



High Availability Guide

Release 7.5 SP1

May 2006



Copyright Notice

High Availability Guide, Release 7.5 SP1

Copyright © 2005 - 2006
Yantra Corporation
ALL RIGHTS RESERVED

WARNING: ANY UNAUTHORIZED DUPLICATION OF THIS DOCUMENTATION SHALL BE AN INFRINGEMENT OF COPYRIGHT

Trade Secret Notice

This documentation, the software it describes, and the information and know-how they contain constitute the unpublished, proprietary, confidential and valuable trade secret information of Yantra Corporation, its affiliated companies or its or their licensors, and may not be used for any unauthorized purpose, or disclosed to others without the prior written permission of the applicable Yantra entity. This documentation and the software it describes have been provided pursuant to a license agreement that contains prohibitions against and/or restrictions on their copying, modification and use. Duplication, in whole or in part, if and when permitted, shall bear this notice and the Yantra Corporation copyright notice.

This documentation and the software it describes are licensed either "AS IS" or with a limited warranty, as set forth in the Yantra license agreement. Other than any limited warranties provided, NO OTHER WARRANTY IS EXPRESSED AND NONE SHALL BE IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR USE OR FOR A PARTICULAR PURPOSE. The applicable Yantra entity reserves the right to revise this documentation from time to time and to make changes in content hereof without the obligation to notify any person or entity of such revisions or changes.

Yantra does not warrant or represent that use of the software described in this documentation will insure compliance with the U.S. Department of Transportation's (DOT) Hazardous Materials Regulations (HMR) found in Title 49 of the Code of Federal Regulations (49 CFR), and users of this software should consult independent legal counsel and technical support to insure compliance with the HMR and other legal requirements.

U.S. Government Restricted Rights. This documentation and the software it describes are each and collectively a "commercial item" as defined in 48 C.F.R. 2.101. Consequently, as and when provided to any agency or instrumentality of the U.S. Government or to a U.S. Government prime contractor or a subcontractor at any tier ("Government Licensee"), the terms and conditions of the customary Yantra commercial license agreement are imposed on Government Licensees per 48 C.F.R. 12.212 or § 227.7201, as applicable, or through 48 C.F.R. § 52.244-6.

Yantra and the Yantra logo are marks of Yantra Corporation. All other services and products or company names are used for identification purposes only and may be marks of their respective owners.

Yantra Corporation
One Park West
Tewksbury, MA 01876
1-978-513-6000

Copyright © 2005 -
2006

Contents

Preface

Structure	vii
Prerequisites	ix
References	ix
Additional Information	ix
Yantra 7x Documentation	x
Conventions	xi

1 Availability

1.1 The 9's	1
1.2 Problem with the 9's	2
1.3 Availability Design and Principles	2
1.3.1 Business Drives High Availability Requirements	2
1.3.2 Keep It Simple Strategy	3
1.3.3 Configuring for Higher Availability or Resilience Is Like Buying Insurance	3
1.4 High Availability Motivation	4

2 Limited Redundancy Single-Site Configuration

3 Yantra 7x Architecture

3.1 Application Server	13
3.2 Agent and Integration Servers	13
3.3 JNDI Service	14

4 High Availability Within A Single Site

4.1	Single Points of Failure.....	15
4.2	Node	16
4.2.1	Active/Passive Cluster Failover Configurations	17
4.3	Database Server	18
4.3.1	DBMS Software Failures	19
4.3.2	Human and Operator Errors.....	19
4.3.3	Hardware Failures	20
4.3.3.1	Active/Passive Failover Configurations.....	20
4.3.3.2	Active/Active Failover Configurations.....	26
4.4	SAN or Disk Subsystem.....	34
4.5	Yantra 7x Application Components.....	34
4.5.1	Application Server	34
4.5.1.1	Stateful Sessions.....	35
4.5.1.2	Stateless Sessions.....	36
4.5.2	Yantra 7x Agent and Integration Server	36
4.6	JNDI.....	36
4.6.1	How to Eliminate SPOF.....	38
4.6.1.1	Deploy the JNDI on a Quiet Server Instance.....	38
4.6.1.2	When the JNDI Server Instance Crashes.....	38
4.6.1.3	When the JNDI Node Fails	38
4.6.2	Limitations/Issues in JNDI Failover	40
4.7	Message Queues	41
4.7.1	Integration Queues for Yantra 7x Integration Servers.....	41
4.7.2	Agent Queues for Yantra 7x Agent Servers	42
4.7.3	WebLogic JMS.....	42
4.7.3.1	Persistence.....	43
4.7.3.2	Message Paging	43
4.7.3.3	Dedicated Integration JMS Server.....	44
4.7.3.4	Shared Disk Subsystem.....	44
4.7.3.5	JMS Server on a Different Application Server.....	44
4.7.3.6	Backup JMS Server	44
4.7.4	WebSphere MQ	45
4.7.4.1	Protecting WebSphere MQ HA Using Cluster Failover	45
4.7.4.2	Implementing Message Persistence Files on SAN	46

4.7.4.3	Risks	46
4.7.5	Clustered MQ.....	47
4.8	Networked File Systems (NFS)	47

5 Architectural Patterns

5.1	Asynchronous Integration.....	50
5.2	Caching	50
5.3	Hot Deployment of Code, Configuration, and Fixes.....	51
5.4	Deployment Processes and Regression Testing	52

6 Disaster Recovery

6.1	Disaster Recovery from a Yantra 7x Perspective	53
6.2	Cold Site Recovery	54
6.3	Warm and Hot Site Recovery	55
6.4	Key to Disaster Recovery	56
6.4.1	Recovery Procedures	56
6.4.2	Database Backups and Transaction Log Files.....	56
6.4.3	Integration Queue Replication	57
6.4.4	Service Names Instead of IP Addresses	57

Index

Preface

The Yantra Distributed Order Management (DOM) and Warehouse Management System (WMS) applications are often deployed in an integrated network of external systems and business partners to form a cohesive business ecosystem. Prolonged application or system outages can have significant business consequences.

This document describes approaches that can increase the resiliency of Yantra 7xt to environmental, hardware or software faults or failure. It also presents techniques or architectural patterns that can minimize the impact on the overall ecosystem in the event of a planned or unplanned Yantra system outage.

This document emphasizes a fairly pragmatic approach to availability recognizing that one cannot implement highly available techniques at the detriment of other architectural considerations such as capital cost, ongoing total cost of ownership, or manageability.

Structure

This manual contains the following sections:

Chapter 1, "Availability"

This chapter introduces availability engineering, the often-quoted “nines” and the “five nines” and the problems with relying on the nines as an availability requirement. It also introduces the design principles that Yantra adheres to when presenting the availability techniques. Yantra strongly encourages you to read through this section.

Chapter 3, "Yantra 7x Architecture"

This chapter provides a very high level overview of the Yantra 7x architecture with a focus on describing the component that needs protection. It assumes that you are familiar with the Yantra 7x architecture. It is preferable that you have also installed and used the Yantra 7x.

Chapter 2, "Limited Redundancy Single-Site Configuration"

This chapter presents a simple, possibly entry-level, Yantra 7x system that was configured with little attention to availability. In some cases, this simple configuration may meet the customer's business or availability requirements. For others, this chapter serves as the baseline for comparison as we examine the configurations with increasingly higher levels of resiliency.

Chapter 4, "High Availability Within A Single Site"

This chapter examines common techniques deployed to ensure application availability against faults incurred within the four walls of the data center. It identifies all the hardware and software components used by the Yantra 7x, all the potential single points of failures (SPOF), and the approaches to protecting these components.

Chapter 5, "Architectural Patterns"

Adding resiliency to the system is not limited to technology. It is important to note that developing a highly available Yantra 7x based solution has as much, if not more, to do with the design of the overall solution and integration points as it does with the Yantra 7x architecture itself. This chapter presents techniques and patterns that can be used to insulate some of these integration points from planned and unplanned Yantra 7x downtime. In this way, even though portions of the solution may be unavailable, there is no downtime for the service as a whole especially as perceived by the end users or customers. More importantly, these architectural patterns can significantly simplify or reduce the Yantra 7x application's availability, requirements and implementation.

Chapter 6, "Disaster Recovery"

With the advent of 9/11 or the Northeast Blackout, events that disaster planners once dismissed as implausible or far-fetched are now required considerations as part of disaster recovery and business continuity

planning. This chapter examines commonly used techniques that can be employed so that the Yantra 7x application can continue running at remote sites in the event after disasters.

Prerequisites

This document assumes you are familiar with the Yantra 7x architecture. It also assumes that you have installed and used Yantra 7x.

For More detailed information, see the following Yantra 7x documents:

- *Yantra 7x Installation Guide* - provides detailed information on how to install Yantra 7x
- *Yantra 7x Performance Management Guide* - provides detailed information on how to configure the Yantra 7x

References

[1] IBM eServer p5 590 and 595 System Handbook, SG24-9119-00, IBM Corp, March 17, 2005

[2] NSM: Often the Weakest Link in Business Availability, AV-13-9473, July 3, 2001

Additional Information

There are many good reference books and material that cover high availability engineering for the data center and the network. This document assumes that you are familiar with high availability engineering for general systems. This document highlights areas in the Yantra system that are single points of failures and the approaches that one can take to making them more resilient.

This document assumes the network is resilient. If your Yantra 7x configuration supports a large number of users from the web, Yantra assumes that your wide-area network deployment is configured such that single (or multiple) faults do not cause an outage.

This document does not address security. Yantra recognizes that in today's connected world, security attacks and security fraud is on the rise. Preventing security hacks from taking down systems is a large and complex area and warrants a separate detailed study.

This document does not address environmental or infrastructure availability. Yantra assumes that the data center is built with redundant power circuits, redundant cooling, and so forth, so that the infrastructure remains available with single or multiple environmental faults. Yantra also assume that the data center is sufficiently equipped with an uninterruptible power supply (UPS) so that all the hardware components can operate under brief power fluctuations and is equipped with power generators to continue working during prolonged power outages.

Yantra 7x Documentation

For more information about the Yantra[®] 7x components, see the following manuals in the Yantra[®] 7x documentation set:

- *Yantra[®] 7x Release Notes*
- *Yantra[®] 7x Installation Guide*
- *Yantra[®] 7x Upgrade Guide*
- *Yantra[®] 7x Performance Management Guide*
- *Yantra[®] 7x High Availability Guide*
- *Yantra[®] 7x System Management Guide*
- *Yantra[®] 7x Localization Guide*
- *Yantra[®] 7x Customization Guide*
- *Yantra[®] 7x Integration Guide*
- *Yantra[®] 7x Product Concepts*
- *Yantra[®] 7x Warehouse Management System Concepts Guide*
- *Yantra[®] 7x Platform Configuration Guide*
- *Yantra[®] 7x Distributed Order Management Configuration Guide*
- *Yantra[®] 7x Supply Collaboration Configuration Guide*
- *Yantra[®] 7x Inventory Synchronization Configuration Guide*
- *Yantra[®] 7x Product Management Configuration Guide*
- *Yantra[®] 7x Logistics Management Configuration Guide*
- *Yantra[®] 7x Reverse Logistics Configuration Guide*

- *Yantra® 7x Warehouse Management System Configuration Guide*
- *Yantra® 7x Platform User Guide*
- *Yantra® 7x Distributed Order Management User Guide*
- *Yantra® 7x Supply Collaboration User Guide*
- *Yantra® 7x Inventory Synchronization User Guide*
- *Yantra® 7x Logistics Management User Guide*
- *Yantra® 7x Reverse Logistics User Guide*
- *Yantra® 7x Warehouse Management System User Guide*
- *Yantra® 7x Mobile Application User Guide*
- *Yantra® 7x Analytics Guide*
- *Yantra® 7x Javadocs*
- *Yantra® 7x Glossary*
- *Yantra® 7x Carrier Server Guide*
- *Yantra® 7x Application Server Installation Guide* (for optional component)

Conventions

The following conventions may be used in this manual:

Convention	Meaning
. . .	An ellipsis represents information that has been omitted.
< >	Angle brackets indicate user-supplied input.
mono-spaced text	Mono-spaced text indicates a file name, an API name, or a code example.
/ or \	Slashes and backslashes are file separators for Windows, UNIX and LINUX operating systems. The file separator for the Windows operating system is "\" and the file separator for Unix and Linux systems is "/". The Unix convention is used unless otherwise mentioned.

Availability

This chapter provides guidelines on the planning, implementation, configuration, monitoring, and tuning of the BEA WebLogic and IBM WebSphere application servers.

1.1 The 9's

The 9's are often mentioned when availability is discussed. In its simplest form, the 9's is an indication of how much downtime an application is allowed to incur.

Table 1–1 The 9's

Percentage Uptime	Percentage Downtime	Amount of Downtime each Year	Amount of Downtime each Month
98.0%	2%	7.3 days	14.6 hours
99.0%	1%	3.7 days	7.3 hours
99.8%	0.2%	17.5 hours	1.5 hours
99.9%	0.1%	8.8 hours	43.8 minutes
99.99%	0.01%	52.6 minutes	4.4 minutes
99.999%	0.001%	5.3 minutes	26.3 seconds
99.9999%	0.0001%	31.5 seconds	2.6 seconds

Table 1–1 dramatically shows that the amount of downtime that you can incur drops significantly as you increase the number of 9's. A system can only be down for 5.3 minutes if that system is required to be available for five 9's (an often quoted number).

1.2 Problem with the 9's

Not all outages are the same. In fact, some customers could architect their solution to tolerate a certain level or type of outage. For example, in [Section 5.1, "Asynchronous Integration"](#) on page 50 a customer could integrate the customer-facing website to Yantra 7x using asynchronous messages. With this architectural pattern, Yantra 7x could be offline without impacting the services provided by the website.

The use of the 9's also do not account for the different strategies or level of availability of certain workloads. For example, during failures, customers may want to consider shutting down lower-priority workloads (see example in [Section , "Target Utilization"](#) on page 33).

Similarly, a slowdown or an outage that ripples or cascades to and affects other integrated systems is a lot more serious than an outage of a low priority workload that affects a few users.

Finally, the impact of the outages could depend on the time that the outage occurred. For example, a five hour outage during a weekend or at night has a different business impact than a half hour outage during a peak period.

1.3 Availability Design and Principles

Availability design is a complex subject that can not be synthesized into a simple number. Instead, availability design is a large optimization exercise balancing many requirements.

To help with the availability design, we propose the following principles:

- [Business Drives High Availability Requirements](#)
- [Keep It Simple Strategy](#)
- [Configuring for Higher Availability or Resilience Is Like Buying Insurance](#)

1.3.1 Business Drives High Availability Requirements

High availability (HA) requirements must be driven by business and not implemented for the sake of technology. Using five 9s as a goal could result in an expensive, complex system that is neither needed or wanted.

In some cases, the business may be able to tolerate long periods of outages and a simple backup and restore may suffice. Others may require that every component from power to the database be made as resilient as possible through redundancy and automated failover. In addition, they may mandate geographically dispersed disaster recovery capabilities.

High availability designs cannot be performed in isolation. As in most worthy engineering endeavors, high availability requirements must be balanced against other architectural choices including acquisition cost, maintenance costs, scalability, maintainability, ease of use, impact to business, and so forth.

1.3.2 Keep It Simple Strategy

If possible, you need to manage the complexity of the system, the approaches to high availability and the recovery procedures. Complex systems:

- make it harder for people to understand and manage
- increase the risk of failures
- could make the fault recovery more difficult and in some cases more risky

1.3.3 Configuring for Higher Availability or Resilience Is Like Buying Insurance

In selecting insurance policies, you typically weigh the cost of the insurance against the likelihood that the insurance is needed, whether the insurance is required by law, and the significance of the potential loss if you don't have insurance.

For example, you would likely not take a flood insurance policy regardless of the premium cost if you live on a hilltop in a desert but you would buy a high premium flood insurance if you live in a hurricane zone along the coast. Similarly, when procuring system hardware, you buy servers with high reliability, availability and serviceability (RAS) built in to ensure that hardware faults do not result in an outage. For example, your servers may come equipped with as many as six redundant cooling fans and power supplies.

In some cases, the law may require you to purchase insurance. Similarly, in some business sectors, regulations require business continuity and disaster recovery plans.

At the extremes, if a business is willing and able to tolerate prolonged outage periods, the HA requirements are few. In some cases, having good backups may suffice.

On the other hand, if a business can only tolerate a down time of less than 30 minutes for each outage, you may have to consider having duplicated or redundant components for any component that can fail especially if they are the SPOF.

At the other end of the spectrum, a company may have very high availability requirements and can only tolerate less than five minute downtime for each outage. In that environment, the data center may have to be staffed around the clock, the failure detection must be quick, failover procedures must be automated, and so forth.

1.4 High Availability Motivation

Architecting highly availability systems is not new. They are, in fact, commonplace in industries such as financial. However, many recent events have heightened interests and requirements in availability:

- Catastrophic events such as September 11, 2001 or the Northeast Power Blackout of 2003 have pushed availability to the foreground. Situations that were unimaginable five years ago are now a serious part of business continuity planning. In fact, many corporate managers reject business continuity plans that do not incorporate wide scale disasters.
- Emerging regulations are forcing availability. In the health care industry, the Health Insurance Portability and Accountability Act (HIPAA) mandates business continuity and availability planning. Section 404 of Sarbanes-Oxley specifies that corporations must protect the systems used to report financial information. At a minimum, corporations are forced to think about the ability to recover those systems.
- Your corporation may be part of a supply chain where inventory needs to be available just-in-time. You may demand or are demanded by your partners to have your systems available to ensure that business partners can communicate. In some situations, trading

partners may demand business continuity plans or disaster recovery plans ensuring that services can be restored within a set period of time after catastrophes.

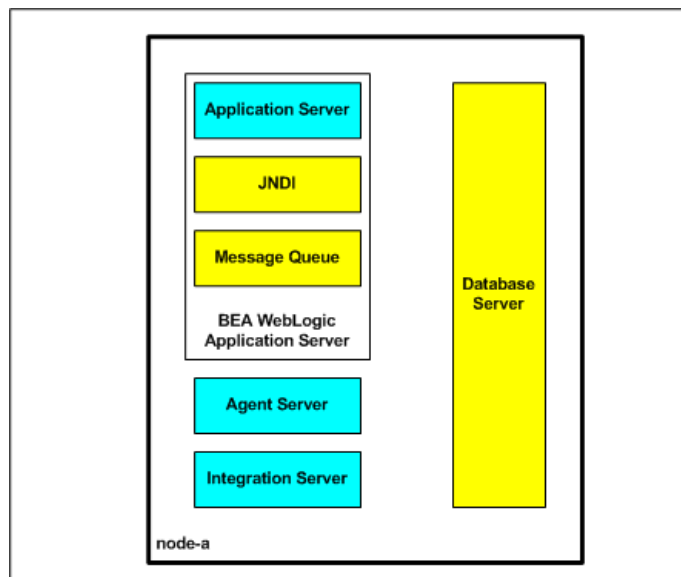
2

Limited Redundancy Single-Site Configuration

The configuration presented in this chapter is a simple entry-level configuration based on standard off-the-shelf products without additional resources or configuration for availability.

An example of this configuration could be as depicted in [Figure 2–1](#).

Figure 2–1 *Limited Redundancy Single-Site Configuration*



In [Figure 2–1](#), the JNDI and Message Queue services are implemented on a single (non-clustered) application server. That application server runs on a single JVM. In addition, all the Yantra 7x time-triggered transactions are configured to run inside a single agent server. Finally, all the integration services run in a single integration server.

The three servers – the application, integration, and agent servers – run on a single node (node-a) along with the database server.

This system has many single-points-of-failure (SPOF) where a single fault can cause a partial or complete Yantra 7x system or application outage. For example, the failure of the node (node-a) will cause a complete system outage. Failure of the Yantra 7x agent server will cause a partial outage – the integration and application server will continue running.

In some cases, this simple configuration may be suitable especially if the system meets the customer’s availability requirements and represents an appropriate balance of risk and benefits. As we mentioned earlier, availability design must be driven from a business perspective.

In practice, we rarely see such systems used in production. Instead, the system above is generally used for development, proof-of-concepts, or demonstrations.

If you want to use a similar system in production, consider the following:

- The ability to recover the database. At a minimum, backup the system and database regularly. Also backup your database transaction logs to allow rollforward recovery from the database backups.
- The backup tapes and archived database transaction logs should be stored off-site. This prevents a data center disaster, such as fire, from destroying not only the database server node but also all the backup tapes.

Even with these considerations, this configuration has the following issues:

- Loss of transactions – if you lose the database server and you have to restore the database to a different server, you will lose recent transactions. After a database restore, you have to rollforward or replay all the transactions found in the transaction logs created after that database backup. Typically, the most current active transaction log, in simple configurations, are only saved when the log closes. If

you lose the log, you have lost all the recent transactions captured in that log after the database restore.

- Loss of transactions in the integration queues – if you lose the disk on which integration queues are kept, you will lose all the unprocessed transactions in those queues.

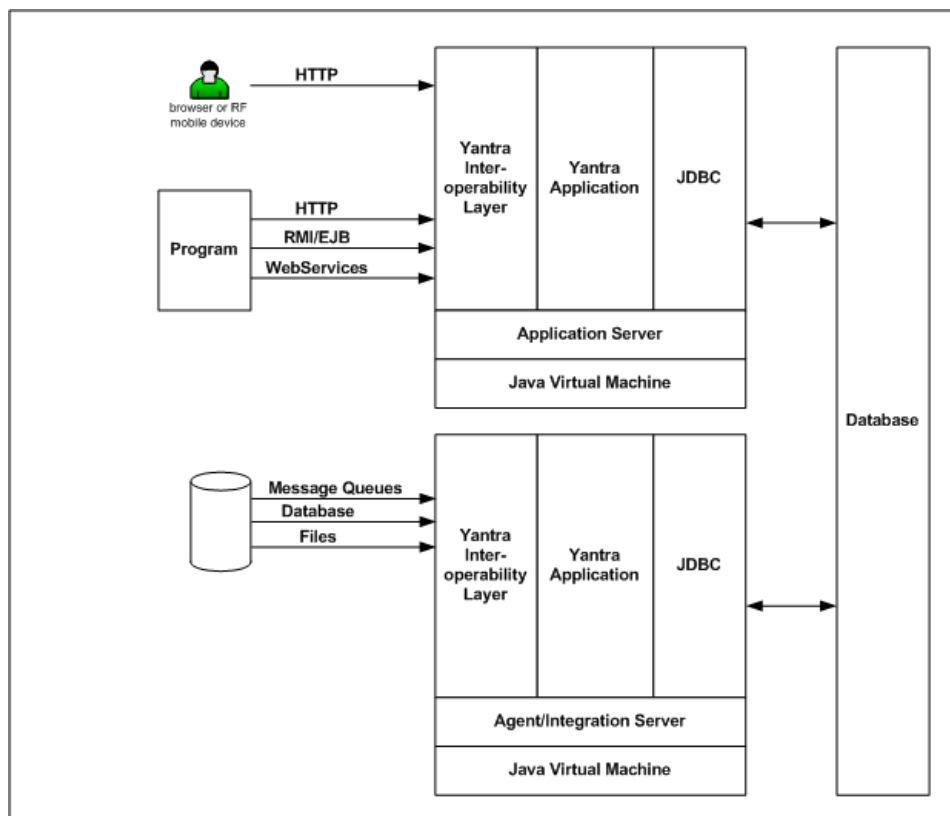
Yantra 7x Architecture

This chapter presents the Yantra 7x application architecture at a high level with a focus on describing the component that you have to protect.

The Yantra 7x application runs on one of the following server components (as depicted in [Figure 3–1](#) the following diagram).

- Application server (such as an IBM WebSphere or BEA WebLogic application server)
- Yantra 7x agent server
- Yantra 7x integration server

Figure 3–1 The Yantra 7x Architecture



These components run inside a Java Virtual Machine (JVM). As a result, each component exists as a process in the system. You can have multiple instances of each component. For example, you can run Yantra 7x in multiple instances of the application server. Each instance is a separate JVM.

The components use the following services:

- Message queue
- JNDI
- Database server
- LDAP (optional)

3.1 Application Server

The application servers are the processes that handle synchronous requests to provide the real-time access to the features and application logic within the Yantra 7x application.

The most common type of requests that an application server handles are the requests originating from clients using the Yantra 7x Application Consoles. The application servers are always deployed using an industrial-strength server application such as BEA WebLogic.

The Yantra 7x application server is the component that runs in Java-based J2EE application servers. Application servers handle real-time requests from users or programs. Requests can be sent in different protocols such as HTTP, servlet calls, EJB/RMI calls, and so forth. The application server runs inside either the IBM WebSphere Application Server or the BEA WebLogic Application Server.

Typical usage scenarios include:

- The call center representative uses the Yantra 7x Application Consoles to interact with the Yantra 7x application. For example, to create, query or modify orders, shipments or inventory. Requests come in as HTTP requests.
- Program runs transactions – calls through Remote Method Invocation (RMI), EJB, servlet calls, and so forth.

3.2 Agent and Integration Servers

Agent and Integration Servers are Java-based processes that run in the background to process work.

An agent server runs Yantra 7x defined “time-triggered” transactions. These include transactions to schedule orders. In the transaction configuration screen, you can designate transactions to an agent server. Multiple transactions could be assigned to an agent server. You can also specify that a transaction should run in multiple threads.

For example, if you associate both the Schedule and Release Order transactions to an agent server (sched_rel_ord_agent) with 3 threads each, when you start an instance of the sched_rel_ord_agent agent server, that server will have six processing threads – three for the Schedule Order transaction and three for the Release Order transaction.

You can also start multiple agent server instances. For example, if you start four `sched_rel_ord_agent` servers, you will see four Java processes running in the system. Each Java process will have 3 threads of the Schedule Order and Release Order transactions. In total, you will get 12 threads of the Schedule Order and 12 threads of the Release Order transaction.

The agent server relies on the JNDI service. At startup, it registers itself to the JNDI. This allows other servers to locate it.

3.3 JNDI Service

All the servers register themselves in the JNDI on startup. This allows servers to locate other servers. One reason is to refresh the reference data cache. The Yantra 7x servers cache reference data records for speed and scalability. When a server modifies a reference data record, it notifies all the servers in the JNDI list to refresh their cache.

The agent server also uses the JNDI to look for the WebSphere MQ message queue service.

High Availability Within A Single Site

From a Yantra 7x system perspective, this chapter identifies:

- All the hardware and software components used by the Yantra 7x application
- All the potential single points of failure (SPOF)
- The approaches to eliminate or reduce the impact of these SPOF

4.1 Single Points of Failure

In the previous chapter, we presented a simple system that was configured with little or no protection against faults. The number of SPOF increases as the system grows in size and complexity. This chapter addresses the following SPOF (within the four walls of the data center) and the means to protect them from faults:

- [Node](#)
- [Database Server](#)
- [SAN or Disk Subsystem](#)
- [Yantra 7x Application Components](#)
- [JNDI](#)
- [Message Queues](#)
- [Networked File Systems \(NFS\)](#)

4.2 Node

The term 'node' refers to the physical computing hardware on which the Yantra 7x application runs.

Fortunately, due to advancements in hardware design, component redundancy, and automatic fault detection and correction, node failures due to hardware fault are rare events. Take for example memory on an industrial-strength computer. Error Checking and Correcting (ECC) codes are built into the memory to correct single bit errors and to detect double bit errors. If needed, parts of the memory can be selectively disabled. Through techniques such as bit-scattering, memory chips are organized such that failure of an entire memory module only affects a single bit within the ECC word. In addition, with techniques such as bit-steering, bits can be dynamically routed to spare memory chips. [1]

Similarly, servers come with multiple critical components such as power and fans so that the server can continue to run after one or more components fail. Most of these components are also hot swappable allowing one to replace failed components without the need to shut down the server.

Unfortunately, if the node fails, the mean-time-to-repair (MTTR) could be very high. In the best case, you may only have to restart the node, restart the services, initiate recovery and make the service available. Depending on the size of the configuration, this could take up to 20 minutes or more.

In the worst case, for example if the fault was due to a hardware failure, you may have to wait for replacement parts. In those situations, the MTTR could be days.

The impact of a node outage depends on the service that runs on that node. For example, the database server is unavailable in the event of a database node outage.

If your tolerance for downtime is measured in minutes, you may have to rely on automated approaches to recover or restore the service on another node. One approach is an active/passive or primary/standby failover configuration where one or more passive or standby nodes are available to take over for failed nodes. See [Section 4.2.1, "Active/Passive Cluster Failover Configurations"](#) on page 17 for more information. We apply this approach in subsequent sections to protect critical components such as the JNDI, message queues and database servers.

In a more specialized case, [Section 4.3.3.2, "Active/Active Failover Configurations"](#) on page 26 describes in the use of an active/active clustered database failover configuration. This approach can be used to protect the Oracle database.

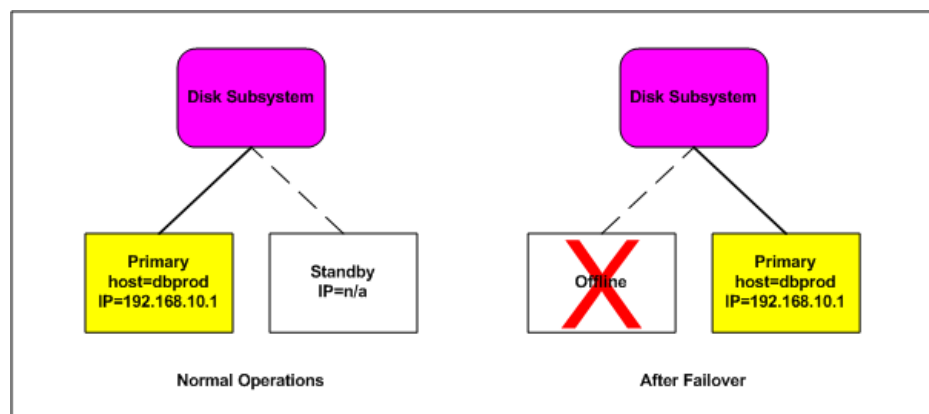
4.2.1 Active/Passive Cluster Failover Configurations

Generally, in an active/passive cluster failover configuration, one or more passive or standby nodes are available to take over for failed nodes. Only the primary node is used for processing. When a node fails, the standby node takes over the resources and the identity of the failed node. The services provided by the failed node are started on the standby node. After the "take over", clients are able to access the services unaware that the services are being provided by a different node.

[Figure 4–1](#) illustrates an active/passive database failover configuration. Both the active/passive nodes share the same disk subsystem although only the primary database server has access to the disk subsystem. The path from the standby node to the shared disk subsystem is not activated.

During normal operations, the application connects to the database server with a hostname of `dbprod` that gets resolved to an IP address of `192.168.10.1`.

Figure 4–1 Active/Passive Database Failover Configuration



During a node failure, the following typically occurs:

On the original primary node:

1. If the primary node is still up, the services on the primary node are brought down.
2. All resources (specifically the disk subsystem) from the primary node are released.
3. The service IP address (192.168.10.1) is released.

On the standby node:

1. The disk subsystem is brought online.
2. File systems are checked and repairs are made if needed.
3. The service IP address (192.168.10.1) is configured.
4. The services are started – database rollforward recovery is initiated as necessary.
5. The database services are opened.

These failover or takeover steps can be automated. Some of the software that can be used include:

- IBM HACMP (only available on AIX)
- Veritas Cluster Service (VCS)
- HP MC/ServiceGuard
- Microsoft Cluster Server (MCS)

Fully automated, the failover could take 5 to 10 minutes.

In subsequent sections, we present the use of active/passive failover configurations to protect many of the Yantra 7x components in more detail.

4.3 Database Server

The database server is a critical system component. The entire Yantra 7x application is unavailable if the database server crashes. There are many reasons why the database server can come down, including:

- [DBMS Software Failures](#)
- [Human and Operator Errors](#)

- [Hardware Failures](#)

4.3.1 DBMS Software Failures

As with any large complex software, there are bugs in the Oracle and UDB database servers. Some of these bugs can cause instance crash or performance degradation. In rare extreme cases, these bugs can corrupt the database.

The best means to protect against software failures is testing. Your testing must exercise transactions from a broad range of application functionality and not a small subset of transactions. The tests must also run at transaction volumes at or higher than anticipated peak production periods. These tests are the only reliable means for identifying load, concurrency, or integrity issues in the database management system and the application.

You should be aware of any support or service alerts associated with or new issues introduced with your database server release. The list of issues is not static – new bugs are discovered as customers use the release, existing bugs are be fixed, and so forth. Therefore, you should check this list periodically to see if there are any new issues that could potentially affect your system.

Additionally, you should be careful that you don't apply all the patches available for that database release. From our experience, you may destabilize a database release when you apply too many individual patches. In some cases, individual patches may conflict with each other.

For software bugs that crash the instance, the fastest recourse is to restart the instance. For a corrupted database, your recourse may range from trying to repair the damage using SQL to restore the database from the last backup and performing rollforward recovery until the point before the corruption. Either way, the MTTR will likely be very high.

4.3.2 Human and Operator Errors

A Gartner report “shows that an average of 80 percent of mission-critical application service downtime is directly caused by people or process failures. The other 20 percent is caused by technology failure, environmental failure or a disaster.” [2]

The best prevention is strict change control, documented procedures, training, and supervision.

Recovery from human-induced outages could range from restarting services to recovering a corrupted database.

4.3.3 Hardware Failures

Node failures are extremely rare. Unfortunately, when they do occur, the MTTR can be unacceptably high for your business.

To protect the database server from node failures, you can use either active/passive or an active/active cluster failover configurations.

4.3.3.1 Active/Passive Failover Configurations

4.3.3.1.1 IBM UDB Active/Passive Using Cluster Failover

Software Conceptually, UDB active/passive failover configurations using cluster failover software operates as described in [Section 4.3.1, "DBMS Software Failures"](#) on page 19. The standby node takes over the primary node's resources (the database files, logs) and identity (IP address, SAN WWNN). The database service is then started on the standby node. During the startup, UDB goes through its normal crash recovery and ensures committed changes are made to the database and incomplete transactions are rolled back. When UDB is finished with crash recovery, the database service is made available.

From Yantra 7x perspective, you can expect the following to occur after the primary node fails (and the database server is unavailable).

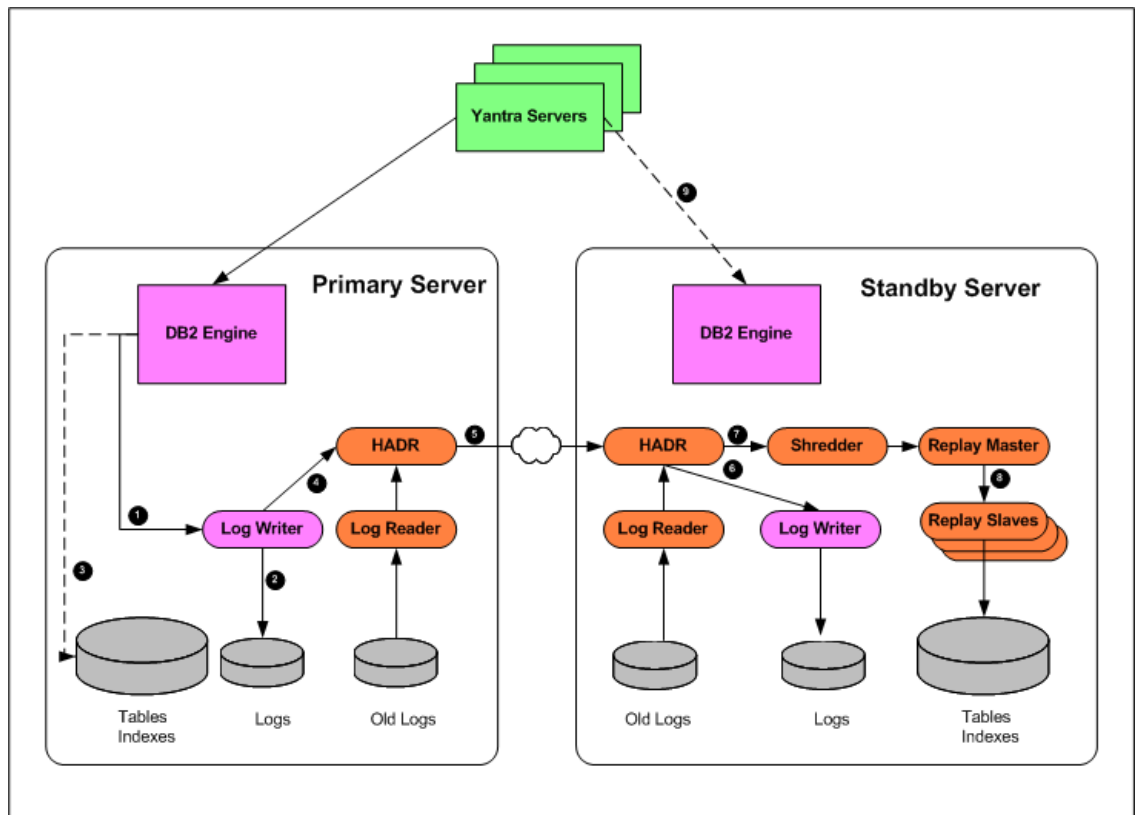
- Transactions in the application, agent and integration servers that were actively processing will throw a SQL error message. The changes from those transactions will be correctly rolled back later when the database server comes up on the standby node.
- The Yantra 7x servers will continually reissue the transactions until the database service is restored. You will not have to restart the Yantra 7x servers during the transition to the standby node.
- If the source of the work request (specifically for the agents and integration servers) came from message queues, the messages remain in the message queue. When the database service is restored, these messages are processed.

Setting up and testing an active/passive failover configuration can be tricky with many interdependencies and related parts. We strongly

encourage you to contact the cluster failover vendors for assistance in planning and implementing your cluster failover.

4.3.3.1.2 IBM UDB Active/Passive Using HADR UDB 8.2's High Availability Disaster Recovery (HADR) is a transaction log replication approach that keeps a standby database server in or near synch with changes in the primary database server. In the event of a failover, HADR on the standby database server takes over and becomes the primary as described in [Figure 4–2](#).

Figure 4–2 UDB Active/Passive Using HADR



At a high level, the log writer on the primary database server records changes to its local transaction logs. These logs are critical for crash and

instance recovery. The primary HADR sends the log records to the standby HADR where the logs are written out to the standby server's transaction logs. The changes are then applied to the standby server's database. At some point in time, the changes on the primary server are asynchronously written to the database.

The standby database server is kept in "perpetual rollforward" mode applying transaction log entries as they are replicated from the primary.

HADR provides many benefits over the traditional active/passive cluster failover provided by software such as HACMP, VCS or MC/ServiceGuard. First HADR recovery is faster because you do not have to start the standby database server – the standby database server is always running and is either in or near synch with the primary database server. Similarly, you do not have to spend a lot of time in database crash recovery because by design, the standby HADR database server is in or near peer state. Also, you do not have to spend time releasing resources on the failed node and acquiring resources on the standby node. With HADR, the standby database is already connected to and using a separate disk subsystem.

Second, the standby HADR database server does not share disk subsystems with the primary database server. Therefore, with HADR, you can survive a disk subsystem failure whereas cluster failover, which relies on a shared SAN, could incur a potentially prolonged outage until the disk subsystem is repaired.

In addition, HADR provides many intangible benefits. For example, HADR allows read-only queries to run from the standby database. With this capability, you can offload your reporting and DSS workloads to the HADR standby database.

HADR is provided as part of ESE. With HACMP or the other cluster failover software, you have to purchase additional software licenses.

From a recovery perspective, the HADR provides a less risky failover approach. With HADR, the standby database is already running. In contrast, with cluster failover, resources have to be acquired, services started, etc. There are potential startup risks during the recovery process.

UDB 8.2's HADR implementation has the following limitations:

- HADR can only replicate to one standby database server – therefore, from the primary database server, you can either HADR to a local

standby for local site failover or to a remote site for disaster recover (but not both).

- HADR is only supported on UDB ESE
- You cannot backup from the standby – you must backup from the primary
- Client reroute does not work with the Yantra 7x agents.

Please refer to the IBM UDB documentation for more detail.

Client Reroute

Client Reroute was introduced in UDB 8.2 along with HADR to enable client applications to automatically reconnect to the standby HADR database server when the primary server fails. Client reroute works by informing the client of the alternate or standby database when it connects to the primary.

The alternate database information is defined on the primary database server with the following command:

```
db2 update alternate server for database <dbname> using
hostname <hhh> port <ppp>
```

For example, if your primary database DB2PROD is on node N1 port 50000 and the alternate is on node N2 port 50000, issue the following command on node N1:

```
db2 update alternate server for database DB2PROD using
hostname N2 port 50000
```

Alternates are propagated from the server to the client dynamically when the client issues a CONNECT or CONNECT RESET. This dynamically propagated alternate server information is stored in global driver memory, and is also updated in the JNDI store of DB2 active servers.

DB2 Universal JDBC Driver client reroute support is available only for connections that use the `javax.sql.DataSource` interface.

Limitations with Client Reroute

Since Client Reroute registers the alternate database when a connection is made to the primary, you can not start the application server cluster when the primary server is down and the standby HADR has assumed

the role of the primary database server. The reason is that the application server only knows the hostname of the original primary database server, which is already down. Since the application server cannot connect to the primary, it also cannot get the information on the alternate database.

Client Reroute does not work with the Yantra 7x agent or integration servers because Client Reroute requires the servers to use `javax.sql.DataSource` to request database connections. The Yantra 7x agent and integration servers use `java.sql.DriverManager` to create database connections.

If the agent and integration servers were to use `javax.sql.DataSource`, they would be given a proxy to the actual database connection object on the application server. All JDBC requests from the agent or integration servers are processed by the connections on the application servers. In addition, all parameters and database results must to be marshaled and un-marshaled between the agent or integration servers and the application servers. This adds a lot of overhead to the SQL processing.

Yantra 7x Servers Failover for IBM UDB

Given the limitations of Client Reroute, the Yantra 7x agent and integration servers are not able to connect to the standby database if the primary database server fails. UDB's JDBC URL only allows you to connect to a database service on a hostname or host IP address. In the example below, the JDBC URL points to a database called DB2PROD on HOSTXYZ at port 50000:

```
jdbc:db2://HOSTXYZ:50000/DB2PROD
```

The value HOSTXYZ is actually an alias that gets resolved to an IP address. It does not have to be a hostname. With that flexibility, we could use an alias to designate a server name – in this case a database server name. In the example above, we could use the value DB2PRDAL to designate the DB2 Production Alias. By using an alias, we decouple the server name from a physical node.

```
jdbc:db2://DB2PRDAL:50000/DB2PROD
```

In the event of a failure (e.g., the node HOSTXYZ has crashed), all we need to do is change the DB2PRDAL alias to point to the IP address of the standby node that has the standby database server. When the agent

and integration servers try to reconnect, the connect requests (with the same DB2PRDAL) now go to the standby database node's IP address.

4.3.3.1.3 Oracle Active/Passive Using Cluster Failover Oracle active/passive failover configurations are very similar to the UDB active/passive configuration described in [Section 4.3.3.1.1, "IBM UDB Active/Passive Using Cluster Failover Software"](#) on page 20. It is our belief that customers prefer to implement an active/active Oracle failover configuration instead. This is described in [Section 4.3.3.2, "Active/Active Failover Configurations"](#) on page 26.

4.3.3.1.4 Microsoft SQL Server Active/Passive Using MSCS Microsoft SQL Server active/passive failover configurations are very similar to the [Section 4.3.3.1.2, "IBM UDB Active/Passive Using HADR"](#) on page 21.

The clustered nodes use the heartbeat to check whether each node is alive, at both the operating system and SQL Server level. At the operating system level, the nodes in the cluster compete for the resources of the cluster. The primary node reserves the resource every 3 seconds, and the competing node every 5 seconds. The process lasts for 25 seconds and then starts over again. For example, if the node owning the instance fails due to a problem (network, disk, and so on), at second 19. The competing node detects it at the 20-second mark, and if it is determined that the primary node no longer has control, the competing node takes over the resource.

From a SQL Server perspective, the node hosting the SQL Server resource does a looks-alive check every 5 seconds. This is a lightweight check to see whether the service is running and may succeed even if the instance of SQL Server is not operational. The IsAlive check is more thorough and involves running a `SELECT @SERVERNAME Transact SQL` query against the server to determine whether the server itself is available to respond to requests; it does not guarantee that the databases are up. If this query fails, the IsAlive check retries five times and then attempts to reconnect to the instance of SQL Server. If all five retries fail, the SQL Server resource fails. Depending on the failover threshold configuration of the SQL Server resource, Windows Clustering attempts to either restart the resource on the same node or fail over to another available node. The execution of the query tolerates a few

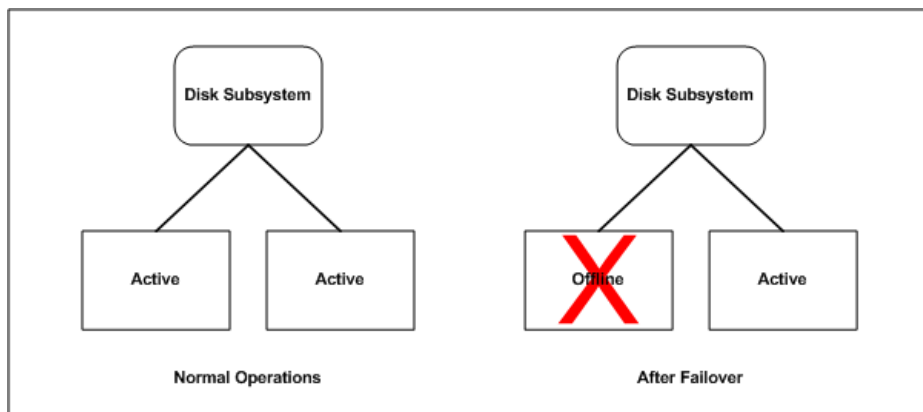
errors, such as licensing issues or having a paused instance of SQL Server, but ultimately fails if its threshold is exceeded.

During the fail over from one node to another, Windows clustering starts the SQL Server service for that instance on the new node, and goes through the recovery process to start the databases. The fail over of the SQL Server virtual server takes a short time (probably seconds). After the service is started and the master database is online, the SQL Server resource is considered to be up. Now the user databases go through the normal recovery process, which means that any completed transactions in the transaction log are rolled forward, and any incomplete transactions are rolled back. The length of the recovery process depends on how much activity must be rolled forward or rolled back upon startup. Set the recovery interval of the server to a low number to avoid long recovery times and to speed up the failover process.

4.3.3.2 Active/Active Failover Configurations

In an active/active failover configuration, two or more database nodes are clustered together to provide a common service to applications. Unlike active/passive failover configurations, all nodes of an active/active failover configuration are actively processing transactions. If a node fails, the remaining nodes continue to provide the service. [Figure 4–3](#) illustrates this concept:

Figure 4–3 Active/Active Failover Configuration



4.3.3.2.1 Oracle RAC Active/Active The Oracle Real Application Cluster (RAC) is a share-everything database cluster with a distributed lock manager. As a share-everything database, all RAC nodes access and update the same database data files. The distributed lock manager controls which node updates the data. It does not matter which node the transaction is performed on. Each node has equal rights to all data in the shared database.

Each RAC node has a listener process that is responsible for processing database connection requests from client programs. When the listener receives a request, it could spawn off a new database process to which the client program connects to. If server-side load balancing is enabled, the listener could send the request to the listener on the least busy RAC node.

Configuration

When configuring Yantra 7x with Oracle RAC, you want the RAC nodes to be reasonably balanced so that all RAC nodes, over a period of time, are about the same utilization. During a node failure, you also want the connections from the failed node to automatically reconnect to the surviving RAC node. You can do this using Oracle features on the client-side and server-side.

Client-Side Load Balancing

On the client-side, set up the JDBCURL parameter, which the JDBC driver uses to connect to Oracle, as follows:

```
"jdbc:oracle:thin:@
  (DESCRIPTION =
    (ADDRESS_LIST =
      (LOAD_BALANCE = yes)
      (ADDRESS = (PROTOCOL = TCP)(HOST = dbnode01)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = dbnode02)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = rac)
    )
  )
```

In this example, the JDBCURL shows two RAC nodes (dbnode01 and dbnode02). The (load_balance=yes) instructs the JDBC Driver to

progress through the list of addresses in a random sequence spreading the load to all the listeners.

In a failure situation, if the connection request goes to the “downed” node, the connection requests timeouts. The driver then transparently sends the connection request to the remaining node. The connection timeout can be tuned down.

With client-side load balancing, the connection request eventually reaches an Oracle listener.

Server-Side Load Balancing

On the RAC nodes, enable server-side load balancing so that the listener routes database connections to the RAC instance on the least busy (loaded) nodes.

The listener on each RAC node is aware of all RAC instances. In addition, the Oracle PMON (process monitor) periodically updates the listeners with the node utilization. The update can occur as quickly as a minute on heavily loaded nodes or as much as ten minutes on lightly loaded nodes. Depending on the load information, the listener decides to which instance to send the incoming client request. The listener typically selects an instance on the least loaded (busy) node. If the least busy node has multiple RAC instances, the listener then chooses the least loaded instance on that node.

You can enable server-side load balancing by setting the following parameters:

hostname	service name	sid name	instance_name
=====	=====	=====	=====
dbnode01	rac	rac1	rac1
dbnode02	rac	rac2	rac2

```
spfile
*.remote_listener='LISTENERS_rac'
rac1.local_listener='LISTENER_rac1'
rac2.local_listener='LISTENER_rac2'

*.db_name='rac'

rac1.instance_name='rac1'
rac2.instance_name='rac2'
```


dbnode01 listener.ora file

```

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = dbnode01)(PORT = 1521))
      )
    )
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = /u01/app/oracle/product/9.2.0.5)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (ORACLE_HOME = /u01/app/oracle/product/9.2.0.5)
      (SID_NAME = rac1)
    )
  )

```

dbnode01 and dbnode02 tnsnames.ora file

```

LISTENERS_RAC =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = dbnode01)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = dbnode02)(PORT = 1521))
    )
  )
RAC1 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = dbnode01)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = rac)
      (INSTANCE_NAME = rac1)
    )
  )
RAC2 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = dbnode02)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = rac)
      (INSTANCE_NAME = rac2)
    )
  )
RAC =
  (DESCRIPTION =
    (LOAD_BALANCE = yes)
    (ADDRESS = (PROTOCOL = TCP)(HOST = dbnode01)(PORT = 1521))
    (ADDRESS = (PROTOCOL = TCP)(HOST = dbnode02)(PORT = 1521))
    (CONNECT_DATA =
      (SERVICE_NAME = rac)
    )
  )

```

```
dbnode01 only tnsnames.ora
```

```
LISTENER_rac1 =  
  (DESCRIPTION =  
    (ADDRESS = (PROTOCOL = TCP)(HOST = dbnode01)(PORT = 1521))  
  )
```

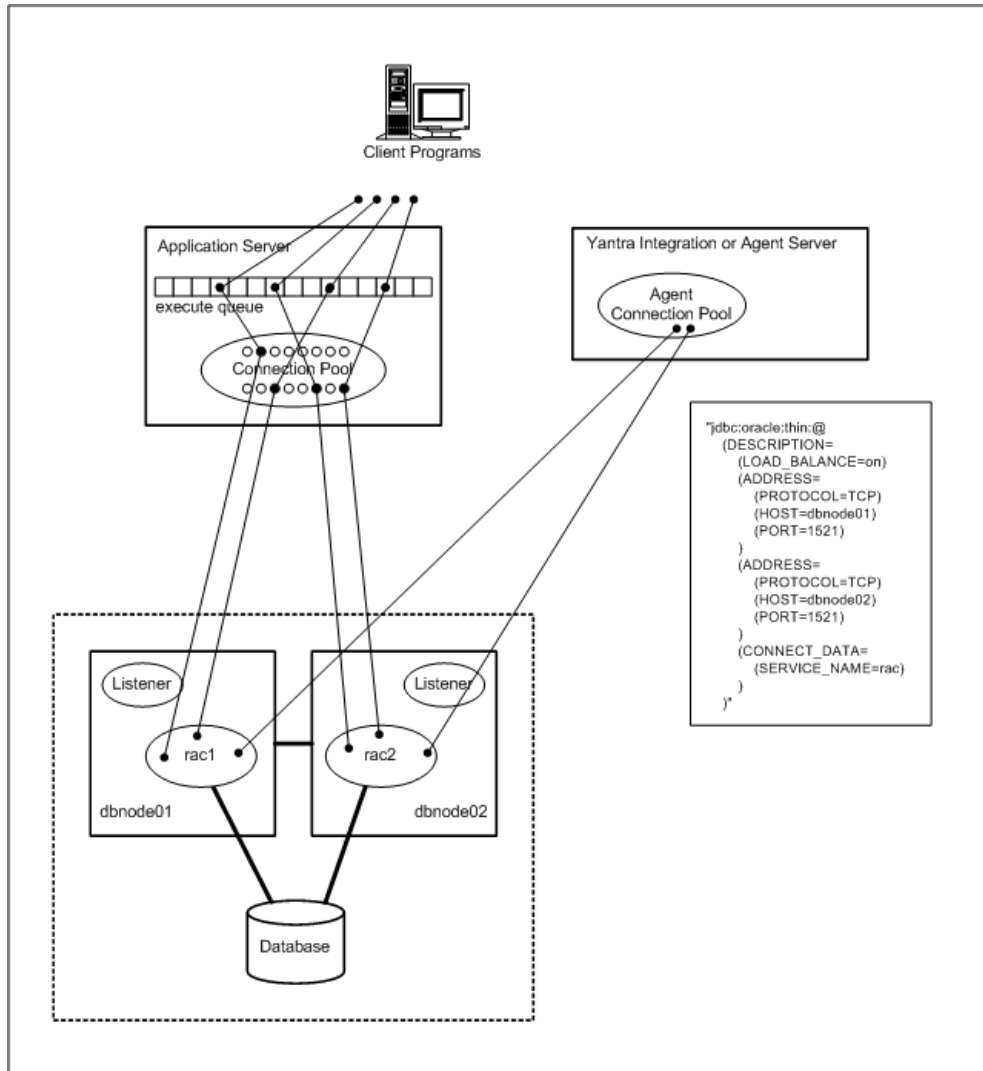
```
dbnode02 only tnsnames.ora
```

```
LISTENER_rac2 =  
  (DESCRIPTION =  
    (ADDRESS = (PROTOCOL = TCP)(HOST = dbnode02)(PORT = 1521))  
  )
```

LISTENERS_RAC, LISTENER_rac1, and LISTENER_rac2, are the net_service_name (connect descriptor) for remote_listener and local_listener. On the client side, you do not need these net_service_name parameters.

RAC is the net_service_name for client-side load balancing as illustrated in [Figure 4–4](#).

Figure 4-4 Load Balancing



From a Yantra 7x application perspective, you can expect the following to occur after a RAC server instance failure:

- Transactions in the application, agent and integration servers that were actively processing throws a SQL error message. The remaining RAC nodes roll the changes from those transactions back.
- The Yantra 7x agent and integration servers continually attempt to reconnect. When it has connected to one of the remaining RAC instances, the failed transactions are reprocessed from the beginning. You do not have to restart the Yantra 7x servers during the transition to the standby node.
- If the source of the work request (specifically for the agents and integration servers) came from message queues, the messages remain in the message queue. When the database service is restored, these messages are processed.

Target Utilization

When deploying an active/active failover configuration like Oracle RAC, take into consideration your target node utilization, especially during the seasonal peak periods. Although RAC is an active/active configuration, you have to be careful when you run the RAC nodes at close to max utilization. First, RAC needs CPU to maintain the Cache Fusion. Secondly, during failover, all the work from the failed node eventually spills over to the remaining node. For example, at the peak hour, the combination of high volume agent processing and application server processing is driving a 2-node RAC configuration to 80% busy. If a node fails, the servers reconnect to the remaining node. The combined workloads drive the database to a point where performance degrades or at worst the system becomes unstable.

Ideally, you should try to keep the average node utilization below 40% utilization (for a 2-node RAC) to reduce the risk of overloading the remaining nodes. In the event of a node failure, the remaining node will likely grow to 80%.

Alternatively, if you typically run the nodes at higher utilization, you should consider classifying workloads into priority groups. For example, transactions that are initiated by customers may be put into the high priority group because they add to the overall customer experience. These could include application server workloads. Customers with short “click to release” service levels may consider agents that schedule and release orders as high priority workloads. Workloads such as order purge or other maintenance work could be categorized into lower priority groups.

4.3.3.2.2 IBM UDB Active/Active

UDB's active/active database configuration (where all nodes are actively processing work) is provided by their UDB ESE Data Partitioning Feature (DPF). Yantra is currently not certified to run on UDB DPF.

4.4 SAN or Disk Subsystem

The disk subsystem is another critical system component. Disk failures could cause outage to parts of or the entire Yantra 7x application.

Careful design and implementation must be placed on the disk subsystem. Your investment in failover configurations could be wasted if the disk subsystem fails.

The different areas where disks are used by Yantra 7x include:

- Database datafiles/objects
- Message queues
- Internal disks

To prevent outages due to disk failures, consider configuring the following:

- Disks as RAID groups (except for RAID-0) for redundancy and performance
- Redundant disk systems to tolerate component failures
- Multiple access paths to the disks

4.5 Yantra 7x Application Components

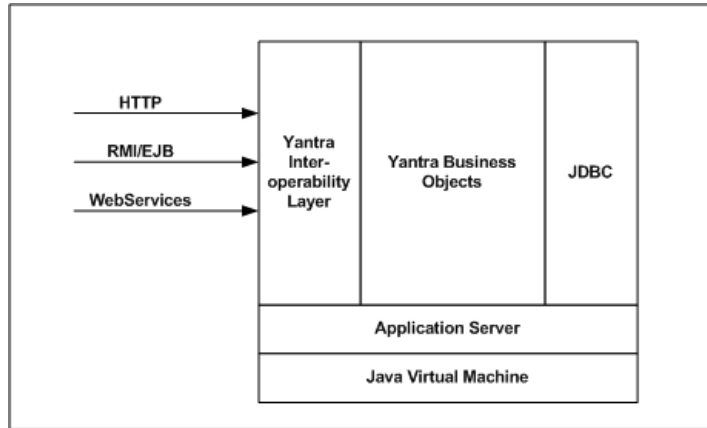
As we described in [Chapter 4, "High Availability Within A Single Site"](#), the Yantra 7x Application runs in the application server, agent server, and the integration server.

4.5.1 Application Server

The application server provides:

- Users with the ability to use the Yantra 7x Application Consoles to create and manage orders, shipments, and inventory manage orders, shipments, and inventory

- Programs to call Yantra 7x business APIs using HTTP, servlets, EJB/RMI or WebServices



Currently, the Yantra 7x application is supported on the following application servers:

- BEA WebLogic
- IBM WebSphere

All application server requests are transactional in that all the work performed is within a transaction boundary and is either fully committed or nothing is committed. This guarantees that there is no partially completed work.

For resiliency, you can configure multiple application server instances in a cluster. If an instance fails, the workloads are sent to the remaining instances.

Generally, the following occurs when an application server instance fails. If there were active transactions running on the failed application server, those transactions can be reprocessed. The transactions are either fully committed or not at all. Subsequent transactions are sent to the remaining application server instances.

4.5.1.1 Stateful Sessions

The HTTP user interface (or Yantra 7x Application Consoles) sessions are “stateful”. When transactions complete, they leave information (state) on the J2EE application server for the subsequent transaction. By being

stateful, all HTTP transactions return to the same application server instance. This can be accomplished by telling the load-balancers or proxies that the sessions are sticky.

By default, WebLogic and WebSphere application servers are configured for memory session persistence where the HTTP session-state is only stored in the application server instance that the transaction ran on. The session information is lost if that application server instance fails. If that happens, the user is redirected to the login page. After logging back into the application, the user is able to continue where they left off.

4.5.1.2 Stateless Sessions

All the other application server transactions, EJB/RMI, servlet calls, and WebServices, are stateless. In contrast, these transactions do not leave behind session state information. As a result, these transactions can be performed on any application server instance where the application is deployed.

4.5.2 Yantra 7x Agent and Integration Server

The Yantra 7x agent and integration servers are completely location or node independent. They can run from any node where the application has been deployed.

4.6 JNDI

The Java Naming and Directory Interface (JNDI) is a standard service in Java for applications to store information that other workloads can query.

The Yantra 7x application uses the JNDI to store information on how to locate each of the server instances (e.g., the application server, agent and integration server instances).

When a server instance starts, it will store its server name and a remote object pointing to itself in the JNDI. Any server instance can query the JNDI to find all the other server instances. Workloads can then use the remote object to query the services provided by the server instance.

For WebSphere, the JNDI is also used to store the name and location of the WebSphere MQSeries queues. When a Yantra 7x server is shut down, its JNDI entry is removed.

The Yantra 7x application uses the JNDI information for the following events:

- Reference data cache refresh – the Yantra 7x application implemented a mid-tier data cache to cache commonly used reference data. If a server instance changes a cacheable record, that instance needs to broadcast that change to instruct all the other server instances to refresh their cache. For more information on the Yantra 7x reference data cache implementation, refer to the *Yantra 7x Performance Management Guide*.
- The system management console uses the JNDI to discover all the application instances. The SMC uses that list to build a list of instances to monitor.
- Yantra 7x workloads in a configuration that is deployed with IBM WebSphere Application Servers and WebSphere MQSeries uses the JNDI to discover the available message queues.

The parameters that govern how the Yantra 7x servers communicate to the JNDI are presented in [Table 4–1](#). These parameters are specified in the `<YFS_HOME>/resources/management.properties` file.

Table 4–1 JNDI Parameters

JNDI	Description
yfs.remote.pingtime	Frequency at which the server checks to see if the JNDI is alive and that it is correctly registered in the JNDI. The default is 600 seconds.
yfs.remote.reconnecttime	Frequency at which a server tries to reconnect to the JNDI to register itself after it discovers that the JNDI is down. The default is 600 seconds.
jndi.nocleanup	Controls whether servers on startup checks and validates all the JNDI entries. Default is true – do not perform this check.

The JNDI is implemented in one of the BEA WLS or IBM WAS server instances. There is only one copy of the Yantra 7x JNDI making that copy a SPOF. If the JNDI server instance fails, the JNDI service will be unavailable.

If the JNDI server is down, the following happens:

- You can not start agents. Existing agents continue processing. This is because the agents need the JNDI to register its service immediately on startup.
- Workloads are not able to change cacheable reference data because they can't contact the JNDI to get a list of servers.

4.6.1 How to Eliminate SPOF

This section covers methods to eliminate SPOF:

- [Deploy the JNDI on a Quiet Server Instance](#)
- [When the JNDI Server Instance Crashes](#)
- [When the JNDI Node Fails](#)

4.6.1.1 Deploy the JNDI on a Quiet Server Instance

As a preventative measure, deploy the JNDI tree to a server instance that is not active or busy. This reduces the risk that a server instance due to out of memory exceptions will cause a JNDI outage.

In WebLogic, consider putting the JNDI on the administration server instance. If the WLS cluster is used by many applications and you don't want to put application specific information on that administration server, consider putting the JNDI on its own server instance in a different cluster.

In WAS, consider putting the JNDI on the deployment manager.

4.6.1.2 When the JNDI Server Instance Crashes

Immediately restart the server instance. After the restart, look at the logs to find the cause. Make sure the `verbosegc` statistics is enabled so you can see if the problem is due to out of memory conditions (this is highly unlikely if you had deployed the JNDI to a quiet server instance). Look for JVM core dumps – the problem may be a JVM bug.

4.6.1.3 When the JNDI Node Fails

If the JNDI node fails, restart the node. If that node needs to be repaired, target the JNDI to another application server instance. You have the following options:

- [Active/Passive JNDI Nodes](#)

- [Target JNDI to Another Application Server Instance](#)

4.6.1.3.1 Active/Passive JNDI Nodes

You could configure the JNDI node into an active/passive failover group, possibly shared by other servers. When the JNDI node fails, you want the JNDI to failover to the standby node. After the failover, restart the application server that is the new home for the JNDI. There are no configuration changes since the new JNDI node assumes the same IP address of the failed JNDI node.

After the JNDI server is up, the application, agent and integration servers automatically reconnect to the JNDI (as dictated by the `yfs.remote.pingtime` and the `yfs.remote.reconnecttime` parameters).

4.6.1.3.2 Target JNDI to Another Application Server Instance

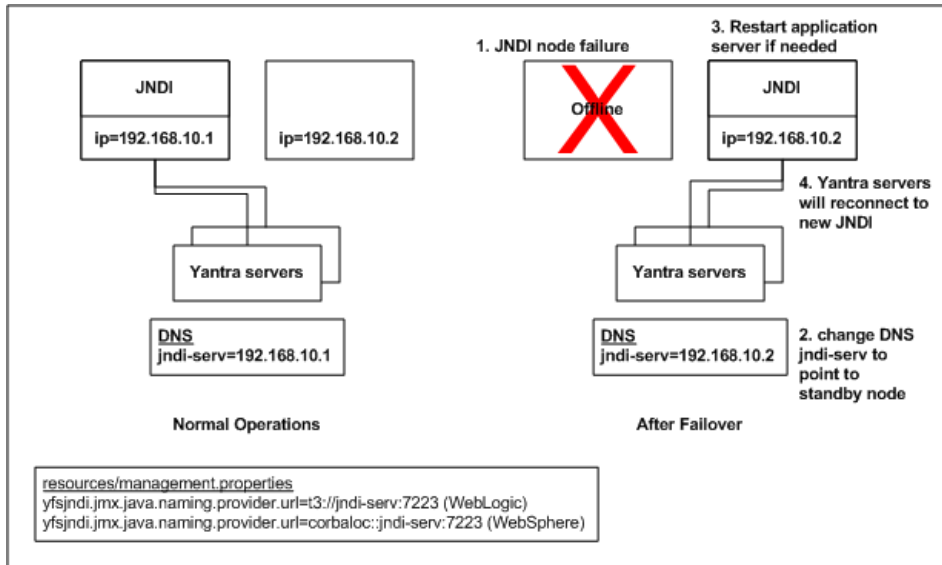
If you do not want to configure an active/passive failover group, you could retarget the JNDI to one of the remaining application server instances by taking advantage of service or server names.

In [Figure 4–5](#), the Yantra 7x servers find the JNDI through the `yfsjndi.jmx.java.naming.provider.url` parameter in the `<YFS/HOME>/resources/management.properties` file. In that parameter, point to the JNDI using a server or service name instead of a hostname (node name) or host IP address. This approach makes the service (in this case the JNDI) location independent.

In [Figure 4–5](#), the JNDI service is initially on the node with the IP address of 192.168.10.1. If that node fails and cannot be restarted, you could change the service name in the DNS to point to another node (192.168.10.2), and then restart the application server (if needed).

Eventually, as dictated by the `yfs.remote.reconnecttime` parameter, the Yantra 7x servers will reconnect back to the JNDI.

Figure 4–5 JNDI Targeting



4.6.2 Limitations/Issues in JNDI Failover

Transactions that do not need the JNDI can continue to run while the JNDI is unavailable. When the JNDI becomes available, servers eventually detect the presence of the JNDI and reregister. The `yfs.remote.pingtime` parameter dictates the frequency at which servers send a heartbeat to the JNDI. Between the time when the JNDI becomes available and when the last active server reconnects, data integrity issues may occur.

For example, the JNDI crashes at time t_0 and is restarted at time t_1 . At that time, there are twenty active Yantra 7x servers. At time t_2 , three servers reconnect to the JNDI. One of those three servers changes several reference data records in the database. As part of that change, that agent gets a list of active (known) servers from the JNDI. It finds the three servers that have reconnected but not the other 17 servers. It then tells those three servers to refresh their cache. The remaining 17 servers are not aware of the data changes and could continue using the stale data.

As a result, at this time, you have the following options:

- Set the ping and reconnect time parameters low (e.g., 1 minute) to reduce the window when the servers rediscover the JNDI
- After a JNDI failure and restart, go to the Yantra 7x System Management Console and issue the global cache clear command. You can issue that command several times after the JNDI restart. You should issue at least clear cache at reconnect time after the JNDI has restarted. This reduces the window when servers are potentially processing with stale reference data (assuming reference data was changed immediately after the JNDI restart).

4.7 Message Queues

Yantra 7x uses message queues extensively. The message queue usage in Yantra 7x can be grouped as:

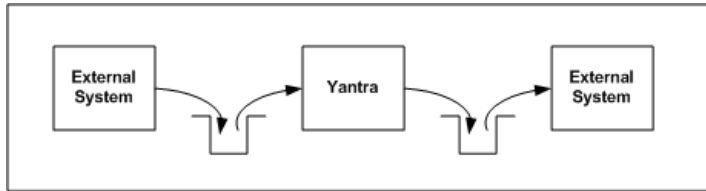
- Integration queues for external communications
- Temporary work-in-progress queues for Yantra 7x's and custom time-triggered transactions

These queues can be implemented in either BEA WebLogic JMS or IBM WebSphere MQ. Yantra 7x does not support IBM WebSphere's embedded JMS server.

4.7.1 Integration Queues for Yantra 7x Integration Servers

Integration-based queues are typically used for communications from or to external systems. For example, in [Figure 4–6](#), the first queue could be used by external systems like a web store frontend to pass order creation requests into Yantra 7x. It could also be purchase orders from a purchasing system, shipment status updates from a logistics management system, and so forth.

Figure 4–6 Integration-based Queues



Similarly, Yantra 7x could use integration queues to send messages to external systems. For example, Yantra 7x could send ship notices to warehouses.

Since these messages are used for communicating between systems, the messages in the integration queues must be protected so that they are not lost in the event of a failure. These messages can be difficult to recreate.

4.7.2 Agent Queues for Yantra 7x Agent Servers

The Yantra 7x agent servers use the messages in the queues as a source of work. These messages are typically read from the database and can be easily recreated. As a result, Yantra recommends creating these queues as non-persistent queues where the messages are kept in memory.

4.7.3 WebLogic JMS

To protect the WebLogic JMS queues, consider the following:

- [Persistence](#)
- [Message Paging](#)
- [Dedicated Integration JMS Server](#)
- [Shared Disk Subsystem](#)
- [JMS Server on a Different Application Server](#)
- [Backup JMS Server](#)

4.7.3.1 Persistence

By default, WebLogic JMS message queues are defined as non-persistent, which means that the messages are only kept in memory (the JVM heap). Non-persistent messages are lost when the JMS server shuts down or crashes.

To protect integration messages, you must define the queues as persistent. WebLogic allows messages to be persisted to files or the database.

The agent queues should be implemented as non-persistent queues. The Yantra 7x agents are designed to be able to recreate the work-in-progress task messages.

4.7.3.2 Message Paging

By default, the messages in WebLogic JMS queues are kept in memory (in the Java heap). You should consider protecting them against situations where a large number of messages could cause the JMS server to fail because it has run out of space in its Java heap. This could happen if there are a significantly large number of messages in the queue or exceptionally large message bodies. In those situations, the JVM could run out of space in its heap.

The best way to find out how much JVM heap you need is to create a large number of messages in your test queues and see how much memory is used. The amount of heap required differs for each implementation.

To protect against situations where the WebLogic JMS heap fills up, you could enable byte or message paging. When the number of bytes or messages exceeds specified thresholds, WebLogic JMS server pages out the message body (but not the message header) to a paging file store. This approach can reduce the likelihood of a JVM crash. You can still run out of JMS Server heap if the queue has a lot of message headers.

As a draconian measure, you can set the maximum message parameter. When set, this parameter puts a hard limit on the number of messages that can be in the queue. When this threshold is reached, new messages are rejected with an error message. As a result, you will have to ensure that message producers are architected to correctly handle message maximum exceptions – for example, the message producers may want to queue the messages and retry later.

4.7.3.3 Dedicated Integration JMS Server

Since integration queues can grow unbounded, you should define the integration queues to one or more dedicated JMS servers. These JMS servers should be targeted to one or more dedicated WebLogic managed servers. For example, assume you have ten integration queues. Some of your options are:

- Define all ten integration queues to a single JMS server which is targeted to a dedicated application server. That application only has the JMS server targeted to it.
- For very large integration queues, target 5 integration queues to one JMS server and the other five to another JMS application server. The two JMS servers are then targeted to two dedicated application servers.

4.7.3.4 Shared Disk Subsystem

The persistence and paging file stores should be implemented on a shared disk subsystem (such as a SAN) and not on local disks. The shared disk subsystem should be accessible from standby servers to prevent disk failures or node outages from causing a prolonged JMS Server outage. This configuration allows you to restart the JMS Server from another node.

4.7.3.5 JMS Server on a Different Application Server

You can protect the task-based queues by ensuring the JMS server can be restarted on a different node or a different application server. You don't have to worry about preserving the content of the queues because they can be recreated from the database. You also don't have to worry about protecting them against large number of messages because the agents only fetch a finite number of messages (default is 5,000).

4.7.3.6 Backup JMS Server

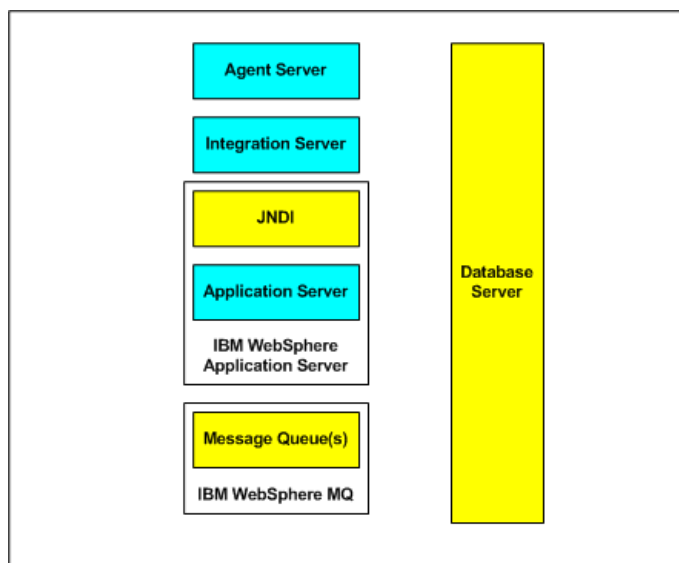
Alternatively, you can specify a backup JMS Server for the Yantra 7x agent servers. In the event that the initial or primary JMS Server fails, the Yantra 7x agent servers migrate to the alternate JMS server. You can specify the backup JMS server using the following parameters in the <YFS_HOME>/resources/yfs.properties file.

```
yfs.agent.backup.providerurl=t3://<ip Address>:<port>
yfs.agent.backup.icf=<InitialContextFactory Name>
yfs.agent.backup.qcf=<QueueConnectionFactory Name>
```


4.7.4 WebSphere MQ

For WebSphere customers, as depicted in [Figure 4–7](#), the integration and task-based queues are implemented in WebSphere MQ, which is an external message queue from the WebSphere Application Server.

Figure 4–7 WebSphere MQ



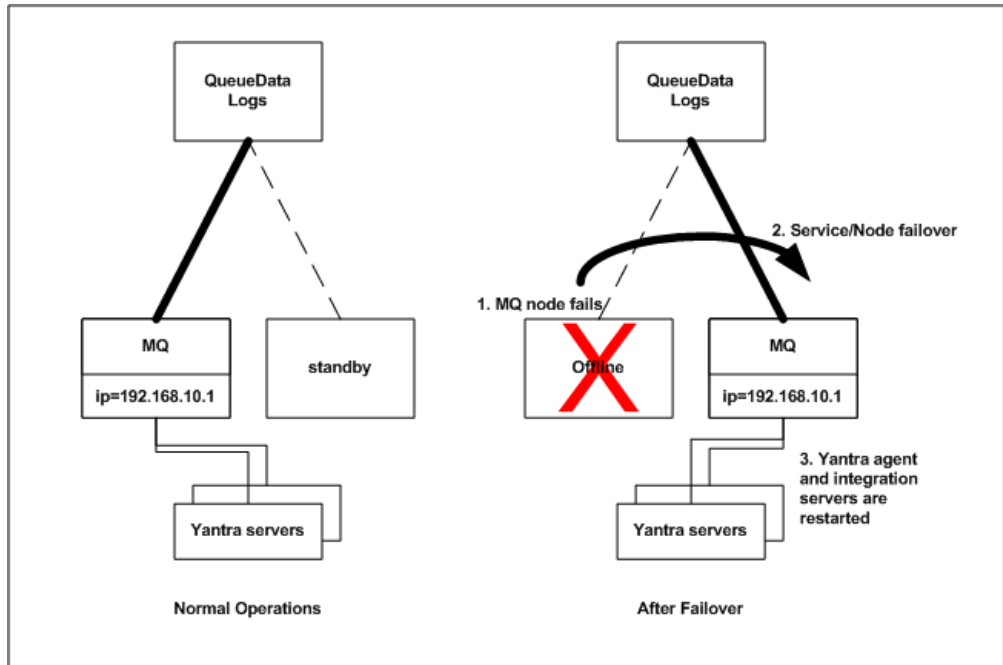
The WebSphere Application Server is the host to the JNDI service and the Yantra 7x application server.

4.7.4.1 Protecting WebSphere MQ HA Using Cluster Failover

One approach to protecting WebSphere MQ is through cluster failover using cluster software such as HACMP, MC ServiceGuard, or Veritas Cluster Service. In [Figure 4–8](#), the MQ (the MQ queue manager and the local queues) on the active node with IP address of 192.168.10.1 is running and accepting and distributing MQ messages. In the event of a node failure and the node cannot be restarted, the cluster software activates the standby node. From the agent and integration servers' perspective, the MQ service was unavailable during the time it took to

failover. Once the failover is completed, the agent and integration servers can be restarted. At that time, the standby node looks the same as the former primary node. Therefore, there are no changes required.

Figure 4–8 Cluster Failover



4.7.4.2 Implementing Message Persistence Files on SAN

To prevent disk failures or node outages from causing a prolonged MQSeries outage, you should consider putting the MQ logs and files used to store messages on a SAN that is accessible from multiple nodes. This allows you to restart MQ from another node.

4.7.4.3 Risks

In the previous examples, the shared storage remains the single point of failure. You could lose data if the shared storage was. One option is to configure active/passive SAN devices with the primary SAN replicating

data to the standby SAN. In the event of a SAN failure, the primary MQ node would release the primary SAN and acquire the standby SAN.

4.7.5 Clustered MQ

Yantra 7x is not certified on clustered MQ.

4.8 Networked File Systems (NFS)

Some customers prefer to implement Yantra 7x on a networked file system (NFS). With this approach, all mounted nodes can access all the shared folders. Changes are made to the shared files. In contrast, changes do not have to be pushed out to every node if files were stored on local disks.

If you choose this approach, consider implementing a highly available NFS to prevent an outage of the NFS server from creating an application outage. Losing the NFS server crashes all the Yantra 7x servers.

Architectural Patterns

The Yantra Distributed Order Management (DOM) and Warehouse Management System (WMS) applications are often deployed in an integrated network of external systems to form a cohesive business ecosystem. Prolonged application or system outages can have significant business consequences.

Decoupling and component independence is an extremely powerful architectural pattern to insulate critical portions of the overall ecosystem solution from downtime or faults in other areas. The availability and uptime of the Yantra 7x based solution can be greatly enhanced by adopting one or more of the following patterns during solution design. Each of these patterns makes it possible to decouple one or more parts of the application from other portions thus providing increased availability and uptime for critical areas like external users and customers.

Each of these design patterns can be applied to provide increased application resilience. While these examples talk about website integration, these patterns can be applied to other areas of integration as well.

A well-designed solution around the Yantra 7x system can actually increase the availability and uptime of the solution as a whole to levels above what Yantra 7x delivers out-of-the-box. In some critical areas for example, the solution can continue to be available even when the product is taking a planned or unplanned outage.

Finally, there are a few other process and deployment related solution design considerations that can actually provide better gains in availability and uptime at a much lower cost than technological and redundancy based solutions.

5.1 Asynchronous Integration

The most common decoupling technique is asynchronous message communication between business entities. Take, for example, the need to send orders created at different external systems to Yantra 7x. These systems could send the order creation requests to Yantra 7x:

- Synchronously using protocols such as HTTP, WebServices, EJB/RMI and so forth; or
- Asynchronously using messages.

Both approaches have their strengths and weaknesses. From a high-availability standpoint, the loosely-coupled asynchronous approach allows Yantra 7x to be unavailable, as a result of a scheduled or unscheduled outage, without affecting the external systems. The external systems can queue up the order creation requests into an integration message queue.

In contrast, if the communications is synchronous, Yantra 7x must be available in order for the external system to create the request. In this architecture, Yantra 7x's availability requirements will have to be the greater of all the availability requirements of all tightly connected systems combined.

This scenario is, of course, simplistic since it may not take into account other synchronous interfaces (like inventory lookups or pricing) that the order creation process is dependent upon. These requirements are addressed in [Section 5.2, "Caching"](#).

5.2 Caching

Another common decoupling technique is the use of local caching. In this pattern, the consuming application (for example, store web store) gets information such as item attributes, inventory balance, or item availability from a local data cache. This approach reduces the need to synchronously query Yantra 7x.

The local information cache can be updated by utilizing a variety of algorithms that offer various degrees of sophistication, performance and accuracy. Not only does this technique provide a way to decouple two areas of the solution, but it also provides significant performance, response time, and scalability advantages that are especially useful in end-user or website scenarios.

As an example, one area where Yantra 7x typically recommends utilizing this algorithm is for caching ATP (Available to Promise) data on the website. In some customer environments where shopping cart abandonment rates are very high, for example 100 item lookups to 1 item ordered, it is better to have Yantra 7x push out item availability to the web storefront using the Yantra 7x Real-time Inventory Monitor. With this approach, most inventory lookups that are part of the customer's browsing and ordering experience can be served from the website without any synchronous calls to Yantra 7x. Based on business requirements, if the inventory levels are sufficiently high, the web storefront can sell that item. The web storefront would revert to synchronous inventory availability check when the inventory levels are below a certain threshold. More importantly, Yantra 7x can be down without affecting the web storefront.

While this cookie cutter approach to inventory caching may not work for all scenarios, techniques such as these can be invariably applied to almost all critical interfaces to provide simplistic but "safe" algorithms to counter planned or unplanned downtime without affecting end users or disabling critical functionality areas altogether.

5.3 Hot Deployment of Code, Configuration, and Fixes

While the methodologies and design patterns presented will insulate critical areas of the solution from downtime, there are deployment techniques provided by Yantra or inherent within Yantra 7x's architecture that allow you to hot deploy incremental changes, configuration, and fixes on critical synchronous application components. Some of these capabilities include the ability to:

- Deploy changes to incremental configuration or master data without having to bring down any application areas.
- Hot deploy incremental software or code changes on the synchronous application components by utilizing capabilities offered by the application server or by utilizing application server independent techniques like clusters and rolling restarts.

Theoretically, there could be scenarios where even a small change to a component may require multiple areas of the application to be updated simultaneously due to interdependencies, thus causing an outage.

However, in reality, a large number of incremental changes and product fixes can utilize these techniques even without any explicit hot deployment design considerations.

Explicitly factoring in requirements to be able to hot deploy changes during incremental solution design phases leads to the ability to hot deploy all changes with a few exceptions. This situation is further mitigated by Yantra 7x's dependence on asynchronous processing for complex algorithms. This significantly reduces the solution footprint that external synchronous interfaces like those from the website, rely on. This, in turn, reduces the probability of many changes or fixes in these areas.

5.4 Deployment Processes and Regression Testing

One of the most important and most overlooked areas that can significantly affect availability and uptime of an application is the presence of a strictly enforced process to promote, characterize, verify, and regression test incremental rollouts or hot-fixes and upgrades. In industry studies and based on Yantra's experience, the lack of sufficient automated integration testing, human and operator error, and lack of appropriate software change management processes to prevent those errors, is the single biggest factor that causes application downtime when there is no actual infrastructure failure. The cost of setting up and investing in a robust and isolated testing environment that mirrors the configuration and a small amount of representative transactional data from production is usually much lower in comparison to implementing redundant systems and complex processes to handle issues with new solution rollouts and software fixes. Any investment in this area goes a long way to prevent issues with failure and downtime.

Disaster Recovery

With the approaches described in [Chapter 4, "High Availability Within A Single Site"](#), you should be able to withstand most single and possible multiple component failures without incurring an outage. With the appropriate architectural patterns described in [Chapter 5, "Architectural Patterns"](#), you may be able to schedule downtime with less impact to the corporations overall availability.

There remains one major contingency you need to consider: what happens if a catastrophic event causes your primary data center to be partially or completely incapacitated? The reasons could range from the commonplace disasters such as fires in the building or natural disasters like floods or earthquakes. It may also be rare events like the Northeast Blackout of 2003 when wide regions covering over eight US states and one Canadian province lost power affecting over 50 million people.

This chapter presents the approaches you could take to ensure that Yantra 7x can continue running after a data center outage.

6.1 Disaster Recovery from a Yantra 7x Perspective

In the event of a data center disaster, you may have almost no option other than to reestablish Yantra 7x in a disaster recovery site. This could be an internal site or an office space at a disaster recovery vendor.

Generally, when dealing with a disaster recovery service site, you have to decide on the level of recovery service - the higher the disaster recovery service, the higher the price. Keeping in mind the Insurance Principle, you need to weigh the likelihood of a disaster occurring, and the cost of the disaster recovery service, against the potential impact to your business due to a prolonged outage.

In the disaster recovery industry the terms cold, warm, and hot site recovery are often used to describe the level of service. A cold site recovery is a term that typically refers to a recovery site that may or may not have equipment provisioned. Depending on your disaster recovery contract, you may have to bring all of your equipment, computing nodes, software, and so forth. In some cases, the disaster recovery vendor may have a pool of equipment that you can draw from. In either case, you have to entertain the possibility that you or your vendors may face a shortage of equipment if multiple customers simultaneously declare disasters.

Typically, the software and equipment are not pre-configured in cold site. Therefore, a cold-site recovery involves a very lengthy and complicated recovery from scratch that could take many days.

A warm recovery site is one where the application may be installed on pre-configured standby equipment and nodes. The data in a warm site are generally updated periodically. Recovery in a warm site typically involves bringing the standby database to the latest consistent state. This generally involves applying all the available transaction logs. A warm-site recovery could take up to a day.

A hot recovery site is one where the application is configured and available at a moments notice. The applications data, ranging from the database to configuration information, are synchronized with the primary data center. A hot-site recovery could take a few minutes to a few hours.

6.2 Cold Site Recovery

A cold site recovery can be daunting especially for a large complex system like Yantra 7x. At a minimum, you will have to procure, install, and configure all the hardware equipment needed by the application ranging from network equipment, load balancers, mid-tier and database nodes, SAN, cabling for the SAN, and so forth.

Next, you have to install and configure all the system software ranging from the operating system, database management system, application server, Yantra 7x, and so forth. It is critically important that the software version and release, and even the same patches be installed the same as the primary data center. Installing different software versions may result in unexpected behavior.

Next, you will have to configure the environment. At a minimum, this includes:

- Defining all the service, host, and server names to DNS
- Defining the message queues
- Setting all the configuration and performance parameters (for example, the operating system kernel parameters, the database parameters). Again, it is important that these parameters be set to the same values as the corresponding parameters in the primary site.
- Installing and preparing the SAN including defining the storage and file systems
- Loading the application database schema

After the infrastructure and environment is available:

- Restore the database from the backup tapes
- Roll forward all the transaction logs to bring the database up to the latest consistent state
- Configure the application servers (for example, connection pool)
- Restore the messages from the integration queues – if you do not have a backup of the messages, you will lose all the unprocessed messages in the integration queues
- Install Yantra 7x and reapply all the custom code, extensions, custom XMLs, and the property files
- Reconfigure the load balancer or proxy to the application server cluster
- Define the service names to the IP address at the recovery site
- Establish connections to all the external systems (for example, credit card companies for credit authorization)

A cold site recovery could easily take days.

6.3 Warm and Hot Site Recovery

Warm and hot site recoveries are much faster and potentially less risky compared to the cold site recovery because the system is already

installed and configured, and the data loaded. Customers who need faster recovery may have to go to with warm or hot recovery sites.

6.4 Key to Disaster Recovery

Consider the following key points for disaster recovery:

- [Recovery Procedures](#)
- [Database Backups and Transaction Log Files](#)
- [Integration Queue Replication](#)
- [Service Names Instead of IP Addresses](#)

6.4.1 Recovery Procedures

Given the extensive list of tasks to recover a system, especially for the cold site recovery, the disaster recovery process must be very well documented and tested. Equally important, these procedures and the entire system must be placed under strict change control and management. Changes to the system must be properly reflected in the recovery procedures. The procedures must be tested as part of the change.

6.4.2 Database Backups and Transaction Log Files

In a cold site recovery, the database has to be restored to the last successful backup and the transaction logs replayed to update the database with all the changes performed since the backup. Given the importance of these files, many companies copy these files for off-site storage. In some cases, transaction logs are immediately copied to a remote site when the logs are closed.

Standard copy utilities can only copy files that are not opened for access. If you rely on standard copy utilities, you will not be able to backup the currently active (and open) transaction log.

For warm and hot sites recoveries, you could use log-shipping technologies to not only replicate but also to apply the log transactions to the standby database.

For Oracle, you could use products such as Oracle Data Guard or Quest Shareplex.

For UDB, you could use UDB HADR. However, with HADR, you can replicate to only one standby database. If you want to have a standby UDB database server at the local site and the disaster recovery site, you may have to use a combination of cluster failover software and HADR respectively.

6.4.3 Integration Queue Replication

Integration queues are used to exchange data messages between Yantra 7x and external systems. The messages could be orders placed between supply chain partners, shipping notices to partner warehouses, and so forth. These messages should be persisted to either files or a database.

If you use file-based persistence, you should consider replicating the files to a remote site to prevent loss of messages from local site faults (for example, JMS server crashing, node crashing). You may also want to consider replicating these messages to a remote site to prevent loss of messages from a data center disaster.

As with transaction logs, you cannot rely on standard copy utilities since these files are continuously opened and updated. Instead, you may have to resort to disk-to-disk replication, such as EMC SRDF, to protect the messages in your integration queue.

6.4.4 Service Names Instead of IP Addresses

You must use service names or host names instead of IP addresses when specifying the location of services such as the JNDI, databases, JMS queues, and so forth. The IP address scheme at the recovery site will not be the same as the primary site. If you use IP addresses, you will point to non-existent nodes.

A

- agent servers, 8, 11, 13, 36
 - queues, 42
- application servers, 13, 34
 - stateful sessions, 35
 - stateless sessions, 35
- architectural patterns, 49
- availability, 1, 2
 - design, 2
 - motivation, 4
 - principles, 2
 - requirements, 2

B

- BEA WebLogic
 - application server, 11
 - implementation, 1
 - monitoring, 1
 - planning, 1
 - tuning, 1

C

- caching, 50
- client reroute, 23
 - limitations, 23
- Clustered MQ, 47

D

- database servers, 8, 18, 22

DBMS

- failures, 19
- deployment processes, 52
- disaster recovery, 3, 5, 21, 53
 - cold site recovery, 54
 - procedures, 56
 - warm and hot site recovery, 55
- disk subsystems, 34
- downtimes, 1, 16

F

- failover configurations, 17, 20, 25, 26, 33
- failovers
 - IBM UDB, 24

H

- hardware failures, 20
- High Availability Disaster Recovery, 21
- hot deployment, 51
- human errors, 19

I

- IBM WebSphere
 - application server, 11
 - implementation, 1
 - monitoring, 1
 - planning, 1
 - tuning, 1
- implementation, 1
- integration

- asynchronous, 50
- integration servers, 8, 11, 13, 36
- queues, 41

J

- Java Virtual Machine, 12
- JNDI, 36
 - failovers
 - limitations, 40
 - parameters, 37
- JNDI service, 14

L

- load balancing
 - server-side, 28
- load balancing
 - client-side, 27

M

- mean-time-to-repair, 16
- message queues, 20, 41
- messages
 - asynchronous, 2
- monitoring, 1

N

- networked file systems, 47
- nodes, 16, 25

O

- operator errors, 19
- outages, 2, 16

P

- planning, 1

R

- RAC server instance failures, 32
- regression testing, 52

S

- single-points-of-failure, 8, 15
 - eliminating, 38
- single-site configuration, 7

T

- target node utilization, 33
- time-triggered transactions, 8
- transactions
 - loss, 8
 - loss in integration queues, 9
- tuning, 1

W

- WebLogic JMS, 42
- WebSphere MQ, 45
- workloads, 2

Y

- Yantra 7x application architecture, 11