 as a solution for high availability, we move onto a more detailed look at the product. Chapters 2-5 look at the WAS-ND architecture, as was outlined in Section 1.2.1. Chapters 6-13, along with the appendices, provide a more detailed look at how to implement the WAS-ND solution in a production environment. We document "hands-on" setup and configuration details for WAS-ND, and additional products that can be used to enhance it, as was presented in Sections 1.2.2 and 1.2.3.

Chapter 2 - Clustering of HTTP/Web Servers

The e-business solutions which include WebSphere generally include static HTML content, servlets, and JSPs. These are invoked using the HTTP protocol and are serviced with an HTTP server, such as IBM HTTP Server (IHS) or some other apache based server . These servers are made highly available by placing them in a cluster. This section describes suggested strategies for creating and managing these WebServer clusters.

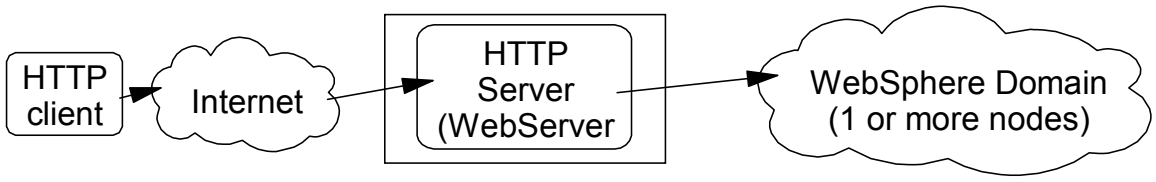


Figure 2.1: Including HTTP WebServer in the e-business solution

The management of the cluster of WebServers is something that is accomplished outside of the WebSphere management tools. However, a general pattern for setting up the three elements of a WebServer cluster (defining the members of the cluster, controlling the routing to the individual members and the management of the data needed by the servers) does exist. The members of the cluster can be defined by setting up the individual servers that will become members of the cluster. Each of these individual members is generally identified with an IP address and port. (However, the IP address may be identified with a level of indirection to the machine name and the port may be a well known port such as 8080). The control of the routing to the members of the cluster can be provided for with an IP sprayer. The data is managed through some sort of replicated/shared data storage device.

Setting up the members of the cluster will define the service level of the e-business solution. This is obvious as one considers the type of failures that may occur. Some examples help illustrate this fact: a failure of a machine is not tolerated if all members of the cluster are on the same machine, a failure of a power grid is not tolerated if all members of the cluster are dependent upon the same power circuit.

Setting up an IP sprayer provides a gateway into the cluster. All client requests (generally browser requests) are directed to the gateway (IP sprayer) and the IP sprayer redirects them to the individual members of the cluster. Most IP sprayers provide several different options for routing requests among the HTTP servers, from a basic round-robin algorithm to complex utilization algorithms. With the introduction of the IP sprayer as the router to the members of the cluster we have also introduced another entity which also needs to be included in a cluster. Figure 1.2 depicts a configuration of clustered HTTP/Web servers.

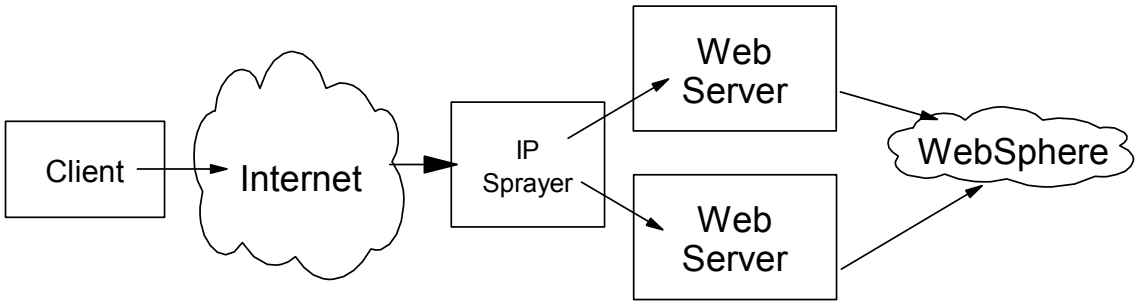


Figure 2.1 Highly Available web servers

2.1 Web Traffic Routing for scalability and failover with WebSphere Edge Server

We will discuss IBM WebSphere Edge Server Network Dispatcher here and compare its functions with DNS capability later.

IBM Edge Server Network Dispatcher has several components that can be used separately and together to provide better failover and high availability support:

- Dispatcher
- Content Based Routing (CBR)
- Site Selector
- Mailbox locator
- Consultant for CiSco CSS Switches

Figure 2.2 shows a basic configuration that implements an IP sprayer as a router and also in a cluster. Each machine in the topology is configured with at least one physical IP address, and a loopback adapter configured with a shared virtual IP address, sometimes called a cluster address. HTTP clients make HTTP requests on this virtual IP address. All requests are routed to the sprayer which in turn sprays them among the members of the cluster of web servers. The cluster of web servers consists of identical web servers running on different physical machines. In the event of a failure of one of the HTTP servers, the IP sprayer discontinues directing work to the failed server.

The cluster for the IP sprayer is created by providing a backup IP sprayer, which maintains a heartbeat with the primary sprayer. If primary sprayer fails, the backup sprayer will take over the virtual IP address and process requests from HTTP clients.

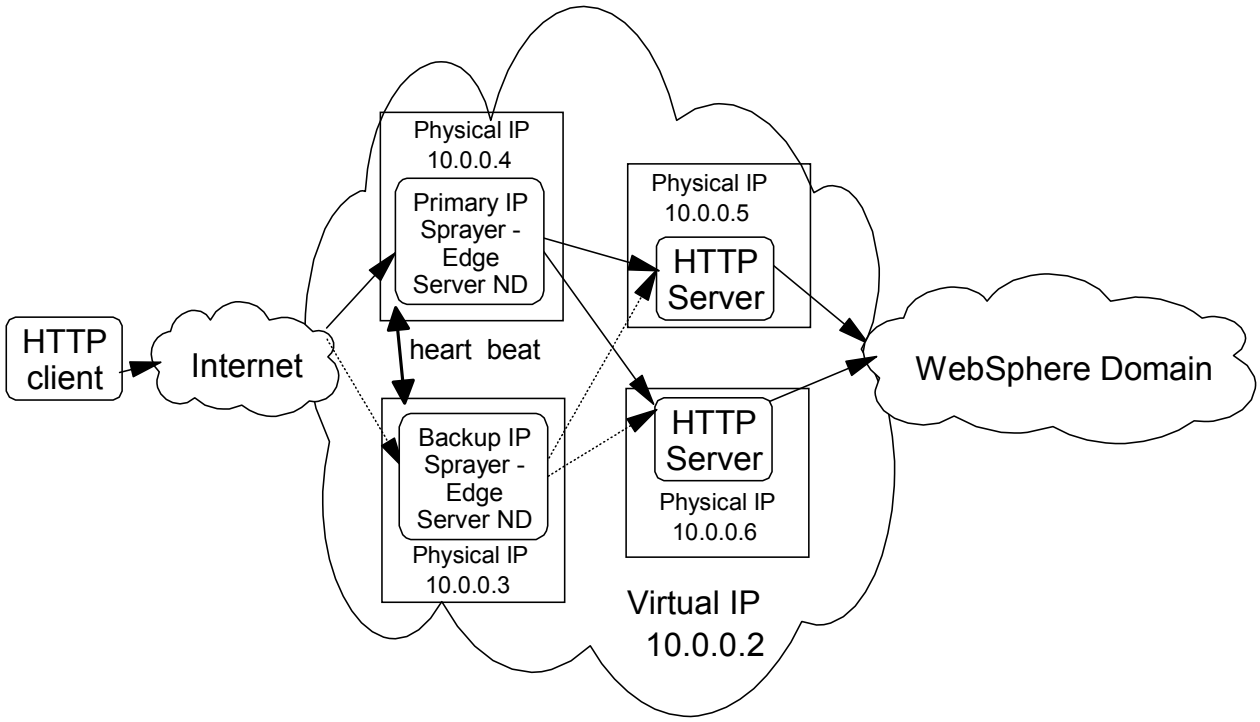


Figure 2.2: Highly Available web server configuration

Some IP Sprayer products also provide mutual high availability for an IP sprayer. In this configuration, each IP sprayer routes requests for a virtual IP address (although both sprayers are not routing the same virtual IP address) to the members of the HTTP server cluster. When one of the sprayers fails, the other will take over responsibility for all of the virtual IP addresses and route all requests to the cluster members.

IBM WebSphere Edge Server Network Dispatcher can be configured to use dynamic weights and measurements to route the request traffic. While the Network Dispatcher supports failover for web servers, this entity itself needs to be highly available. The cluster for the IP sprayer is created by providing an active/standby configuration. A backup IP sprayer maintains a heartbeat with the primary sprayer, and if primary sprayer fails, the backup sprayer will take over the virtual IP address and process requests from HTTP clients.

The Edge Server also provides an active/active configuration, that allows mutual high availability for an IP sprayer. See Figure 2.3 below. In this configuration, each IP sprayer routes requests for a virtual IP address (although both sprayers are not routing the same virtual IP address) to the members of the HTTP server cluster. When one of the sprayers fails, the other will take over responsibility for all of the virtual IP addresses and route all requests to the cluster members.

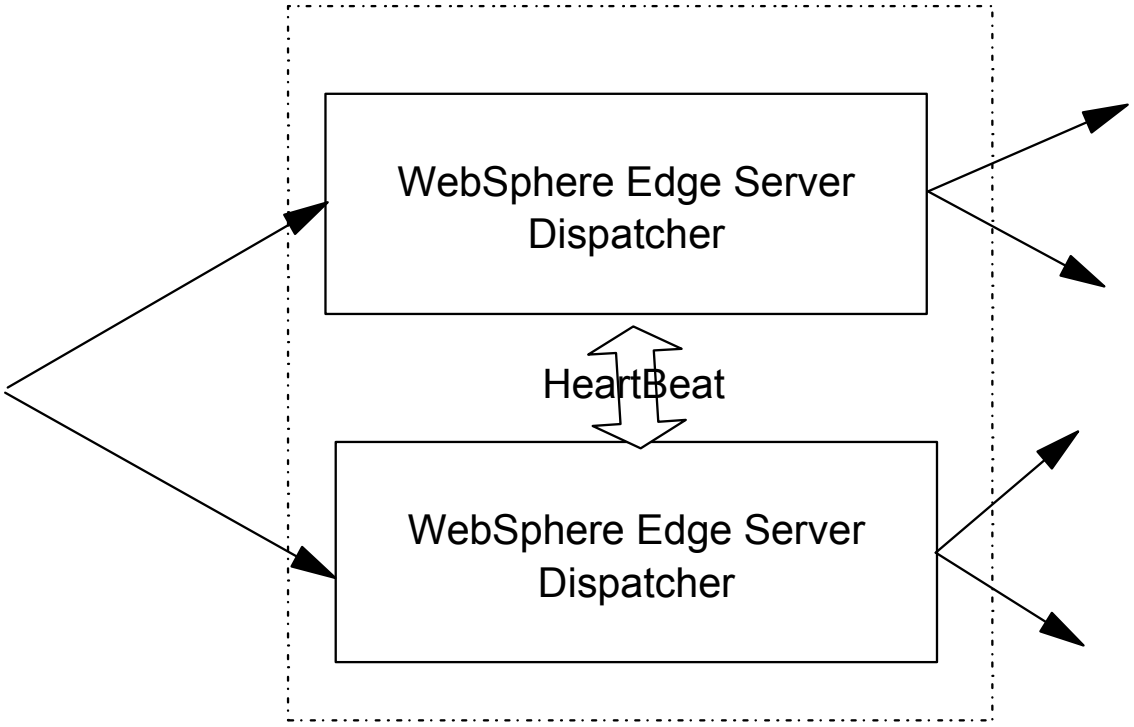


Figure 2.3 Active/Active Configuration of the Edge Server Dispatcher

There are several IP sprayers available, including IBM's Network Dispatcher, a part of the WebSphere Edge Server product. For more information on working with Network Dispatcher, see Appendix A and "WebSphere Edge Server: Working with Web Traffic Express and Network Dispatcher" (SG246172) available from http://www.redbooks.ibm.com.

2.1 Failure Detection

The Manager component of Dispatcher uses lightweight clients that runs inside the dispatcher. It sends transactions to determine the status of web servers. If the transaction succeeds, the web server is considered available. However, if a transaction fails, the server is considered unusable, and its weight is set to zero (i.e. no requests will flow to it). Future requests will not flow to this server, until a transaction succeeds, indicating its availability.

2.2 Server Affinity

If server affinity is not enabled, WebSphere Edge Server Network Dispatcher routes the traffic according to server weights (weighted round-robin). However, client requests with affinity are required for due to transactions or to improve performance. There are several ways for the Dispatcher to maintain server affinity:

- Binding client IP address to server port, for a configurable time period ("sticky IP")
- SSL session ID
- Passive cookie
- Active cookie

The easiest way to configure the server affinity is to enable a "sticky IP". Once enabled, all subsequent connections from the same source (client) IP address are dispatched to the same server until the timeout expires for binding the source IP address and server port. While there is no hard and fast rule for setting the timeout, one can estimate the maximum/average duration of client requests, and set it to this value.

While the Sticky IP approach is simple and can satisfy most requirements for server affinity, it does not, for example, parse the content of individual requests. Content Based Routing (CBR) provides a more advanced technique to support server affinity, as it checks the information contained in the HTTP data stream, such as URLs, paths, cookies, and make routing decision based on the information. Enabling CBR in ND is similar to the WebSphere V4.0 Advanced Edition. Refer to "IBM WebSphere V4.0 Advanced Edition Scalability and Availability" (SG24-6192-00) available at http://www.redbooks.ibm.com.

2.3 DNS Spraying

DNS round-robin may also be used to allow HTTP requests to be served by multiple HTTP servers. However, this solution can introduce failover problems because DNS resolution may be cached by intermediate DNS servers. In the case of failure, the cache may not be flushed quickly enough to allow resolution to a different and functional server. Further more, some DNS round-robin implementations do not provide failure detection of nodes being resolved. Without this capability, a subset of clients of the DNS may be rerouted to the failed node.

How the data needed by the web servers is made available depends upon how (where) the members of the cluster are placed. When all of the HTTP server machines are located in close geographic proximity then a mirrored or RAID protected, distributed file system can be accessed by all of the members of the cluster. When the web servers are located around the world then some other mechanism to replicate the data must be employed. High availability network file systems will be discussed in chapter

Chapter 3 - Clustering Application Servers within the Web Container

The mechanism used to route HTTP requests to WebSphere application servers is a technology termed the "HTTP plug-in". As in WAS versions prior to WAS V5.0 , the plug-in receives the data needed for the routing decisions from a configuration file. The administrative actions to generate the file and distribute the file to the HTTP server machines is typically handled via a script that utilizes the WebSphere tools for plug-in file creation in conjunction with FTP or some other file distribution mechansim.

In theWAS-ND V5.0 product, we will continue to use an xml configuration file for the plug-in. When the file is generated, a weighting value is associated with each server. These weights are used by the routing policy support to make the decision as to the specific server to handle the request.

Communication of the routing information between the HTTP plug in and the WebSphere application servers is accomplished by adding special header information to the HTTP request and response. Attached to the HTTP request is special header information that identifies how current the routing information is in the HTTP plug in. When the WebSphere application server receives the HTTP request, it strips the special header information from the HTTP request. If the WebSphere application server's routing information is more current than the HTTP plug in routing information, the WebSphere application server's routing information is sent to the HTTP plug in by placing the routing information in a special header that is attached to the HTTP response. The HTTP plug-in strips off the routing information from the HTTP response and updates it's version of the server list and server weights. The HTTP plug in will compare the dynamic routing server list to the HTTP plug in file server list. Only those servers that have an entry in both lists will be considered active.

As will be seen in Chapter 4, the WebSphere application server's routing information will be updated using a similar mechanism used for IIOP (e.g. application servers in the EJB container) requests. One main contrast is that the updates to the HTTP server list require a manual editing, while the deployment manager updates the routing table for the IIOP requests automatically. For example, suppose a server group had four servers named A, B, C and D and there were two HTTP plug ins directing requests to these servers. By deleting servers A and B from the plug in file, the first HTTP plugin would only direct requests to servers C and D. The second HTTP plug in could direct its requests only to servers A and B by deleting servers C and D from the HTTP plug in file.

The diagram below depicts request distribution at a high level.

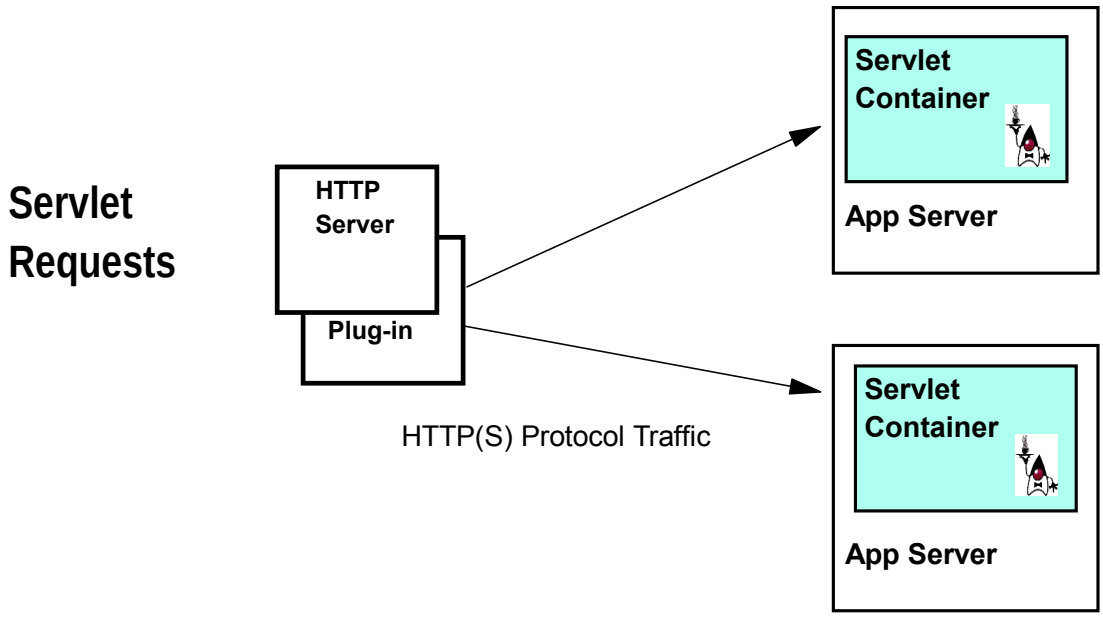


Figure 3.1: WebSphere Servlet Request Mechanism

In ND, several new functions have been added to enhance the WebSphere plug-in failover and recovery: server weight that minimizes the unavailability due to overloading, ordered server cluster failover, which eliminates two cluster servers being selected as the session failover target concurrently, primary servers and backup servers groups which provide another level for failover support, non-blocking connection that reduces the impact of operating system's TCP/IP keep-alive timeout, and memory-to-memory session replication that enhances session data availability.

The web container failover support in ND is provided by three mechanisms:

- Server clusters
- HTTP plug-in that controls the (weighted round robin) routing of client requests among redundant server processes
- Session management that provides preservation of http session data

We will discuss all of these three aspects below.

3.1 Web Container Server Clusters and Failover

WAS-ND provides support for creation of server cluster, as defined in Chapter 1, section 1.2. These servers can all reside on a single node, or can be distributed across multiple nodes in the WebSphere Domain (or cell). WebSphere servers can share application workload and provide failover support. All servers can be divided manually into primary servers and backup servers. If one of the primary servers fails, work can continue to be handled by other primary server in the cluster, if available. If all of the primary servers are unavailable, the request will fail over to the first backup server. If the first backup server fails, the second backup server will be used. A highly available topology is depicted in Figure 3.2

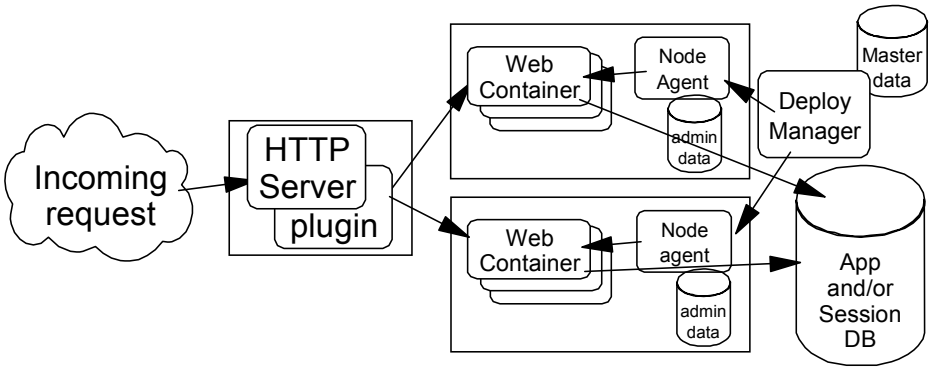


Figure 3.2: High Availability Web Container Configuration

3.2 The HTTP Plug-in

3.2.1The HTTP Plug-in basics

The plug-in uses an XML configuration file to determine information about the WebSphere domain it is serving. This configuration file is initially generated by the WebSphere Administrative Server. If the web server is on a machine remote from the WebSphere Administrative Server, as in the above diagram, the plugin-cfg.xml file will need to be moved to the web server machine.

When the HTTP server is started, the plug-in reads the information from the plugin-cfg.xml configuration file into memory and each server is assigned with a server ID and builds an ordered server list. The plug-in also uses this configuration file to determine how to route HTTP requests to WebSphere web containers. To more closely examine how the plug-in makes this determination, let's look at an example. Assume the plug-in receives a request for the URL http://www.mycompany.com/myapplication/myservlet.

The plug-in parses the request into two pieces, a virtual host (www.mycompany.com:80) and a URI (/myapplication/myservlet). The plug-in then checks the plugin-cfg.xml file searching for a match for these two items:

```
<UriGroup Name="myapplication web module">
   <Uri Name="/myapplication"/>
</UriGroup>
<VirtualHostGroup Name="default_host">
   <VirtualHost Name="*:80"/>
   <VirtualHost Name="*:9080"/>
</VirtualHostGroup>
```

Next the plug-in searches for a route entry that contains both the <UriGroup> and the <VirtualHostGroup> entry. This route is used to determine the cluster which can handle the request.

```
<Route VirtualHostGroup="default_host"  UriGroup="myapplication web module"
ServerCluster="HACluster" />
```

A cluster represents either a standalone server or a group of WebSphere clones which are available to service the request. Once the cluster has been determined, the plug-in must choose a server within the cluster to route the request to.

```
<ServerCluster Name="HACluster">
   <Server CloneID="u307p2vq" LoadBalanceWeight="2" Name="HAClusterServer1">
      <Transport Hostname="9.5.90.22" Port="9081" Protocol="http"/>
   </Server>
   <Server CloneID="u307p48r" LoadBalanceWeight="3" Name="HAClusterServer2">
      <Transport Hostname="9.5.90.22" Port="9082" Protocol="http"/>
   </Server>
```

The plug-in first checks to see if the client request has an HTTP session associated with it. If so, the CloneID is parsed from the end of the session ID and compared with the CloneIDs for the servers in the cluster until a match is found. The request is then routed to that server. If no session ID is associated with the request, the plug-in sends the request to the next server in its routing algorithm.

Once the server has been determined, the <Transport> must be chosen for communications. Transports define the characteristics of the connections between the web server and the application server, across which requests for applications are routed. If the server only has one <Transport> configured, this decision is easy. It is possible, however, for the server to have two transports configured, one for HTTP communication and one for HTTPS communication.

```
      <Transport Hostname="myhost1.domain.com" Port="9089" Protocol="http"/>
      <Transport Hostname="myhost1.domain.com" Port="9079" Protocol="https">
         <Property name="keyring" value="C:\WebSphere\AppServer\etc\plugin-key.kdb"/>
         <Property name="stashfile" value="C:\WebSphere\AppServer\etc\plugin-key.sth"/>
      </Transport>
   </Server>
```

In this case, the <Transport> communication between the plug-in and the web container is matched to the communication protocol used between the browser and the web server. The URL http://www.mycompany.com/myapplication/myserlvet would be sent from the plug-in to the web container using the HTTP transport while https://www.mycompany.com/myapplication/myserlvet would be sent using the HTTPS transport.

The rule to chose either HTTP or HTTPS depends on client security context and server security support as described in the following table:

Client request with	Only HTTP transport in the AppServer	Only HTTPS transport in the AppServer	Both HTTPS and HTTP in the AppServer
HTTP	HTTP	HTTPS	HTTP
HTTPS	HTTP	HTTPS	HTTPS

We can add and modify the plugin-cfg.xml file for better failover performance: 1) partition all servers into primary server group and backup server group. In ND, this is called two-level failover support. When plugin-cfg.xml file is generated, all servers are in the primary server group; one can manually

move some of these servers into backup server group; 2) adjust plugin-cfg.xml refresh time; (3) adjust RetryInterval; 4) add ConnectTimeout attribute for each server.

3.2.2 Plugin-cfg.xml hot updates

The plug-in periodically checks to see if the file has been modified, and reloads the information if necessary. If the change is detected, the new plug-in.xml will be loaded:

How often the plug-in checks this file is based on the RefreshInterval property, defined in the XML file. If this attribute is not present, the default value is 60 seconds.

```
<Config RefreshInterval=180>
```

In a production environment, the RefreshInterval should be set to a relatively high number, as the overhead of the plug-in checking for a new configuration file frequently can adversely affect performance.

3.2.3Weighted Round-Robin Routing

WAS-ND employs a weighted routing algorithm. This algorithm will allow customers to assign weights to each of the server clones and have requests distributed to those server clones based on the assigned weight. Over time, the proportion of the requests that are directed to any one server is the weight of that server divided by the sum of the weights of all of the servers in the cluster.

The weighted round-robin algorithm will be implemented as follows. A weight will exist for each server in a server cluster. The weights for all of the servers in a server cluster will be the “weight set” for that server cluster. The client request is routed to each server in the server cluster in a round-robin fashion. For each client request the weight of a server is decremented by 1, and when the weight of the server becomes zero no more requests are sent to that server. This process continues for each server in the cluster until the complete “weight set” reaches zero, at which time the weight set is reset and the whole process starts over again.

The weight is specified within the admin console for each server, in a given cluster. At server startup, each server has a weight set to the value input in the admin console. The generated plugin-cfg.xml file includes an entry for the server weight, for example:

<Server CloneID="u307p71m" **LoadBalanceWeight="5"** Name="HAClusterServer4">

For example, for two servers with weights of 2 and 4, the client request routing sequence and server weight change are shown in the following table.

Server servicing request	ClusterServer1 weight	ClusterServer3 weight
ClusterServer3	2	3
ClusterServer1	1	3
ClusterServer3	1	2
ClusterServer1 (no new requests to server1 allowed)	**0**	2
ClusterServer3	0	1
ClusterServer3	0	**0**
ClusterServer1 (reset weights)	2	4

3.2.4 Web App Server Failure Modes

When a web container fails, it is the responsibility of the HTTP server plug-in to detect this failure and mark the web container unavailable. Web container failures are detected based on the TCP values (or lack of) to a plug-in request. There are five types of failover scenarios for web containers:

I. Expected server process failure, for example, stop the server
II. Unexpected server process failure, for example, the server process dies
III. Server network problem, for example, network cable is disconnected
IV. Unexpected and/or expected machine problems, for example, machine is shutdown, operating system crashes, power is turned off
V. Overloading of web clients, for example, denial of service attack, machine is too small to handle large number of clients

In the first two cases, the physical machine where the web container was operating remains available, however the port over which the web container processes was accepting requests will not. When the plug-in routes a request to this machine, the connection over this socket will be refused, causing that application server to be marked unusable.

In the second two cases, however, the machine is no longer available to provide any kind of response. In these cases, if non-blocking connection is not enabled, the plug-in will wait for the local operating system to timeout the request before marking the server unavailable. While the plug-in is waiting for this connection to timeout, requests routed to the failed server appear to hang. You can enable non-blocking connections, to eliminate this loss of processing time on a server that can be reached (as is discussed in the next section).

In the fifth case, client overloading can make a server appear to be unavailable. Often this is due to incorrectly setting a weight for a server. We will discuss server overloading in an upcoming section.

3.2.5 Setting TCP Timeout Values and Adjusting the Retry Interval

The default value for the TCP timeout varies based on the operating system. While these values can be modified at the operating system level, adjustments should be made with great care. Modifications may result in unintended consequences in both WebSphere and other network dependent applications running on a machine. See Appendix B for details on viewing and setting the TCP timeout value for each operating system.

If a request to a server in a cluster fails, and there are other servers in the group capable of servicing the request, the plug-in will transparently reroute the request to the next server in the routing algorithm. The unresponsive server is marked unavailable, and will not participate in the routing of requests for some interval of time. The amount of time the server remains unavailable is configured by the RetryInterval property on the <ServerGroup> attribute. If this attribute is not present, the default value is 60 seconds.
When this interval expires, the plug-in will attempt to send a requests to this server. If the request fails or times out, the server is again marked unavailable.

The proper setting for a RetryInterval will depend on the environment, particularly the value of the operating system TCP timeout value and how many servers are available in the cluster. Setting the RetryInterval to a small value will allow a server, which becomes available again quickly, to begin serving requests. However, too small of a value can cause serious performance degradation, or even cause the plug-in to appear to stop serving requests.
To explain how this can happen, let's look at an example configuration with two machines, A and B. Each of these machines are running two cloned servers. The HTTP server and plug-in are running on machine A and B, with TCP timeouts of 75 seconds and RetryInterval of 60 seconds, each. If machine A fails, the following events occur when a request is intercepted by the plug-in:

1. The plug-in accepts the request from the HTTP server and determines the cluster.
2. Request routed to clone 1 on machine A.
3. Because machine A is down, 75 seconds expire during the operating system TCP timeout interval, before clone 1 is marked unavailable.
4. Request now routed to clone 2 on machine A. Because A is still down, the 75 seconds pass before clone 2 is marked unusable.
5. Request now routed to clone 1 on machine B, which successfully returns a response to the client. However, over 2 minutes have passed since the request was first handled by the HTTP server.
6. Note that during the TCP timeout interval, while the client was awaiting a response from clone 2 on machine A, the 60 second RetryInterval for clone 1 on machine A expired, and the clone was added back into the routing algorithm. A new request will soon be routed to this clone, which may still be unavailable, thus beginning lengthy process again.

To avoid this problem, we recommend a more conservative RetryInterval, related to the number of clones in your configuration. A good starting point is 10 seconds + (# of clones * TCP_Timeout). This ensures that the plug-in does not get stuck in the situation of constantly trying to route requests to the failed clones. In the scenario above, this setting would cause the two clones on machine B to exclusively service requests for 235 seconds before the clones on machine A were retried, resulting in a another 150 second wait.

3.2.6 Non-blocking Connections as Means of Boosting Performance

WAS-ND provides a means of configuring a non-blocking connection, that eliminates the impact of operating system TCP/IP timeout. There are side effects to reducing the operating system TCP/IP timeout, and customers should do so with great care. For example, the TCP/IP timeout also controls the incoming traffic for some operating systems; if this is too small, client requests from slow network connections (for example dial-up services) will timeout before they can eve reach an application server!

You can enable socket non-blocking connection to avoid this problem. In plugin-cfg.xml file, add an attribute ConnectTimeout=<value> in each server as:

```
    <Server CloneID="u307p2vq" LoadBalanceWeight="2" Name="HAClusterServer1"
ConnectTime=”5”>
    </Server>
```

```
      <Server CloneID="u307p48r" LoadBalanceWeight="3" Name="HAClusterServer2"
ConnectTime=”10”>
     </Server>
     <Server CloneID="u307p62u" LoadBalanceWeight="4" Name="HAClusterServer3"
ConnectTime=”15”>
     </Server>
     <Server CloneID="u307p71m" LoadBalanceWeight="5" Name="HAClusterServer4">
ConnectTime=”20”
      </Server>
```

If the server cannot respond before the ConnectTimeout, the Plugin will mark it as unusable. A small value leads may result in faster detection of faults, but may cause a false detection, and the ConnectTimeout value should be larger than “normal” server response times, in order to avoid false detection of a fault. Testing with WAS-ND revealed a 10 second ConnectTimeout is reasonable.

3.2.7 Server Overloading

WAS-ND is designed to handle large number of clients concurrently. However, sometimes over driving a server can cause the appearance that a server is unusable, even though the server is capable of handling requests. The server weights reflect the relative capacity of the members of the cluster to accept work, the server weights also reflect the distribution of work among the members of the cluster. This claim can be easily verified by plotting the CPU utilization as a function of workload -- Trans/sec of 100, 200, 300, 400, 500, 600, 700, etc. However, this distribution of workload is only guaranteed when the weights accurately reflect the relative capacity of the members of the cluster. When some member of the cluster reaches its bottleneck prior to all the other members of the cluster, then the enterprise is operating in an uncontrolled region of the curve. The behavior in this uncontrolled region may be to throttle all work to the cluster (as implemented with the HTTP plugin router) or the behavior may be to route work away from the bottle necked server (as implemented with the IIOP router, see Chapter 4).

If there are more connections than available threads to service a request, the connections backlog, awaiting free threads. If the maximum number requests in the backlog is reached, new connections will be refused and the plugin will treat the server as unusable. This is known as an server overloading.

There are several parameters you can tune to reduce server overloading, for example server weights that more accurately reflect a servers capacity, or by increasing connection backlog, the maximum keep-alive connections, maximum requests per keep-alive connection, keep-alive timeout, I/O timeout, minimum thread size, maximum thread size, and thread inactivity timeout.

3.2.8 Primary and Backup Server Clustering (Two-Level Failover)

When the plugin-cfg.xml is generated, all servers are under the Primary Server group. The plug-in file can be manually edited to move some servers into the backup server group. For example:

<PrimaryServers>
<Server Name="HAClusterServer1"/>
<Server Name="HAClusterServer3"/>
</PrimaryServers>
<BackupServers>
<Server Name="HAClusterServer2"/>
<Server Name="HAClusterServer4"/>
</BackupServers>

The HTTP plug-in will not route requests to any server in the Backup Server group as long as there are servers available in the Primary Server group. When all the servers in the primary group are not usable, the plug-in will route the traffic to the first available server in the backup group, or failover the request to the next available server in the Backup Server group. Requests are not distributed via weight-based Round Robin in the Backup Server group, the plugin uses only server, as long as it is available. The Two-level failover mechanism provides more robust failover support, allowing customers to plan for server "over-capacity", for example during high load spikes.

3.3 HTTP Session Affinity and Failover

HTTP session objects can be used within a web application to maintain information about a client across multiple HTTP requests. For example, consider a website that employs the concept of "shopping carts". The web application needs to maintain information about what each client has placed in his or her shopping cart. Session information is stored on the server, and a unique identifier for the session is sent back to the client as a cookie, or through URL rewriting mechanisms.

3.3.1Server and Session ID

3.3.1.1Server ID

When the server is created, WebSphere assigns a unique cloneID for this server, and the cloneID is shown in the plugin-cfg.xml file, for example:

```
<ServerCluster Name="HACluster">
  <Server CloneID="u307p2vq" LoadBalanceWeight="2" Name="HAClusterServer1"></Server>
  <Server CloneID="u307p48r" LoadBalanceWeight="3" Name="HAClusterServer2"></Server>
  <Server CloneID="u307p62u" LoadBalanceWeight="4" Name="HAClusterServer3"></Server>
  <Server CloneID="u307p71m" LoadBalanceWeight="5" Name="HAClusterServer4"></Server>
</ServerCluster>
```

A CloneID is used to create a server group object. The plugin will bind cloneID and client's sessionID when it receives the client's first request.

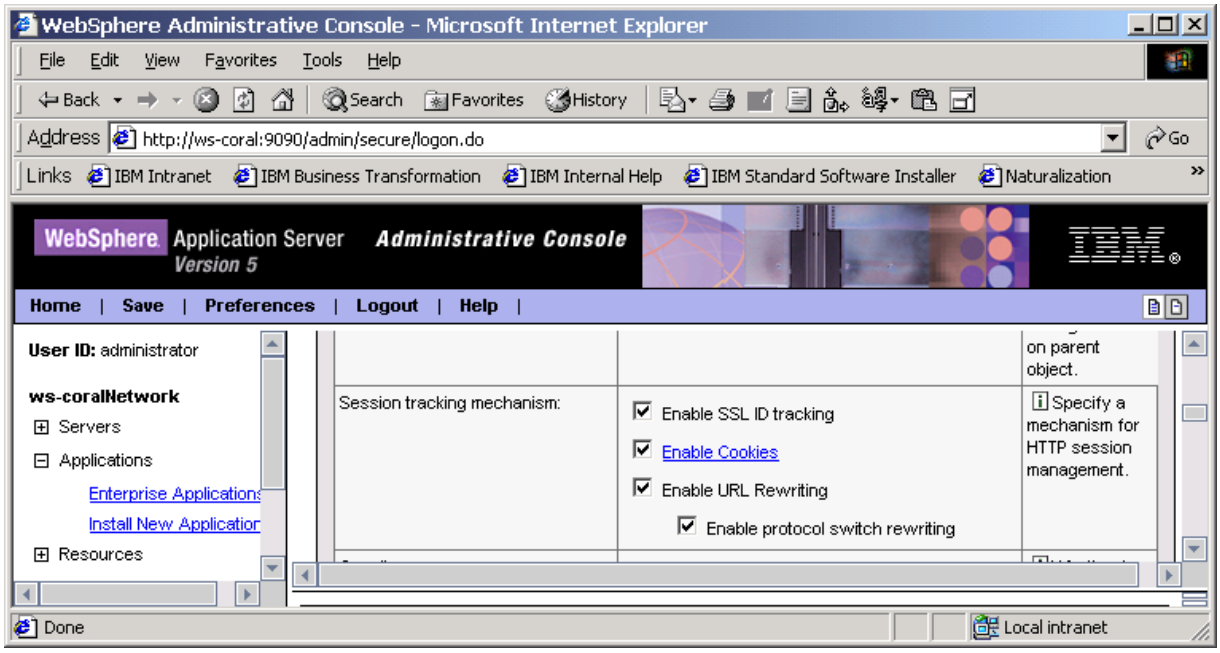
3.3.1.2Session ID

The HTTP protocol is stateless and cannot carry state information. ND attempts to remedy this by providing HTTP session support, as well as session failover. A user's session information can be kept

on a server in two ways: in memory, which provides failover support, or into a database or message brokers, which does provide failover support. A the user's session information is recognized in three ways:

- SSL session ID
- Cookies
- URL rewriting

You can enable any combination of these within the admin console, as depicted in this screen shot:



If you set more than one session tracking mechanisms, ND will check them in the following preference: SSL session ID, cookies, and URL rewriting, respectively.

The following is a typical session ID before and after a request has failed over:

HTTP failover	HTTP Session ID
Before failover	001PK1SK3PWIEATJT12Z2K1WXI:u307p2vq
After failover	0001PK1SK3PWIEATJT12Z2K1WXI:u307p2vq:**u307p62u**

The first 4 characters are used to determine the validity of the cache entries. The middle part before ":" is the session ID to identify a session. The last part is the cloneID, as discussed above. After a failover has occurred, the plugin appends the session information with the cloneID of the next server to which the request was sent. Note. the default cache timeout is 30 minutes, however a smaller value will improve performance. You can adjust the timeout value within the admin console. If a client is not active during that time, its session information will be lost during a failover event.

3.3.2 Session affinity and failover

3.3.2.1 Ordered routing process

To achieve session affinity, the Plugin will parse the client's session information, extract the previous cloneID, and match this cloneID with cloneID(s) in the server cluster object. When the matched clone is found, the server will retrieve the client's session from the in-memory cache. If the matched server clone is not available, the plugin will route the client to another server, and appends the new server's cloneID into the client's session. ND employs the weight-based ordered routing, which eliminates the possibility of two different servers being selected after a failover occurs, for concurrent clients.

3.3.2.2 No session support in the server

By default, the session affinity is enabled. If applications do not require session affinity, remove the cloneIDs in the plugin-cfg.xml. Disabling session affinity allows for faster request processing, since the time-consuming task of parsing client's session information is eliminated.

However, if an a request has built in-memory state, needed to complete a transaction, all state information is lost if the request needs to be failed over to another server. For example, a shopping cart with wanted purchases, will be empty after a request is failed lover when, if server session affinity is disabled. Note, the setting for server session affinity is for the whole cluster, not for a single web server.

3.3.2.3 No session information in the client

When server session affinity is enabled, the plugin will look for the session information for each incoming request. If the plug-in fails to find any session information from the incoming client, the plugin will assume that no session affinity is needed for this client. Session affinity is bypassed, and the request is routed according to the weighted Round Robin algorithm.

3.3.2.4Session affinity without session persistence

When the server session affinity is enabled, and an incoming client has session information, the sessions are held in-memory of the application server. The plug-in will route multiple requests, for the same HTTP session, to the same server by examining the CloneID information which is stored with the session key. In a failover scenario, the plug-in will route the client to the alternative server. If the alternative server cannot retrieve the session information, the session data is lost.

3.3.2.5Session persistence and failover

In order to have successful session failover, session data must be persisted, and for that data to be available for other servers during failover. ND as feature of memory-to-memory session replication, in addition to database session persistence. When Session Affinity and Session Persistence are both enabled, HTTP sessions are held in memory of the application server containing the web application, and are also periodically persisted to a database, or published to a message broker(s). The plug-in will route multiple requests for the same HTTP session to the same server. This server can then retrieve the information from its in-memory session cache. If this server becomes unavailable, the request is routed to another server, which can read the session information from a database or receives the session information from a broker (locally or remotely). The WAS-ND HTTP routing algorithm does not have, a priori, information about which servers have the session information locally.

The session manager can be configured at each web module, or each enterprise application, level. All web modules inside the enterprise application inherit the enterprise level configuration. The session manager can also be configured at the level of the application server. Therefore, different persistence mechanisms can be configured in the web containers at the same time, to tune failover for each web module. How much session information is lost depends on the frequency of the session persistence, which is configurable on the application server from the admin console.

3.3.3 Session Update Methods and Session Data Persistence

The WAS-ND session manager drives session data replication and persistence. There are three methods to trigger session data persistence, and each has different failover implications. The fewer the updates and the longer the update interval, the better the server performance, but the more session data can be lost in the event a fail over is required.

3.3.3.1 End of Servlet Service Method

When the write frequency is set to begin at the end of servlet's service method, the data will be written/published into database/replication stores. Therefore all subsequent session data, between two service method calls, can be lost if a request must be redirected to another server. The server now handling the request retrieves the session data from the last HTTPServlet.service() method call.

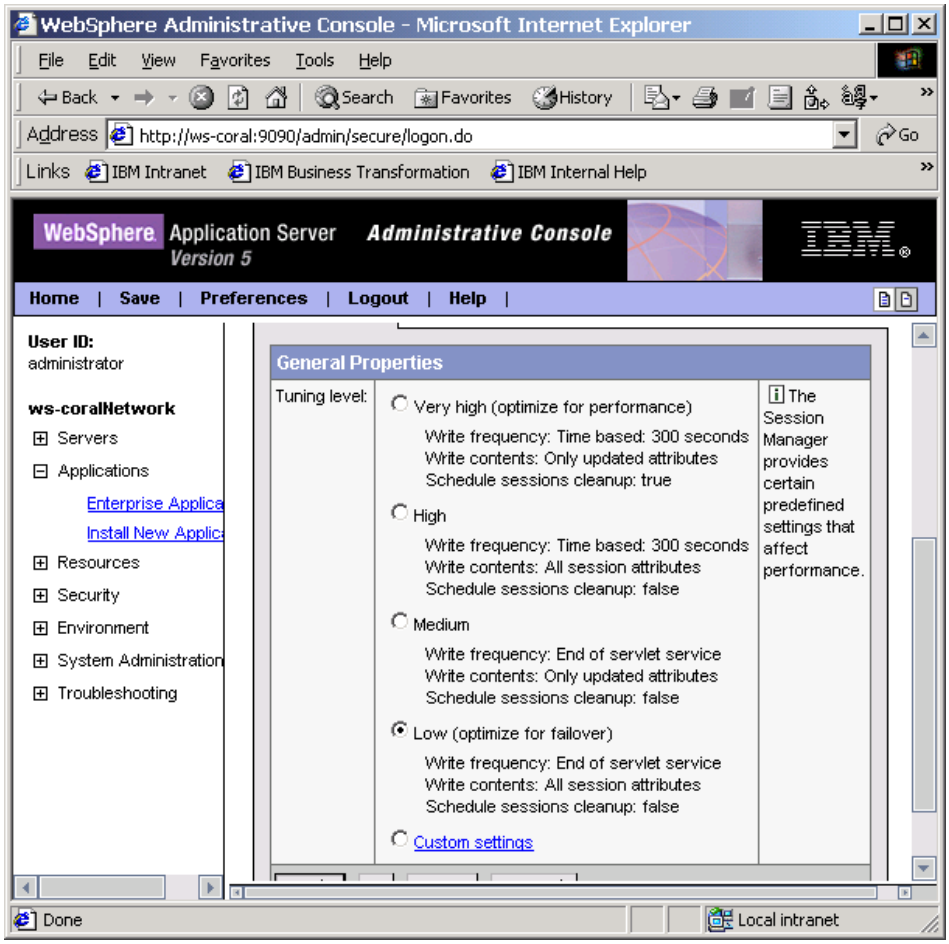
3.3.3.2Manually sync() in the code with the IBM extension

IBM extends the Servlet Specification to have a capacity to let an application decide when to update its session data. The application needs to invoke the sync() method explicitly to write/publish the session into database/replication stores. Any new session data since last invoking sync() can be lost during a failover event.

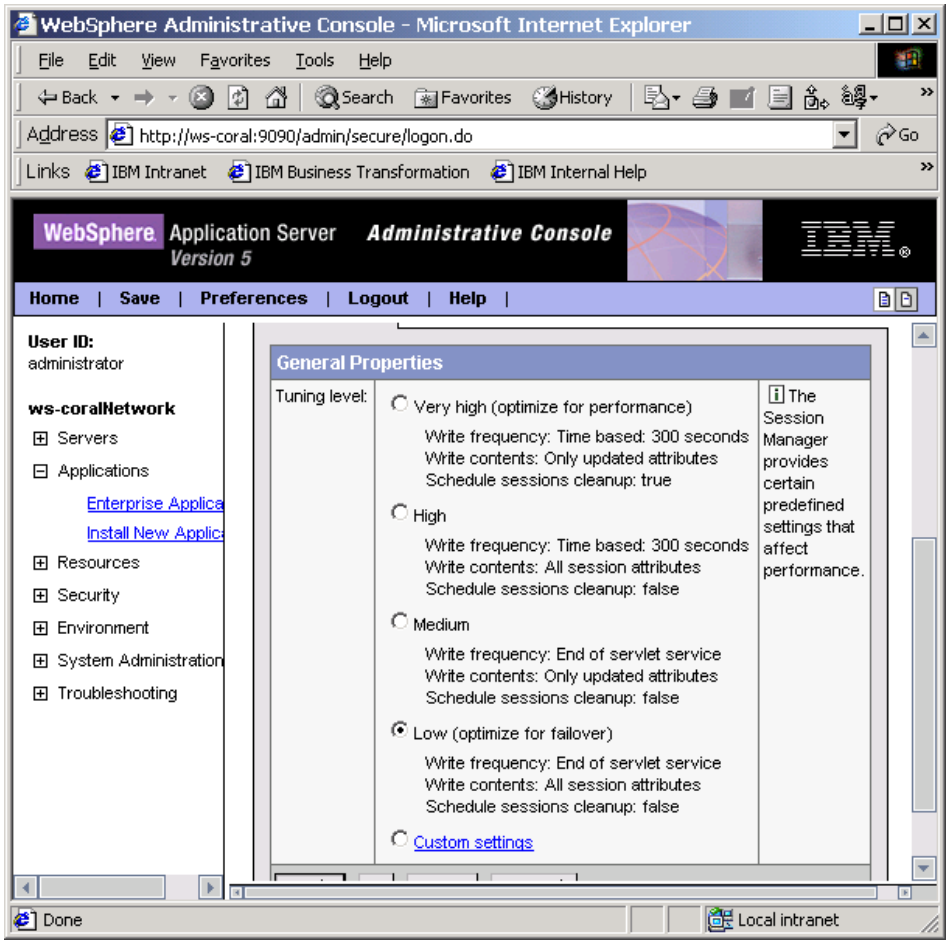
3.3.3.3Time-based

The session is written/published into database/replication stores at a configurable time interval. All new session data since last update can be lost during a failover.

These three choices can be configured from the admin console, as shown in this screen shot:



You can also use customer settings to fine tune these parameters:



3.3.4 Session persistence and failover

3.3.4.1 In memory

In-memory session management provides the best performance for HTTP requests, to keep session data, as long as the user's requests are always directed to the same server and this server never fails. However, a (local) in-memory session manager doesn't persist session data. If a server dies, all the session information is lost with the loss of the server process.

3.3.4.2 Persistent to database

The session manager provides a mechanism to write session data to a database. If the server fails, another server may retrieve the session data, thus allowing client requests to continue to be processed. The session data loss amount spans from the last update time (as discussed above), to the time the failure occurred, with the maximum data loss occurring over the entire update interval.

If the database is not available, obviously all session data is lost. Therefore, database availability can be a single point of failure. We will discuss how to ensure database availability in chapter 10.

3.3.4.3 Memory-to-memory

WAS-ND provides another mechanism for session management: memory-to-memory replication through WebSphere Internal Messaging. There are 3 topological models for memory-to-memory replication: buddy system of single replica, dedicated replication server, and peer-to-peer replication (which is the default).

If the original server fails, the HTTP plug-in will route the user's request to another server in the server cluster (the request's "adoptive server"). The session manager in the adoptive server will inspect to see if there is a copy of the user's session locally. If not, the manager will go to another store to retrieve the session data.

Considerations for configuring replication topology are: 1) at least two replication stores are needed for high availability in case that single replicated copy is also unavailable, 2) the message complexity of session replication should be minimized. While dedicated replication servers may have fewer replication messages, they are also prone to failures if many servers fail at the same time. The peer-to-peer or client-server models provide more resistance to multiple server failures. "Channel partitioning", available in WAS-ND, can be used when there are a large number of servers in a cluster, to reduce the replication message complexity.

The HTTP plug-in does not have any information on which servers have a replicated session locally, and routes a request to an adoptive server in a cluster according to server weight. In the peer-to-peer model every web application server has access to replicated session data on other web application servers. Thus, no matter how the plug-in routes a request, the server will (potentially) always have current session data locally, however replicating session data to all servers in a cluster can exhaust JVM heap in cases where there are large number of users and a large session object.

Chapter 4 - Clustering Application Servers within the EJB Container

Many J2EE applications require Enterprise JavaBeans (EJBs). High availability of application servers in the EJB container is achieved using a combination of the WebSphere server cluster support and the Workload Management (WLM) plug-in to the WebSphere Object Request Broker (ORB). As discussed in the previous chapter, WebSphere server clusters allow multiple (clone) instances of an WebSphere application server to be created, with the ability to share workload thus providing a single, highly responsive system.

4.1 Routing to WebSphere Application Servers via IIOP

The implementation of the routing pattern for IIOP requests to an application server is a combination of the two client routing patterns: client and client with advisor. This implementation is to support both those clients with our Distributed platform WebSphere code and also those clients that support the IIOP interface without the WebSphere extensions.

It is expected that the Non-WebSphere distributed clients will consist of WebSphere/390, CORBA C++, or non-IBM ORBs. The WebSphere distributed clients are server group aware and have WLM support built into the client. This support has knowledge of the servers in the server group and implements the routing policy resulting in the client making the routing decision and directing the request to a specific server . The Non-WebSphere distributed clients are server group unaware clients and do not have any WLM routing or server group knowledge and must be given a specific server to direct the client's request to. In this case we employ a Location Service Daemon (LSD), which acts as an advisor to a client, directing incoming requests to the appropriate server.

Within the LSD approach, a client obtains an indirect IOR (possibly from a look up in JNDI, or as a return argument) which contains routing information to the LSD rather than to the application server. The client request is then sent to the LSD to determine a direct IOR to be used for the object request. As a part of the resolution of the IOR some WLM logic is added to the LSD. The LSD's ORB will contain hooks that will be used to the WLM code to determine how to handle the request. The following sections describe the different functions based on client type.

Server group aware

If the client contains the distributed WebSphere WLM code and has yet to obtain the initial routing information, the initial request will flow to the LSD. The WLM director code in the LSD will determine the correct server and return the direct IOR to the client. This direct IOR will then be used to route the request to the specific server. However, since this client contains the distributed WebSphere WLM code, the request to the server will contain WLM service context data, and the WLM code on the server will look at this context data to determine whether the client has back level server group information. If required, the WLM code on the server adds server group information into the service context list on the response back to the client. The WLM code on the client then updates its server

group information (if newer data was available). From this point on the WLM code on the client has updated server group information, thus for all subsequent requests the client has the capability to choose the specific server to handle the request. This client based routing is basically how our current Java EJB clients work in WebSphere today.

Server group unaware

If the client does not have the distributed WLM code active, the LSD will invoke the WLM director where it is the director's responsibility to handle all of the WLM policy and affinity decisions that need to be made. The client does not have any knowledge of the server group so the LSD must determine the specific server the request will be sent to. This direct IOR will be then be returned to the client and be used to invoke all requests on the object. What this means is that every request the client makes on this object will go to the server that the LSD determined. This is different from server group aware clients which will determine what server to use on every request. For server group unaware clients a load balancing mechanism would be added to the server which would detect when it was getting over loaded and would stop a request by returning an error to the client. This error would trigger the client to return to the LSD again to determine another direct IOR to use. In addition, if a server being used by a server group unaware client goes down or is disconnected again an error will be returned and the client will return to the LSD and start the process over.

Location Service Daemon(LSD)

If only one LSD is configured for each server group the potential for a single point of failure exists. To prevent the single point of failure a set of LSDs per server group will be configured and if an LSD fails one of the other LSDs in the group would be used.

4.2 Routing algorithm

While the weighted round-robin algorithm described in Chapter 3 is the main routing algorithm, there are situations that will override how the request is routed. These situations are described in the paragraphs below.

In Process

If a request from a client is to an object that is already in the same process as the client, the request will automatically be routed to the in-process object. In this case no weights are decremented. The in-process optimization overrides all other selection criteria.

Prefer Local

"Prefer Local" will be an additional attribute that can be selected on a given server cluster. This attribute will instruct the selection logic to keep all client requests local to the same node as the client if possible. Thus, if the prefer local setting is enabled, and a request from a client can be serviced by a server clone that is local to the same machine as the client, the request will be routed to that local server clone. Prefer local still decrements the weight for each request, and if multiple server clones are local to the client the standard "weighted round-robin" algorithm will be used to route requests between all of

the local server clones. However, the "weight set" will be refreshed when all the local weights reach zero.

Affinity

It is possible to establish an affinity between a client and server clone such that all client requests are sent to the same server clone for the life of the affinity. Examples:

Transactional (strong) affinity:

The first request after the BEGIN of the transaction will determine which Application Server requests will be routed to. All subsequent requests from this client will be routed to the selected Application Server until the END of the transaction.

Applied (weak) affinity:

The first time a server group unaware client needs to use an object, the ORB (via an indirect IOR) will route the request to the LSD. The LSD will return the Application Server (via a direct IOR) to route request to. All subsequent requests from this client for that object will be routed to the selected Application Server. Thus, the standard weighted round-robin algorithm can be overridden by various affinities. In affinity situations, the server weight value is still decremented for each request. However, requests with affinity continue to be sent to the associated server clone even when the weight value reaches zero.

4.3 Clustering of Application Servers in the EJB Container

4.3.1 The WLM Client Runtime

Workload Management strives to balance work between a set of server clones in order to increase scalability and aid in failover capabilities. When a client makes a remote request, the WLM client runtime code determines which available clone should be used for the request. For the ND release this WLM client runtime will exists in (as described above):

- WLM code is embedded in the client, this is where the WLM runtime (delivered through the IIOP request/response sequence) will actually run in the client ORB process. We consider these to be "WLM-aware" clients.

- Clients without any WLM runtime code, non-IBM Orb based clients and WebSphere 390 clients. These clients will utilize the client routing with directives pattern and utilize the LSD for running the WLM client runtime. We consider these to be 'WLM-unaware" clients.

4.3.2 Bootstrapping

The J2EE 1.3 programming model uses ejb-ref and resource-ref definitions in the deployment information to provide a level of indirection between the application code doing a JNDI java:comp/env lookup and the actual object that is returned. The IBM extensions to the ejb-ref and resource-ref

provide a JNDI name which is used to lookup the actual object in the name space of the CosNaming server. At the time an application is assembled or deployed, the value for this JNDI name must be specified. The JNDI name that is specified may take one of the following forms:

- Unqualified simple name - this will work when the code doing the java:comp/env lookup is running in the same server.
- Qualified name containing the name of the server or cluster in which the target object exists - this will work from anywhere within the cell
- corbaname URL identifying host, port for the server and a name appropriate to the InitialContext position. This will work from anywhere in the cell, other cells and from non-WebSphere environments. This is the mechanism typically used by J2EE 1.3.

Bootstrapping into the name space and the positioning of the JNDI InitialContext are controlled by URLs (iiop, corbaloc and corbaname URLs) and other properties that are passed on the creation of the InitialContext. The view of the name space seen by the program is controlled by this. The user can position themselves in the name space at the cell root, the cell persistent root, the node root or the server root. When positioned at the cell root or cell persistent root, no matter which server in the cell the client bootstrapped into, the logical view of the name space is identical. However, if positioned at the node root or server root, the view of the name space that is visible is dependent upon the server that was bootstrapped into and the node on which it is running. The URLs used to identify the server in which to bootstrap can contain multiple host/port addresses. This eliminates single points of failure from bootstrapping, which is a problem in previous releases of WebSphere. Therefore, it is convenient to write multiple bootstrap address together, for automatic retries in case any one location is faulty.

Node agents are not the recommended bootstrap location in a clustered environment, because requests to node agents that are down will not be automatically be redirected. , Requests to application servers that are down are automtically redirected to the next server in the list hence it's recommended to use the application servers as bootstrap servers, as illustrated below.:

```
prop.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");

prop.put(Context.PROVIDER_URL, "corbaloc::host1:9810,:host1:9811,
:host2:9810, :host2:9811");
```

However, for WLM-unaware clients bootstrapping to the node agent is expected:

```
prop.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");

prop.put(Context.PROVIDER_URL, "corbaloc::host1:2809,:host2:2809");

Context initialContext = new InitialContext(prop);

try { java.lang.Object myHome =
initialContext.lookup("cell/clusters/MyCluster/MyEJB");

myHome = (myEJBHome) javax.rmi.PortableRemoteObject.narrow(myHome,
myEJBHome.class);
```

```
} catch (NamingException e) { }
```

The client automatically tries bootstrap servers on host1 and host2 until an initialContext object is returned (the order, host1 → host2, is not guaranteed).

4.3.3Failover

LSD Failover

If the failure occurs after the routing table is available on the client, the WLMClient code does not go to the LSD to determine what server to route WLM'able objects. The WLMClient code handles the routing decisions. If the failure occurs before the first client request which obtains the WLM information, then WLM depends on the LSD request to fail over to another LSD.

The WLM component is dependent on LSD failover because the first request for a client will require going to the LSD to obtain the first direct IOR. In order to support clients that do not have any WLM code running on them, z/OSclients, other ORB providers and C++ CORBA clients the WLM routing decisions will utilize the LSD.

- Performance of clients that run in WebSphere Java ORBs, with the WLM client code, will not be affected at all. The IORs are manipulated and routed by the WLM client code so indirect and direct IOR are handled the same.

- Clients without the WLM code would only route to the LSD on the first request of each object. The direct IOR will be used by the client until the client either drops the reference to the object or the server becomes unavailable

Application server fail over

If the failure is on the first initial request where the routing table information is not available, a COMM_FAILURE exception will be returned and the ORB will recognize that it has an indirect IOR available and so re-send the request to the LSD to determine another server to route to. If the failure is after the client has the routing table information, the WLM Client will handle the COMM_FAILURE. The server will be removed from the list of selectable servers and use the routing algorithm to select a different server to route the request to.

Consider the following sequence of a client making a request to an application server in the EJB container:

1. No server cluster and routing information are available initially in the WLM client's runtime process, for the initial client request. The request is therefore directed to LSD that is hosted on a node agent to obtain routing information. If LSD fails, the request will redirect to an alternative LSD (i.e. LSD failover, see below). If this was not the first request, the WLM client would already have routing information for WLM-aware clients. However, WLM-unaware clients, the LSD will always route requests to available servers. For future requests from the client, if there is a mismatch of the WLM client's routing information from what is on a server's, new routing information (as service context) will be added to the response.

2. After getting the InitialContext, a client does a lookup the EJB's home object (an indirect IOR to the home object). If a failure occurs at this time, the WLM code will transparently redirect this request to another server in the group that is capable of obtaining the Bean's home object.

3. Server(s) become unusable during the life-cycle of the request
   i. If the request has strong affinity, there cannot be request failover. The request will fail if the original server becomes unavailable. The client must perform recovery logic, and resubmit the request.
   ii. If the request is to an overloaded server, its unresponsiveness makes it seem as though the server is stopped, which may lead to a timeout. In these circumstances it may be helpful to: change server weight, tune orb and pool properties such as **com.ibm.CORBA.requestTimeout**, **com.ibm.CORBA.RequestRetriesCount, com.ibm.CORBA.requestRetrisDelay, com.ibm.CORBA.locateRequestTimeout.** These can be command line properties, or changed in the admin console.
   iii. If the **com.ibm.ejs.wlm.MaxCommFailures** threshold has been reached for a clone, it is marked unusable. By default, the MaxCommFailures threshold is 0, such that after the first failure the clone is marked unusable. This property can be modified by specifying -Dcom.ibm.ejs.wlm.MaxCommFailures=<number>, as a command line argument when launching a client.
   iv. If a machine becomes unreachable (network and/or individual machine errors), before a connection to a server has been established, the operating system TCP/IP "keep-alive" timeout dominates the behavior of the system's response to a request, since a client will wait the OS-specific "keep-alive" timeout, before a failure is detected. This value can be modified, but as described in Chapter 2, only with caution. If a connection is already established to a server, **com.ibm.CORBA.requestTimeout** dominates (the default value is 180 seconds), and a client will wait this length of time before a failure is detected. The default value should only be modified if an application is experiencing timeouts repeatedly, and great care must be taken to tune it properly. If the value is set too high, the failover will be very slow, and set too low, requests will time out before the server has a chance to respond. The two most critical factors affecting the choice of a timeout value are the amount of time to process a request, and the network latency between the client and server. The time to process a request in turn depends on the application, and the load on the server. The network latency depends on the location of the client, for example those running within the same LAN as a server may use a smaller timeout value to provide faster failover. If the client is a process inside of a WebSphere Application Server (i.e. the client is a servlet), this property can be modified by editing the request timeout field on the Object Request Broker property sheet. If the client is a java client, the property can be specified as a runtime option on the Java command line, for example:
   ```
   java -Dcom.ibm.CORBA.requestTimeout=<seconds> MyClient
   ```
   v. A failed server is marked unusable, and a JMX notification is sent. The routing table is updated. WLM-aware clients are updated during request/response flows. Future requests will not route requests to this clone until new server group information is received (i.e. Server process restarted), or until the expiration of the **com.ibm.ejs.wlm.unusable.interval**. This property is set in seconds. The default value is 300 seconds. This property can be set by specifying -Dcom.ibm.ejs.wlm.unusable.interval=<seconds> on the command line arguments for the client process.

vi. when a request results in an org.omg.CORBA.COMM_FAILURE or org.omg.CORBA.NO_RESPONSE, the return value of COMPLETE_STATUS determines whether a request can be transparently redirected to another server. In the case of COMPLETED_NO the request can be rerouted. If the completed status is COMPLETED_YES, no failover is required - the request was successful, but some communication error was encountered during the marshaling of the response. In the case of COMPLETED_MAYBE, WLM cannot verify whether the request was completed successfully, and cannot redirect the request. For example, consider a transaction that must be "at most once". In that case had WLM redirected the request, it is possible the request would be serviced twice. The programming model is for the client to receive this exception, and to have logic in place to decide whether or not to retry the request.

vii. If all servers are unavailable, the request will result in a org.omg.CORBA.NO_IMPLEMENT. At this point, either the network is down, or some other error has caused the entire cluster to be unreachable.

4.4 Fault Isolation and Data Integrity

There are many data and process interactions between, and outside, the EJB container(s). One important aspect of maintaining a highly available enterprise is that a failure for one client in the EJB container should not impact another client. Therefore the data used by various clones for an application must be kept consistent.

4.4.1 EJB Caching

An entity bean represents persistent data. It is common for a client, or several independent clients, to make a succession of requests targeted at the same bean instance. Therefore, the state of an entity bean must be kept consistent across multiple sequential or concurrent requests. Between transactions, the state of the entity bean can be cached. ND supports option A, and option B, and option C caching.

1. **Option A caching:** The EJB container caches an instance of the EJB in a ready state between transactions. The data in the EJB is also cached, not requiring the data to be reloaded from the data store at the start of the next transaction. The entity bean must have exclusive access to the underlying database. Multiple clones, using the same entity bean, could potentially modify the entity bean data asynchronously. In the ND release there are no built mechanisms to ensure data integrity for asynchronous data access.
2. **Option B caching**: The EJB container caches an instance of the EJB in a ready state between transactions, but invalidates the state information for the EJB. At the start of the next transaction, the data will be reloaded from the database.
3. **Option C caching (default):** No ready instances are cached by the container, they are returned to the pool of available instances after the transaction has completed.

The recommended caching option in ND is either option B or option C (the default), since these options ensure that the entity bean is always reloaded from the database at the start of each transaction, ensuring data integrity

In WAS-ND, the caching options are configured using the Application Assembly Tool (AAT), and are set by the combination of options selected in the drop down menus **Activate at** and **Load at**. WAS-ND also supports optimistic concurrent control, where cached data is checked for collisions during the commit. Loading data from the database may not be required at the beginning of a transaction if there is a design in place to stamp the cached entity bean.

The table below shows the values that represent the three caching options

Option	Activate Bean Instance at	Call ejbLoad() on bean instance at
A	Once	Once at activation
B	Once	Start of transaction
C	At start of transaction	Start of transaction

4.4.2 EJB Types

When a client accesses an EJB, it *may* be routed to one of a number of instances on any number of servers. However, not all EJB references can be utilized this way. The table below shows the types of EJBs that can be "workload managed" (i.e., requests can be routed, and redirected if need be).

EJB Type		Able to be Workload Managed
Entity Bean, Commit time caching Option A	Home	Yes
	CMP	No
	BMP	No
Entity Bean, Commit time caching Options B and C	Home	Yes
	CMP	Yes
	BMP	Yes
Message-Driven Bean	MDB	Yes
Session Bean	Home	Yes
	Stateless	Yes
	Stateful	No

As indicated in the table, the only EJBs that can not be workload managed are stateful session beans and entity beans with option A caching. A stateful session bean is used to capture state information that must be shared across multiple client requests that are part of a logical sequence of operations. To ensure consistency in the logic flow of the application, a client must always return to the same instance of the stateful session bean, rather than having multiple requests workload managed across a group of available stateful session beans.

Because of this restriction, stateful session bean instances are not considered failover-tolerant. If the process running the stateful session bean fails, the state information maintained in that bean is

unrecoverable. Otherwise the application must somehow store and reconstruct the required state data. An alternative implementation is storing state in an entity EJB**.**

4.5 Resource availability

EJB failover relies on data and resource availability (databases, JMS resources, LDAP). If the data and resources are unavailable, a request to the EJB container will fail. Beginning with Chapter 7 we discuss methods and best practices for helping to ensure resources beyond the ND administrative domain are available.

Chapter 5 - Deployment Manager and Node Agent High Availability Best Practices

WAS-ND provides a distributed administrative domain (the administrative cell in Figure 1.1). Deployment managers are administrative agents that provide a centralized management view for all nodes in a cell, as well as management of clusters and workload balancing of application servers across one or several nodes. A deployment manager hosts the administrative console. A deployment manager provides a single, central point of administrative control for all elements of the entire WebSphere Application Server distributed cell. A node agent manages all WebSphere Application Server servers on a node. The node agent represents the node in the management cell.

All node agent, and the deployment manager servers use XML files as their configuration repository, stored on the local network file system. The "master" repository data are stored on the deployment manager's node, and are replicated to each node in the cell. The deployment manager is responsible for maintaining the synchronization of the configuration files across the administrative domain, where the process is unidirectional (deployment manager ⇨ nodes), with the strict scope, in order to maintain the repository integrity. Hence, a change made (solely) on a node agent XML, and not the master copy on the deployment manager, will result in that change being overwritten by the deployment manager during the next synchronization. Changes to the XML are best made using the master repository on the deployment manager, allowing these changes to be pushed to the node agents.

The node agents provides runtime support for the Location Server Daemon (LSD), file transfer and synchronization, JMX, distributed logging, naming, performance monitoring services (PMI), EJB workload management configuration. The deployment manager provides runtime support for setting server weights, file transfer and synchronization, configuration management (either through the administrative console or the wsadmin scripting interface), JMX, distributed logging, naming, enabling security, PMI services, and EJB workload management master configuration.

The loss of the deployment manager, or node agent processes, reduces the availability of the administrative features of the network deployment cell. The next two sections describe how to help ensure these processes are available.

5.1 Enhanced Deployment manager High Availability

The deployment manager server can be a single point of failure in ND, however the loss of the deployment manager process does not have a catastrophic affect on the WebSphere domain's ability to route requests, and provide failover, as the configuration data are replicated to every node agent.

5.1.1 Add the Deployment manager to OS Daemon Service

On the windows platform, during the installation, check the box allowing the process to run as a windows daemon service. One can also use:

<install_root>\WebSphere\DeploymentManager\bin\WASService.exe

to add the Deployment manager server to the windows daemon service.

For other platforms, you can do so by adding an entry in the inittab file:

was:2:once:/usr/WebSphere/Deployment manager/bin/rc.was >dev/console 2>&1

Once you have added the Deployment manager server as OS daemon service, if the deployment manager server process terminated abnormally, the operating system will restart it. This will minimize the impact of process failure.

5.2 Enhanced Node agent High Availability

5.2.1Node Agent Clustering

As the first line of protection, have more than one node agent in a cell, to ensure LSD failover will occur in the event a node agent process is lost.

Expecting an application server to run when the machine is disconnected from the network requires that the loopback port address (127.0.0.1) be configured as an alias endpoint for the machine.

The reason behind this requirement is due to the fact that the node agent normally pings all of the application servers on the node. When the node agent detects that the application server is not available then the node agent stops the application server then tries to restart the application server. This goes on for 10 minutes, at which time the node agent no longer tries to restart the application server. The ping from the node agent to the application server occurs using the internet protocol (IP). When the machine is disconnected from the network the communication then occurs on the loopback port. This loopback port is only considered to a valid port for the application server when the loopback port address (127.0.0.1) is configured as an alias for the machine.
The behavior when the loopback port is not configured as an alias is that the application server is stopped (because the node agent cannot reach the application server) and the node agent will not be able to restart it until the machine is again connected to the network. When the time required to reconnect to the network is less than ten minutes then the application servers will be started on the next retry of the node agent. However, when the reconnection time exceeds the ten minute interval then an explicit start to the application servers will need to be performed.

5.2.2 Add the Node agent as OS daemon

For the Windows platform, you can add Node agent as OS daemon by using the WASService.exe:

WASService -add <Node agent> -serverName nodeagent -restart true

For other platforms, you can do so by adding an entry in the inittab file:

was:2:once:/usr/WebSphere/AppServer/bin/rc.was >dev/console 2>&1

When a node agent process is killed, or ends abruptly, the operating system will restart the node agent automatically.

5.3 Other Solutions

Running these processes as OS daemon services will not prevent permanent loss of the deployment manager, or node agent processes due to network, disk, operating system, JRE, and host machine failures. Therefore, we explore the use of non-WebSphere software, and hardware configurations, to help maintain a highly available deployment manager in Chapter 6, and for node agents in Chapter 8.

Chapter 6 - Techniques for Clustering the Deployment Manager

6.1 Introduction

As noted in the previous chapter, the deployment manager server is a single point of failure in ND. This chapter describes how to make the deployment manager highly available with clustering software and hardware.

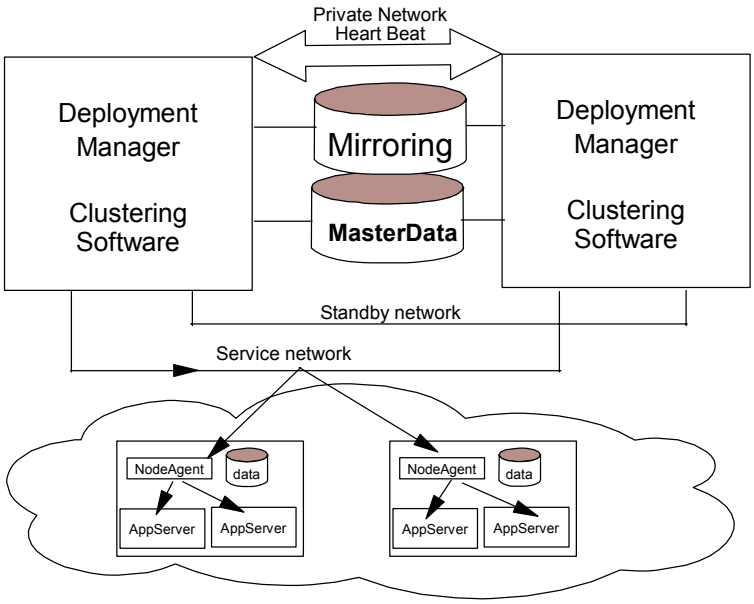


Figure 6.1: Clustered Deployment Manager

As shown in Figure 6.1, a software and hardware system includes high availability clustering software (like High Availability Cluster Multiprocessing (HACMP), MSCS), shared disk array subsystem, two host machines, serial private network, public primary and backup IP networks in different LANs (that connect to other WebSphere nodes and the resources employed by WebSphere). The deployment manager is installed into the shared disk and is configured with a high availability resource group. Consider the scenario where a machine in a cluster has had an unrecoverable error. The loss of this machine can lead to a decrease in the capacity of the cluster to service requests, or could impact the ability to perform administrative tasks. Utilization of an HACMP cluster allows for another machine to be brought on-line, without an system administrator manually intervening. As has been shown, and s will be shown in Chapters 6-8, if any application server, node agent, or deployment manager process fails unexpectedly, there features within, and outside, the WebSphere domain which will attempt to restart these processes. However, if the restart fails, a backup process can become available on a “hot standby” node/machine. In addition to the details provided below, and in the following Chapters, refer to Appendix D for HACMP setup instructions. Note, the file based config repository is not designed for

access from concurrent active deployment managers. The difficulty arises in that there is locking in-memory, such that there is no cross-process coordination for concurrency control.

6.2 Deployment Manager Failover

In case any failure occurs in the Deployment manager process, operation system, JRE, network, disk subsystem, and host machine, the heartbeat system will detect these failures and initiate a failover by:

- Stopping the Deployment manager server process in the failed host
- Release the shared disk and other resources from this node
- Remove the service IP address from the failed node
- Mount the shared disk to the other healthy node
- Add the same service IP address to the adopted node
- Start the Deployment manager server

The order between disk and IP operations can be changed in the configuration.

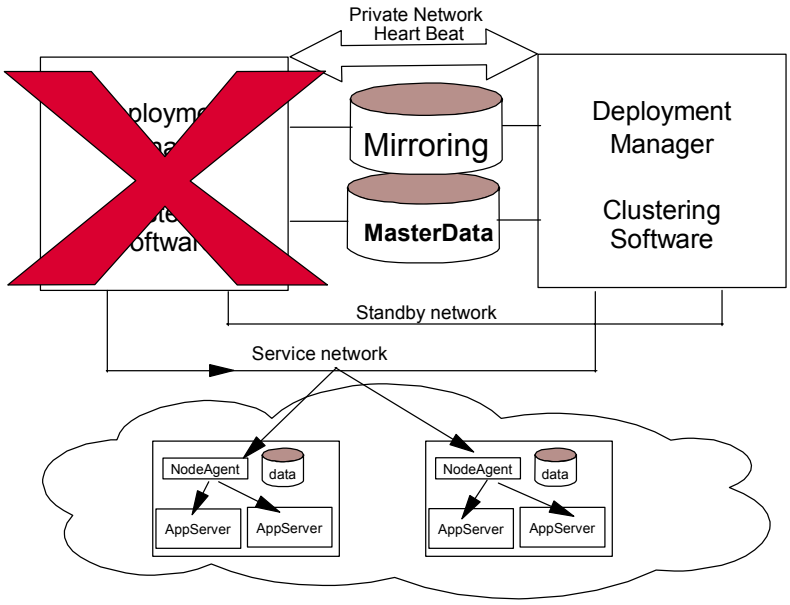


Figure 6.2 Failover of the Deployment manager with the assistance of HACMP

Note, when this configuration was tested, it was determined that there were timing issues when stopping the deployment manager (this resulted in a HACMP error code of rc207).

The deployment manager is stopped via a JMX interface, and due to the asynchronous nature JMX messaging, the deployment manager process was not ended before the attempted release of shared disk resources on that node. In this situation, the disk resource is still owned and held by deployment manager process. A time delay was inserted such that the stop operation could finished before disk resource release could be attempted For example, here is a snippet of a shell script that might be used:

```
#/usr/bin/ksh
/HA/WebSphere/Deployment manager/bin/stopManager.sh
sleep X
```

Where X should range from 30 to 180 seconds. Adjust the sleep time according to the system employed. The correct value will take some fine tuning such that the rc207 error is avoided, but not so long that deployment manager failover is too lengthy.

6.3 The Deployment Manager Failover System Configuration

The following are the complete HACMP configuration files for configuring a highly available deployment manager.

```
Cluster Description of Cluster hacluster
Cluster ID: 99
There were 2 networks defined : haserial, tr157
There are 2 nodes in this cluster.

NODE hacmp1:
    This node has 2 service interface(s):

    Service Interface hacmp1-tty:
        IP address:     /dev/tty0
        Hardware Address:
        Network:        haserial
        Attribute:      serial
        Aliased Address?:       False

    Service Interface hacmp1-tty has no boot interfaces.
    Service Interface hacmp1-tty has no standby interfaces.


    Service Interface hacmp1s:
        IP address:     9.5.157.68
        Hardware Address:
        Network:        tr157
        Attribute:      public
        Aliased Address?:       False
    Service Interface hacmp1s has 1 boot interfaces.
        Boot (Alternate Service) Interface 1: hacmp1b
        IP address:     9.5.157.207
        Network:        tr157
        Attribute:      public
    Service Interface hacmp1s has 1 standby interfaces.
        Standby Interface 1: hacmp1sb
        IP address:     10.10.0.30
        Network:        tr157
```

Attribute: public

NODE hacmp2:
This node has 2 service interface(s):

Service Interface hacmp2-tty:
IP address: /dev/tty0
Hardware Address:
Network: haserial
Attribute: serial
Aliased Address?: False

Service Interface hacmp2-tty has no boot interfaces.
Service Interface hacmp2-tty has no standby interfaces.

Service Interface hacmp2s:
IP address: 9.5.157.196
Hardware Address:
Network: tr157
Attribute: public
Aliased Address?: False
Service Interface hacmp1s has 1 boot interfaces.
Boot (Alternate Service) Interface 1: hacmp1b
IP address: 9.5.157.207
Network: tr157
Attribute: public
Service Interface hacmp1s has 1 standby interfaces.
Standby Interface 1: hacmp1sb
IP address: 10.10.0.30
Network: tr157
Attribute: public

NODE hacmp2:
This node has 2 service interface(s):

Service Interface hacmp2-tty:
IP address: /dev/tty0
Hardware Address:
Network: haserial
Attribute: serial
Aliased Address?: False

Service Interface hacmp2-tty has no boot interfaces.
Service Interface hacmp2-tty has no standby interfaces.

Service Interface hacmp2s:
IP address: 9.5.157.196
Hardware Address:
Network: tr157
Attribute: public
Aliased Address?: False
Service Interface hacmp2-tty has no boot interfaces.
Service Interface hacmp2-tty has no standby interfaces.

Service Interface hacmp2s:
IP address: 9.5.157.196
Hardware Address:
Network: tr157
Attribute: public
Aliased Address?: False

Service Interface hacmp2s has 1 boot interfaces.
Boot (Alternate Service) Interface 1: hacmp2b
IP address: 9.5.157.206
Network: tr157
Attribute: public
Service Interface hacmp2s has 1 standby interfaces.
Standby Interface 1: hacmp2sb
IP address: 10.10.0.40
Network: tr157
Attribute: public

Breakdown of network connections:

Connections to network haserial
Node hacmp1 is connected to network haserial by these interfaces:
hacmp1-tty

Node hacmp2 is connected to network haserial by these interfaces:
hacmp2-tty

Connections to network tr157
Node hacmp1 is connected to network tr157 by these interfaces:
hacmp1b
hacmp1s
hacmp1sb

Node hacmp2 is connected to network tr157 by these interfaces:
hacmp2b
hacmp2s
hacmp2sb

Resource Group Name ND
Node Relationship cascading
Participating Node Name(s) hacmp1 hacmp2
Dynamic Node Priority
Service IP Label hacmp1s
Filesystems /ha
Filesystems Consistency Check fsck
Filesystems Recovery Method sequential
Filesystems/Directories to be exported
Filesystems to be NFS mounted
Network For NFS Mount
Volume Groups havg
Concurrent Volume Groups
Disks
Connections Services
Fast Connect Services
Shared Tape Resources
Application Servers CellManager
Highly Available Communication Links
Miscellaneous Data
Auto Discover/Import of Volume Groups true
Inactive Takeover false
Cascading Without Fallback true
9333 Disk Fencing false
SSA Disk Fencing false
Filesystems mounted before IP configured true
Run Time Parameters:

Node Name hacmp1
Debug Level high
Host uses NIS or Name Server false
Format for hacmp.out Standard

Node Name hacmp2
Debug Level high
Host uses NIS or Name Server false
Format for hacmp.out Standard

Chapter 7 - JMS/MQ Server Clusters

7.1 Introduction

WebSphere ND V5.0 includes an embedded JMS server, and supports all messaging functions as described in the J2EE 1.3 specification.

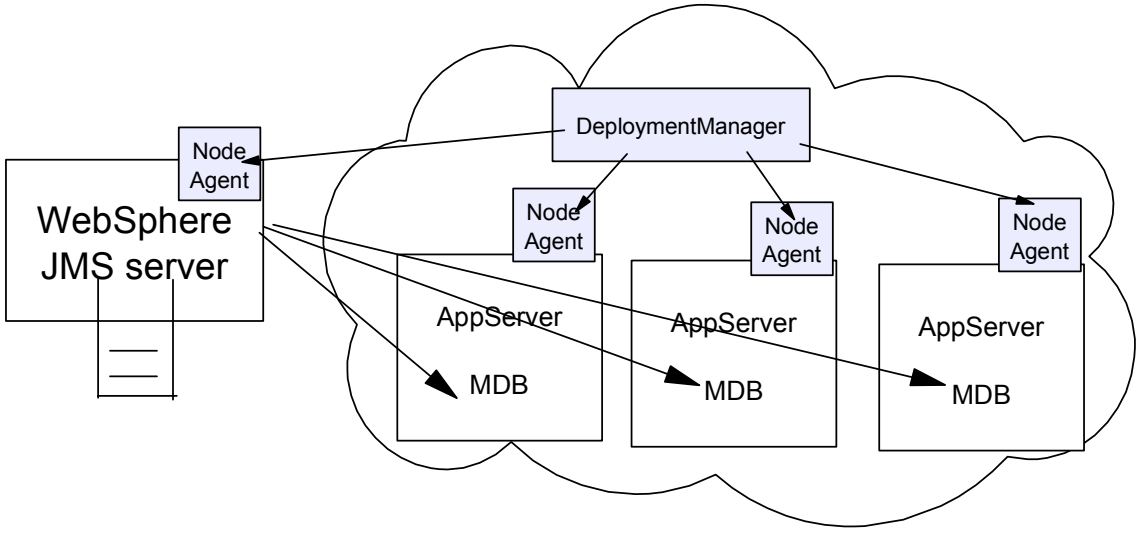


Figure 7.1 The embedded JMS in an WebSphere Network Deployment Environment

Figure 7.1 shows a non-clustered JMS server configuration. When the JMS (queue manager) server fails, all application servers loose their messaging service, the JMS server can be a single point of failure. This chapter documents 4 techniques to alleviate this situation: 1) software-hardware clustering for the JMS server, 2) software-hardware clustering for WebSphere MQSeries, 3) use of the WebSphere MQSeries queue clustering, and 4) combined software-hardware clustering for the MQSeries built-in queue clustering.

7.2 The Embedded JMS Server High Availability with HACMP Clustering

As shown in Figure 7.2, a software, network, and hardware system can be configured with the embedded JMS server, to build a high availability system within WebSphere domain. Such a system can tolerate any failure from the JMS server process, disk, operating system, host, or network. Therefore, such a system reduces the effect of the loss of a JMS server. Additionally, this system provides a transparent and single image view to application clients that need to put or retrieve messages. Note that the “rc207”error described in Chapter 6 is also an issue in this configuration

Customers should be aware that the embedded JMS server provides limited subset of functions contained within MQSeries. For example, the embedded JMS server doesn't support QMgr to QMgr channel, message flows and message transformation. It is not possible to communicate outside of the WebSphere domain with the JMS server, nor does it support database persistence.

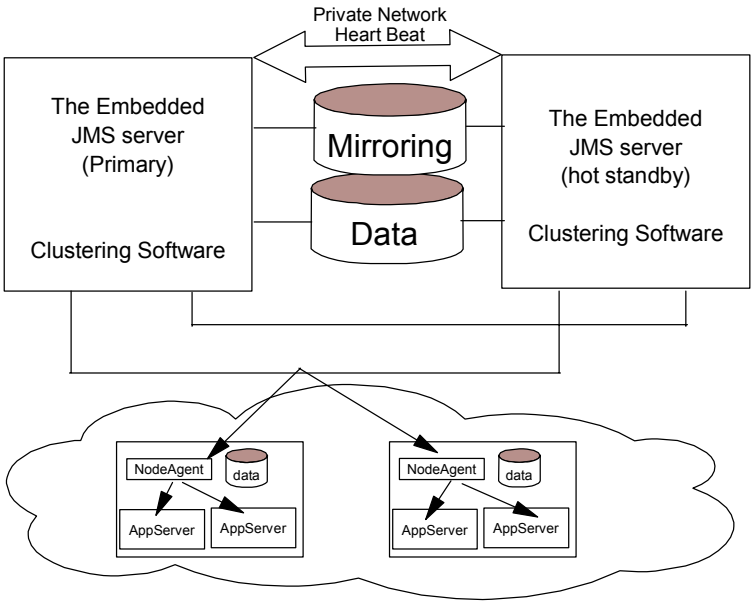


Figure 7.2: Clustered JMS Server with Clustering Software

7.3 MQSeries High Availability with HACMP Clustering

Figure 7.3 shows a system that uses (HACMP) hardware clustering for a highly available WebSphere MQSeries Message Oriented Middleware system. If the queue manager process or network or primary host fails, the queue manager, as well as the queue data and log, will fail over to the hot standby machine with the same service IP address.

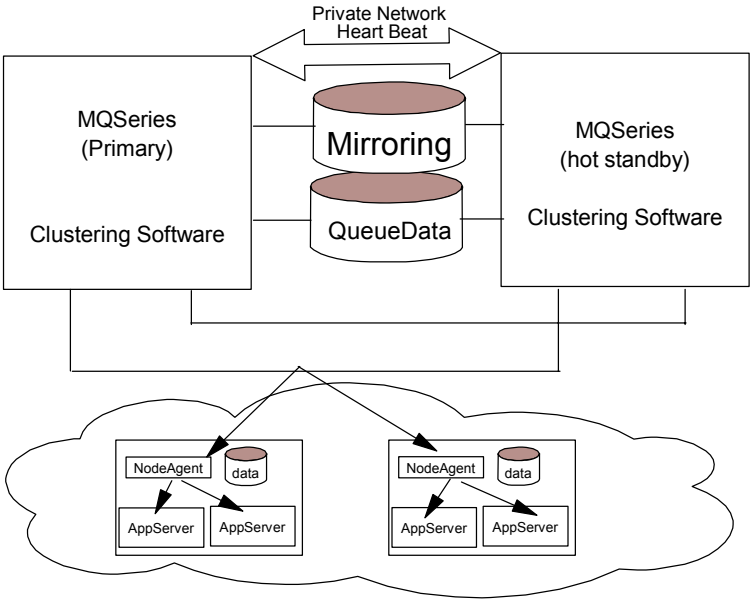


Figure 7.3 Clustered WebSphere MQSeries

7.4 MQSeries Built-in Queue Clustering

WebSphere MQSeries Queue Cluster reduces administration effort and provides load balancing of messages across instances of cluster queues. Therefore, Queue clustering offers "built-in" high availability. Such a cluster is depicted in Figure 7.4.

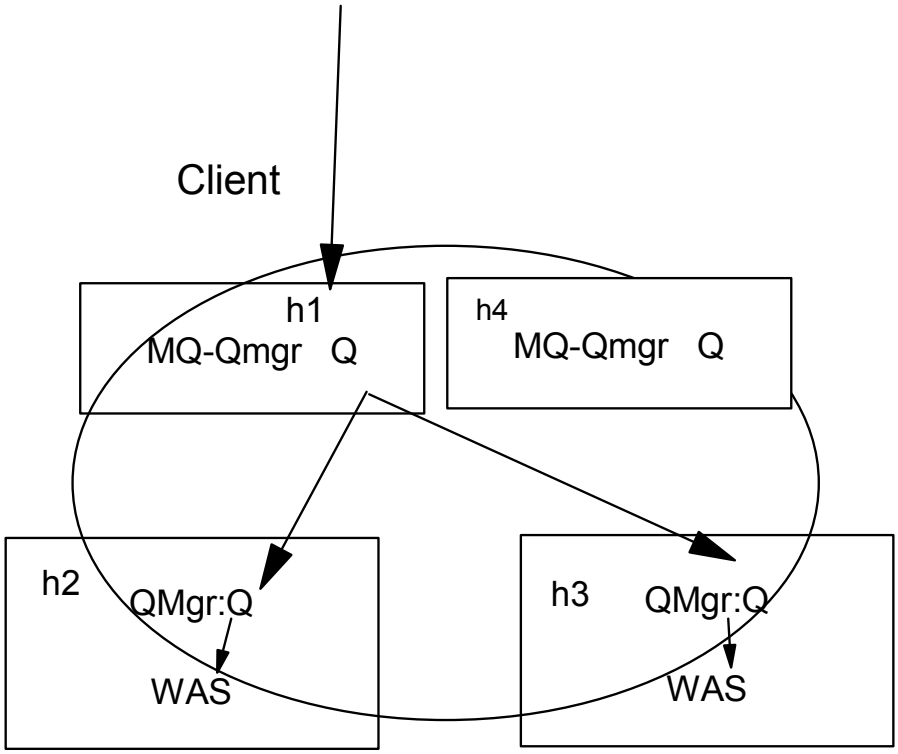


Figure 7.4 WebSphere MQSeries built-in Queue Cluster

If the queue manager on host "h2" fails, there will be a functioning queue manager on host "h3". Therefore an application server on host h3 will be capable of processing its local queue. If all messages are produced locally and consumed locally, this is a perfect high availability solution because of messaging client redundancy. These were the conditions tested using the Trade3 and Big3 applications.

However, locally produced and locally consumed messages are atypical in production e-business sites, where messages are generally transferred long distances (i.e. credit card verification). This reduces the value of asynchronous messaging.

There are two ways to fulfill messages covering large distances:

1) clients connect synchronously to servlet and session EJBs, using the beans to place messages locally to be consumed by message driven beans (MDBs). Since servlet and EJB requests are workload managed, a clustered queue approach is an excellent high availability solution.

2) clients send messages asynchronously to the queue to host "h1" (see Figure 7.4), and an MDB processes the messages from a local queue (on, h2 or h3), that is clustered with the queue on h1. If, in this case, the queue manager on host h1 fails clients cannot push messages into the queue. Messages cannot be distributed to the queues on hosts h2 and h3. Another queue manager on h4, however, allows clients to connect to either of queue manager. However, clients do have not "server transparency", since h1 and h4 will have different IP addresses. In addition, the WebSphere MQSeries

queue cluster does not provide automatic detection of queue manager failures, and thus trigger the failover of one queue manager to another.
An MQSeries queue cluster may cross a wide range of geographical boundaries. For an MQSeries cluster, at least two repository queue managers are needed for high availability. At any site or access point, at least 2 producers/publishers or 2 consumers/subscribers are needed for high availability. There must be ways to increase either client chain redundancy (such as single client-> Servelt/EJB WLM->multiple copies of the clients-> put/consume messages into/from the clustered queue, else place the addresses of multiple queue managers directly into the client code) or provide a single image of the queue manager for single clients, using a hardware-based high availability queue manager.

The best approach is to combine the WebSphere MQSeries build-in queue cluster and HACMP (or other kind of clustering software).

7.5 Combined Approach: MQSeries Built-in Queue Clusters and HACMP Clustering

Figure 7.5 depicts a highly available system that contains both the WebSphere MQSeries built-in queue cluster and HACMP based clustering. This configuration not only provides a higher level of availability, but also scalability.

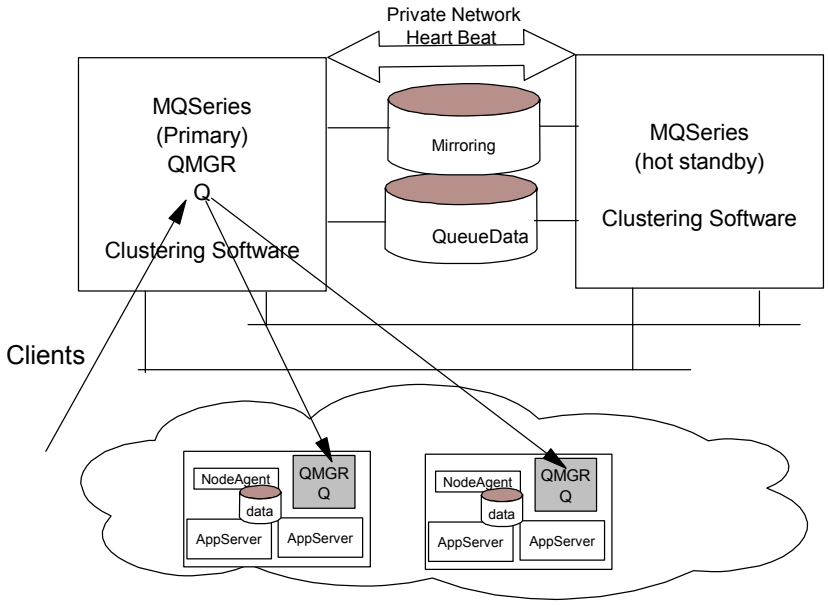


Figure 7.5 The combined MQSeries Queue Cluster and HACMP clustering

Chapter 8 - WebSphere Application Servers on clustered platforms

8.1 Introduction

WebSphere ND has built-in workload management mechanism, which provide excellent scalability and availability. This chapter discusses the combining WebSphere's Workload Management and HACMP clusters to further enhance the high availability of an e-business. This two-pronged approach helps resolve 1) two phase transaction and transaction log failover issues, and 2) automatic fault detection by employing private heartbeat, hot server standby and automatic server resource failover.

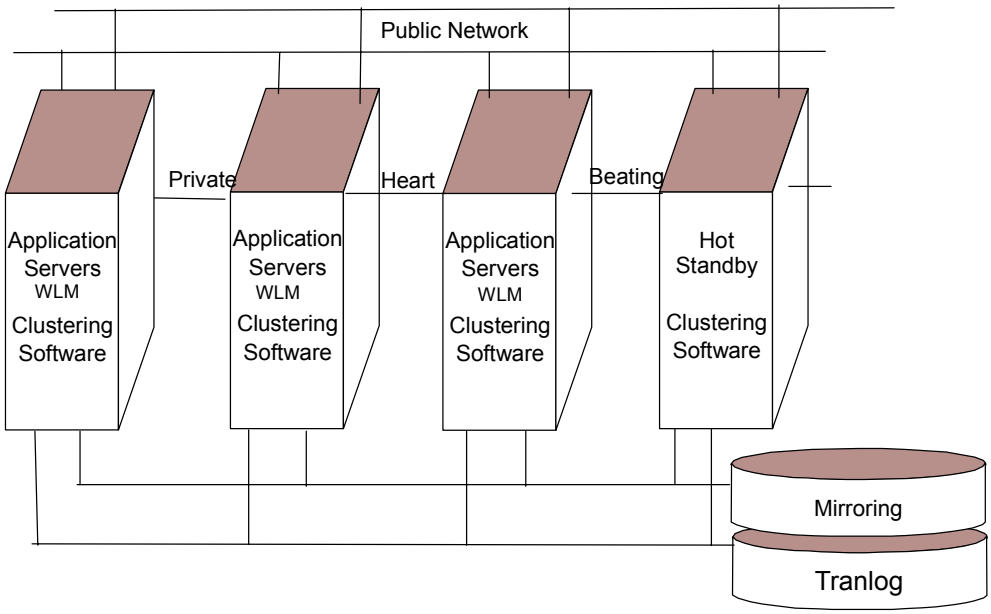


Figure 8.1: WebSphere Application Server topology for the XA transaction failover

8.2 Failover Process

If a network disk fails or applications, logs, or data cannot be retrieved locally, a process can be available automatically on the hot standby node, where these resources have been replicated, and are immediately available. The maximum loss of any given resource depends on the frequency of replication of information. Further, if a node fails due to CPU load, an operating system or power failure, or JRE, the process can be available on a hot standby node. Finally, if the network fails, a backup network connection can be made available. If the backup network is also experiencing difficulties, a process can become available on a hot standby node on a different network.

8.3 Advantages and Disadvantages

WebSphere's workload management provides excellent failover and scalability support for almost all applications. However, with the addition of HACMP, there are some added advantages:

- Provides 2-phase commit transaction and transaction log automatic failover. WebSphere application servers act as coordinators of distributed transactions and write the transaction logs in the <Install_Root>/WebSphere/AppServer/tranlog directory. If an application server fails after the start of the 2-phase commit, the transaction manager will wait for the application to be restarted to complete the transaction. If the node hosting the application server fails, the transaction cannot be completed, and the corresponding database rows are locked until the transaction can be completed. With IP takeover, a new application server is restarted, with access to the tranlog, and the transaction can be completed automatically.
- Provides automatic application server failover in the event a machine has an unrecoverable error such that the client will not perceive a performance impact during failover. Without an HACMP cluster, the machine that encounters an error (and hosting an application server that is servicing requests) needs to be manually rebooted to recover the server process.

Although such configuration has advantages mentioned above, it is highly complex configuration.

Chapter 9 - WebSphere Database Failover

9.1 Introduction

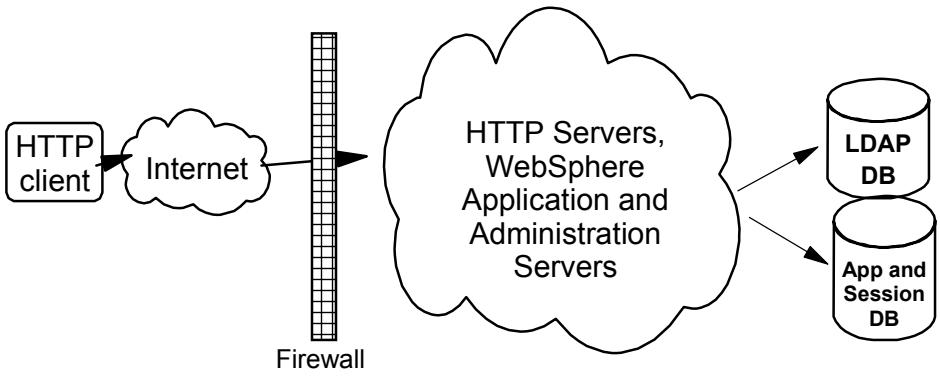


Figure 9.1: Single WebSphere Database Topology

As noted in the introduction, there are two types of availability: data and process. High availability for database servers encompasses both. Data availability is the resonsibility of the database manager, while process availability is the responsibility of a choosen clustering technology (such as HACMP) employed to make the database server process highly available. The specifics of configuring your database server for high availability are addressed in chapter 10. However, it is important to note that even in a WebSphere environment, which employs one of the HA mechanisms discussed in chapter 10, there is still an interruption in service while a database is switched from a failed server to an available server. This chapter will focus on application code (i.e. programming model) implications for database high availability.

WebSphere allows (client) application code to recover to recover from the interruption and subsequent restoration of a database service, through the use of IBM extensions to the JDBC 2.0 API. These extensions allow clients connecting to, or applications running within, an application server the capability to: 1) reconnect to a database server, if it has to recovered from a failure, or 2) recognize that the database is not responding to requests.

The first JDBC extension allows client code to catch a com.ibm.websphere.ce.cm.StaleConnectionException. This exception maps multiple SQL return codes, which indicate a database server failure/outage, to a single exception. Catching this excpetion allows application code the ability to recognize the need to attempt a reconnection to a database, after service is restored, and is a part of the WebSphere programming model fpr application components (Servlets, JSPs, and EJBs).

The second extension allows for a com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException to be caught by application code. This exception occurs when the connection timeout parameter, for a datasource, is exceeded. The timeout parameter specifies how long an application will wait to obtain a connection from the pool, if the maximum number of connections is reached and all of the connections are in use. When exception is received, an application should resubmit a request to obtain a free connection.

Sample application code is provided below.

9.2 Application Databases

9.2.1StaleConnectionException

From an application code perspective stale connections are connections which are no longer usable, for example, if the database server is shut down or the network is experiencing problems. When a StaleConnectionException is detected in the application server runtime, the connection pool is flushed and repopulated.

Explicitly catching a "StaleConnectionException" is not required, because applications already catch java.sql.SQLException and StaleConnectionException extends SQLException. However, specifically coding to catch a StaleConnectionException can trigger an application to perform additional (custom) recovery steps. However, in many instances application programmers that develop on WebSphere need not (normally) need custom recovery, as there will be an automatic reconnection with the server runtime. The use of StaleConnectionException should be limited to applications that needs an extra level of transparent recovery, in addition to that provided by WebSphere. Custom recovery code can be very difficult to develop, especially in complex multiple resources configurations or logic flows. The following discussion is aimed at application programers that feel the need for this additional complexity is warrented.

Since connections are pooled, the most common scenario to encounter a StaleConnectionException is the first time database communication is attempted with a connection retrieved from the pool. Stale connections occur less frequently when applications retrieve new connections to a databse more frequently, since the connection pool will be flushed and repopulated more frequently (assuming a stale connection is encountered).

Generally, when a StaleConnectionException is caught, the transaction in which the connection was involved needs to be rolled back and a new transaction begun with a new connection. Details on how to do this can be broken down into three categories:

- A connection in auto-commit mode
- A connection not in auto-commit and transaction begun in the same method as database access
- A connection not in auto-commit and transaction begun in a different method from database access

9.2.1.1Connections in auto-commit mode

By default, any connection obtained from a one-phase datasource (implementing

javax.sql.ConnectionPoolDataSource) is in auto-commit mode, when there is no scoping transaction. When in auto-commit mode, each database action (statement) is executed and committed in a single database transaction. Servlets often use connections in auto-commit, because transaction semantics are not necessary. Enterprise applications do not usually use auto-commit connections, because they frequently require multiple statements to be executed, and serially committing each would be quite cumbersome. Auto-commit can be explicitly disabled by calling setAutoCommit() on a Connection object. When a StaleConnectionException is caught from a connection in auto-commit mode, recovery is a simple matter of closing all of the associated JDBC resources and retrying the operation with a new connection. However, in some cases the cause of a database outage may be transient. In these cases, adding a delay in the retry logic may allow a database service to be restored. The number of retries as well as any delay should be small, so as to keep a client from waiting an inordinate amount of time.

Here is some sample code:

```
public void myConnPool() throws java.rmi.RemoteException
{
	// retry indicates whether to retry or not
	// numOfRetries states how many retries have been attempted
	boolean retry = false;
	int numOfRetries = 0;
	java.sql.Connection conn = null;
	java.sql.Statement stmt = null;
	do {
		try {
			//Assumes that a datasource has already been obtained from JNDI
			conn = ds.getConnection();
			stmt = conn.createStatement();
			stmt.execute("INSERT INTO ORG VALUES (10, 'Pacific', '270', 'Western', 'Seattle')");
			retry = false;
		} catch(com.ibm.websphere.ce.cm.StaleConnectionException sce)
		{
			//if a StaleConnectionException is caught rollback and retry the action
			if (numOfRetries < 2) {
				retry = true;
				numOfRetries++;
				// add an optional pause
				Thread.sleep(10000) ;
			} else {
				retry = false;
			}
		} catch (java.sql.SQLException sqle) {
			//deal with other database exception
		} finally {
			//always cleanup JDBC resources
			try {
				if(stmt != null) stmt.close();
			} catch (java.sql.SQLException sqle) {
				//usually can ignore
			}
			try {
				if(conn != null) conn.close();
			} catch (java.sql.SQLException sqle) {
				//usually can ignore
			}
		}
```

```
        } while (retry) ;
}
```

9.2.1.2Connections with Auto-Commit Disabled

When a connection has auto-commit disabled, multiple database statements can be executed in the same transaction. Since transaction tend to use a significant number of resources, fewer transactions result in better performance. Therefore, if an application requires executing mulitple statements, best practice is to disable auto-commit, and use transactions to group a number of statements into one unit of work. Keep in mind that if a transactions very large numbers of statements, the database can experience memory resource issues.

9.2.1.2.1Transactions started in the same method

If a transaction is begun in the same method as the database access, recovery is straightforward and similar to the case of using a connection in auto-commit mode. When a StaleConnectionException is caught, the transaction is rolled back and the method retried. As was the case with connections where auto-commit is enabled, the time delay between, as well as the number of, retries should be limited. This is illustrated in the following snippet of code:

```
do {
        try {
                //begin a transaction
                tran.begin();
                //Assumes that a datasource has already been obtained from JNDI
                conn = ds.getConnection();
                conn.setAutoCommit(false);
                stmt = conn.createStatement();
                stmt stmt.execute("INSERT INTO ORG VALUES (10, 'Pacific', '270', 'Western', 'Seattle')");
                tran.commit();
                retry = false;
        } catch(com.ibm.websphere.ce.cm.StaleConnectionException sce)
        {
                //if a StaleConnectionException is caught rollback and retry the action
                try {
                        tran.rollback();
                } catch (java.lang.Exception e) {
                        //deal with exception in most cases, this can be ignored
                }
                // deal with other database exceptions and clean up as before
        }
}
```

9.2.1.2.2Transactions Started in a Different Method from Database Access

When a transaction is begun in a method different from where a database connection is accessed, an exception needs to be thrown from the database access method, to indicate a failure. In an ideal situation, a method can throw an application-defined exception, indicating that the transaction logic can

be retried. However this is not always allowed, and often a method is defined only to throw a particular exception. This is the case, for example, in the ejbLoad and ejbStore methods on an enterprise bean.

9.2.1.3Connection Error Recovery within WebSphere

Internally within WebSphere a Bean Managed Persistent (BMP) entity EJB, or a Container Managed Persistence (CMP) entity EJB should transparently recover from a StaleConnectonException, and not be reflected in the client programming model.

9.2.2ConnectionWaitTimeoutExceptions

If servlet threads wait an inordinate amount of time for free connections from the connection pool, the website appears "hung" to a client. A com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException occurs if the connection timeout parameter for a datasource is exceeded. The parameter specifies the amount of time an application will wait to obtain a connection from the pool, if the maximum number of connections is reached and all available connections are in use. Tuning the connection timeout value for a datasource can significantly improve the end user experience with a web site under heavy load. The default value for this parameter is 180 seconds. The parameter may set to any non-negative integer value, where a 0 disables the connection timeout (the timeout is infinite).

This value can also be changed programmatically by calling setLoginTimeout() on a datasource. However, if setLoginTimeout is called on the datasource, the timeout value given is set for all applications using that datasource. Thus, the best practice is to set the connection timeout property during configuration. The value for this parameter is specified in the admin console, under JDBC Providers for each data source.

An example of catching this exception is given in the following code snippet:

```
java.sql.Connection conn = null;
javax.sql.DataSource ds = null
try {
        //Retrieve a DataSource through the JNDI Naming Service
        java.util.Properties parms = new java.util.Properties();
        setProperty.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
        //Create the Initial Naming Context
        javax.naming.Context ctx = new
        javax.naming.InitialContext(parms);
        //Lookup through the naming service to retrieve a
        //DataSource object
        javax.sql.DataSource ds = (javax.sql.DataSource) ctx.lookup("java:comp/env/jdbc/SampleDB");
        conn = ds.getConnection();
        //work on connection
} catch (com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException cw) {
        //notify the user that the system could not provide a
        //connection to the database
} catch (java.sql.SQLException sqle) {
        //deal with exception
}
```

9.3 Session Database

The default behavior in WebSphere is to store HTTP session objects in memory. As described in Chapter 3 (section 3.3.4) WebSphere provides two mechanisms for session failover: 1) memory-to-memory replication, and 2) session persistence into a database.

The Session Manager in WebSphere provides an option for utilizing a database to store HTTP session objects. This implementation utilizes updates to HTTP Session objects in memory, with periodic updates of session information to a database. This allows all application servers in a cluster to participate in the failover of requests invovling HTTP session objects. The database represents a potential single point of failure, in this scheme.

ND provides a number of mechanisms that allow for tuning of the frequency of updates to the database used for persistence of the HTTP session object. In brief, the Session Manager may be tuned to update the database:

- At the end of the servlet service method (the default).
- Manually (this requires use of an IBM extension to the HttpSession API).
- At the end of a specified time interval

Updating the database at the end of a servlet service method requires a database be continuously available. Tuning the Session Manager, in conjunction with use of a highly available database (see Chapter 10) are the keys to minimizing disruptions in service, in the event of a database outage.

9.3.1Expected Behavior - Servlet Service Method

In the event of a database becoming unavailable, servlet requests become queued waiting for the database service to be restored. The Session Manager uses the programming model extension by triggering off the StaleConnectionException to recover from database connectivity errors. Once the database service is restored, the database connection pool is destroyed and recreated. All queued requests are then serviced. The Session Manager handles all database connection failures, and subsequent their recovery, transparently.

9.3.2Expected Behavior - Manual Update

An option for session persistence updates in WebSphere involves the use of an IBM extension to the HTTPSession API , known as “manual update”. Manual update allows the application to decide when a session should be stored persistently. With manual update , the Session Manager sends changes to the persistent data store, only if the application explicitly requests this action.

Manual update requires the use of the IBMSession class for managing sessions. When an application invokes the sync() method, the Session Manager writes the modified session data and last access time to the database. The session data that is written out to the database is controlled by the Write Contents option selected.

If the servlet or JSP terminates without invoking the sync() method, the Session Manager saves the contents of the session object into the session cache (if caching is enabled), but does not update the modified session data in the database. The Session Manager will update just the last access time in the database asynchronously at a later time.

Recovery from database connectivity failures is the same as described in the previous section.

9.3.3Expected Behavior - Time Based Update

Time based updates of the HTTP Session object occur in memory, as is the case with the default and manual update, but database persistance is deferred until the end of the specified time interval. By deferring the updates the probability that a disruption in database service will less likely disrupt servlet requests (assuming the in memory session data is current). For example, the default interval for time based updates is 300 seconds, thus a database failure that occurred immediately after the last update interval would not impact servlet requests for 300 seconds. In the best case, database service will be restored in between updates, and therefore would not have any impact.

Recovery from database connectivity failures is the same as described in section 9.3.1, however, tuning for the “correct” update interval depends on an individual customer’s requirements for failover and recovery. While the default time interval should prove adequate, it can be cutomized to a given environment via the manual persistence configuration dialog in the session manager. The benefits of a longer time interval, which lowers the probability of an update coinciding with a database failure, should be weighed against the amount of traffic on the website in that interval. The more data that is in memory and unpersisted, the greater the potential for high loss of data. Also, large amounts of session data, cached by the session manager, may decrease performance during persistance due to the attending JVM memory impact.

Despite the above, there remains the potential for loss of HTTP session data. For example, if an application server process were to fail between updates, all cached in memory data would be lost. In cases where the state information being stored is deemed valuable (“state liability is high”), such loss would be unacceptable. Rather than relying on the persistence mechanism used by the Session Manager, a recommended alternative, to the storage of state information in HTTP sessions, is for an application to store state information in a database via direct JDBC calls, or within an entity EJB. This alternative provides for transactional updates of state information, but also requires the high availability of database resources, which is now covered in the next chapter.

Chapter 10 - Software and Hardware Clustering for Database High Availability

Protecting data integrity and enhancing database availability is critical for WebSphere, which can be achieved utilizing software and hardware cluster solutions.

WebSphere ND supports many different databases technologies. This chapter will discuss high availability database solutions, which have been tested in WebSphere labs. WebSphere should perform properly with these solutions, assuming proper database availability is maintained, and that client application follow the programming model for database connections, discussed in Chapter 9.

10.1WebSphere with Oracle Parallel & Real Application Cluster Server

10.1.1Introduction

In Oracle 8i Oracle Parallel Server (OPS) offers superior scalability and high availability through distribution of database workload across multiple nodes. Parallel database servers, such as OPS, facilitate faster failover in the event a database process is unavailble, since copies of that instance exist on backup node or nodes. Platform-specific software such as MC/Serviceguard or Sun Cluster is not required for OPS, since it is not (by itself) registered with, or managed by, a cluster resource "group manager"Although Oracle 8i OPS employs non-IP failover, it requires platform-specific cluster software to provide Cluster Manger and Inter-Process Communication. OPS has an operating system dependent layer (platform-specific cluster software), Oracle 8i Enterprise Edition, Oracle Parallel Server Option, and Oracle Parallel Server Management. Further, within OPS there are Integrated Distributed Lock Management (IDLM), Parallel Cache Management (PCM), and Oracle Parallel Query.

OPS can be configured to use a shared-disk architecture. In this configuration, a single database can be shared among multiple instances of OPS that access the database concurrently, and the Distributed Lock Manager controls conflicting access to the same data. If a process (or node) becomes unavilable, the DLM is configured to recover from the failure, since there are copies of the database on other nodes available. Therefore, Oracle clients can failover to another instance of a database server without recourse to IP failover. Multiple Oracle instances cooperate to provide access to the same shared database, and clients can use any database server instance to access data. This elminates any one process as a single point of failure.

In Oracle 9i Real Application Cluster (RAC), provides not only the functionality provided by OPS in Oracle 8i, but Oracle 9i RAC(OPS) also includes groupware, and OPS(RAC) failover takes a very short time (non-IP takeover without the need to restarting database instances).

10.1.2Setup and Configuration

10.1.2.1Shared Disk Subsystem

OPS stores all common data such as log, control files, and Oracle data on a shared disk, accessable by all OPS instances in different nodes. Each instance of an OPS database has its own log files that should be on the shared disk, such that other instances can access them during failover events. Since control and data files are shared by all OPS instances, these files are also placed on the shared disk. OPS binaries should be installed onto the shared disk, or local disks of in each node. During testing, the choice was to install the binaries onto the local disks of each node. The shared disk subsystem must be partitioned to be shared raw, and the raw devices must be created by root.

You can configure OPS into a failover mode, as shown in Figure 10.1, or a failover with load balance, as shown in Figure 10.2. We will discuss the configuration details in the next section.

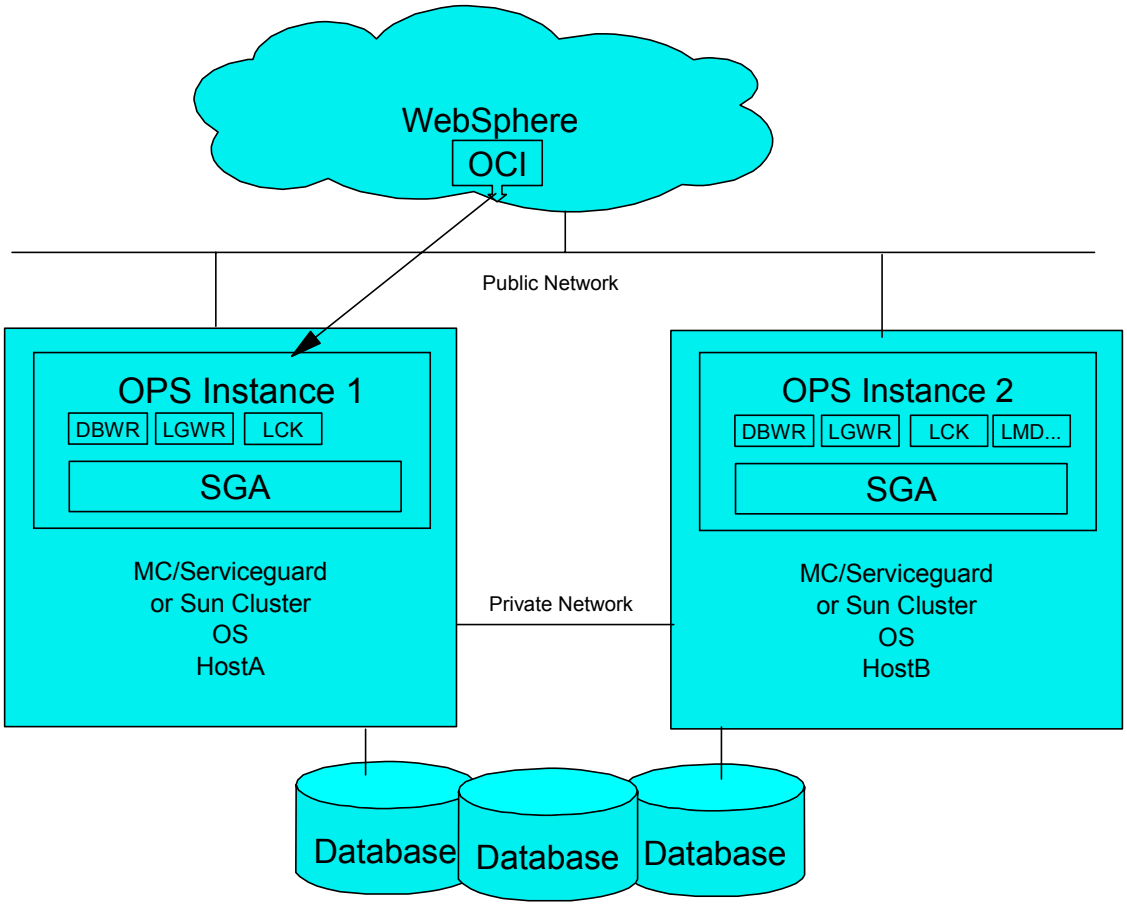


Figure 10.1 OPS configuration without load balance for WebSphere

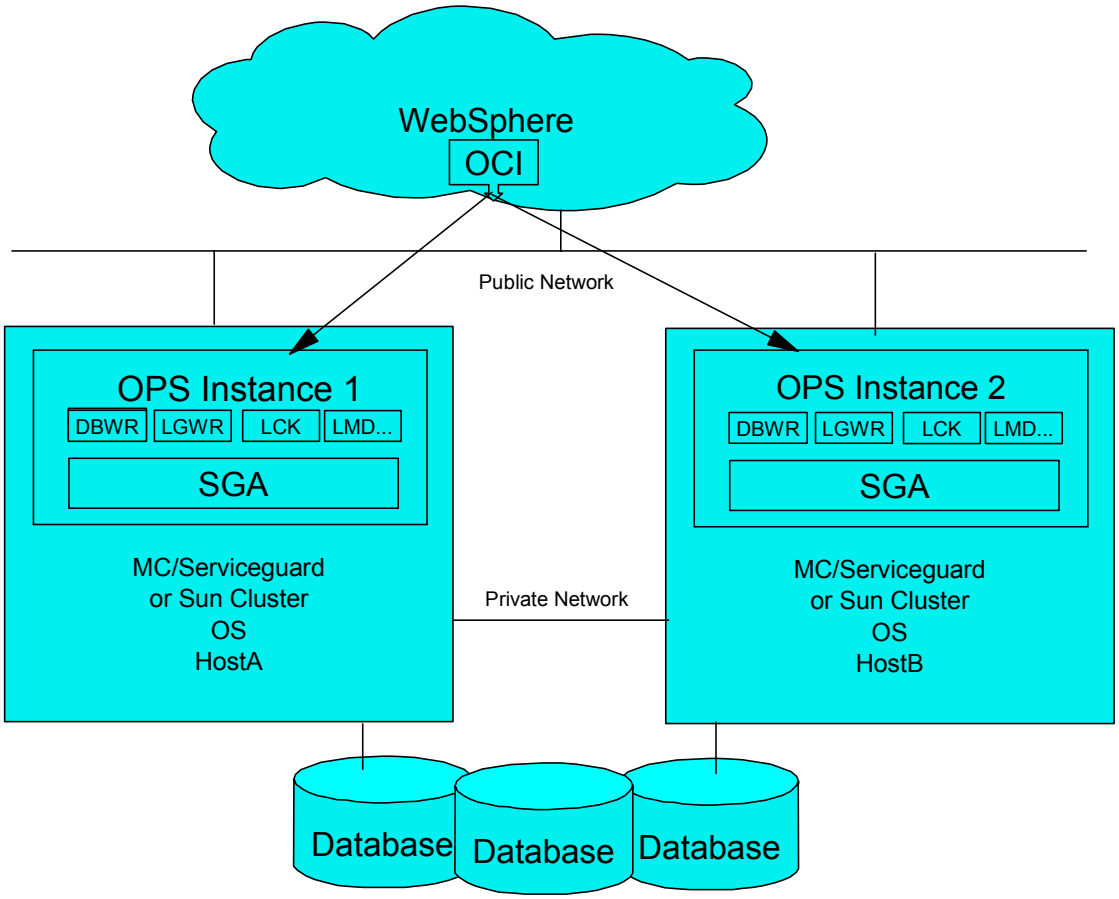


Figure 10.2 OPS configuration with load balance for WebSphere

OPS can also be configured as a part of an Oracle distributed database system for partitioning data, or isolating applications. As shown in Figure 10.3, four oracle instances are created to serve 3 databases; instance-1 and instance-2 share the common data files and control files on multihost shared disks.

As shown in Figure 10.4, when OPS becomes unavailable due to failed host hardware or software, network connectivity problems, or its Oracle instance dies, OPS will automatically failover to another OPS, and its Oracle instance will have access to the log files of the failed Oracle instance. The OPS will complete some recovery work to keep database integrity. Since OPS failover does not need require starting another Oracle instance, the technique of OPS clustering is much faster than IP failover.

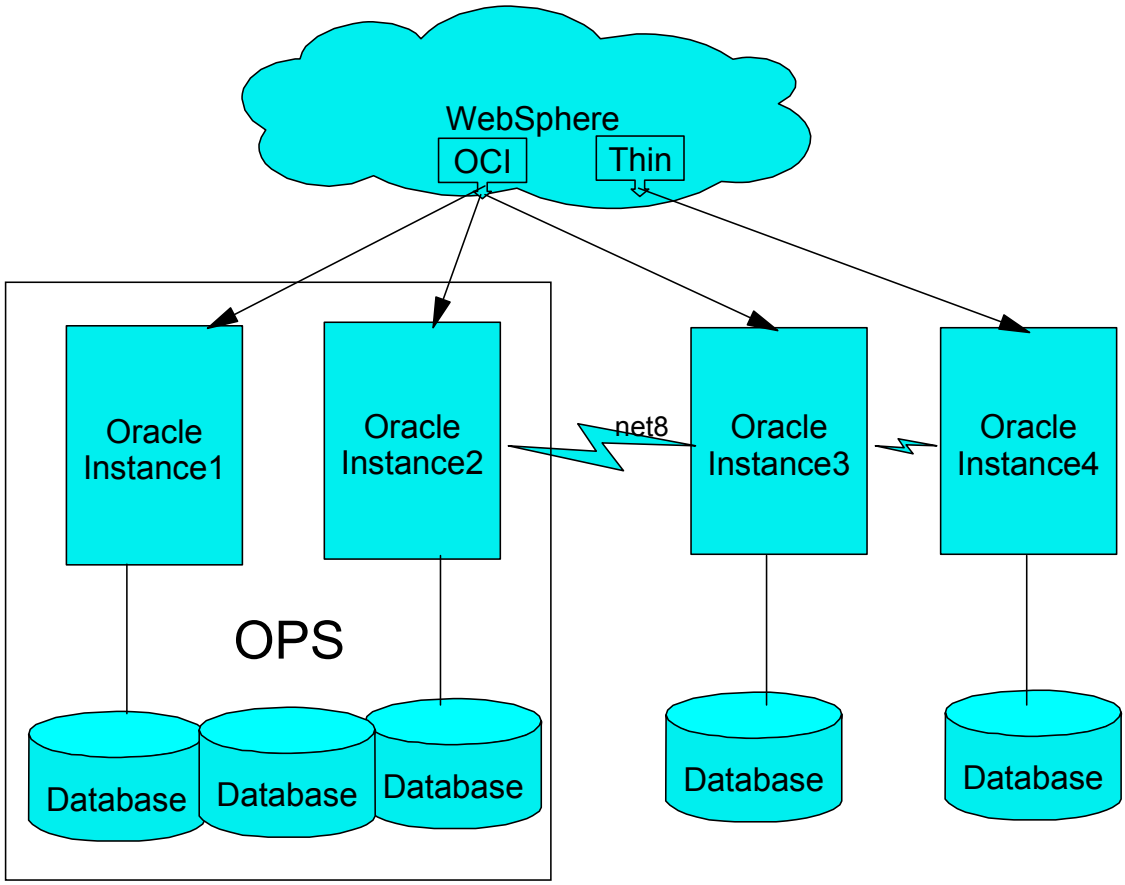


Figure 10.3 OPS and Distributed Database Configuration

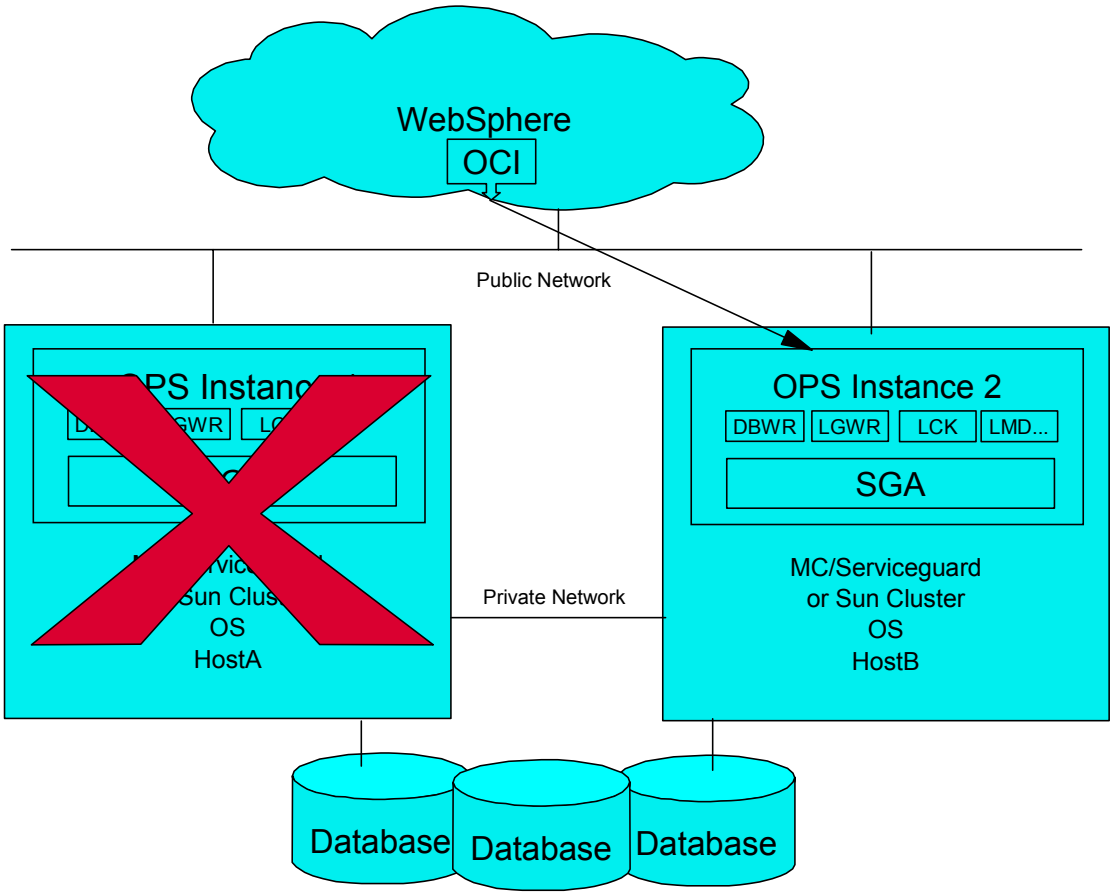


Figure 10.4 OPS failover for WebSphere

Here are the neccessary steps to configure OPS for WebSphere:

10.1.2.2Create raw devices.

Create ASCII files with entries for each raw device file name for the Oracle Universal Installer, and an environment variable DBCA_RAW_CONFIG that points to the location of the ASCII file.

HACMP and MSCA require the use of the Volume Manager. For Sun Cluster HA for Oracle Parallel Server disks, you can use either of the following configurations, VERITAS Cluster Volume Manager or Sun Cluster Solstic Volume Manager. For MC/Serviceguard Cluster HA for Oracle Parallel Server disks, use the HPCluster Volume Manager configurations.

10.1.2.3Operating System Dependent Layer

Install either HACMP, Sun Cluster 2.2 or Sun Cluster 3.0 for Solaris or MC/Serviceguard OPS edition for HP UX or MSCS. Configure the cluster software. Diagnose the cluster configuration. For installation and configuration details, see above sections in this chapter. You don't need to do so if you use Oracle 9i Real Application Cluster since it is supplied with groupware.

10.1.2.4 Install and configure OPS

Install Oracle 8i/9i Enterprise Edition, Net8 Server, Oracle Parallel Server Option, Oracle Intelligent Agent (if Oracle Enterprise Manager is used), Oracle Data Gatherer (if Oracle Diagnostics Pack is used for generating performance reports) and a web browser (if not installed).

- Create dba user group and oracle oracle that belongs to dba group.
- Create mount point directory on each node so that its name is identical for each node. Ensure that the oracle account has read, write and execute privileges.
- Give the permission to use the rcp command. Set up user equivalence on the node from which OPS is installed by adding entries for all nodes in the cluster to the .rhosts file of the oracle account, or the /etc/hosts.equiv file.
- Install Oracle 8i/9i Enterprise Edition with OPS/RAC option. Please note that the Oracle Universal Installer will not make Oracle Parallel Server Option visible unless Cluster Manager cluster software has been properly configured. If a problem is encountered, please see the platform-specific cluster software configurations discussed above.
- Choose a node for installation, on the cluster node selection screen. For example, opshp1 or opshp2. Enter a SID on the database Identification screen.
- Follow the typical installation and create database for the WebSphere.
- Start the listener of each node by lsnrctl start
- Start the database on each of node by connect internal/password; startup
- Verify all of defined OPS instances are running by query select * from v$active_instances.

10.1.2.5Configure OPS net service for failover

There are three files to be configured: LISTENER.ORA, TNSNAMES.ORA, and SQLNET.ORA Use the Oracle netasst utility to configure them graphically, or edit these files by hand if very familiar with OPS. They must be manually edited to implement a new listener for failover, as netasst doesn't permit using the same port number for another listener.

Implementing failover doesn't allow the use of static service configuration parameters in the LISTENER.ORA file. Therefore, in order to implement OPS failover, an additional listener must be specified on each node in the LISTENER.ORA file without service registration information. Edit SQLNET.ORA file to ensure that TNSNAMES.ORA is used to resolve a net service name.

Upon setting load_balance=on in tnsnames.ora, OPS will balance the load over the list of listener addresses by picking one at random.

There are two different kinds of OPS failovers: Connection-Time Failover (CTF) and Transparent Application Failover (TAF). For TAF, there are several different configurations by setting different TYPE and different METHOD.

10.1.2.6Connection-Time Failover (CTF)

Connection-Time Failover refers to client attempting to connect to a second listener, when the attempted connect to the first fails. Implementation involves the creation of a new net service name

with the database name as its identifier, such opswas, instead of opshp1 or opshp2. Put the address information of all nodes in the cluster into this newly created global net service name. Use one of two option

FAILOVER=ON

Setting this means that try each address in order, until one succeeds

FAILOVER=ON LOAD_BALANCE=ON

Setting this means that try each address randomly, until one succeeds

Sample OPS CTF failover tnsnames.ora files:

```
WASOPSCTF.somecorp.com=
 (description=
  (load_balance=on)
  (failover=on)
  (address=
      (protocol=tcp)
      (host=idops1)
      (port=1521))
 (address=
      (protocol=tcp)
      (host=idops2)
      (port=1521))
  (connect_data=
    (service_name=WASOPSCTF.somecorp.com))))
```

WebSphere works with either setting mentioned above.

10.1.2.7Transparent Application Failover (TAF)

Transparent Application Failover enables an application to automatically reconnect to a database if the connection is broken. Active transactions roll back, and the new database connection is identical to the original one no matter how the connection was lost.

Implementing Transparent Application Failover (TAF) requires manual configuration of tnsnames.ora, with the specification of a primary/backup node, failover type, and failover method. One can combine OPS CTF with OPS TAF. There are 12 configurations (2 CTF x 3 TYPE x 2 METHOD) for the OPS TAF implementations.

CTF	Type	Methodb
FAILOVER=ON	SESSION	BASIC
FAILOVER=ON LOAD_bBALANCE=ON	SELECT	PRECONNECT
	NONE	

Parameter TYPE specifies the type of failover. This is a required parameter for TAF failover. There are three options: SESSION, SELECT, and NONE. Selection of SESSION means that OPS fails over the session. A new session will be created automatically on the backup database server if a user's connection is lost. This type of failover doesn't attempt to recover selects after a failover. Selection of

SELECT means that OPS allows users with open cursors to continue fetching on them after a failover. Although performance for SELECT failover is good, it has an overhead during normal select operations. Selection of NONE means that no failover function is used, which is the default.

The parameter METHOD specifies how fast a failover should occur from the primary to the backup node. There are two options: BASIC and PRECONNECT. Selection of BASIC means that OPS establishes connection at failover time. This option requires almost no work on the backup server until a failover occurs. Selection of PRECONNECT means that OPS pre-establishes connections all the time, and is good for the failover performance. However, PRECONNECT is bad for the normal operational performance, since it requires that the backup instance supports all connections from every primary instance.

Sample tnsnames.ora for OPS/TAF configurations with CTF is as follows

```
WASOPSTAF.somecorp.com=
 (description=
  (load_balance=on)
  (failover=on)
  (address=
      (protocol=tcp)
      (host=opshp1)
      (port=1521))
 (address=
      (protocol=tcp)
      (host=opshp2)
      (port=1521))
  (connect_data=
    (service_name=WASOPSTAF.somecorp.com)
    (failover_mode=
      (type=session)
      (method=preconnect)
        (retries=0)
        (delay=10))))
```

Or sample tnsnames.ora for the OPS/TAF primary and secondary servers configuration:

```
WASOPSTAF.somecorp.com=
(description=
(address=
      (protocol=tcp)
      (host=opshp1)
      (port=1521))
  (connect_data=
    (service_name=WASOPSTAF.somecorp.com)
    (failover_mode=
       (backup=opshp2)
      (type=session)
      (method=preconnect)
        (retries=0)
```

(delay=10))))

WebSphere works well with OPS/CTF failover configurations, and with some OPS/TAF failover configurations, for example, TYPE of SESSION, and METHOD of both BASIC and PRECONNECT.

10.1.2.8 Connect to WebSphere servers

Install an Oracle client on the same hosts where WebSphere is installed. Use the OCI driver to connect the WebSphere application servers to the databases instances created. Since we use the global net service name, which is independent on individual nodes, failover will be transparent to WebSphere. For CTF, WebSphere will receives a handful of StaleConnectionExceptions during OPS failover, but WebSphere will handles these exceptions automatically as described in Chapter 9. Therefore OPS/TAF failover should not impact Websphere function.

In addition to the Oracle 8i/9i OPS/RAC non-IP failover, we will discuss how to configure Oracle 8i and 9i to employ IP-takeover failover with HACMP (and other clustering software) below.

10.2HACMP and HACMP/ES on IBM AIX

10.2.1Introduction and Considerations

HACMP and HACMP Enhanced Scalability (HACMP/ES) provides the AIX platform with a high-availability solution for mission critical applications. HACMP allows the creation of clusters of RS/6000 servers to provide high availability of databases and applications in spite of a hardware, software, or network failure. HACMP/ES includes all of the same functionality as HACMP, but enables larger clusters of machines and more granular event control.

The unit of failover in HACMP or HACMP/ES is a resource group. A resource group contains all the processes and resources needed to deliver a highly available service and ideally should contain only those processes and resources. HACMP for AIX software supports a maximum of up to 20 resource groups per cluster.

As with MC/ServiceGuard, an HACMP cluster requires two network connections for a highly available heartbeat mechanism. A public network connects multiple nodes and allows clients to access these nodes, while a private network connection allows point-to-point communications between two or more nodes.

Each node is typically configured with at least two network interfaces for each network to which it connects: a service interface that handles cluster traffic, and one or more standby interfaces to allow recovery from a network adapter failure. Recovery by a standby adapter is typically faster than recovery by another node, and thus provide less down time.

The AIX IP address takeover is utilized to facilitate node failover. If one node should fail, a backup node acquires the failed node's service address on its standby adapter, making failure transparent to clients. To enable IP address takeover, a boot adapter IP address must be assigned to the service adapter on each cluster node. Nodes use this boot address after a system reboot and before the HACMP for AIX

software is started. When the software is started on a node, the node's service adapter is reconfigured to use the service address instead of the boot address. In addition a RAID disk array can be shared among the nodes in the cluster to provide data redundancy.

When a failover occurs, HACMP invokes a script to stop the resource group and a second script to start the resource group. Samples of these scripts are included in Appendix D, but need to be customized to match a particular runtime environment.

10.2.2Expected Reactions to Failures

If a failure occurs, the quickest way to recover is failover from the service adapter to a standby adapter. In many cases, for example a node power failure, this is not an option. In this case, the resource group will need to fail over to another node in the cluster. The node on which the resource group is recovered depends on the resource group configuration. There are three types of resource groups supported:

- Cascading resource groups - A resource may be taken over by one or more nodes in a resource chain according to the takeover priority assigned to each node. The available node within the cluster with the highest priority will own the resource. If this node fails, the node with the next highest priority will own the resource. One can choose to allow the highest priority node recover its resources when it is reintegrated into the cluster, or set a flag requiring the resource to remain on the active lower-priority node.
- Rotating resource groups - A resource is associated with a group of nodes and rotates among these nodes. When one node fails, the next available node will acquire the resource group.
- Concurrent access resource groups - A resource can be managed by the HACMP for AIX cluster Lock Manager and may be simultaneously shared among multiple applications residing on different nodes.

Cascading resources provide the best performance sincethey ensure that an application is owned by a particular node, whenever that node is active within the cluster. The ownership allows a node to cache data the application uses frequently. Rotating resources may minimize the downtime for failover. Concurrent access, while providing very quick response times to failures, is a performance drain.

When this failover occurs, the following steps are run on the new active node:

- run the stop script
- release the disk volume groups and other shared resources held by the primary node.
- take over the service IP address
- mount the shared disk array and take over any other resources necessary
- run the start script

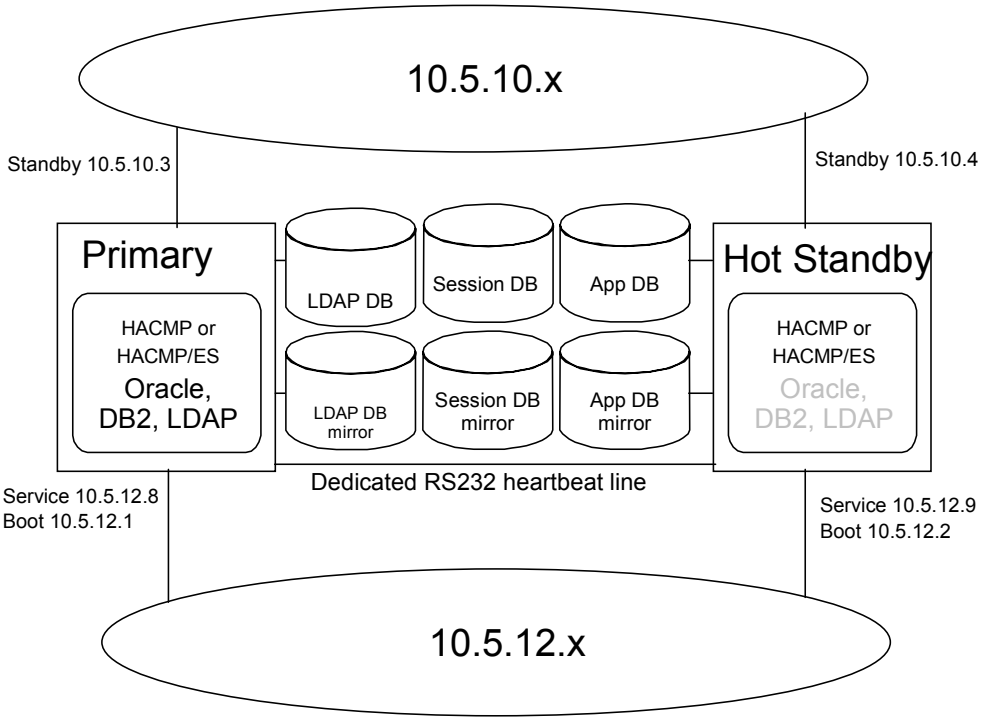


Figure 10.5: HACMP Cluster Configuration for the WebSphere

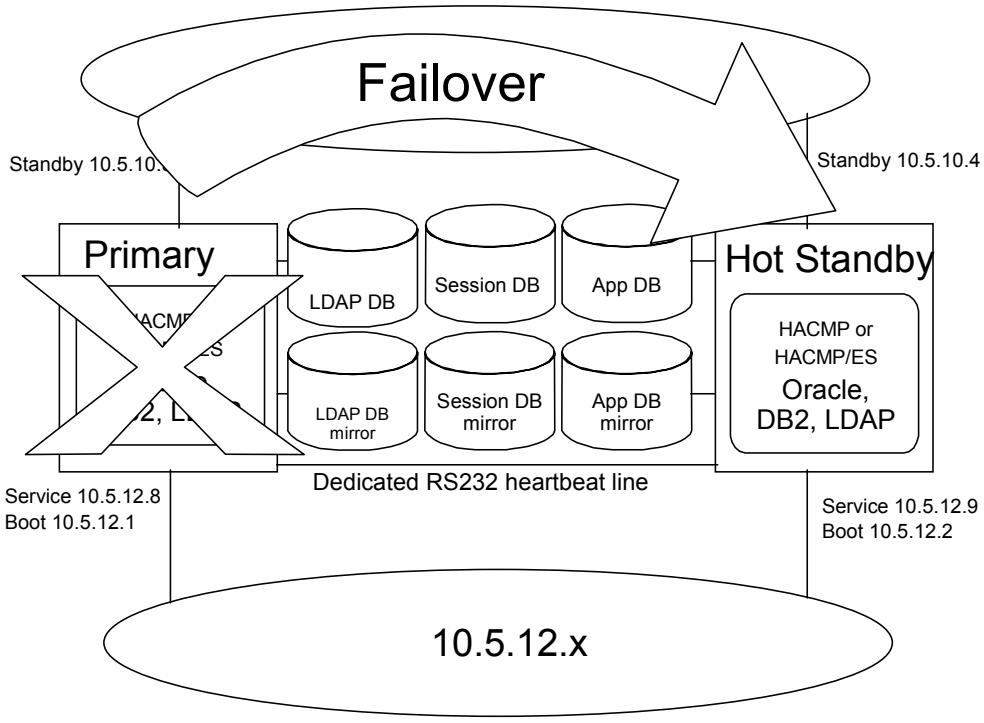


Figure 10.6 HACMP Cluster for WebSphere after the primary node fails

10.2.3WebSphere HACMP configuration

The lab environment used to test this configuration consisted of two IBM RS/6000 machines running AIX 4.3.3. These machines were installed with the HACMP software. Testing covered HACMP 4.3.1, HACMP 4.4, HACMP/ES 4.4 ptf 3, and HACMP/ES 4.4.1. Both DB2 7.2.1 and Oracle 8i/9i were installed. The machines shared an IBM 7133 Serial Storage Architecture Disk Subsystem. The public network connection was supplied by a 100 MB Ethernet LAN, while the private connection was supplied by a dedicated RS232 connection.

WebSphere resources requiring database access were configured to communicate with the service IP address, and programmed in accordance with the recommendations in Chapter 9.

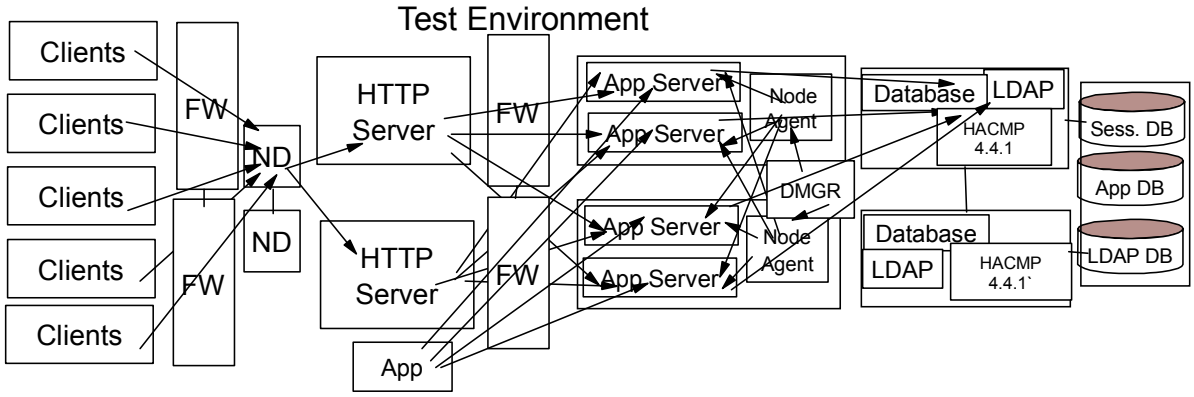


Figure 10.7: Test environment for integrated WebSphere and HACMP HA system

10.2.4Tuning heartbeat and cluster parameters

HACMP 4.4.1 provides control over several tuning parameters that affect the cluster's performance. Setting parameters correctly to ensure throughput and adjusting HACMP failure detection rate can help avoid sluggish server response caused by heavy network traffic.

- *Adjusting high and low watermarks for I/O pacing*. The default value is 33-24.

- *Adjusting syncd frequency rate*. The default value is 10.

- *Adjusting HACMP failure detection rate*. There are two parameters that control HACMP failure detection rate.
  - HACMP cycles to failure: the number of heartbeats that must be missed before detecting a failure.
  - HACMP heartbeat rate: the number of microseconds between heartbeats

  For examples, heartbeat rate=1 second, cycles=10, the failure detection rate would be 10 seconds. Faster heartbeat rates may lead to false failure detection, particularly on busy networks. Please note that the new values will become active the next time cluster services are started.

- *AIX deadman switch timeout*. When HACMP for AIX does not have enough CPU capacity to send heartbeats on IP and serial networks, other nodes in the cluster will assume the node has failed, and

initiate a takeover of that node's resources. In order to ensure a lean takeover, the deadman switch crashes the busy node if it is not reset within a given time period.

10.3MC/Serviceguard on the HP-Unix Operating System

10.3.1Introduction and Considerations

Hewlett Packard Multi-Computer/ServiceGuard (MC/ServiceGuard) is a high-availability solution available for HP-UX systems. This product allows the creation of clusters of HP 9000 series computers that provide high availability for databases and applications in spite of a hardware, software, or network failure.

MC/ServiceGuard ensures availability of units called packages; a collection of services, disk volumes, and IP addresses. When a cluster starts, a package starts up on its primary node. The MC/ServiceGuard package coordinator component decides when and where to run, halt, or move packages. User-defined control scripts are used to react to changes in the monitored resources.

A heartbeat mechanism is used to monitor node and database services, and their dependencies. If the heartbeat mechanism detects a failure, recovery is started automatically. To provide a highly available heartbeat mechanism, two connections must be available between the MC/ServiceGuard nodes. This can be provided by dual network connections, or, in a two node configuration, a single network connection and a dedicated serial heartbeat. This heartbeat redundancy will prevent a false diagnosis of an active node failure.

Network failures are handled by the network manager component. Each active network interface on a node is assigned a static IP address. A relocatable IP address, also known as a floating or a package IP address, is assigned to each of the packages. This provides transparent failover to the clients by providing a single IP address to connect to, no matter which node in the cluster is hosting the service. When a package starts up, the cmmodnet command in the package control script assigns this relocatable IP address to the primary LAN interface card in the primary node. Within the same node, both static and relocatable IP addresses will switch to a standby interface in the event of a LAN card failure. Only the relocatable IP address can be taken over by an adoptive node if control of the package is transferred. At regular intervals, the network manager polls all the network interface cards specified in the cluster configuration file. The polling interface sends LAN packets to all other interfaces in the node that are on the same bridged net and receives packets back from them. If a network failure is detected, the network manager has two options:

- Move the package to a backup network interface on the same node. This is often referred to as a local switch. TCP connections to the relocatable IP address are not lost (except for IEEE 802.3 that does not have the rearp function).
- Move the package to another node in the cluster. This is often referred to as a remote switch. This switch causes all TCP connections to be lost. If a remote switch of a WebSphere database package is initiated, connections held in the WebSphere connection pools will be lost. These connections will be marked stale and will need to be recovered as described in Chapter 6.

Data redundancy is also an important part of this failover mechanism. There are two options available to provide data redundancy with MC/ServiceGuard:

- Disk mirroring with MirrorDisk/UX - MirrorDisk/UX is a software package for the HP-Unix Operating System which provides the capability to create up to three copies of your data on different disks. If one disk should fail, MirrorDisk/UX will automatically keep the data available by accessing the mirror. This access is transparent to the applications utilizing the data. To protect against SCSI bus failures, each copy of the data must be accessed over a separate SCSI bus.
- Redundant Array of Independent Disks (RAID) levels and Physical Volume (PV) links - An array of redundant disks and redundant SCSI interfaces protect against a single point of failure in the I/O channel. PV links are used to configure the redundant SCSI interfaces.

One can monitor disks through the HP Event Monitoring Service, a system monitoring framework for the HP environment.

10.3.2Expected Reactions to Failures

If a package needs to be moved to another node, the general MC/ServiceGuard process involves the following steps:

- Stop services and release resources in the failed node
- Unmount file systems
- Deactivate volume groups
- Activate volume groups in the standby node
- Mount file systems
- Assign Package IP addresses to the LAN card on the standby node
- Execute start up of services and get resources needed

During the time this failover process is occurring, the package is unavailable for client connections. If the package contains one or more of the WebSphere databases, errors may be reflected in the client. Once the failover is complete, WebSphere components and applications should successfully reconnect, assuming proper programming practices as outlined in Chapter 9 are followed.

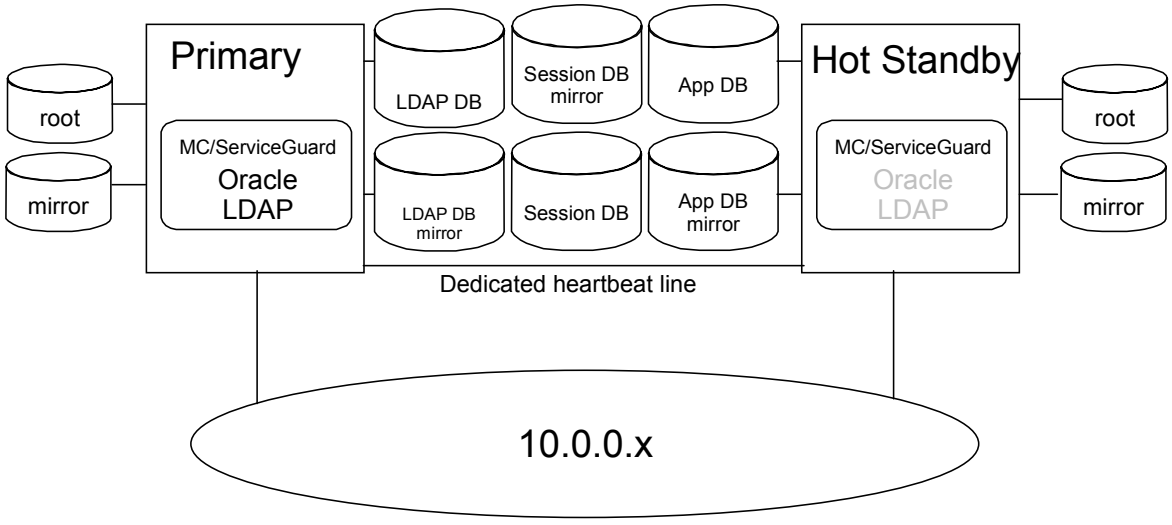


Figure 10.8: MC/ServiceGuard cluster database configuration

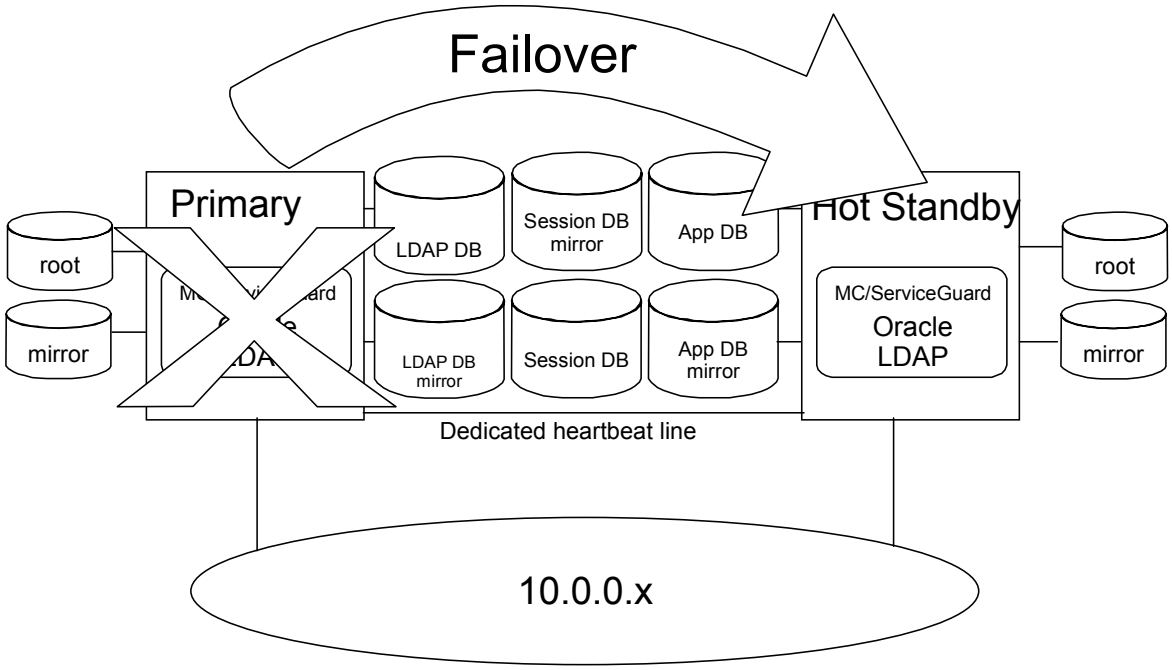


Figure 10.9: MC/ServiceGuard cluster configuration after failover

10.3.3WebSphere MC/ServiceGuard configuration

The lab environment used to test this configuration consisted of two HP 9000 k-class machines running HP-UX 11.0. These machines were installed with MC/ServiceGuard A11.12. Both DB2 7.2.1 and Oracle 8i/9i were installed. The machines shared an AutoRAID disk array, where the DB2 or Oracle database instance were created. Fast/Wide SCSI interfaces were used to connect two nodes to the disk array. The redundant heartbeat mechanism was set up on two Ethernet LANs, a public LAN connected to the rest of the machines in the test configuration, the other a private LAN for heartbeat only. WebSphere resource requiring database access were configured to communicate with the relocatable IP address, and were programmed in accordance with the recommendations in Chapter 6.

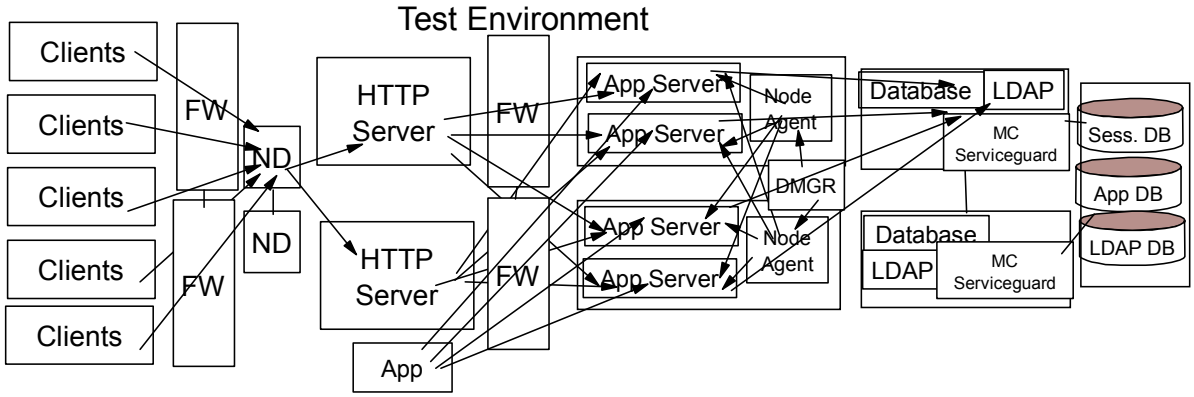


Figure 10.10: Test environment for integrated WebSphere and MC/Serviceguard HA

10.3.4Tuning heartbeat and cluster parameters

Sending and receiving heartbeat messages among the nodes in a cluster is a key management technique. If a cluster node does not receive heartbeat messages from the other node(s) within a prescribed time, a cluster reformation is initiated. At the end of the reformation, if a new set of nodes form a cluster, that information is passed to the package coordinator. Packages which were running on nodes that are no longer in the new cluster are transferred to their adoptive nodes.

There are several MC/ServiceGuard parameters you can tune for performance and reliability:

- *Heartbeat interval* is the normal interval between the transmission of heartbeat messages from one node to the other in the cluster. The default value is 1 second, with a maximum value of 30 seconds.

- *Node timeout* is the time after which a node may decide that the another node has become unavailable and initiate cluster reformation. The default value is 2 seconds, with a minimum of 2* (heartbeat interval) and a maximum of 60 seconds. Small values of node time out and heartbeat interval may increase the potential for spurious cluster reformation due to momentary system hangs or network load spikes.

- *Network polling interval* is the frequency at which the networks configured for MC/Serviceguard are checked. The default value is 2 seconds. Changing this value can affect how quickly network health is polled, ranging from 1 to 30 seconds.

- *Choosing switching and failover behavior:* Switching the IP address from a failed LAN card to a standby LAN card on the same physical subnet may take place if Automatic Switching is set to "enabled" in SAM. One can define failover behavior and if there is failback automatically as soon as primary node is available.

- *Resource polling interval* is the frequency of monitoring a configured package resource. Default value is 60 seconds, with the minimum value of 1 second.

10.4 Microsoft Clustered SQL Server on Windows 2000

10.4.1Introduction

In the Windows 2000 Advanced Server and Datacenter Server operating systems, Microsoft introduces two clustering technologies that can be used independently or in combination, providing organizations with a complete set of clustered solutions that can be selected based on the requirements of a given application or service. Windows clustering technologies include:

- **Cluster service.** This service is intended primarily to provide failover support for applications such as databases, messaging systems, and file/print services. Cluster service supports 2-node failover clusters in Windows 2000 Advanced Server and 4-node clusters in Datacenter Server. Cluster service is ideal for ensuring the availability of critical line-of-business and other back-end systems, such as Microsoft Exchange Server or a Microsoft SQL Server (TM) 7.0 database acting as a data store for an e-commerce web site.

- **Network Load Balancing (NLB).** This service load balances incoming IP (Internet Protocol) traffic across clusters of up to 32 nodes. Network Load Balancing enhances both the availability and scalability of Internet server-based programs such as web servers, streaming media servers, and Terminal Services. By acting as the load balancing infrastructure and providing control information to management applications built on top of Windows Management Instrumentation (WMI), Network Load Balancing can seamlessly integrate into existing web server farm infrastructures. Network Load Balancing will also serve as an ideal load balancing architecture for use with the Microsoft release of the upcoming Application Center in distributed web farm environments.

See Introducing Windows 2000 Clustering Technologies at http://www.microsoft.com/windows2000/techinfo/howitworks/cluster/introcluster.asp for more information.

The failover mechanism in the Windows 2000 active/passive clustering works by assigning a virtual hostname / IP address to the active node. This is the hostname exposed to all external clients accessing the clustered resource. When a resource on the active node fails, the cluster moves to a passive standby node, making it the active node and starting all the necessary services. Additionally, the virtual hostname/IP address is moved from the first node to the newly activated node, and handling all new requests from the client. During this transition, database resources will be unavailable. However, once the transition is complete, the only indication to the client that something has failed is that the pooled connections to the database are no longer valid and must be reestablished.

10.4.2 Expected Reactions to Failures

The Microsoft cluster can fail in several ways.

- **Manual Push to Passive Node** - Within the cluster administrator, you can right-click on a group and click "Move Group" to move it to the passive node.
- **Clean Shutdown of Active Node.** Without manually moving any of the groups, go to Start->Shutdown and power down the Active Node.
- **Unexpected power failure on Active Node.** Pull the power cable from the Active Node.
- **Public network failure on the Active Node.** Pull the public network cable from the Active Node.

When these failures occur, the cluster service should recognize that one (or more) of the resources failed on the active node and transition all of the components from the failing node to the alternate node. At this point, all the connections to WebSphere processes are broken. The next request (from WebSphere) result in a StaleConnectionException (as described in Chapter 6). After the transition to the new active node has completed, WebSphere will flush and refill the connection pool. Applications properly programmed according to the guidelines in Chapter 6 would also reconnect to the database.

If both the private and public networks on the active node fail, the cluster service will not transition the components to the alternate node. When both networks go down, and all communication is lost between nodes, each will attempt to take control of the shared disk. However, the disk is already locked by the active node, so the standby node's takeover request will fail. Meanwhile, the active node will attempt to move the cluster to another node. However, since all of its network connections were lost, the active node will seeappear to be the only node in the cluster.

10.4.3WebSphere Microsoft Cluster Server Configuration

The lab environment used to test this configuration consisted of two IBM Netfinity 7000 M10 machines running Microsoft Windows 2000 and Microsoft SQL Server 2000 Enterprise Edition.

	Hardware	***Software***
Database Server Node	IBM Netfinity 7000 M10 • 4 Internal 16 GB SCSI Hard drives • Adaptec AIC-7895 SCSI controller • 2 GB RAM • 4x500 MHz Pentium III Xeon processors	• Microsoft Windows 2000 Advanced Server with Service Pack 2 Applied (includes clustering software) • Microsoft SQL Server 2000 Enterprise Edition OR IBM DB2 7.2 Enterprise-Extended Edition
Shared Disk	IBM EXP15 • 10 Drive Array • 2 SCSI interfaces	
WebSphere Server	IBM M Pro • 1x933 MHz Processor • 512 MB RAM	Windows NT 4.0 SP 6a

Notes:

- Even though the WebSphere server was tested on an NT platform, the same functionality occurs on any of the distributed WebSphere platforms.
- The nodes of the cluster were on identical boxes, with identical software installed on each, hence only one column for hardware and software, above.

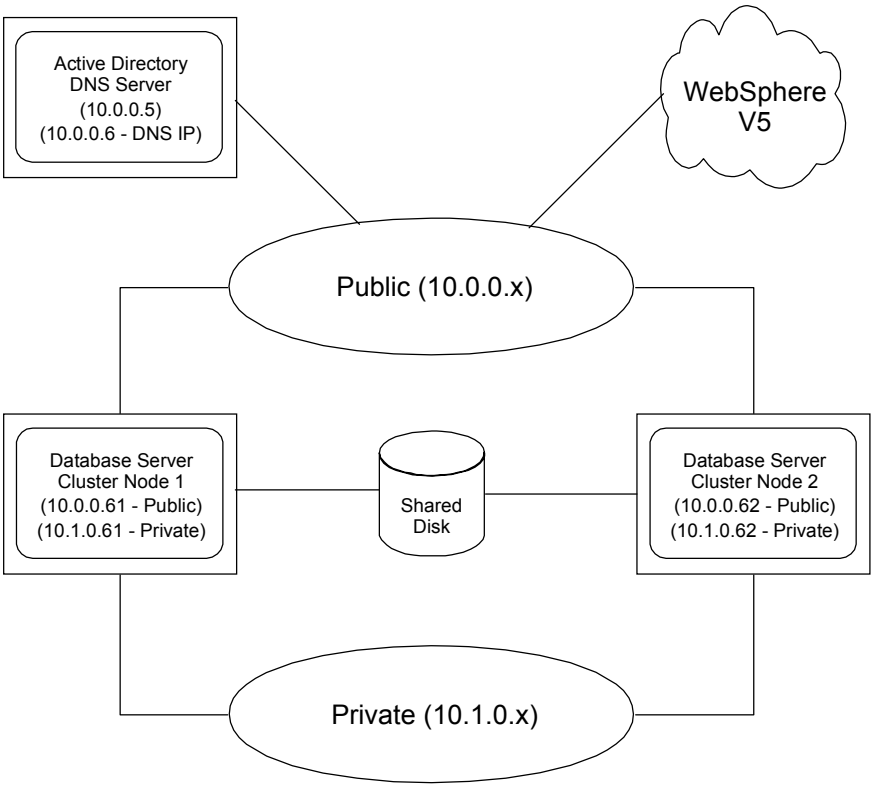


Figure 10.11: Microsoft Cluster Configuration

10.4.4Tuning heartbeat and cluster parameters

To ensure proper performance of the heartbeat communications, it is recommended that disabling any unnecessary network protocols, and ensure TCP/IP is configured correctly. It is also recommended that the "Media Sense" configuration option be disabled in Windows 2000. For more information, see the Microsoft Support Article **"**Recommended Private "Heartbeat" Configuration on a Cluster Server (Q258750)**"** at http://support.microsoft.com/default.aspx?scid=kb;en-us;Q258750.

Chapter 11 - LDAP Failover, High Availability and Scalability

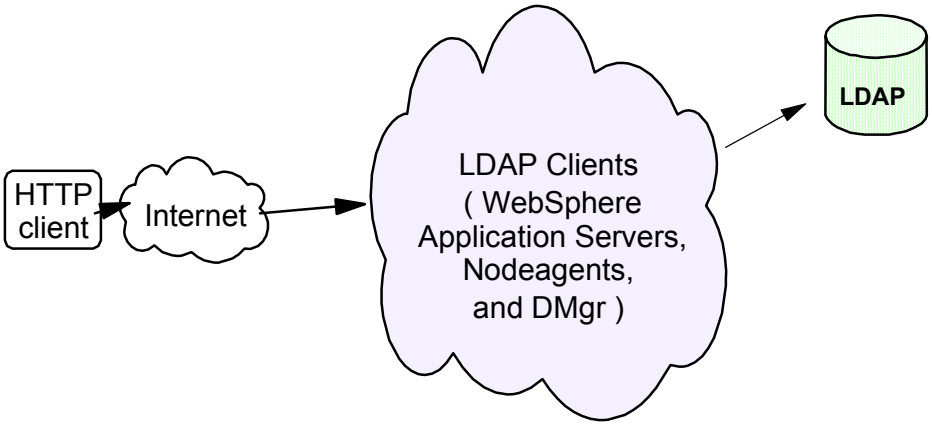


Figure 11.1 LDAP is a Single Point of Failure in the System

The WebSphere ND V5 uses LDAP for security and directory services. A single LDAP server can be a single point of failure. This chapter discuss how to build a highly available (and scalable) WebSphere LDAP-enabled system, based on hardware clustering, referral, replication, and IP sprayer.

11.1LDAP Server and High Availability

A directory is often described as a database, albeit one that is highly specialized. Directories have characteristics that set it apart from a general purpose relational databases. One characteristic is they are accessed (read or searched) much more frequently than they are updated (written). Lightweight Directory Access Protocol (LDAP) is a fast growing technology for accessing common information. LDAP is implemented in most network-oriented middleware. Building LDAP-enabled networks and applications is now common practice in many enterprises. WebSphere is dependent upon a highly available LDAP server, without which there is no access to directory and security data

11.2Clustered LDAP Servers

As in previous chapters, a highly available LDAP service can be enabled with clustering software such as HACMP, MC/Serviceguard, Microsoft Cluster Service, as shown in Figure 11.2. Two nodes are interconnected via public and private networks, and a shared disk is used to store common directory data. The private networks are dedicated to heart beat detection. Clustering software and LDAP are installed onto each node, and a resource group is created. One node can now fail over to the other, under the control of clustering software. A clustering IP address, instead of individual physical host IP addresses, is used to access the LDAP service. A clustering IP address is called service IP for HACMP, package IP for MC/Serviceguard, or logical host IP for Sun Cluster and Veritas Cluster. A clustering IP will be moved to the backup node under the control of clustering software, when the primary node fails. In such an event, the LDAP client (WebSphere) still utilizes the same IP address (clustering IP

address) to access the LDAP service on the backup node, as shown in Figure 11.3. You can configure whether the service failsback automatically to the primary node, once it is up again.

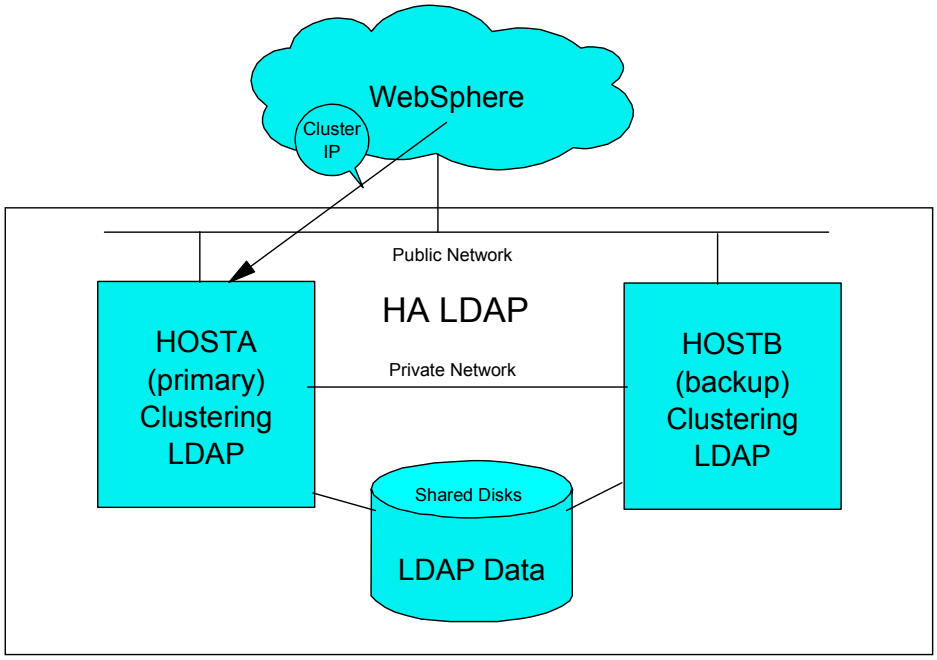


Figure 11.2 Clustered LDAP with shared disks configuration

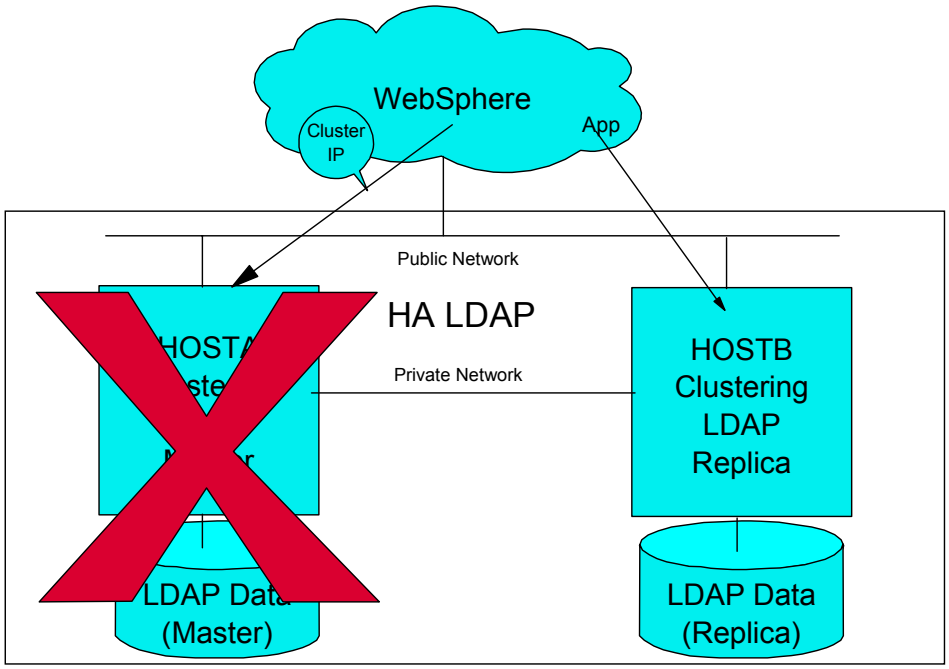


Figure 11.3 Clustered LDAP with shared disks after a failover

11.3LDAP Master and Replica Cluster

LDAP provides a master and replica architecture that makes it possible for the configuration of a highly available LDAP server, without needing shared disks on multiple hosts. Install clustering software on

two nodes, and configure the LDAP servers to use local data. A primary node is configured with an LDAP master server, and a backup node is configured with a replica, as shown in Figure 11.4. If a request is sent to the backup that impacts LDAP data, it is redirected to the primary LDAP server, since the replica server cannot change the data. However, the master server synchronizes its data with the backup periodically. When the primary server (master) becomes unavailable, the LDAP backup service becomes the master service under the control of clustering software, as shown in Figure 11.5.

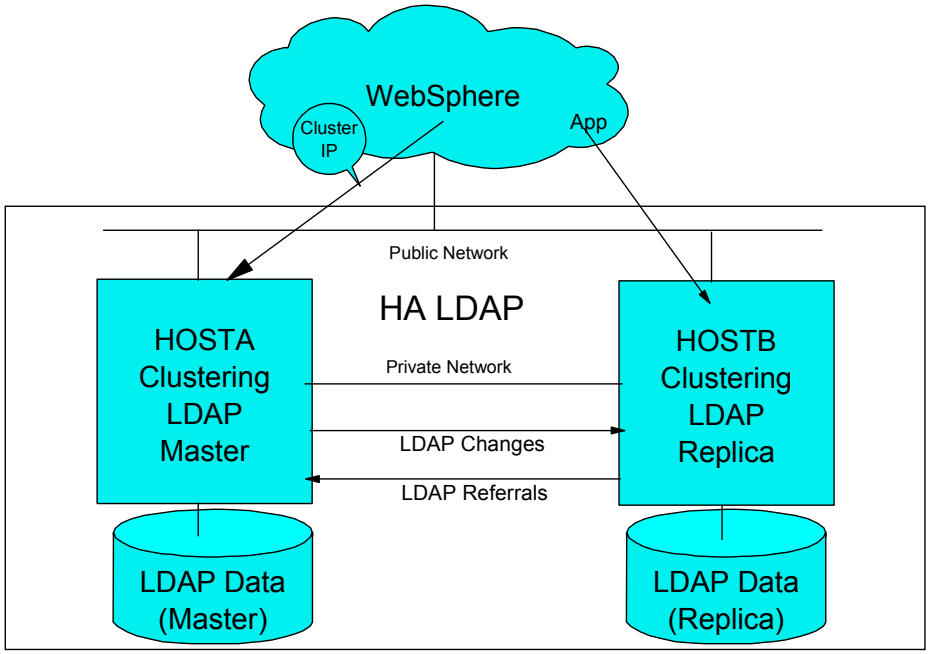


Figure 11.4 Clustered master-replica LDAP without shared disks

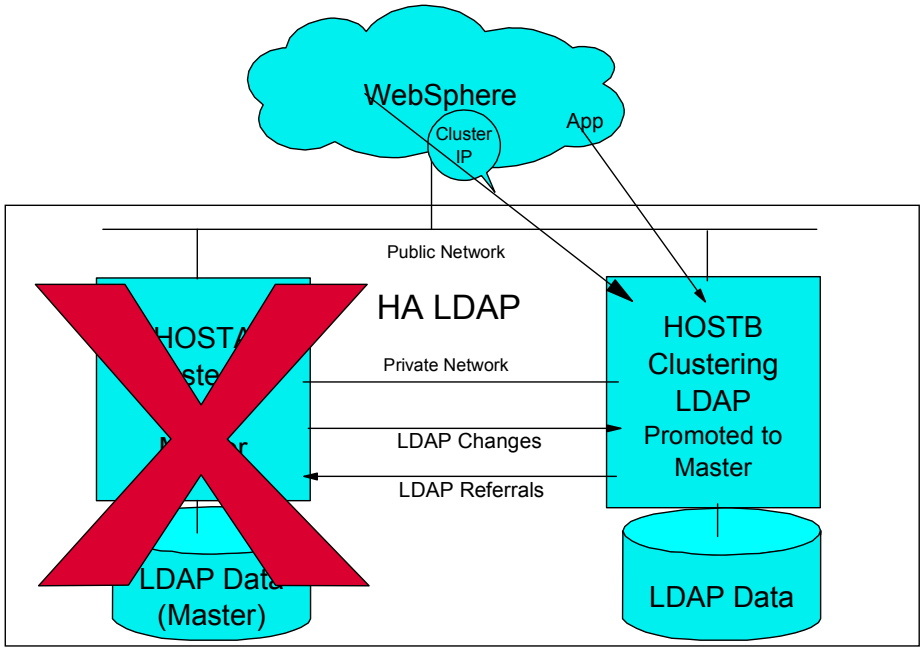
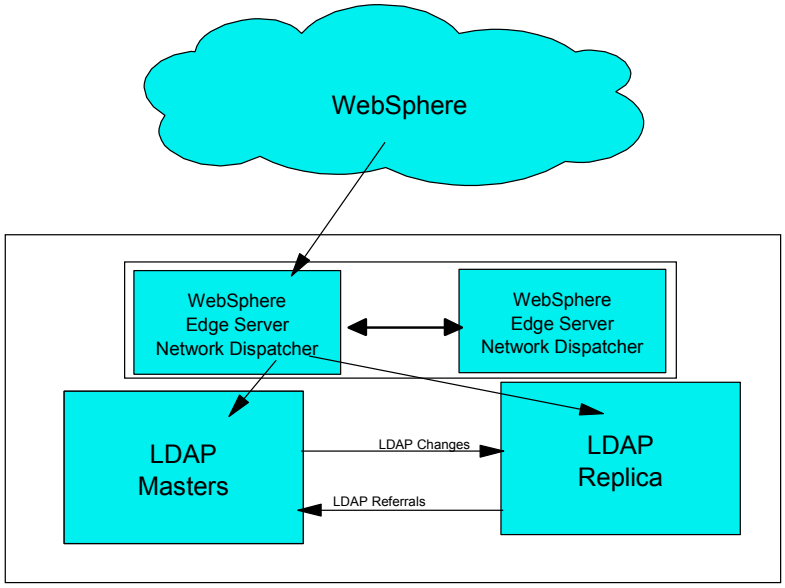


Figure 11.5 Clustered master-replica LDAP without shared disks after failover

The LDAP master/replica scenario just described need not use clustering software. While the backup process cannot be promted to the primary, it can still provide some level of service. However, in case of that the primary service is lost, all write operations will fail. Therefore, this permutation is discouraged.

11.4IP Sprayer-Based Clustering LDAP

A low-cost, easy-to-configure highly available LDAP server can be built with a network IP sprayer such as the IBM Edge Server Network Dispatcher, or a DNS server that has load balancing function (i.e. DNS Round Robin), as shown in Figure 11.6. Network Dispatcher distributs client requests to both servers. When a server fails, request will continue to be directed at the other server, as shown in Figure 11.7.



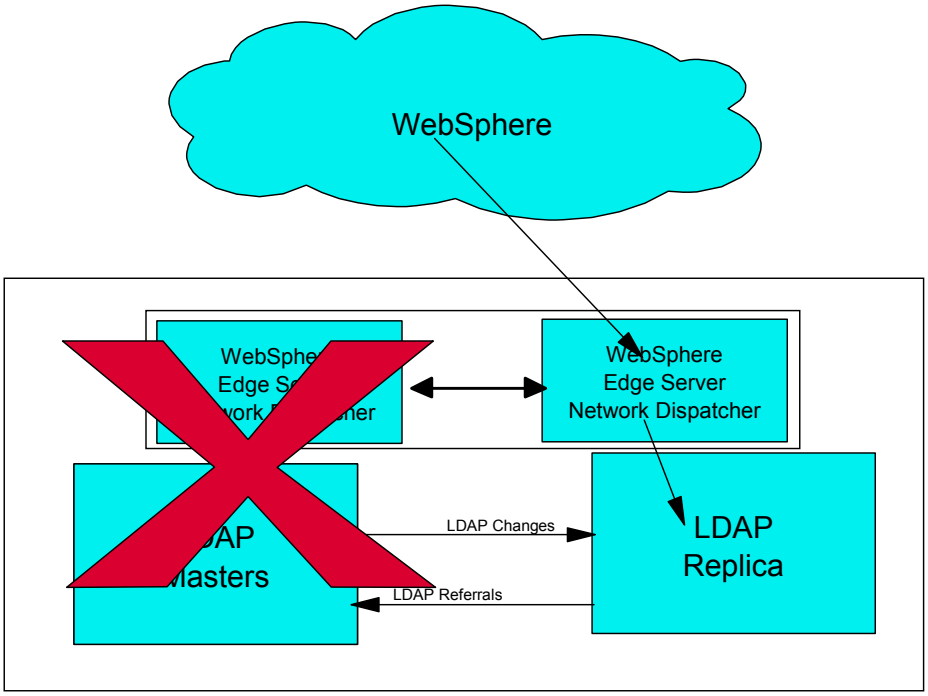


Figure 11.7 LDAP and ND configuration after failover

11.5 Combined ND and clustered LDAP for high availability and high scalability

For high-end enterprise applications, combined clustering software and network sprayer can improve LDAP availability and scalability by reducing downtime and providing more servers, as shown in Figure 11.8. During the clustering transition downtime, LDAP servers (read only) remain available. Partitioning directory structures enhance scalability, when used in conjunction with the approaches discussed in this Chapter.

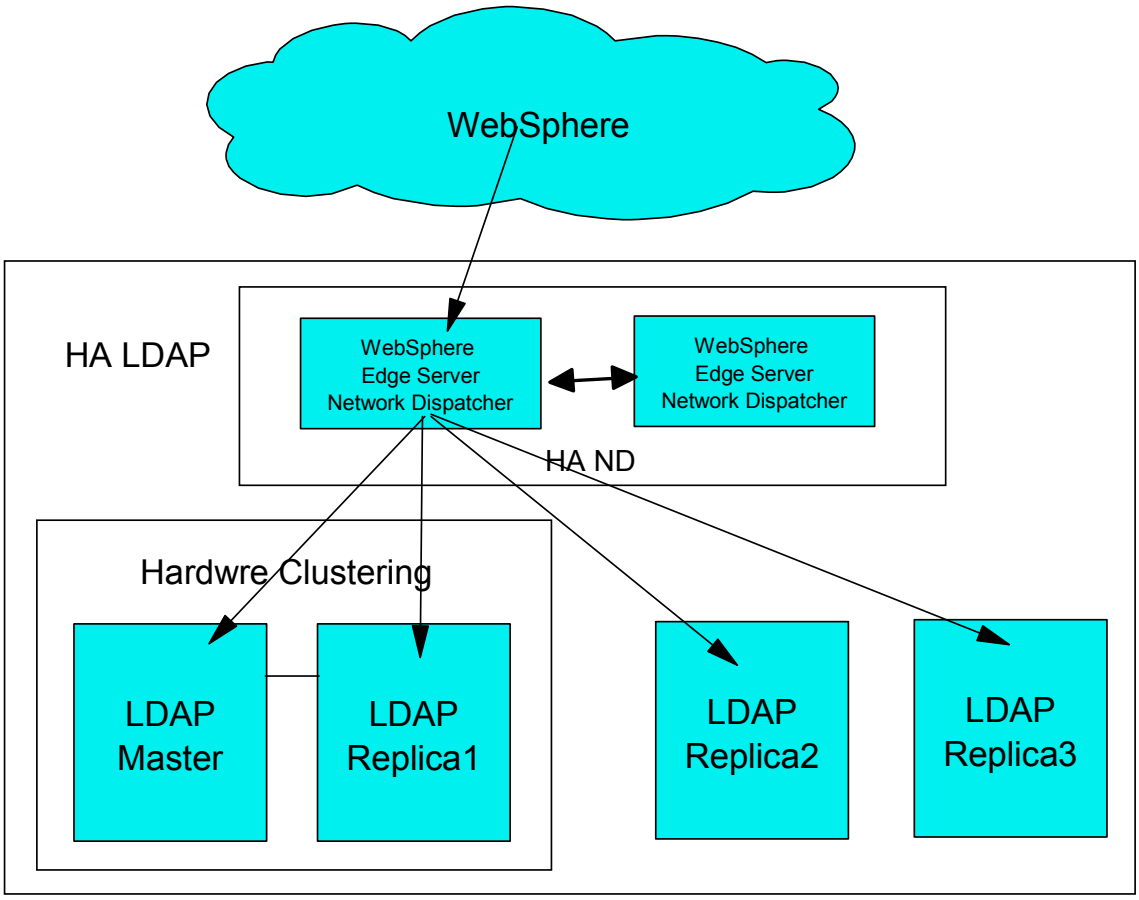


Figure 11.8 Combined ND and clustered LDAP for high availability and high scalability

Chapter 12 - High Availability for Firewalls and Network File Systems

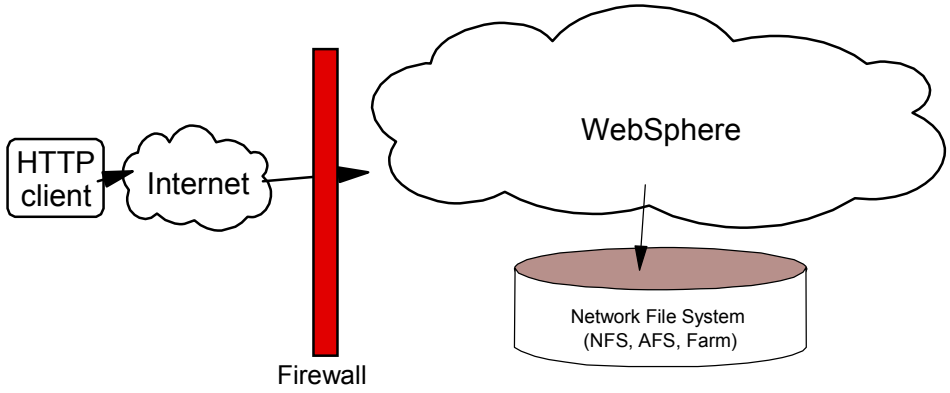


Figure 12.1: Firewall and NFS in WebSphere Topology as Single Points of Failure

Firewalls and Network File Systems (NFS) are critical to the proper functioning of any enterprise. This chapter addresses how to make these components highly available.

12.1Firewall Failover and High Availability

Any production web site includes at least one firewall. More common still are the use of two firewalls to create demilitarized zone (DMZ) to enhance the system security. The weakest point in any system is arguably the firewall. If the firewall fails, the DMZ and/or the enterprise is exposed to security risks, in particular mission critical data may be stolen or destroyed.

A highly available firewall can be configured with IBM SecureWay eNetwork firewall or CheckPoint VPN-1/FireWall-1, by employing two firewalls on separate hosts. The CheckPoint VPN-1/FireWall-1 product provides several high availability features, such as state synchronization of the firewall modules that allow active connections to continue after failover. However, there is no built-in mechanism in VPN-1/FireWall-1 to synchronize the security policy across two VPN-1/FireWall-1 management stations. The VPN-1/FireWall-1 management workstation is therefore a single point of failure.

In this section, we discus two advanced solutions:

Build HA firewall with clustering software such as HACMP, MC/Serviceguard, Sun Cluster or MSCS.

Build HA firewall with network sprayer such as IBM Edge Server Network Dispatcher

12.1.1Clustered Firewalls

As shown in Figure 12.2, clustering software such as HACMP, MC/Serviceguard, or MSCS can be used to provide a highly available service IP address, resource group, and fault-monitoring mechanism. For clustering configurations, please refer to previous chapters. Each node has a complete installation of IBM Secureway eNetwork Firewall, CheckPoint FireWall-1, or another firewall product. Configure both nodes such that there are equal and interchangeable configurations on both node. When firewall a process becomes unavailable, the clustering software will detect this and relocate the resource group, including the service IP address to a backup node, starting a firewall service, as shown in Figure 12.3. As soon as the primary firewall is up, the firewall service can failback to the primary node, as determined by the clustering configuration settings, or through manual means.

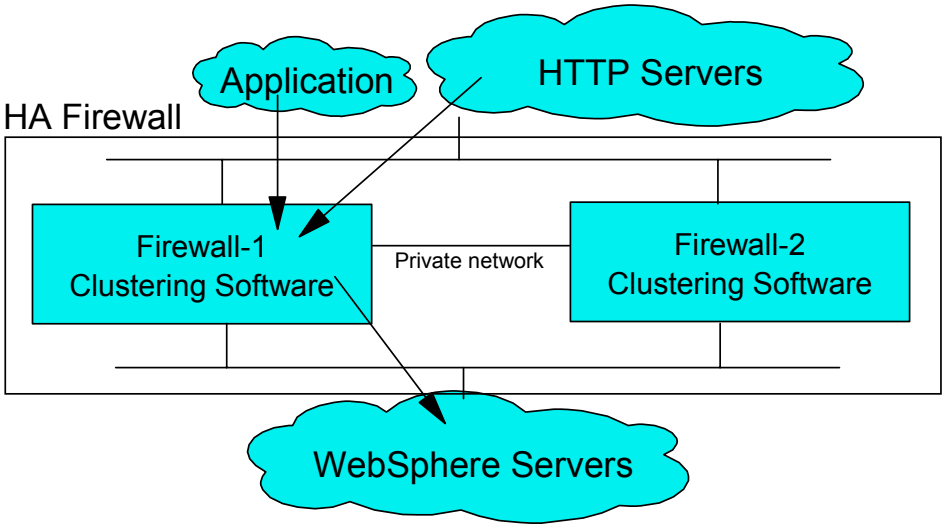


Figure 12.2 Clustered Firewall Configuration for High Availability

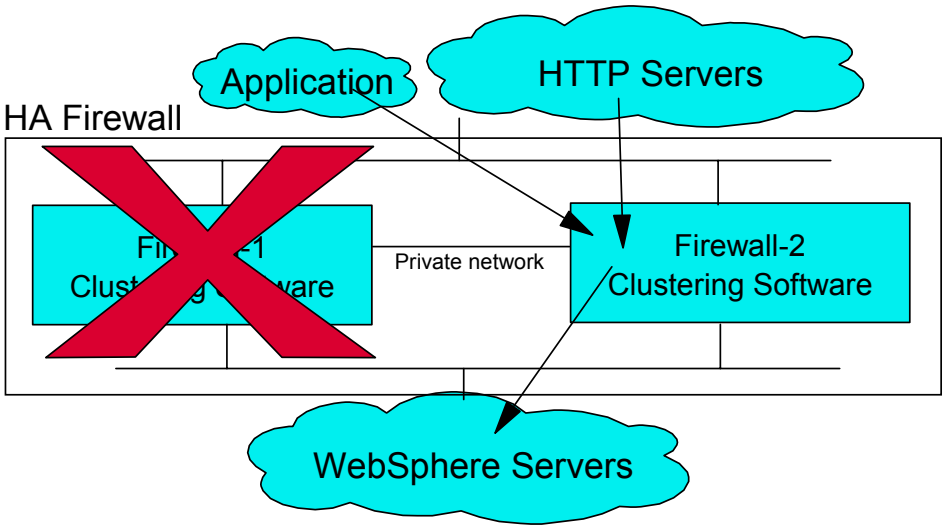


Figure 12.3 Clustered Firewall after failover

12.1.2Sprayer-based Firewall Clusters

The configuration discussed in the previous section can be very costly and is complicated. IBM Edge Server Network Dispatcher, as shown in Figure 12.4, can be used to provide a highly available firewall.

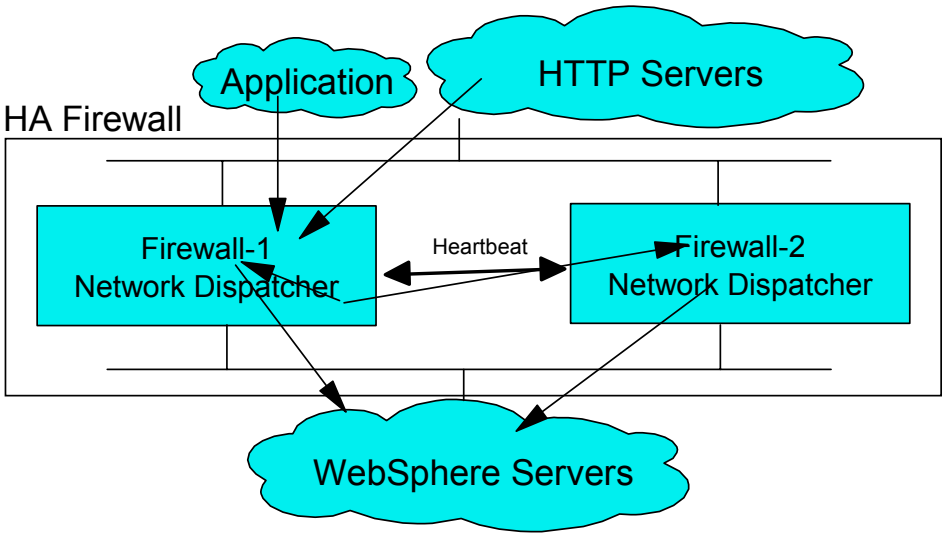


Figure 12.4 Network dispatcher configured firebwall

The Network Dispatcher is a load balancing product, which divides up the incoming workload to a group of servers. This is accomplished by either changing the assignment between the hostname and the IP address (DNS redirection) or by rerouting TCP and UDP traffic directly to the server with the lowest work load. The Dispatcher recognizes server failures and automatically keeps new requests from being dispatched to the failed server.

There are two approaches using IBM Edge Server Sprayer to build a highly available firewall environment: Interactive Session Support (ISS) and Network Dispatcher. The ISS is responsible for the load balancing in conjunction with a DNS server. Load balancing is achieved with either a round-robin mechanism or by calculating the load on servers from key system parameters, such as CPU load or memory utilization. A name server keeps DNS entries representing a group of available servers. ISS constantly monitors the workload on servers, and directs requests to a server with the lowest workload for some period of time.

ISS works fine with TCP/IP connections that are open for a few minutes, but will pose problems with very short connections, such as requests for static HTTP pages. After the Dispatcher starts routing requests to another server, the servers experiencing heavy loads will begin to completed requests, remaining idle until the Dispatcher calculates the next routing path. This approach will not be very efficient in regard to load balancing, but is easy to implement. A dedicated machine running as an ISS monitor is nlot needed, because monitors run on one of the production servers. Further, if this instance of the ISS server fails, another production servers is chosen as the new ISS server. This makes the system automatically highly available, as long as the DNS server does not fail.

A serious issue with ISS is application clients, and other DNS servers, which cache DNS entries returned by ISS. By default, these entries are cached for 24 hours, depending on the time to live (TTL) information for the DNS entry. The time to live can be changed to the average duration of connections. However, using short time to live intervals will increase the number of DNS requests to servers dramatically, therefore increasing load and network traffic. Keeping the load of your DNS server to a reasonable level requires (optimally) changes are propagated in 15 to 30 minute intervals, which is not adequate for highly availability. Using the name server module shipped with ISS, you can build up a DNS server to serve only ISS requests. However, this custom DNS server will be a single point of failure.There will not be other DNS servers which can be employed as a backup, since the ISS DNS server updates its zone information frequently. Hence, for this type of configuration, a secondary DNS servers is impractical, as due to synchronized between the primary ISS and secondary name server. While ISS provides a simple solution for load balancing, the drawbacks are it may not provide high availability.

In contrast to ISS functionality, where clients are exposed to the IP addresses of the application servers, is IBM's Network Dispatcher where client's use a common cluster IP address. The dispatcher calculates the workload of the servers by collecting metrcis such as the number of active and pending connections, or system information (collected by ISS software running locally on each server), such as CPU load or memory utilization. Since the workload distribution is calculated for every new connection, requests should go to the least loaded server. The Network Dispatcher can be a single point of failure in the system. To prevent this, the Dispatcher allows for the configuration of a backup server that automatically takes over in case of a failure. Load information and client routing tables are shared between the two Dispatcher servers, hence all requests should be preserved.

The Dispatcher executes shell scripts for startup and failover events, which configure the cluster IP addresses to a loopback device or a network card, depending on the state of the system. These scripts can be customized for a particular environment, and can include commands to start and stop application proxies and generate alerts when a failover occurs. Because Network Dispatcher only uses one TCP connection on a dedicated port for the heartbeat, there are few modifications needed in the firewall configuration.

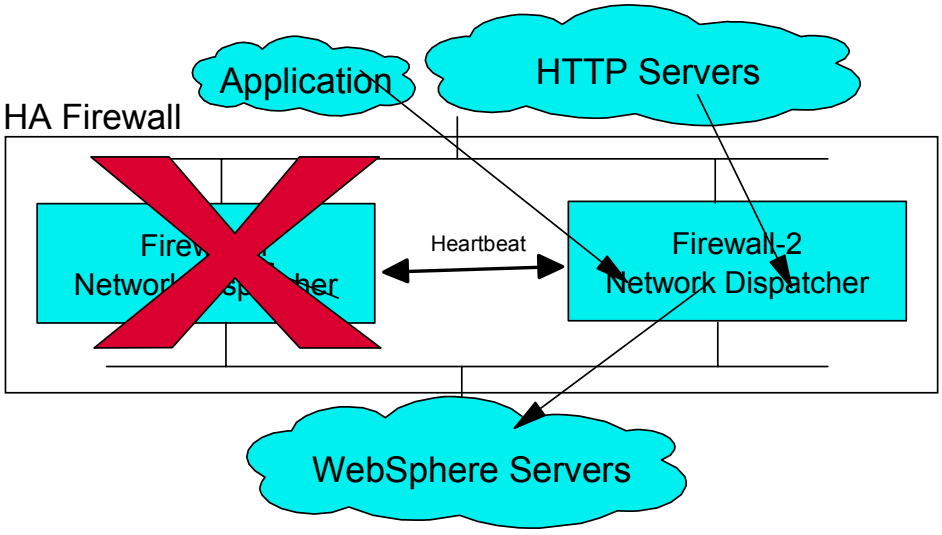


Figure 12.5 Network Dispatcher configured firewall after failover

In summary, HACMP is very complex, and needs to establish multiple connections between two servers, must be allowed by the firewall. If SSH is used for HACMP for these connections, there is the benefit of encryption and authentication. However, SSH can cause some HACMP scripts to report errors in cluster verification, which can be ignored. Configuration and troubleshooting of HACMP is not an easy and produces highly complex environments. Therefore, a firewall administrator must have a good understanding of HACMP.

12.2Network File System High Availability

A network file system is critical for the function of any enterprise application, i.e. for storage of data, files, and logs. Again, one can use clustering software such as HACMP, MC/Serviceguard, Sun Cluster to implement a highly available NFS server.

A file system that is exported by one node can be acquired by another through NFS cross mounts, i.e., one node mounts a file system from another. In the case where one node's file system server fails, there will be backup copy available.

A clustered system for NFS as shown in Figure 12.6. When clustering NFS with HACMP, do not use the standard /etc/exports, but a /usr/sbin/cluster/utils/cl_export_fs script. In an NFS file system, the server system passes a file handle to the client system, and a file handle consists of major device number, minor number, i-node, and a generation number. These four file handle components should be identical on each node that participates in the cluster, since if a failover occurs, the file handle held by an NFS client should still be valid on the new server. When we create a logical volume, the minor number, i-node number and generation number are the same on each node, since they are self-contained within the logical volume. However, the default major device number may be different since the logical volume inherits its number from the major device number of the volume group that contains it. The default major device number is assigned with the lowest unused number. In a cluster system, lowest unused major device numbers need not necessarily be the same. Major device numbers are assigned to TTYs, printers, disks, etc. Therefore, it is a key to configure the volume group in the shared disks with the same major device number for all nodes in the cluster, such that the file handles remain valid after failover.

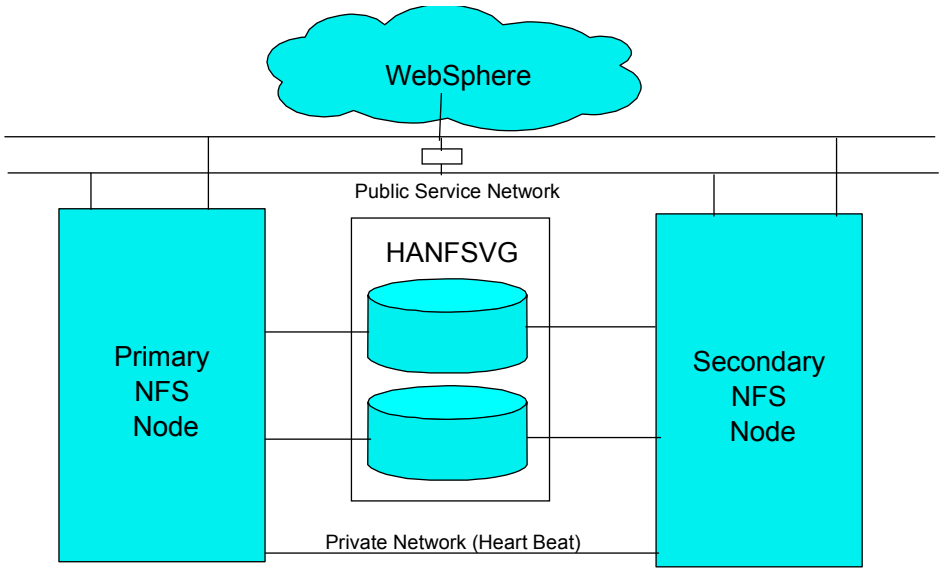


Figure 12.6 High Availability Network File System for WebSphere

Chapter 13 - Suggested Topologies

As the previous chapters have shown, there are many hardware and software configurations which provide high availability. This chapter presents the "gold standard" topology/configuration that provides the best, most flexible solution. Figure 13.1 depicts various possible layers in a WebSphere topology, and the mechanisms employed to provide high availabilit within these layers.

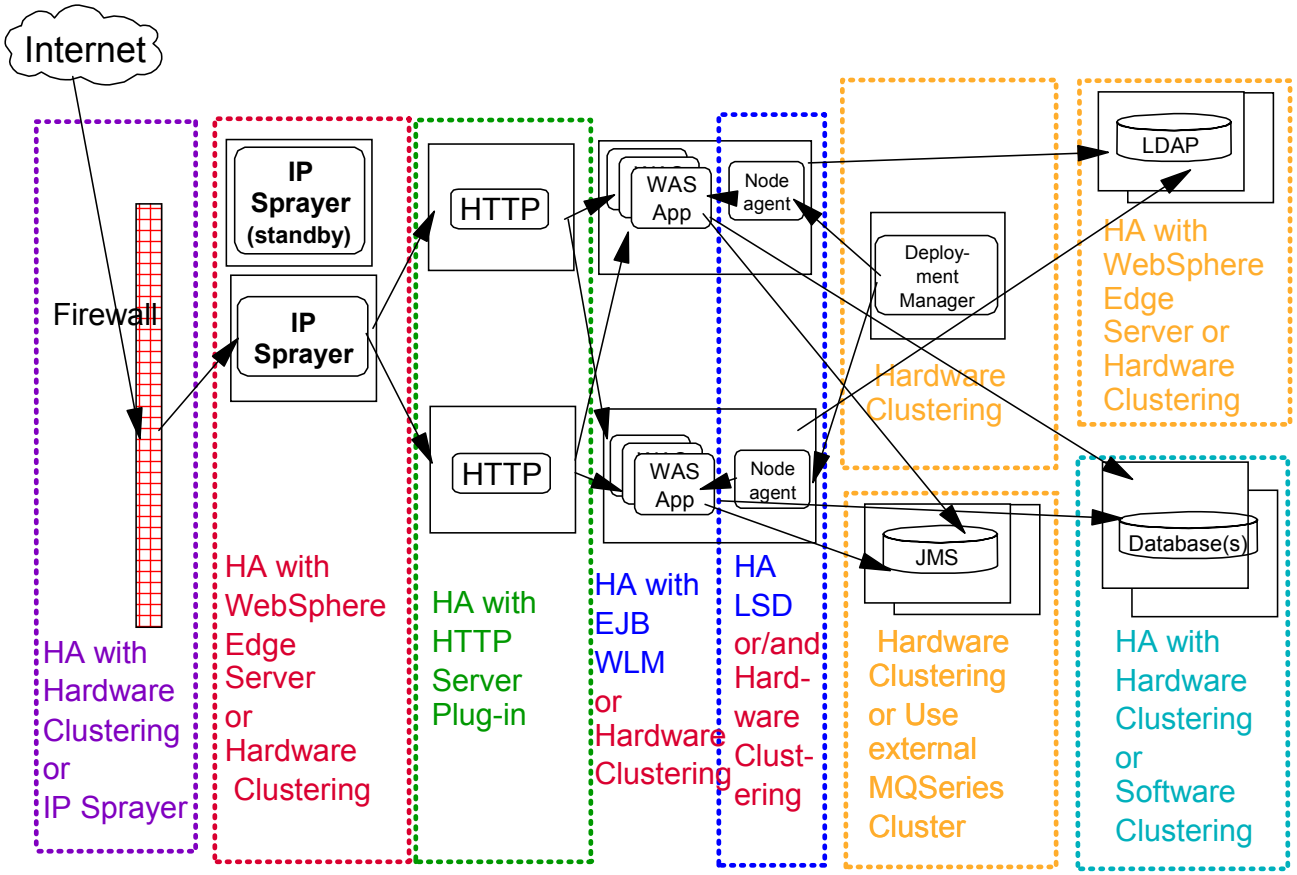


Figure 13.1: High Availability Mechanisms by Layer

This depicts a minimal topology for high availability, with a total of 16 servers from the firewall to the database/LDAP servers. While the number of servers could be reduced further, e.g. Network Dispatcher might be co-located on each HTTP server machine, but administrative and security issues generally preclude this. A more robust implementation is depicted in Figure 13.2.

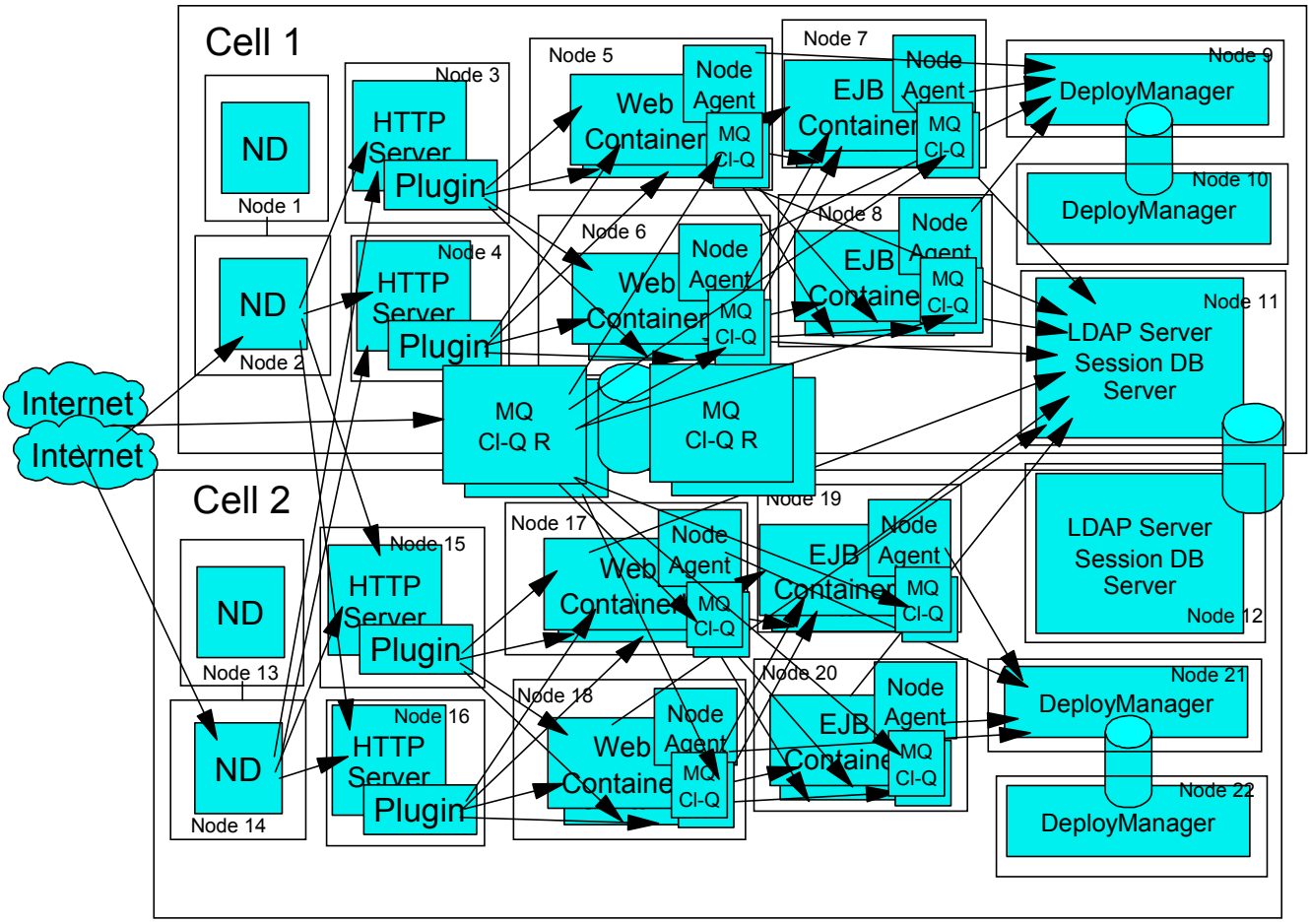


Figure 31.2: "Gold Standard" WebSphere Domain

While there are no "hard" limits on the number of nodes that can be clustered in a WebSphere domain. One may want to consider creating multiple WebSphere domains for a variety of reasons:

- Two (or more) domains can be employed to provide hardware and software failure isolation. This can come into play in a variety of situations:
  - Planned Maintenance
    - When deploying a new version of WebSphere.
    - When applying an e-fix or patch.
    - When rolling out a new application or revision of an existing application.

  - In cases where an unforeseen problem occurs with newly deployed software, multiple domains prevent a catastrophic outage to an entire site. A rollback to the previous software version can also be accomplished more quickly. Of course, multiple domains require software be deployed more than once

- Multiple smaller domains may provide better performance than a single large domain, since there will be less interprocess communication.

- WebSphere ND V5 requires repository synchronization, and uses JMX messaging for administration. Therefore a smaller domain may reduce message complexity and hence improve performance

Multiple domains will require more administration for day-to-day operations, which can be mitigated through the use of scripts (wsadmin). Multiple domains also requires multiple administration repositories, requiring multiple backup of data.

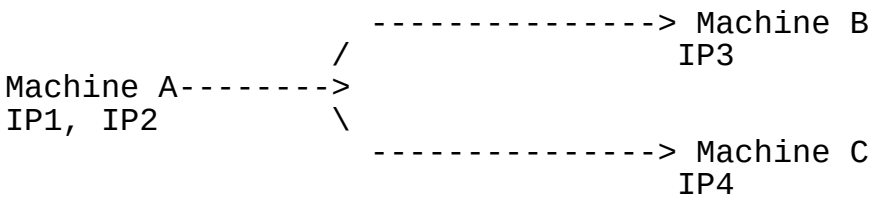
Appendix A - Installing Network Dispatcher on Windows NT

Introduction

This appendix discusses the installation of Network Dispatcher on an NT machine. The procedure for other platforms is similar. This configuration is based on ethernet. If a network employs token-ring, the procedure would be similar.

Instructions are also provided in the Redbook "WebSphere Edge Server: Working with Web Traffic Express and Network Dispatcher", SG24-6172-00, July 2001. Download from http://www.redbooks.ibm.com before starting the installation, as there will be references to that volume below.

Obtain 3 NT machines, and install:
Machine A: Network Dispatcher.
Machine B and C: one or more web servers



Obtain 4 IP addresses. All four IP addresses must be in the same subnet. For example:

1st IP: **25.34.145**.A1	This is the primary IP address of machine A
2nd IP: **25.34.145**.A2	This is the IP that is to be load balanced. It will be used by machine A. - also known as the **Cluster Address** or **Virtual IP Address** - typed in a web browser's URL field to initiate the HTTP request
3nd IP: **25.34.144**.B	IP address of machine B
4nd IP: **25.34.145**.C	IP address of machine C

Pre-Installation Setup

Machine A setup:
None needed, simply ensure the IP addresses are valid. The cluster address is only after Network Dispatcher is installed, and will not be added to the Network Properties on the NT system.

Machines B and C setup:

- **Install a loopback adapter on each web server machine**

Before installing ND on machine A, enable a loopback adapter configured with the Cluster Address that will be used by machine A. Refer section titled "***Configuration of the back-end servers***" in Chapter 3.2, page 95 of the Redbook. No additional hardware or software is needed to install a loopback

adapter. Once the loopback adapter is enabled, web server machines are equipped to accept requests sent to the cluster address.

- **Delete a route from the routing table**

Continue in the same section of the Redbook. Follow instructions to delete an extra route table

Installation of Network Dispatcher

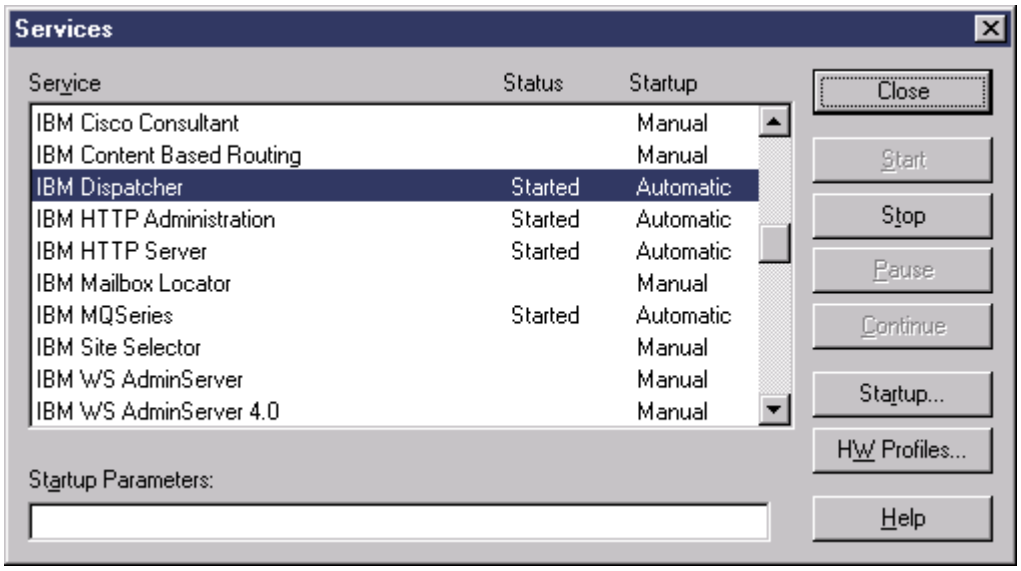
Currently, Network Dispactcher is packaged as a part of WebSphere Edge Server, which can be downloaded from http://www-4.ibm.com/software/webservers/edgeserver/

The Java Development Kit (JDK) 1.3 is a prerequisite. If there are instances of WebSphere V4.0, or higher, installed on the network, copy the directory C:\WebSphere\AppServer\java to Machine A, else download JDK 1.3 from http://www.ibm.com/developerworks/java/jdk/index.html. Follow instructions in the section titled **"Installing Network Dispatcher on Windows NT"** on page 40 of the Redbook to install the JDK. After installation, include the JDK 1.3. If there are other JDK instances on the system, make sure the JDK 1.3 is first in the PATH environment variable. Continue with the installation as described in the Redbook. At the end of the installation restart the system.

In the next section discusses configuration of the Dispatcher to accept requests for the Cluster Address (25.34.145.A2).

Post installation configuration of Dispatcher

Dispatcher starts automatically as a service. Before starting the configuration process, make sure that its status is "Started" by viewing the **Services** menu in the **Control Panel**.



- **Create Keys**

Open a command prompt and run the command:

```
C:\create ndkeys
```

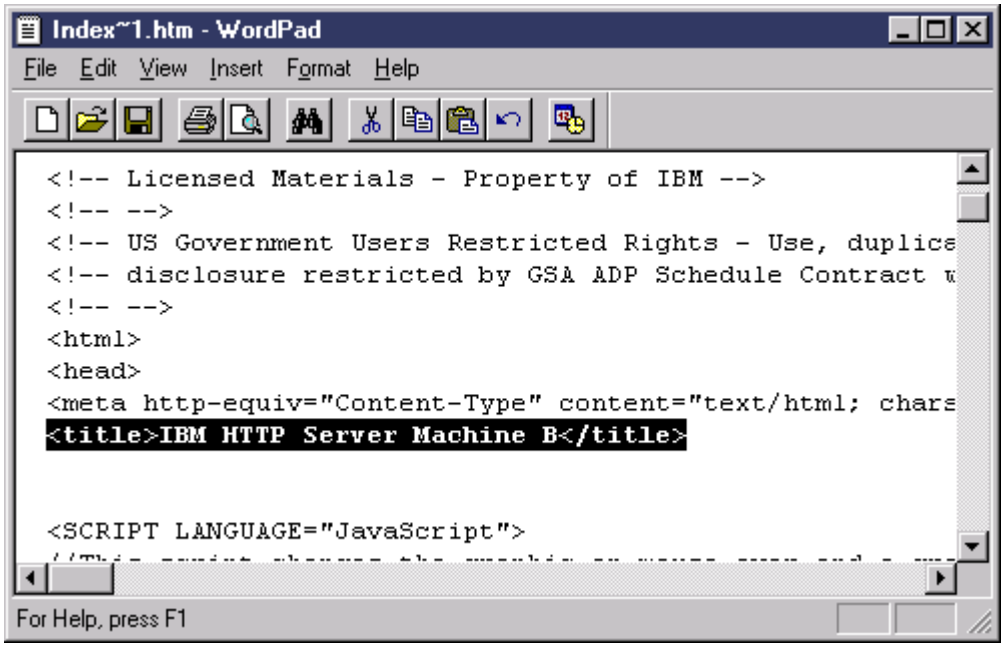
The successful completion message is: “Key files have been created successfully.”
This will create a key file required for administration purposes.

- **Configure the Web Server cluster**

Configure Dispatcher to receive requests on behalf on the Cluster Address, using a command line utility called **ndcontrol** or using a GUI based tool. The command line tool is useful for the automation of administrative tasks. Refer to section 4.1.2 of the Redbook, entitled “Configuration”, on page 71.
The configuration consists of the following steps:
1. Create a cluster for the IP address 25.34.145.A2
2. Create a port number to be load balanced, in most cases, this is the default HTTP port, 80
3. Add servers to which Dispatcher will forward the HTTP requests (B and C, in this example).

After finishing the configuration, test basic functionality:

1. Start the web servers on machines B and C. Make sure default html pages are accessable on the web servers by pointing a web browser directly to their IP addresses. In this example a IBM HTTP Server was used, and the **<title>** tag in the file C:\IBM HTTP Server\htdocs\index.html was edited on both machines to uniquely identify which web server machine was accessed using the Dispatcher. For example:



2. Open a web browser and type in the URL corresponding to the Cluster Address. If everything is installed and configured correctly, the HTML page will render, indicating either machine B or C was accessed. Refreshing the page should result in both machines being accessed.

Appendix B - Configuring TCP Timeout parameters by OS

Windows NT/2000- On Windows NT/2000, the TCP timeout value defaults to 3000 milliseconds. However, the operating system retries the each connection three before failing. Each subsequent retry interval is doubled. Hence, with a default of 3 seconds, 3+6+12 or 21 seconds, must elpse before the machine is deemed unreachable with a given connection.

In Windows NT Service Pack 5 and higher, or Windows 2000, the following procedure can be used to view/modify this value:

1. Start Registry Editor (Regedt32.exe).
2. Locate the following key in the registry:

   HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

3. On the Edit menu, click Add Value, and then add the following registry value:

   Value Name: InitialRtt
   Data Type: REG_DWORD
   Value: <value> decimal (where <value> is the timeout value in milliseconds. For example, 5000 to set it to 5 seconds)
   Valid Range: 0-65535 (decimal)
   Default: 0xBB8 (3000 decimal)
   Description: This parameter controls the initial retransmission timeout used by TCP on each new connection. It applies to the connection request (SYN) and to the first data segment(s) sent on each connection.

4. Exit Registry Editor.
5. Restart the computer.

AIX - On AIX, the TCP timeout value is specified in half-seconds, and can be viewed or modified using the network options (no) command. The default value is 150 half-seconds (75 seconds).

To view the timeout value enter the command

    /usr/sbin/no -o tcp_keepinit

To set the timeout value enter the command

    /usr/sbin/no -o tcp_keepinit=<value>

where <value> is the timeout in half-seconds. This is a runtime value, which must be set every time the machine reboots.

Solaris - On Solaris, the timeout value defaults to 180000 milliseconds, or 3 minutes. This value can be viewed or set using the ndd command.

To view the timeout value enter the command

ndd -get /dev/tcp tcp_ip_abort_cinterval

To set the timeout value enter the command

ndd -set /dev/tcp tcp_ip_abort_cinterval <vablue>

where <value> is the timeout in milliseconds.

HP-UX - On HP, the timeout value defaults to 75000 milliseconds, or 75 seconds. This value can be viewed or set using the ndd command.

To view the timeout value enter the command

ndd -get /dev/tcp tcp_ip_abort_cinterval

To set the timeout value enter the command

ndd -set /dev/tcp tcp_ip_abort_cinterval <value>

where <value> is the timeout in milliseconds.

Appendix C - MC/ServiceGuard setup instructions

- Install MC/Serviceguard software oo each node with swinstall, choosing the B3935DA package. For MC/Servicegard installation details, see the MC/Serviceguard software installation manual.

- Configure and update each node for the MC/Serviceguard cluster

  Give security permissions for both machines by adding entries into */etc/cmclustercmclnodelist* file:

  Hp1.somecorp.com root # WebSphere database cluster
  Hp2.somecorp.com root # WebSphere database cluster

  Non-root users who wish/need to run cmviewcl, should have their user IDs placed in this file.

  Define the name resolution service. By default, MC/ServiceGuard uses */etc/resolv.conf* to obtain the addresses of the cluster nodes. In case DNS is not available, configure the */etc/hosts* and */etc/nsswitch.conf* files to search the */etc/hosts* file when other lookup strategies are not working.

- Set up and Configure the shared disk array

  Connect the shared disk array to both nodes.

  Create volume groups, logical volumes and mirrors by `pvcreate, vgcreate, vgextend, lvcreate, lvextend.`

  Create cluster lock disks.

  Distribute volume groups to other nodes. Distribute volume groups either by SAM or by LVM commands

- Configure MC/Serviceguard cluster for WebSphere databases

  Use SAM, select Cluster -> High Availability Cluster
  Choose Cluster Configuration
  In the Selection Actions menu, choose create cluster configuration, and follow the instructions
  Verify the cluster configuration by

  `cmeckconf -k -v -C` */etc/cmcluster/webspheredb2.config* for IBM DB2
  `cmeckconf -k -v -C` */etc/cmcluster/websphereoracle.config* for Oracle

  Distribute binary configuration files to other nodes via SAM or by the command line
  Backup volume group and cluster lock configuration data, since it may need to be restored at a later time

- Configure packages and their services

Install DB2 or Oracle in both machines and LDAP onto the shared disk, create needed database instances
Use SAM to configure packages
Customizing the package control scripts for vg activation, service IPs, volume groups, service start, and service stop. Since the control scripts are very long, we give key functions of sample scripts for DB2 and Oracle below.

For DB2, our sample service start script is

```
function customer_defined_run_cmds
   {
         su - db2inst4<<STARTDB
         Db2start
         STARTDB
   test_return 51
   }
```

And our sample DB2 service stop script is

```
function customer_defined_halt_cmds
   {
         su - db2inst4<<STOPDB
         db2 force applications all
         sleep 1
         Db2stop
         STOPDB
   test_return 52
   }
```

For oracle, our sample service start script is

```
function customer_defined_run_cmds
   {
         su - oracle<<STARTDB
         lsnrctl start
         export SIDS="APP ADMIN SESSION"
         for SID in $SIDS ; do
         export ORACLE_SID=$SID
         echo "connect internal\nstartup\nquit" | svrmgrl
         Done
         STARTDB
   test_return 51
   }
```

And our sample Oracle service stop script is

```
function customer_defined_halt_cmds
   {
         su - oracle<<STOPDB
         export SIDS="APP ADMIN SESSION"
         for SID in $SIDS ; do
         export ORACLE_SID=$SID
         echo "connect internal\nshutdown\nquit" | svrmgrl
         Done
         lsnrctl stop
         STOPDB
```

```
test_return 52
}
```

Distribute the package configuration via SAM

- Verify that cluster operation and configuration. Make sure all settings are what was intended, and that there are no errors in the logs. Also ensure the following are functioning properly

  Network, and network heartbeat mechanisms
  Nodes
  Services such as DB2, Oracle, and LDAP

- Verify system failover from SAM by moving packages from one node to another node.

Appendix D HACMP setup instructions

- Install HACMP 4.3.1, 4.4 or HACMP/ES 4.4 ptf3

  Use smit to install HACMP 4.3.1, 4.4 or HACMP/ES 4.4pft3 onto both nodes. For installation details, please see the HACMP for AIX Installation Guide. You also can install HACMP after you configure network adapter and shared disk subsystem.

  Before you configure HACMP, network adapters must be defined and the AIX operating system must be updated. Assign permission for nodes in the cluster to access one another. Modify the following configuration files: */etc/netsvc.conf, /etc/hosts,* and */.rhosts*. Make sure that each node's service adapters and boot addresses are listed in the */.rhosts* file, on each cluster node, so that the */usr/sbin/cluster/utilities/clruncmd* command and the */usr/sbin/cluster/godm* can be executed.

- Service network configuration

  Public networks are used to provide services to clients (WebSphere, Applications, LDAP), hence define two TCP/IP public networks. Public networks consists of a service/boot adapter and any standby adapters. It is recommended for using one or more standby adapters. Define standby IP addresses and boot IP addresses. For each adapter, use smit mktcpip to define IP label, IP address, and network mask. HACMP will define the service IP address. Since such configuration processes also changes hostname, configure the adapter with the desired default hostname last. Use smit's chinet to change service adapter so that they will boot from the boot IP addresses. Check the configuration by `"lsdev -Cc if".` Finally try to ping nodes to test the public TCP/IP connections.

- Serial network configuration

  A serial network is needed to exchange heartbeat messages between the nodes in a HACMP cluster, and allows a Cluster Manager to continuously exchange keepalive packets should the TCP/IP-based subsystem, networks, or network adapters fail. The private network can be either a raw RS232 serial line or target mode SCSI or SSA loop. The HACMP for AIX serial line (a null-model, serial to serial cable) is used to connect the nodes. Use smit tty to create the tty device. After creating the ttydevice on both nodes, test communication over the serial line by entering the command stty </dev/ttyx on both nodes (where /dev/ttyx is the newly added tty device). Both nodes should display their tty settings and return to prompt if the serial line is OK. After testing, define the RS232 serial line to HACMP for AIX.

- Shared disk array installation and LVG configuration

  The administrative, application, session, and LDAP data, log files and other file systems that need to be highly available are stored in the shared disks that use RAID technologies or are mirrored to protect data. The share disk array must be connected to both nodes with at least two paths to eliminate a single point of failure. We use IBM 7133 Serial Storage Architecture (SSA) Disk Subsystem.

Configure the shared volume group to have either concurrent or nonconcurrent access. Nonconcurrent access environment typically uses journaled file systems to manage data, while concurrent access environments use raw logical volumes. There is a graphical interface called TaskGuide to simplify creating shared volume groups within an HACMP cluster configuration. In version 4.4, the TaskGuide has been enhanced with automatic creattion of JFS logs and displaying the physical location of available disks. After one node has been configured, import volume groups to the other node by using smit importvg

- DB2 or Oracle and LDAP installation, configuration, instance and database creation

  For installation details, see the manuals for these products. You can install these products on the disks of both nodes, or a shared disk. All the shared data, such as database files and transaction logs, must be kept on the shared disk array so that any node can access these data when the primary node fails. We recommend installing these products on each node, in which case, one must install the same version products on both nodes.

  Create DB2 or Oracle instances on the shared disk subsystem. We created three DB2 instances for administrative database, application database, session database for WebSphere. In other tests with Oracle, we also created three Oracle Instances for administrative database, application database, session database for WebSphere. One may need to modify applheapsz for the created DB2 database or cursor parameters for Oracle. See the WebSphere installation guide for details.

  Install database clients on the WebSphere nodes, and configure database clients to connect to database server if "thick" database clients are used. For example, install DB2 clients on all WebSphere nodes and catalog the remote node and database server.

- Define the cluster topology and HACMP application servers

  The cluster topology is comprised of cluster definition, cluster nodes, network adapters, and network modules. The cluster topology is defined by entering information about each component in HACMP-specific ODM classes. These tasks can be done by using smit HACMP. For details see the HACMP for AIX Installation Guide

  An "application server", in HACMP or HACMP/ES, is a cluster resource that is made highly available by the HACMP or HACMP/ES software. For example, DB2 or Oracle databases, and LDAP servers. Hence do not confuse HACMP application servers with WebSphere application servers. Use smit HACMP to define the HACMP (HACMP/ES) application servers by a name, and its start script and stop script.

- Start and stop scripts for both DB2 and Oracle as the HACMP application servers

  Sample DB2 service start script:

  ```
  db2start
  ```

  Sample DB2 service stop script:

  ```
  db2 force applications all
  ```

```
db2stop
```

For oracle, the sample service start script is

```
lsnrctl start
export SIDS="APP ADMIN SESSION"
for SID in $SIDS ; do
export ORACLE_SID=$SID
echo "connect internal\nstartup\nquit" | svrmgrl
Done
```

Sample Oracle service stop script:

```
export SIDS="APP ADMIN SESSION"
for SID in $SIDS ; do
export ORACLE_SID=$SID
echo "connect internal\nshutdown\nquit" | svrmgrl
done
lsnrctl stop
```

You must be db2 or oracle users to use the above scripts, otherwise you need to become such user by su - <super-username>

- Define and configure resource groups

  The resource group is configured, using three kinds of node relationships: cascading, concurrent access, or rotating. For the cascading resource group, setting cascading without fallback (CWOF) attribute will minimize the client failure time. This configuration was in our tests. Use smit to configure resource groups and resources in each group. Finally, synchronize cluster resources on the nodes.

- Cluster verification

  Use /usr/sbin/cluster/daig/clverify on one node to check that all cluster nodes agree the cluster configuration, and assignment of resources, looks consistent. Use smit HACMP to verify all nodes agree on the cluster topology. If not, and one wishes to define the cluster as it appears on the local node, force agreement of cluster topology onto all nodes by synchronizing the cluster configuration. After the cluster verification is OK, start the HACMP cluster services by using smit HACMP on both nodes. Monitor the log file by $tail -f /tmp/HACMP.out, and check database processes by $ps -ef | grep db2 or ora

- Failover verification

  To test failover, use smit HACMP to stop cluster service with the failover option. On the other node, enter the following command to watch the failover activity: #tail -f /tmp/HACMP.out

  There are several places to find log information: */usr/adm/cluster.log* provides a high-level view of current cluster status. This is the best place to look first when diagnosing a cluster problem. The /tmp/HACMP.out log is the primary source of information when investigating a problem.

One can configure a remote machine to be able to connect to a HACMP cluster and use clstat (or xclstat) to monitor HACMP clusters.

Appendix E - Microsoft Clustering Setup Instructions

The installation process must be followed exactly as presented here in order to assure accurate installation of the products. Read through the entire procedure and make sure you understand each aspect before starting the process. Once you start, it will be difficult to go back without starting from scratch.

Verifying the Hardware/Software

Before installing any aspect of this environment, you want to verify that the hardware and software you are about to work with is the proper version.

The following links provide a list of prerequisites necessary for this installation.

- Hardware Compatibility List (HCL) for Microsoft Windows 2000 Advanced Server
  http://www.microsoft.com/windows2000/server/howtobuy/upgrading/compat/default.asp
- HCL for Microsoft Clustering Service
- http://www.microsoft.com/hcl/default.asp (Search on "cluster")
- SQL Server Enterprise Edition Prerequisites
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/instsql/in_overview_74vn.as
  NOTE: You must use SQL Server Enterprise Edition to get Clustering support. The Developer Edition and Professional Edition are not cluster aware.

Setup SCSI Hardware for Shared Disk

The IBM EXP 15 disk array was used as the shared disk, when testing this configuration. In order for both nodes to utilize the disk array appropriately, SCSI cards in each node were configured to avoid a conflicts. For example, all of the DIP switches were set to off, which provided one chain of 10 drives in the array. On node 1 of the cluster, the SCSI ID was set (in the SCSI BIOS) to 6 and on the other node to 7. This was the only change from the default SCSI setup we needed to do.

Installing Microsoft Clustering Services

Once you have all of your prerequisites accounted for, installing the Microsoft Clustering Services in Windows 2000 is fairly straightforward. Read through the documents provided in the Useful Resources section of this document to learn more about the process, if there are any questions.

During setup of our environment, we followed the **Microsoft Clustering Step-by-Step guide** (see Useful Resources). A list of additional notes were gathered while running through this process:

1. Make sure there is a server running in the Active Directory service, and a DNS server. This is necessary for adding the cluster nodes to a Windows 2000 domain and for providing domain wide user accounts, that many of these services will need to run under.
2. Add the Cluster IP Address and Hostname to the DNS, or hosts file, of the backup node allowing it to find the active cluster during installation.

3. SQL Server depends on the Microsoft Distributed Transaction Coordinator for distributed queries and two-phase commit transactions, as well as for some replication functionality. Iinstall MSDTC by opening a command prompt and running **comclust.exe** on each node in the cluster. (comclust.exe can be found in the Winnt\system32 directory.)

Installing IBM DB2 Enterprise Extended Edition on the Clustered Windows Servers

Installing IBM DB2 in a Windows 2000 cluster is a little different than installing on a single node. Follow the instructions in the IBM Whitepapers on Implementing IBM DB2 on a Windows Cluster found in the resources section along with the IBM DB2 documentation for installing to a cluster. Reading both of the papers (2 node and 4 node), since they each have information not covered in the other.

A list of notes were gathered while running through this process:

1. When creating a new datasource, or WebSphere repository which will use the databases on the cluster, be sure to catalog the database using the Virtual IP address of the DB2 servers as the remote host IP.
2. Add the virtual name/IP, identified in the DB2 Server setup, in the DNS server or hosts files on all nodes that need to have availability to this cluster.
3. The account used for the DB2 Server service should be a domain login.
4. When running through the Verification tests (later in this paper), be sure to set the DB2 Environment variable DB2_FALLBACK to ON using db2set DB2_FALLBACK=ON otherwise the system may not failover when there is a client is connection.
5. Be sure to add the DB2MPP-1 service as a clustered resource, if it is not done through the DB2MSCS tool.

Installing SQL Server on the Clustered Windows Servers

Installing SQL Server in a Windows 2000 cluster is a little different than installing on a single node. Follow the instructions in the SQL Server documentation for installing to a cluster.

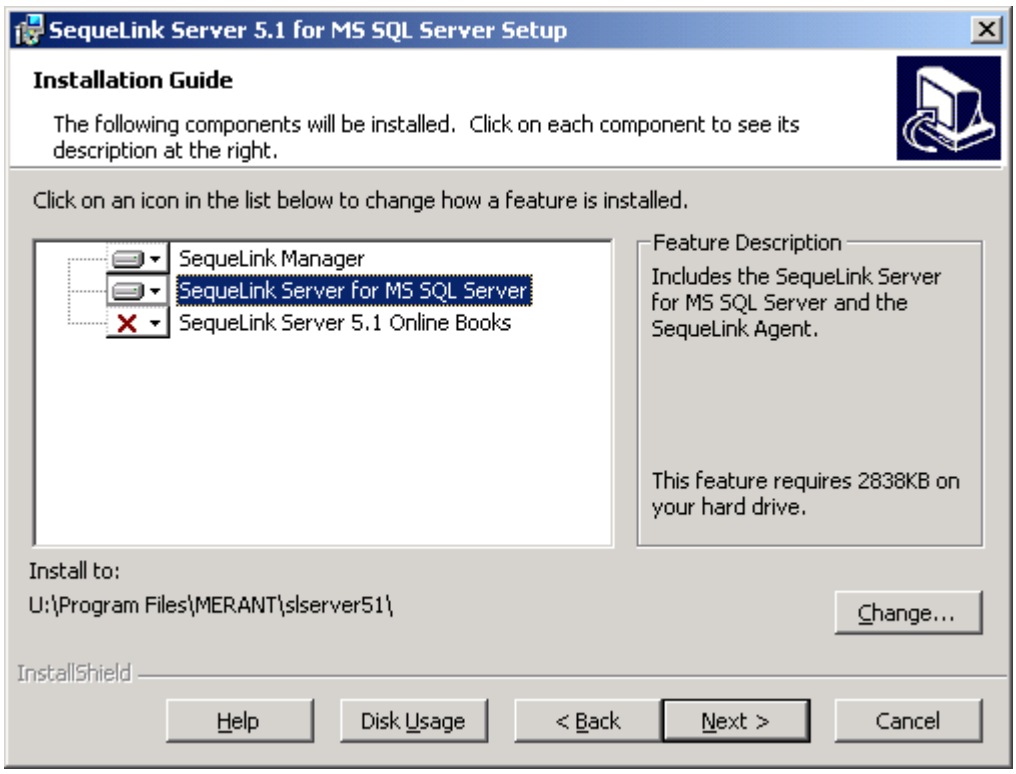
A list of additional notes were gathered while running through this process:

1. Make sure nodes can see each others c$ shares before starting the install. This requires having the "File and Print Sharing" installed on the network driver.
2. Add the virtual name/IP, identified in the SQL Server setup, to the DNS server or hosts files on all nodes that need to have availability to this cluster.
3. The login for SQL Server should be a domain login.
4. Use both the SQL Server authentication and NT authentication to allow for the WebSphere JDBC driver to login.

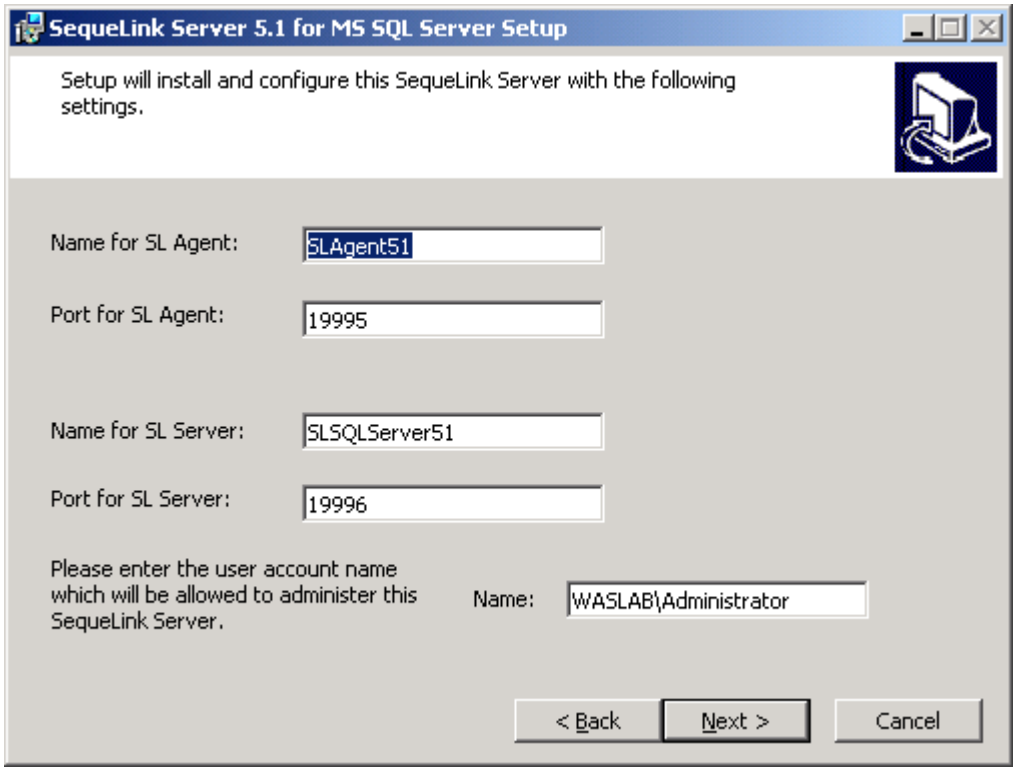
Installing Merant SequeLink Service on the SQL Cluster Servers

After installing Microsoft SQL Server on the cluster, install the Merant SequeLink Server on both nodes of the SQL Server cluster. Since the Merant SequeLink Server is not cluster aware, you need to install it separately on each node of the cluster.

1. Download the Merant SequeLink Server from the WebSphere e-fix page. (A link to this page can be found in the Useful Resources section at the end of this document.)
2. Unzip the Merant SequeLink files to a temporary directory.
3. Run the SequeLink setup by opening a command prompt and executing **setup /v"IPE=NO"** (This command prevents the setup from asking for a registration key.)
4. Accept the license.
5. At the following step in the install, change the drive that the Merant server is installed on, to the shared drive. Here the shared drive is drive U.



6. The next step asks for the Agent and Server name/port, and an account to administer the server. Leave the default settings, unless this would cause a port conflict or there are security requirements that force a change. The most important aspect of this page is the user account. Be sure to enter an account from the domain, not from an indivdual computer. This is so that when a failover occurs the account will still have the proper permissions to administer the new node, and ensures this account will be available after a failure of the primary node.



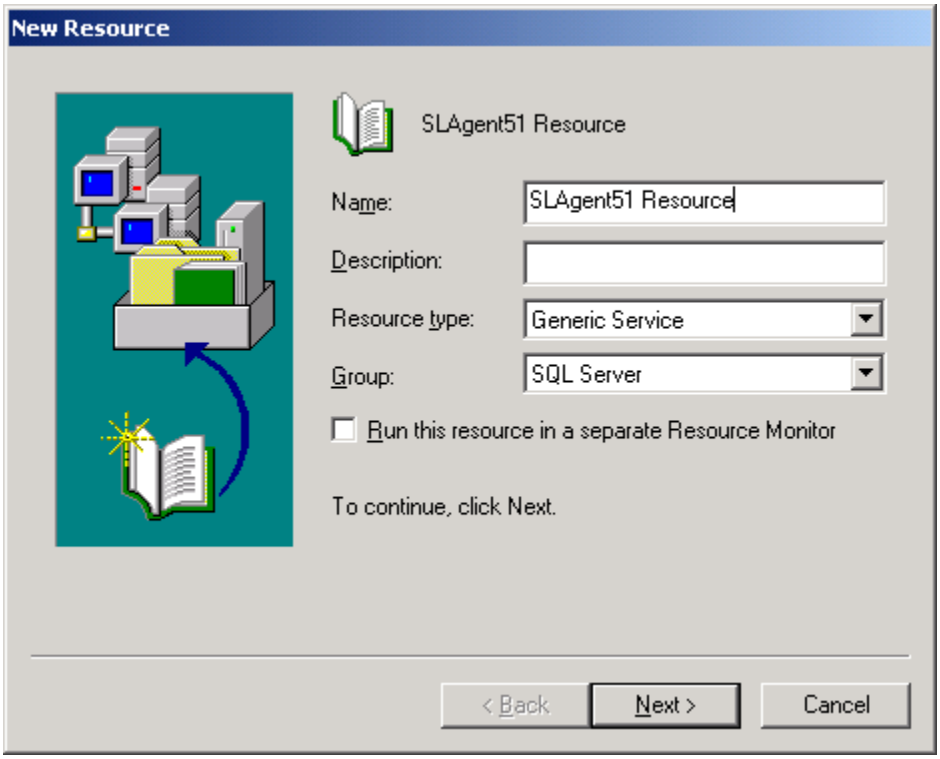
7. After clicking the Next button, the install will begin.
8. This process only installs Merant SequeLink on one node. Although we are installing the files to the shared drive, values are entered in the registry on each system. At this point, it is recommended that you backup the **swandm.ini** file found on the shared drive in the **Program Files\MERANT\slserver51\cfg** directory.
9. To finish the cluster install of Merant, repeat on the backup node. Run through the exact same process to install the registry settings. Be sure to use the EXACT same install path location. This will overwrite the files on the shared drive but doing this does not cause any damage.
10. After installation has completed, go to the Windows Services on each system and change both the SLAgent51 and SLSQLServer51 to start manually.

Configuring Merant as a Cluster Resource

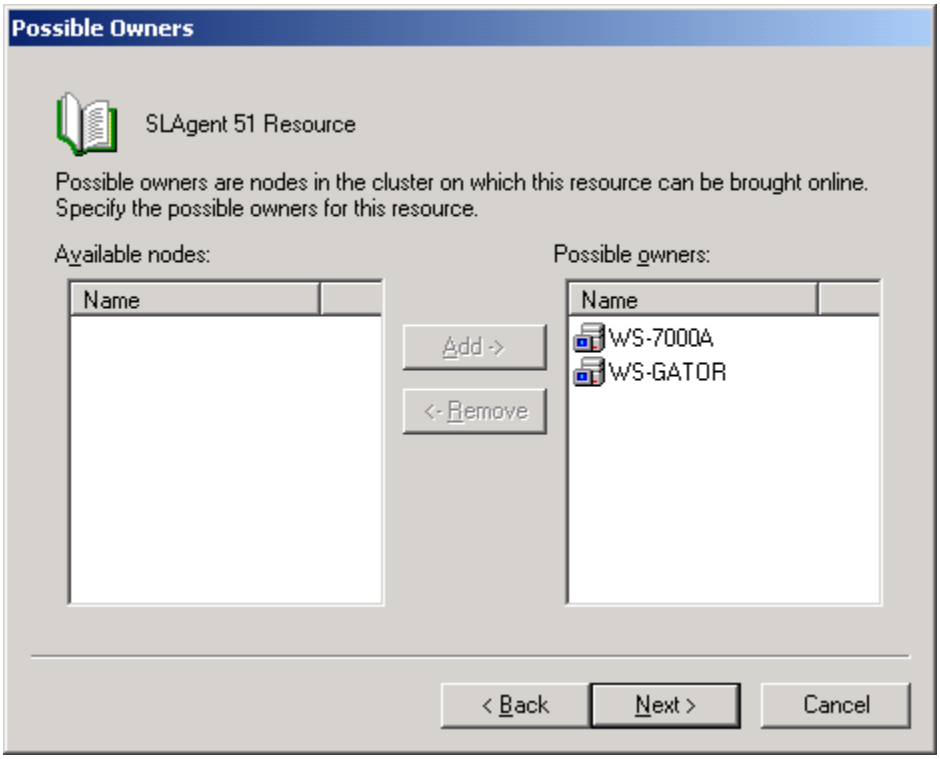
Once the Merant Server is installed on each node, you need to identify it as a cluster resource so that if it would fail on a node, the cluster will force a failover.

1. Start up the Cluster Administrator. (Start->Programs->Administrative Tools->Cluster Administrator)
2. Right-click on the SQL Server Group and select New->Resource.
3. Fill in the appropriate values:
   Name: **SequeLinkAgent**
   Resource Type: **Generic Service**

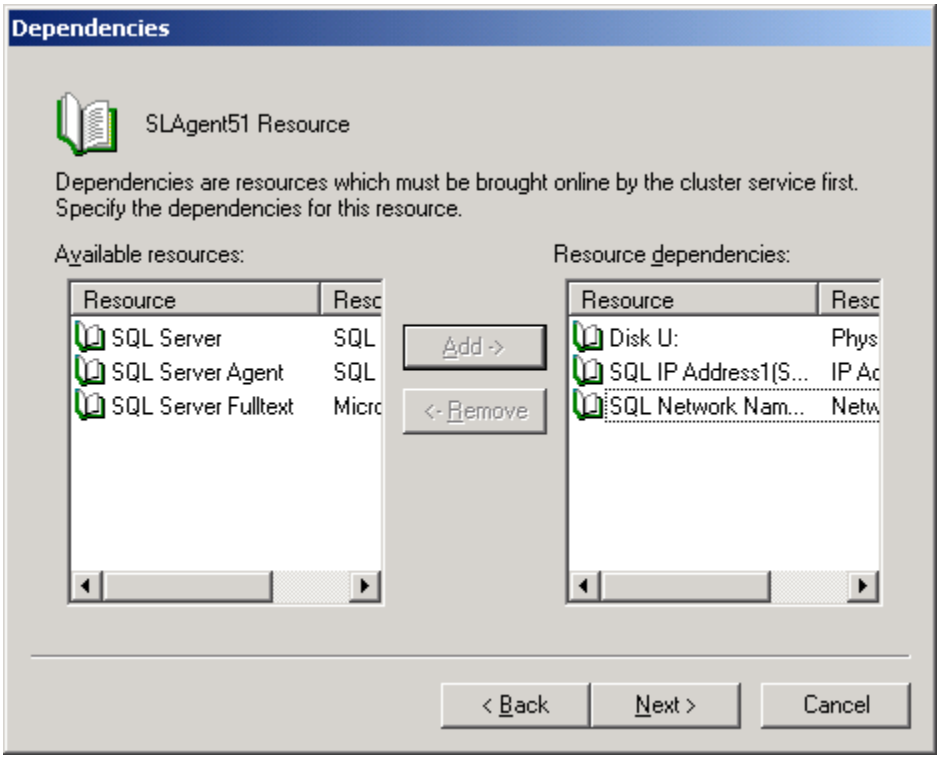
Group: **SQL Server**



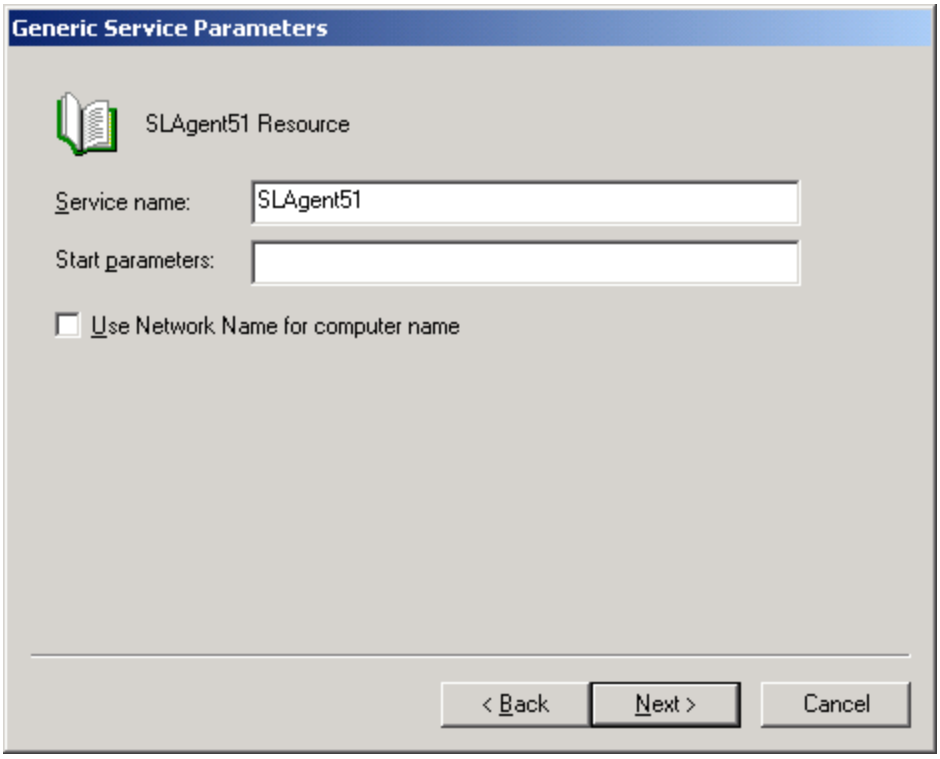
4. Assign both nodes as possible owners of this resource.



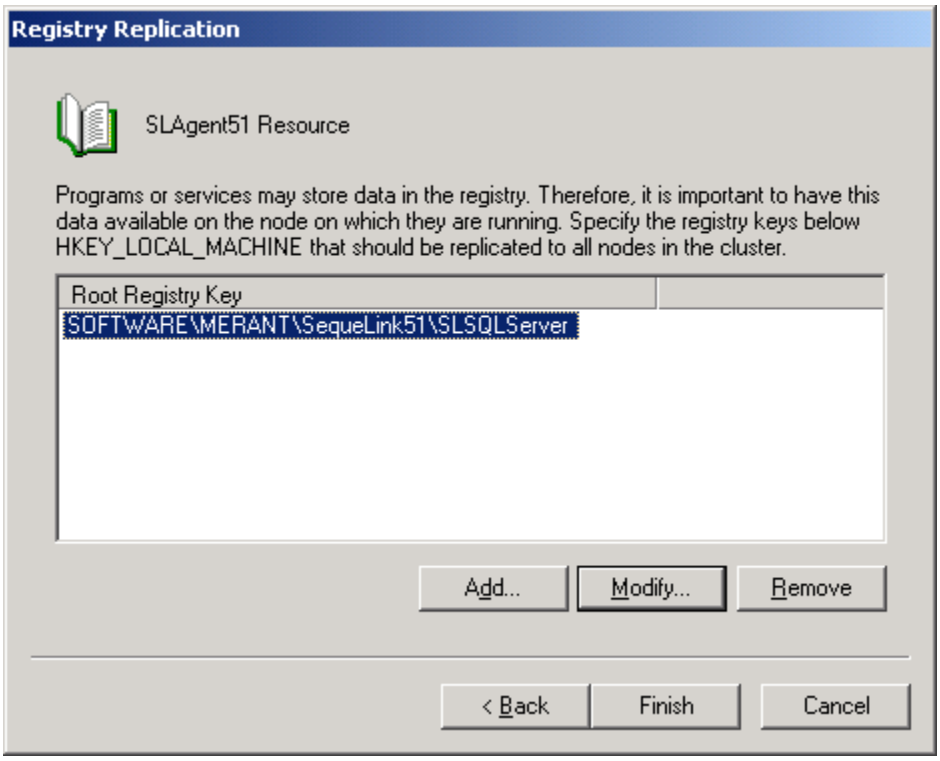
5. Add the following to the Resource Dependencies:
   Disk U: (Quarum Disk)
   SQL IP Address
   SQL Network Name



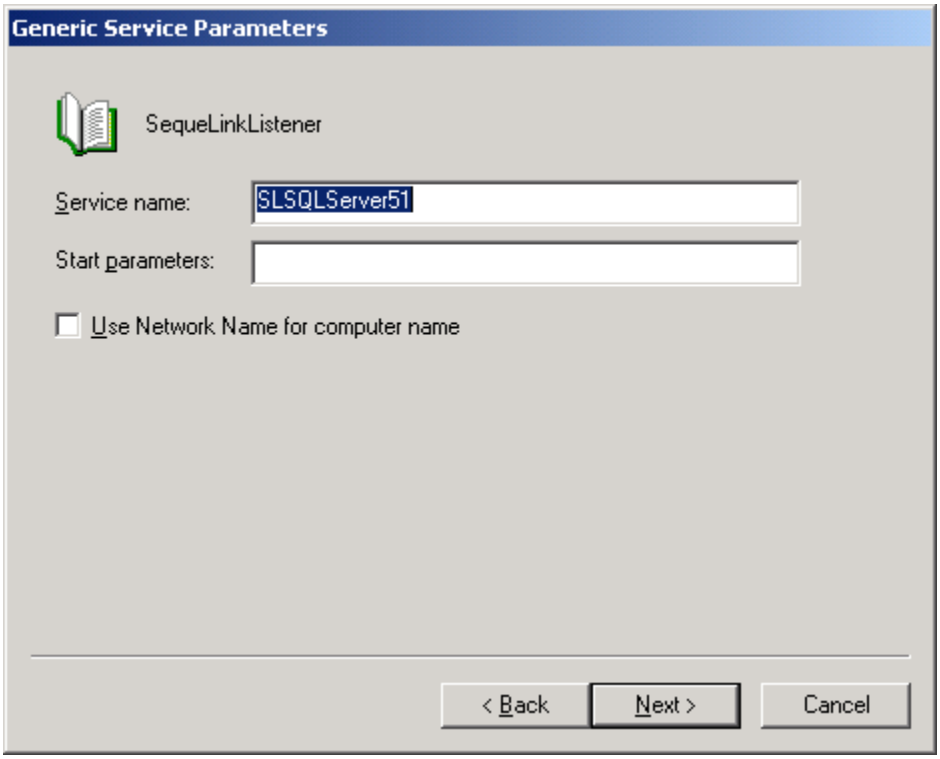
6. Setup the service parameters.
   Service Name: **SLAgent51** (Must match the service that is installed)



7. You need to add a registry key to be replicated on a failover. Click ADD and enter in the string **SOFTWARE\MERANT\SequeLink51\SLSQLServer**.



8. Click Finish.
   Now you must follow the same process for the **SequeLinkListener** Service.



9. Follow steps 2-8 (above), but when one reaches the Generic Service Parameters (Step 6) , enter the following:
   Service Name: **SLSQLServer51**
10. Click Finish. Merant is now setup as a resource in the cluster.

Additional Setup Steps For the Merant Driver

In addition to the setup already performed, the following steps need to be performed to get the environment to function properly.

Changing Merant SequeLink to use TCP/IP Sockets

Merant SequeLink 5.1 Server defaults to using Named Pipes to connect to MS SQL server. To change it to use TCP/IP, follow the steps outlined in "**How to Configure MERANT SequeLink Server for TCP/IP Sockets Net-Library**", referenced in the Useful Resources section below.

Change the Node Hostnames to Virtual Hostname in the swandm.ini File

After installing the Merant SequeLink Server, the configuration file swandm.ini (found at **SharedDrive:\Program Files\MERANT\slserver51\cfg**) was written using the hostnames of each node in the cluster. This will not work for the clustered environment since everything needs to be referenced using the Virtual Hostname of the SQL Server.

Change all instances of specific nodes in **swandm.ini** to virtual IP/hostname (Some examples of where this occurs are the serviceHost and ServiceConnectInfo listings.)

Setting up SQL Server Tables / Usernames

To create a database and user for WebSphere, follow these steps:

1.) Start the SQL Server Enterprise Manager on the active SQL node.
2.) Expand the Console Root until you get to the running SQL Server object.
3.) Expand this object and right-click on the Databases Folder.
4.) Select **New Database**.
5.) Enter the database name, change any necessary parameters and click the OK button.
6.) This will create the new database to use.

Now you need to create a new user for this database that WebSphere will employ:

1.) Using the tree in the left pane, expand the Security folder and right-click the Logins object.
2.) Select **New Login**.
3.) Enter a login name and have it use SQL Server Authentication. Select the database you just created as the Default database.
4.) Click on the Database Access tab and checkmark the database you just created. Give this user the appropriate permissions to this database.
5.) Click OK and test that you can access the database with this user by using the Query Analyzer or some other database client.

Configuring WebSphere to use SQL Server as an application Database

Configuring WebSphere to use the Merant Drivers to access a clustered SQL Server is almost identical to setting up any other database driver/datasource. The main thing to remember is that clients will be referencing the virtual IP of the SQL Server cluster, not each node individually.

1. Start out by installing the Merant database driver to each node within the WebSphere domain. To do this, start up the WebSphere Administrative console and expand the tree to **WAS Domain->Resources->JDBC Providers**
2. Right-click on the JDBC Providers folder and select New.
3. Enter a name for the Driver such as "Merant Driver" and choose the Merant driver **com.merant.sequelink.jdbcx.datasource.SequeLinkDataSource** as the Implementation Class.
4. Click on the Nodes tab and install the Merant Database Driver to the WAS Node, selecting the appropriate jars files **D:\WebSphere\AppServer\lib\sljc.jar;D:\WebSphere\AppServer\lib\sljcx.jar**
5. After the driver has been installed, expand it and right-click on the Data Sources folder. Select New.
6. Fill in the appropriate information such as the Data Source Name, JNDI Name, Database Name, User ID/password, etc. Additionally, be sure to add the following parameters to the Custom Properties.
   serverName = <SQLServer Virtual Name or IP>
   portNumber = 19996 (or whatever you identified when installing Merant)
   disable2Phase = true

7. Setup any beans in your test application to use this datasource with the user specified JNDI name.

Configuring WebSphere to use SQL Server as an Administrative Repository Database

During the installation of WebSphere select the option to use the Merant drivers as the jdbc driver for the repository, as a means to specify a clustered SQL Server as the repository database. Alternatively, install WebSphere using another supported database, and then use the Database Conversion tool to change the appropriate settings and have WebSphere point to a different database. (See the Useful Resources for a link to the Database Conversion tool.) Be sure to get the dbconfig**4**, not the dbconfig conversion tool. Then follow the included instructions.

Verifying your configuration

Once everything is setup, verify that the environment failsover correctly, and the following scenarios can be used:

- **Manual Push to Passive Node** - Within the cluster administrator, you can right-click on a group and click "Move Group" to move it to the passive node.
- **Clean Shutdown of Active Node.** Gracefully shutdown the power on the primary (active) node.
- **Unexpected power failure on Active Node.** Pull the power cable from the primary (active) node.
- **Public network cable failure on the Active Node.** Pull the public network cable from the primary (active) node.

The Microsoft cluster should recognize that one (or more) of the resources failed on the active node and transition all of the components to the alternate node. At this point, all connections to WebSphere are broken, the next database request from WebSphere will result in a StaleConnectionException, thrown (as described in Chapter 9). After the transition backup (and now active) node is completed, WebSphere will reestablish connections to the database. Applications programmed according to the guidelines in Chapter 9 would also reconnect to the database.