Sterling Selling and Fulfillment Foundation

High Availability Guide

Release 9.1.0.18

Sterling Selling and Fulfillment Foundation

High Availability Guide

Release 9.1.0.18

Note
Before using this information and the product it supports, read the information in "Notices" on page 43.

Contents

Chapter 1. Introduction to High Availability

IBM® Sterling Selling and Fulfillment Foundation and the IBM Sterling 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.

Approaches can be taken to increase the resiliency of Sterling Selling and Fulfillment Foundation to environmental, hardware, or software faults or failure. Techniques and architectural patterns can minimize the impact on the overall ecosystem in the event of a planned or unplanned Sterling Selling and Fulfillment Foundation system outage. At the same time, pragmatic approach to availability recognizes that one cannot implement high availability techniques at the detriment of other architectural considerations such as capital cost, ongoing total cost of ownership, or manageability.

Assumptions and Requirements

If your Sterling Selling and Fulfillment Foundation configuration supports a large number of users from the Web, your wide-area network deployment must be configured in such a way that single (or multiple) faults do not cause an outage.

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 that warrants detailed study and is not addressed here.

Environmental and infrastructure availability is also assumed. The data center must be built with redundant power circuits, redundant cooling and so forth so that the infrastructure remains available with single or multiple environmental faults. The data center must also be sufficiently equipped with an uninterrupted power supply (UPS) so that all hardware components can operate under brief power fluctuations. It must also be equipped with power generators to continue working during prolonged power outages.

Availability Design and Principles

Availability design is a complex subject that covers a large optimization exercise balancing many requirements.

The following principles underlie availability design:

- Business Drives High-Availability Requirements
- Keep-It-Simple Strategy
- Configuring for Higher Availability or Resilience is Like Buying Insurance

Business Drives High-Availability Requirements

High availability (HA) requirements should be driven by business and not implemented for the sake of technology. Seeking to use a new high availability technology, supporting a clustered file system, etc. may be interesting and

intellectually satisfying. The most important consideration is that these technologies advances the business goals without making the system overly complicated or expensive.

In some cases, the business may be able to tolerate a certain amount of outage and a simple backup and restore may suffice. Of course, there are others where an hour of downtime is very expensive and as a result 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.

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

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 the automated, and so forth.

The 9s

Specifying availability requirements is not as simple as the often quoted "99.999% availability." In its simplest form, the 9s is an indication of how much downtime an application is allowed to incur. The following table shows that each additional 9 drops the amount of time the application could be down for by an order of magnitude:

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

Therefore, specifying that a system have 99.999% availability means that the system can be down for less than 5.3 minutes in a year.

Problem with the 9s

The problem with using the 9s as a requirements is that 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, a customer can integrate the customer-facing Web site to Sterling Selling and Fulfillment Foundation, using asynchronous messages. With this architectural pattern, Sterling Selling and Fulfillment Foundation can be taken offline for maintenance (such as upgrades) without impacting the services provided by the Web site.

The use of the 9s also does 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.

In general, if architected correctly, the Sterling Selling and Fulfillment Foundation, which is typically used as a backroom order processing engine, does not have high availability requirements. In contrast, some applications, for example, Internet facing applications, have very high availability requirements because they are customer facing.

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.

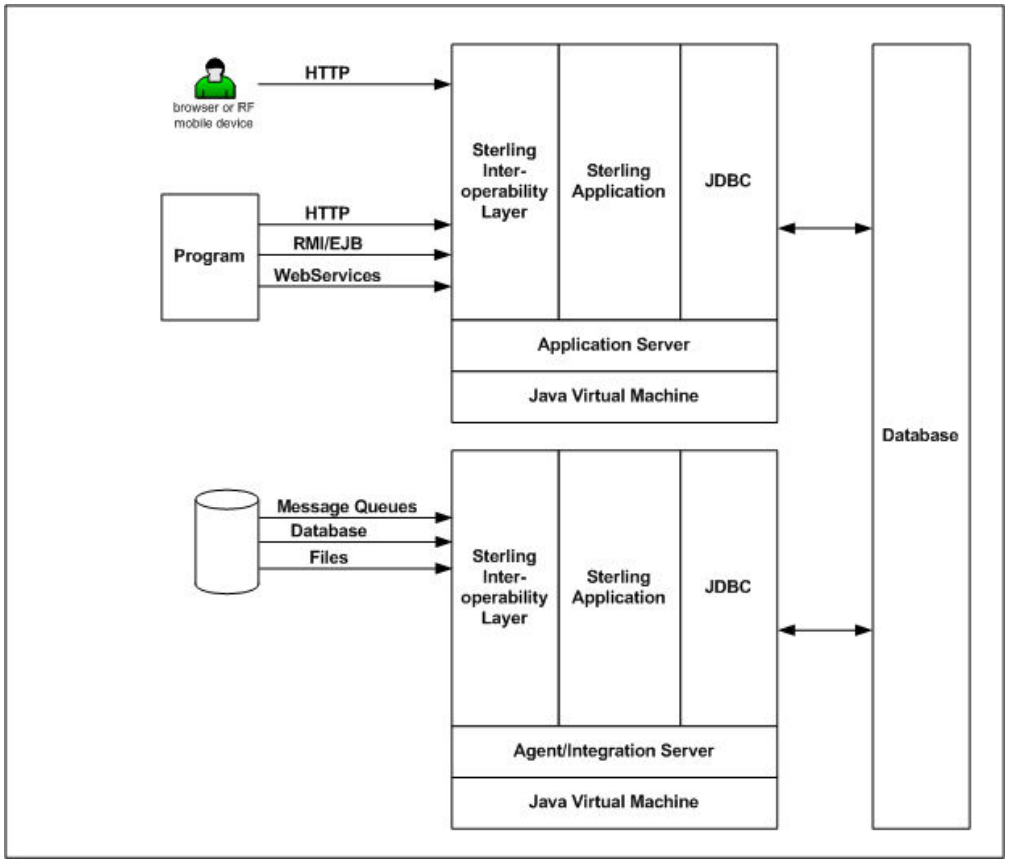
- Emerging regulations are forcing availability. In the health care industry, the Health Insurance Portability and Accountability Act (HIPAA) mandates business continuity and availability planning. Section 404 of Sarbanes-Oxley specifies that corporations must protect the systems used to report financial 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.

Chapter 2. IBM Sterling Selling and Fulfillment Foundation Architecture

Sterling Selling and Fulfillment Foundation Server Components

Sterling Selling and Fulfillment Foundation runs on one of the following server components, as depicted in the following figure:

- Application server
- Sterling Selling and Fulfillment Foundation agent or integration server



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 Sterling Selling and Fulfillment Foundation 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)

Application Server and Application Console

The application servers are the processes that handle synchronous requests to provide real-time access to the features and application logic within Sterling Selling and Fulfillment Foundation.

The most common type of requests that an application server handles are the requests originating from clients using the Application Console. The application servers are always deployed using an industrial-strength server application such as Oracle WebLogic, IBM WebSphere® or JBoss Application Server.

The 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.

Typical usage scenarios include:

- The call center representative uses the Application Console to interact with Sterling Selling and Fulfillment Foundation. 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.

Agent or Integration Servers

Integration Servers are Java-based processes that run in the background to process various tasks. Integration servers allow Sterling Selling and Fulfillment Foundation to collaborate with different systems, organizations, and businesses—all through a standard, uniform interface to all systems. Integration Servers and the tasks that they perform are configured through the means of the Service Definition Framework. For more information, see the *Sterling Selling and Fulfillment Foundation: System Management and Administration Guide*.

The integration servers that process information from external systems can get work from message queues, database tables, and files. Integration servers that send work to external systems do so through a variety of transport mechanisms such as message queues, email, database tables, files, etc.

An agent server is a specialized sub-class of the integration server that runs the Sterling Selling and Fulfillment Foundationdefined “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 has 3 threads of the Schedule Order and Release Order transactions. In total, you 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.

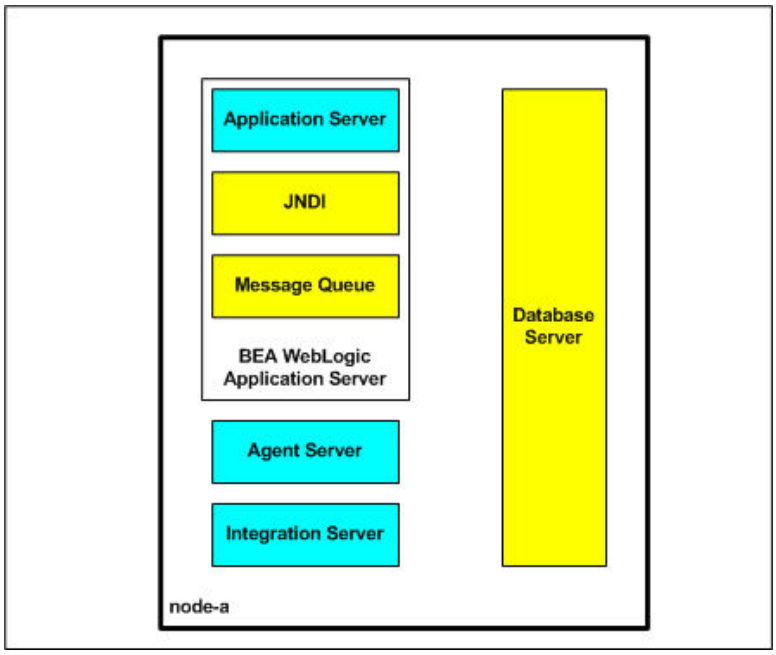
JNDI Service

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 Sterling Selling and Fulfillment Foundation 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 IBM WebSphere MQ message queue service and JBoss Messaging service.

Chapter 3. Limited Redundancy Single-Site Configuration

We will start the discussion on availability by discussing the attributes of a simple entry-level configuration based on standard off-the-shelf products without additional resources or configuration for availability. It is unlikely that this configuration will be used in production. Its value is as a baseline from which we can build in availability.



This figure illustrates:

- a single database instance where application and configuration data are persisted to. The database files are implemented on non-redundant internal drives.
- a single (non-clustered) application server where the JMS message queues are also implemented in. This application server runs on a single JVM.
- a single agent server where all the Sterling Selling and Fulfillment Foundation time-triggered transactions are configured to run in
- a single integration server where all integration services run in

As an entry-level configuration, all three servers – the application server, integration server and agent server – run on a single node (node-a) along with the database server.

Single Point of Failure

As expected, this system has many single-points-of-failure (SPOF) where a single fault can cause a partial or complete Sterling Selling and Fulfillment Foundation system or application outage. For example,

- failure of the node (node-a) will cause a complete system outage.
- failure of the Sterling Selling and Fulfillment Foundation agent server will affect all services provided by the application server. In addition, it would halt all services running the integration and application servers that depend on the message queues.

- failure of the database instance due to software errors or disk errors will cause a complete system outage since the application is strongly reliant on the data in the database.

Loss of Data

These SPOFs will, at a minimum, cause a system outage. Assuming the system is backed up on a regular basis, one should be able to restore services. However, some of these SPOFs in this entry-level system can result in loss of application or business data. We will discuss these error conditions so that you can protect against them in your system.

Loss of Database

First, it should be obvious that losing the disks that the database files are implemented on will result in the loss of the database data. One can easily protect the data by ensuring that the database files are stored on external redundant storage. These devices could range from entry-level storage devices (like Dell's PowerVault MD1000) to high end SAN storage devices (like EMC Symmetrix).

Loss of Database Transaction Logs

First, failure of the non-redundant internal disk can result in the loss of transaction data in the database which can result in loss of transaction data. A database management system (like Oracle, DB2® and Microsoft SQL Server) guarantees the integrity of its data. When a transaction commits its work, the DBMS guarantees that all the changes are either in the database disks or can be recovered from transaction logs.

When a database instance crashes, the DBMS is designed to automatically perform "rollforward or instance recovery". In a nutshell, when you restart the instance, the DBMS will ensure that "committed" changes in its retransaction log are applied to the database files. Similarly, if you had to recover the database from backup, you could also initiate a rollforward recovery from the transaction logs to reapply all the transactions since the backup was taken.

The loss of the transaction logs typically means that at best, the DBMS cannot perform rollforward recovery and at worse, you have to recover the database from the last backup. In either case, this system could lose transaction data.

Never place transaction logs or database files on non-redundant internal disks. These critical files should be placed on redundant storage devices.

Applicability of Limited Redundancy Single-Site Configuration

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.

Chapter 4. High Availability Within a Single Site

Single Points of Failure

The number of SPOF increases as a system grows in size and complexity. Within the four walls of the data center, potential single points of failure include:

- Node
- Database Server
- SAN or Disk Subsystem
- Sterling Selling and Fulfillment Foundation Components
- Server Registry
- Message Queues
- Networked File Systems

Ensuring Against Node Failure

The term ‘node' refers to the physical computing hardware on which Sterling Selling and Fulfillment Foundation 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. (Source: IBM eServer™ p5 590 and 595 System Handbook, SG24-9119-00, IBM Corp, March 17, 2005.)

Similarly, nodes typically are configured with multiple critical components such as power and fans so that they 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 node.

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. If the node was running a few agent servers, the impact could be isolated to just the services provided by those agents. In contrast, if the outage was in the database server node (and the database is not clustered), the outage will be to the entire application.

If your tolerance for downtime is low, you have the following options

- Ensure that your nodes are composed of high redundant servers (as described above) to reduce the likelihood of a node outage caused by hardware faults

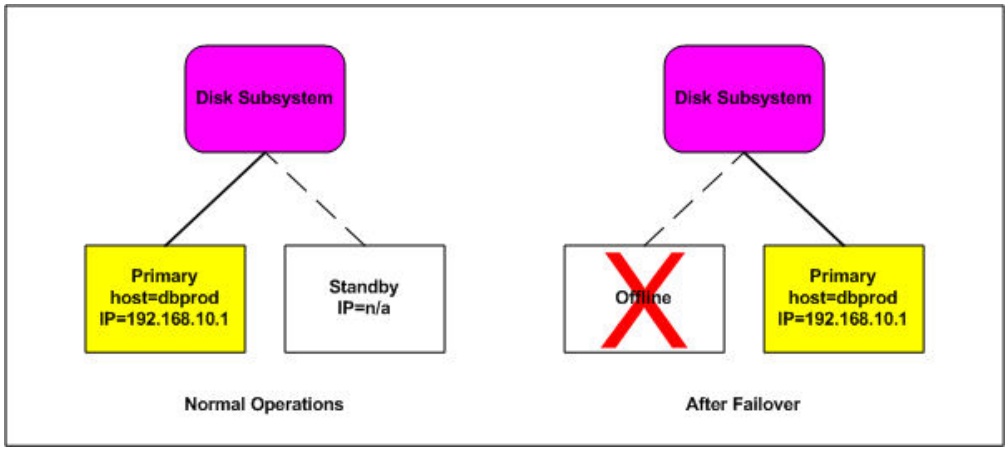
- Use active/passive or primary/standby failover configuration where one or more passive or standby nodes are available to take over for failed nodes. See "Active/Passive Cluster Failover Configurations" for more information. You can use this approach in subsequent sections to protect critical components such as message queues and the application and agent servers. In "Active/Passive Failover Configurations", we present active/passive configurations for database servers.
- Use the clustering capabilities built into application servers and in Oracle Real Application Cluster to protect against outage from a single node failure. "Active/Active Failover Configurations" describes in the use of an active/active clustered database failover configuration.

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.

The following figure 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.



Active/Passive Database Failover Process

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 Sterling Selling and Fulfillment Foundation components in more detail.

Database Server as a Critical System Component

The database server is a critical system component. The entire system 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

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 fix packs available for that database release. From our experience, you may destabilize a database release when you apply too many individual fix packs. In some cases, individual fix packs 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 is likely to be very high.

Human and Operator Errors

A Gartner report says 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." (NSM: Often the Weakest Link in Business Availability, AV-13-9473, July 3, 2001)

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.

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.

Active/Passive Failover Configurations

Active/passive failover configurations provide a fully redundant instance of each node that is brought online only if its associated primary node fails.

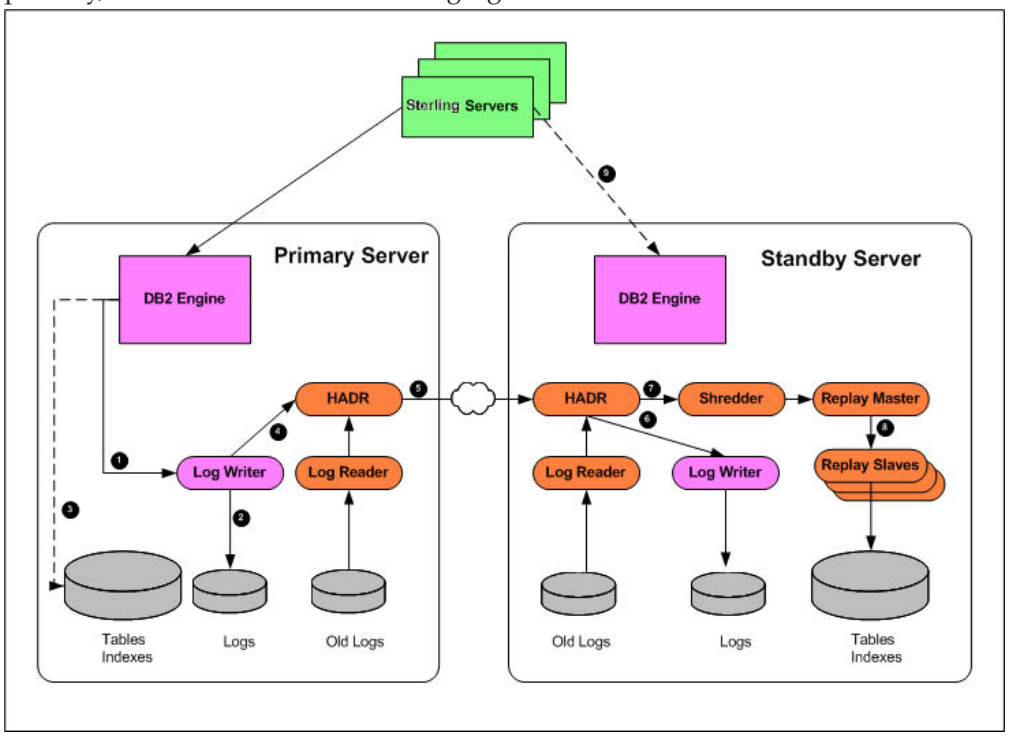
IBM UDB Active/Passive Using Cluster Failover Software: Conceptually, UDB active/passive failover configurations using cluster failover software operates as described in "Active/Passive Cluster Failover Configurations". The standby node takes over the primary node's resources (the database files, logs) 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 the Sterling Selling and Fulfillment Foundation 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 throw a SQL error message. The changes from those transactions are correctly rolled back later when the database server comes up on the standby node.
- The Sterling Selling and Fulfillment Foundation servers continually reissue the transactions until the database service is restored. You do not have to restart the Sterling Selling and Fulfillment Foundation 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.

IBM UDB Active/Passive Using HADR: 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 the following figure:



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.

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 9.1'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

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.

Initially, DB2 Universal JDBC Driver client reroute support was available only for connections that use the `javax.sql.DataSource` interface. In DB2 9.1 FP3, IBM added client reroute support to `java.sql.DriverManager`.

Oracle Active/Passive Using Cluster Failover: Oracle active/passive failover configurations are very similar to the UDB active/passive configuration described in "IBM UDB Active/Passive Using Cluster Failover Software". Given the popularity of Oracle Real Application Cluster (RAC), it is our belief that customers are trending towards implementing active/active Oracle failover configuration instead. This is described in "Active/Active Failover Configurations".

Microsoft SQL Server Active/Passive Using MSCS: Microsoft SQL Server active/passive failover configurations are very similar to the"IBM UDB Active/Passive Using HADR".

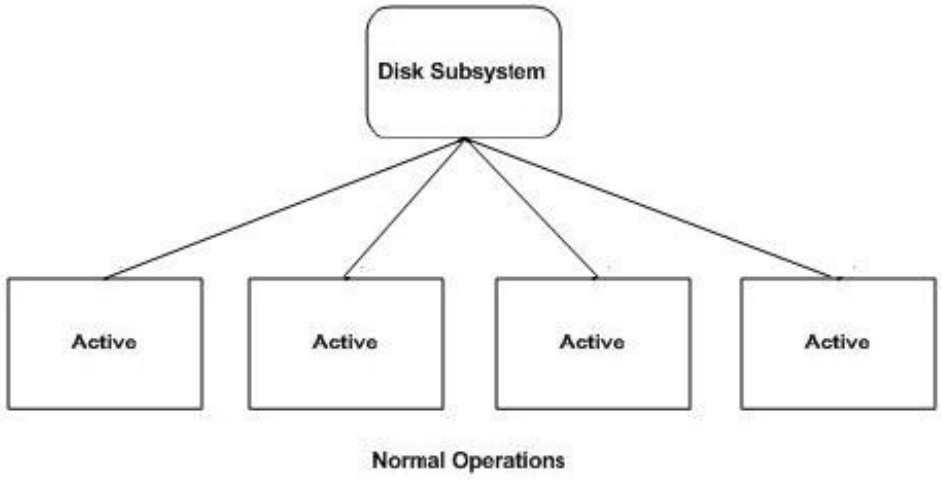
The clustered nodes use the heartbeat to check whether each node is alive, at both the operating system and Microsoft SQL Server level. At the operating system level, the nodes in the cluster compete for the resources of the cluster. The primary node reserves the resource every 3 seconds, and the competing 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 Microsoft SQL Server perspective, the node hosting the Microsoft 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 Microsoft 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 Microsoft SQL Server. If all five retries fail, the Microsoft SQL Server resource fails. Depending on the failover threshold configuration of the Microsoft 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 Microsoft SQL Server, but ultimately fails if its threshold is exceeded.

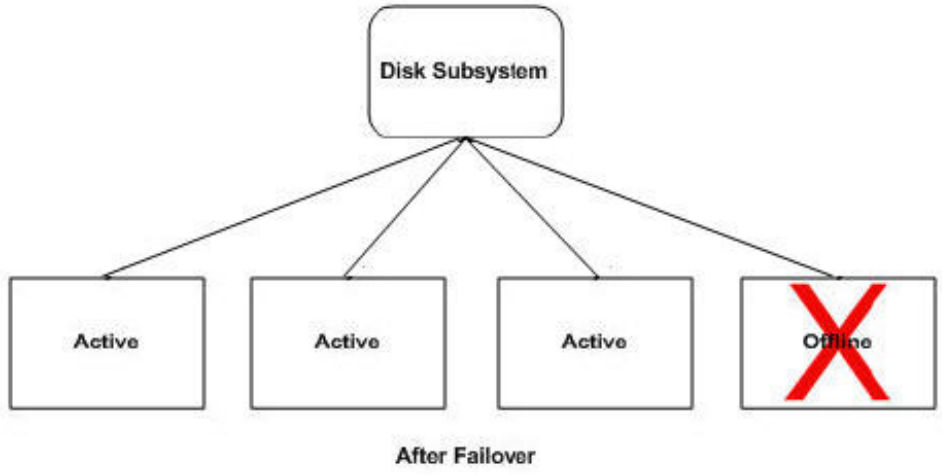
During the fail over from one node to another, Windows clustering starts the Microsoft SQL Server service for that instance on the new node, and goes through the recovery process to start the databases. The fail over of the Microsoft SQL Server virtual server takes a short time (probably seconds). After the service is started and the master database is online, the Microsoft 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.

Active/Active Failover Configurations

In an active/active failover configuration, two or more database nodes are clustered to provide a common service to all the 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. The following figure illustrates this concept:



Normal Operations



After Failover

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 data files. The distributed lock manager controls which node updates the data. It does not matter on which node the transaction is performed. Each node has equal rights to access the data in the shared database.

Each RAC node has a listener process that is responsible for processing the 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.

Oracle has introduced the Grid Infrastructure from Oracle RAC 11g Release 2. With this, Oracle has integrated two of its products – Oracle ASM and the Oracle Clusterware in one software bundle. Oracle Grid Infrastructure provides the necessary functions for volume management, file system, and server pool management to run an Oracle RAC database.

Configuring Sterling Selling and Fulfillment Foundation with Oracle RAC

When configuring Sterling Selling and Fulfillment Foundation 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 using SCAN

Oracle RAC 11g Release 2 introduces the Single Client Access Name (SCAN) to simplify client access to databases. SCAN allows a single name to be used in the client connection requests that does not change as the cluster expands, or if any of the nodes in the cluster changes over time. This allows the use of simplified connect strings for for JDBC. You must define the SCAN in your DNS as a single name that round robins to different IP Addresses of the nodes. These IP addresses must be on the same subnet as the public network for the cluster.

The jdbc url are:

jdbc:oracle:thin:@(DESCRIPTION=(LOAD_BALANCE=ON)(FAILOVER=ON)
(ADDRESS=(PROTOCOL=TCP)(HOST=racscan)(PORT=1521))(CONNECT_DATA=
(SERVER=DEDICATED)(SERVICE_NAME=racdb)))

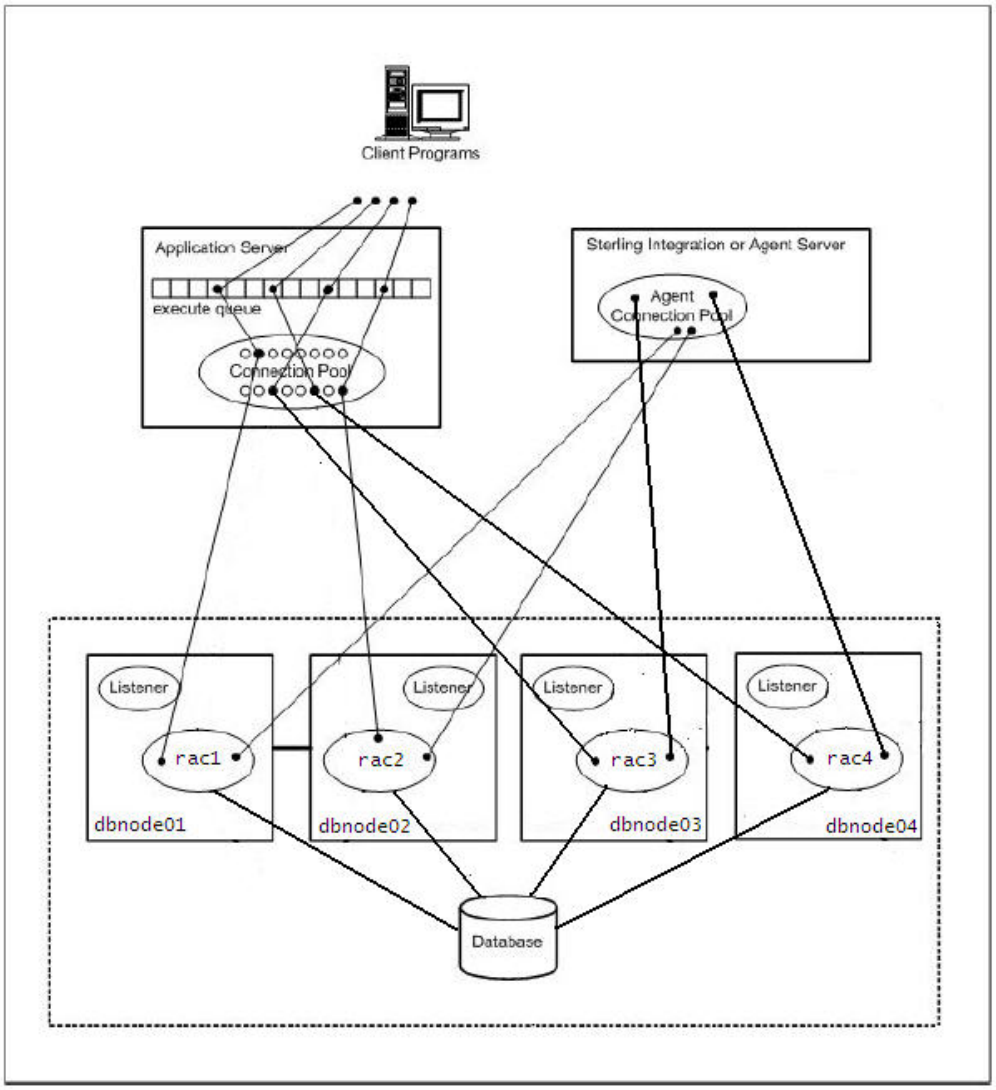
Oracle RAC 11g Release 2 introduces the Grid Naming Service (GNS), which makes it easier to scale by automating the VIP management for Oracle RAC. Sterling Selling and Fulfillment Foundation 9.1 has not been tested with GNS enabled.

For more information on SCAN, please refer Oracle RAC documentation.

All the database nodes will have the following values in the `tnsnames.ora` file

```
RACDB =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = racscan)(PORT = 1521))
   (CONNECT_DATA =
   (SERVER = DEDICATED)
   (SERVICE_NAME = racdb)
  )
   )
```

RACDB is the net_service_name for client-side load balancing, as illustrated in the following figure:



From a Sterling Selling and Fulfillment Foundation 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 back the changes from those transactions.
- The Sterling Selling and Fulfillment Foundation agent and integration servers continually attempt to reconnect. When it has connected to one of the remaining RAC instances, the failed transactions are reprocessed. You do not have to restart the Sterling Selling and Fulfillment Foundation servers during the transition to one of the other active nodes.
- 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 in an Oracle RAC Active/Active Configuration: 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 4-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 degrade or at worst the system becomes unstable.

Ideally, you should try to keep the average node utilization below 60% utilization (for a 4-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%.

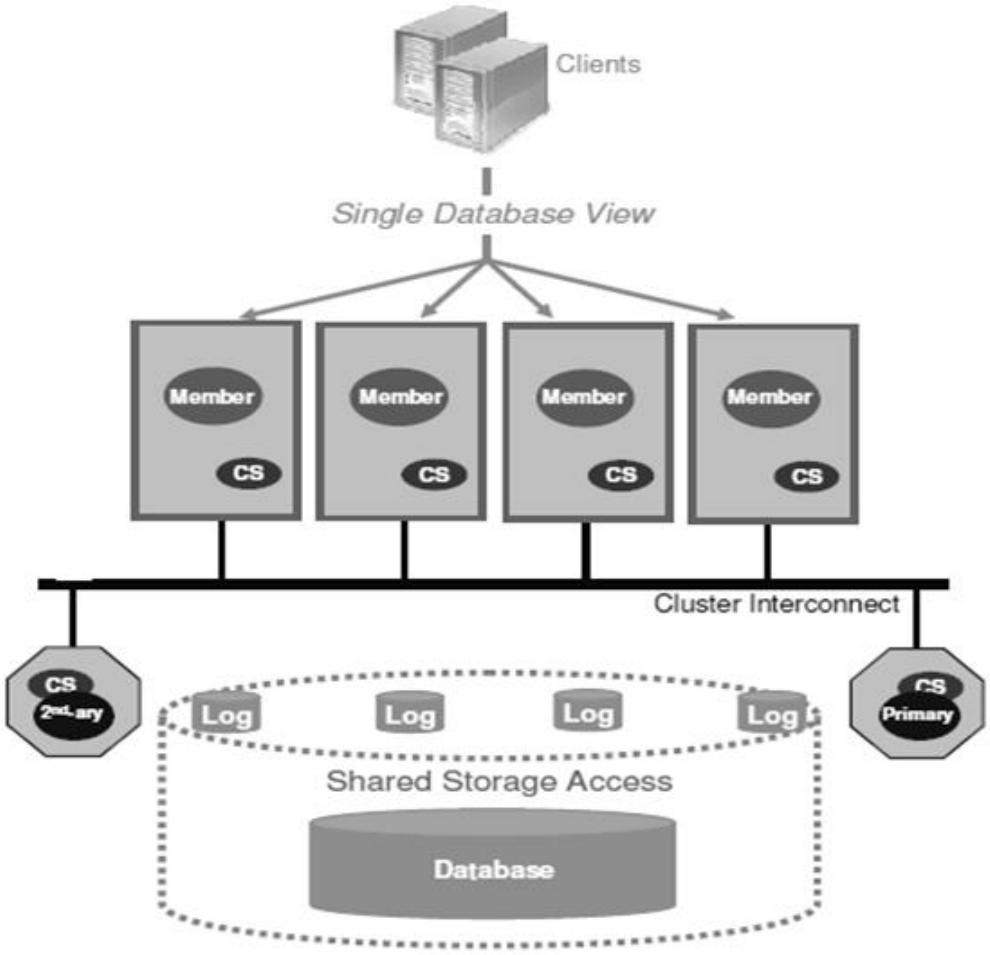
Alternatively, if you typically run the nodes at higher utilization, you should consider classifying workloads into priority groups. For example, transactions that are initiated by customers may be put into the high priority group because they add to the overall customer experience. These could include 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.

IBM DB2 Active/Active Using DB2 Purescale: This topic presents the Sterling specific recommendations and guidelines, that the customers must follow in order to run the Sterling Selling and Fulfillment Foundation applications with a IBM DB2 pureScale database cluster.

For more information about DB2 pureScale, refer to the following documents:

- DB2 pureScale Feature Installation and Upgrade Guide
- DB2 client considerations with the DB2 pureScale Feature

Workload Balancing: DB2 pureScale can be configured to dynamically balance the database workload across all cluster members at the transaction or connection level. When enabled, a DB2 connection that was processing on one member could automatically shift to a less busy member in the next transaction. For example, when you add or restart a pureScale member, DB2 can gradually shift connections from the existing members to a new member to balance the utilization across each cluster members.



To enable workload balancing, ensure to set the IBM DB2 Driver for JDBC property, `enableSysplexWLB` to true. For the Sterling agents and integration servers, set the following parameter in the JDBC property file:

- `jdbcService.db2Pool.prop_enableSysplexWLB=true`

In the WebSphere Application Server administrative console, set the `enableSysplexWLB`property for the data source that your application uses to connect to the Database server.

Automatic Client Reroute: The Sterling Selling and Fulfillment Foundation application uses the following DB2 JDBC URL to connect to one of the DB2 pureScale members.

jdbc:db2://perfdb101:50000/ster_DB

After connecting, the DB2's Automatic Client Reroute (ACR) provides a list of the available cluster members to the DB2 JDBC driver. If a cluster member fails, the driver will automatically send transactions to the surviving members.

In addition to the ACR notification, the JCC driver provides the following parameters to allow you to specify a static list of alternate servers to the DB2 JDBC driver, so that the driver knows to which cluster member to attempt, if the member that the JDBC URL is pointing to is slow.

- jccAlternateServer
- jccAlternateServerPort

You can set these parameters for agent and integration servers with the following parameters in the JDBC property file:

- `jdbcService.db2Pool.prop_clientRerouteAlternateServerName=perfdb101, perfdb102, perfdb103, perfdb104`
- `jdbcService.db2Pool.prop_clientRerouteAlternatePortNumber=50000,50000,50000,50000`

In the WebSphere Application Server administrative console, set the `enableSysplexWLB` property for the data source that your application uses to connect to the Database server.

IBM DB2 Active/Active Using UDB ESE DPF: Sterling Selling and Fulfillment Foundation is not certified to run on UDB DPF.

IBM DB2 Active/Active Using xkoto GRIDSCALE: Sterling Selling and Fulfillment Foundation is not certified to run on xkoto GRIDSCALE.

SAN or Disk Subsystem as a Critical System Component

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

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

Areas where disks are used by Sterling Selling and Fulfillment Foundation 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. We prefer RAID 10 or RAID 5 for performance and redundancy.
- Redundant disk systems to tolerate component failures
- Multiple access paths to the disks

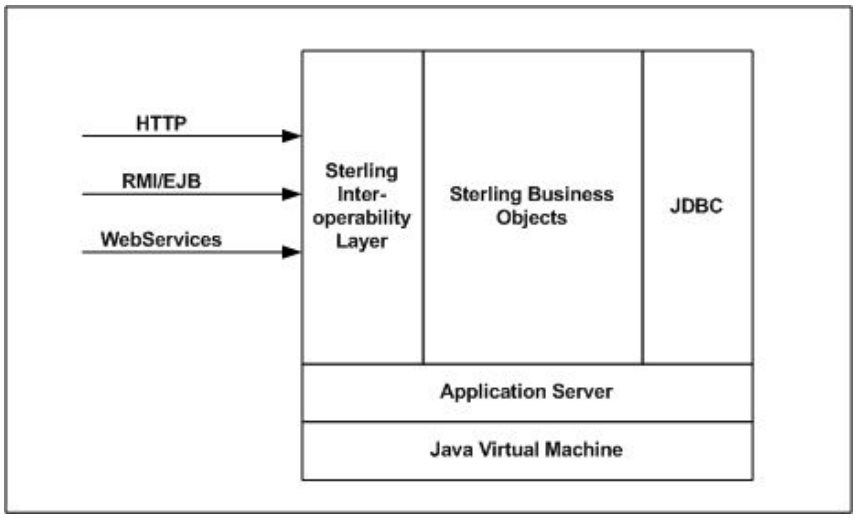
Sterling Selling and Fulfillment Foundation Components

Sterling Selling and Fulfillment Foundation runs in the application server, agent server, and the integration server.

Application Servers in Sterling Selling and Fulfillment Foundation

The application server provides:

- Users with the ability to use the Application Console to create and manage orders, shipments, and inventory manage orders, shipments, and inventory
- Programs to call the Sterling Selling and Fulfillment Foundation business APIs using HTTP, servlets, EJB/RMI, or Web services



Currently, Sterling Selling and Fulfillment Foundation is supported on the following application servers:

- Oracle WebLogic
- IBM WebSphere
- JBoss Application Server

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.

Stateful Sessions on WebLogic, WebSphere, and JBoss Application Servers

The HTTP user interface (or the Application Console) 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.

Types of Stateless Sessions

All the other application server transactions, EJB/RMI, servlet calls, and Web services, are stateless. 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.

Sterling Selling and Fulfillment Foundation Agent and Integration Server

The Sterling Selling and Fulfillment Foundation agent and integration servers are completely location or node independent. They can run from any node where the application has been deployed.

Server Registry in Sterling Selling and Fulfillment Foundation

Sterling Selling and Fulfillment Foundation maintains information on how to locate each of the server instances (e.g., the application server, agent and integration server instances) for system management purposes. For example, a server that changes cached reference data must notify its peers to update their cache. See the *Sterling Selling and Fulfillment Foundation: Performance Management Guide* for more details.

When a server instances starts, it stores its server name and the URL to itself in a registry. In Sterling Selling and Fulfillment Foundation, Release 9.1, the registry resides in the YFS_HEARTBEAT database table.

Any server instance can query the YFS_HEARTBEAT to find all the other server instances.

Sterling Selling and Fulfillment Foundation uses the JNDI information for the following events:

- Reference data cache refresh – Sterling Selling and Fulfillment Foundation implements 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 Sterling Selling and Fulfillment Foundation reference data cache implementation, refer to the *Sterling Selling and Fulfillment Foundation: Performance Management Guide*.
- The System Management Console uses the registry to discover all the application instances. The SMC uses that list to build a list of instances to monitor.

In addition to the initial registration on start up, every server has to periodically update its registry record to indicate that it is still alive.

In past releases, if the JNDI service was unavailable, you would not be able to start servers or notify peers of system management changes. In this release, the registry's availability posture is the same as the database service. If the database service is down, the application would be down therefore, there would be no need for the registry.

Message Queues in Sterling Selling and Fulfillment Foundation

Sterling Selling and Fulfillment Foundation uses message queues extensively. Message queue usage in Sterling Selling and Fulfillment Foundation can be characterized as:

- Integration queues for external communications

- Temporary work-in-progress queues for Sterling Selling and Fulfillment Foundation and custom time-triggered transactions

These queues can be implemented in:

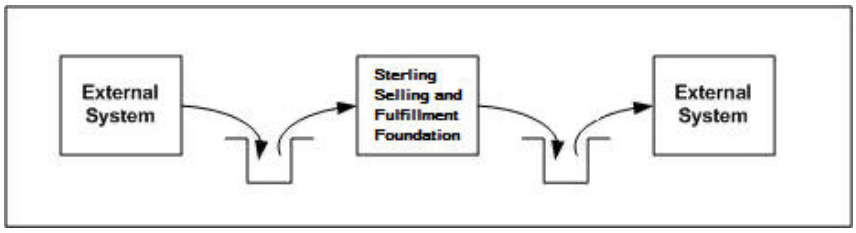
- Oracle WebLogic JMS
- IBM WebSphere MQ
- IBM WebSphere default messaging providers
- JBoss Messaging

Sterling Selling and Fulfillment Foundation agent and integration servers use messaging primarily for two reasons:

- Integration queues
- Agent work queues

Integration Queues for Sterling Selling and Fulfillment Foundation Integration Servers

The integration servers use integration-based queues to communicate from or to external systems. For example, in the following figure, the first queue could be used by external systems such as a Web store frontend to pass order creation requests to Sterling Selling and Fulfillment Foundation. It could also be purchase orders from a purchasing system, shipment status updates from a logistics management system, and so forth.



Similarly, Sterling Selling and Fulfillment Foundation integration servers can use integration queues to send messages to external systems. For example, Sterling Selling and Fulfillment Foundation can send ship notices to warehouses.

Since these messages are used for communicating between systems, the messages in the integration queues should be protected so that they are not lost in the event of a failure. In some cases, these messages can be difficult to recreate. For example, as described above, if messages from a Web store is lost, the Web store will have to resend the missing orders. The recovery will involve having to determine what orders have already been processed to find what is missing. Care will have to be taken to ensure that orders that have already been processed are not resent.

In general, integration queues should be implemented on reliable redundant persistent stores.

Agent Queues for Sterling Selling and Fulfillment Foundation Agent Servers

Sterling Selling and Fulfillment Foundation agent servers use the messages in the queues as a source of work. In contrast to integration queue messages, agent messages are typically read from the database and can be easily recreated. As a result, we recommend creating agent queues as non-persistent queues where the messages are kept in memory.

Protecting WebLogic JMS Queues

To protect WebLogic JMS queues, consider the following:

- Defining WebLogic JMS Message Queues as Persistent
- Enabling Message Paging in WebLogic JMS queues
- Using a Dedicated JMS Server for Integration
- Implementing a Shared Disk Subsystem
- Locating the JMS Server on a Different Application Server

Defining WebLogic JMS Message Queues as Persistent

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 integration 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 Sterling Selling and Fulfillment Foundation agents are designed to be able to recreate the work-in-progress task messages.

Enabling Message Paging in WebLogic JMS queues

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 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.

Using a Dedicated JMS Server for Integration

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 server. The two JMS servers are then targeted to two dedicated application servers.

Implementing on a 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.

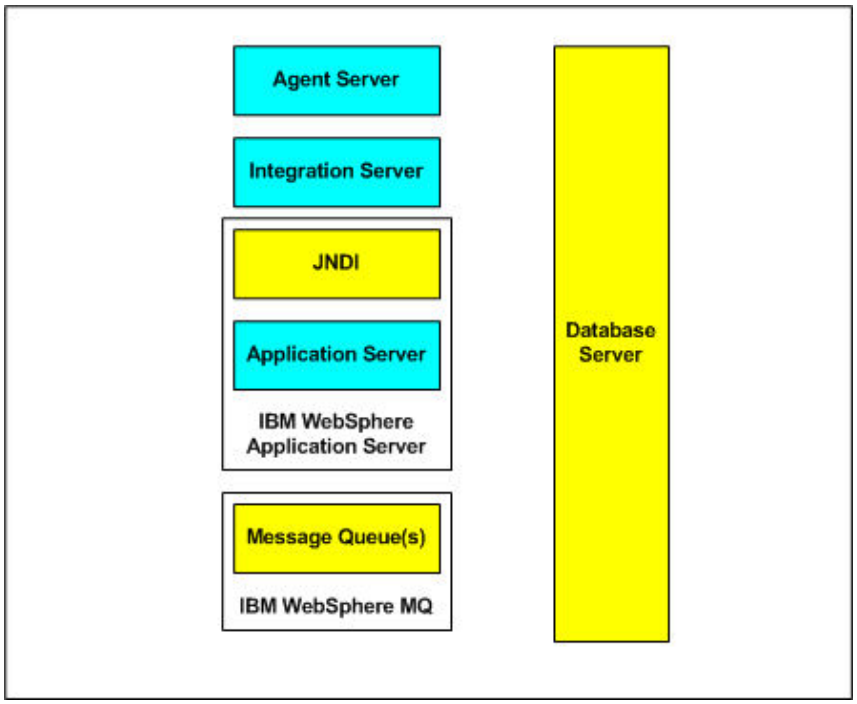
Locating the JMS Server on a Different Application Server

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).

Implementing IBM WebSphere MQ Queues

As shown in the following figure, integration and agent queues can be implemented on IBM WebSphere MQ using either of the following methods:

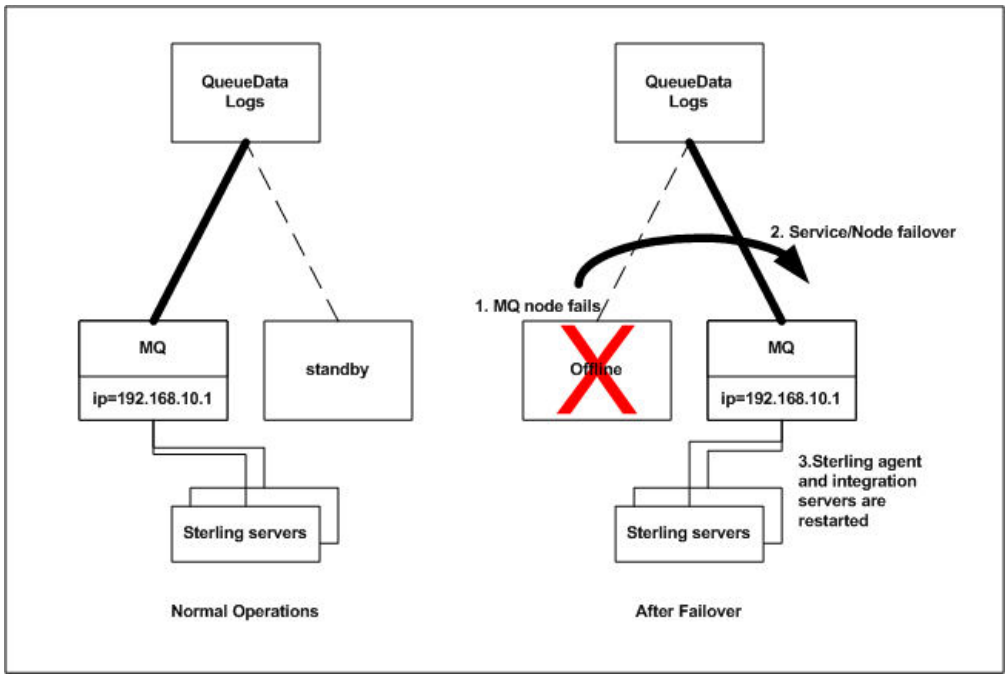
- WebSphere MQ, which is an external message queue
- WebSphere default messaging, which is the messaging component inside the WebSphere Application Server.



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 the following figure, 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, no changes are required.



Implementing Message Persistence Files on a 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.

Risks in Implementing Message Persistence Files on SAN

In the previous examples, the shared storage remains the single point of failure. 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.

Important Notes for IBM WebSphere MQ

- Configuring IBM WebSphere MQ in an active/standby configuration with HACMP requires specialized knowledge. Please consult your IBM representative for assistance.
- IBM on clustered MQ.

Implementing JBoss Messaging Queues

Protecting JBoss Messaging Queue Using Persistence

By default, JBoss Messaging is shipped with a Persistence Manager which uses Hypersonic database for persistence. This database is included in the JBoss EAP installation. However, Hypersonic is not supported in a production environment and should not be used. You can use any of the supported databases to configure

message Persistence. Also, the persistence manager allows implementation of non-relational stores or file stores instead of database persistence.

JBoss also has a Null persistence manager which can be configured when persistence is not required.

In Sterling Selling and Fulfillment Foundation, to protect the integration queue messages, define the integration queues as persistent.

The agent queues should be implemented as non-persistent queues. The Sterling Selling and Fulfillment Foundation agents are designed to be able to recreate the work-in-progress task messages.

Configurable Dead Letter Queues

Dead Letter Queue (DLQ) is a destination for messages that the server has failed to deliver more than a certain number of times. If DLQ is not specified, the message will be removed after the maximum number of delivery attempts. This will help you identify the message delivery exceptions and ensure that the messages are not lost. You can read the messages from DLQ and fix the application accordingly.

By default, DLQ is configured in JBoss Messaging at the global level for all queues.

Locating the 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).

Implementing on a Shared Disk File System

To prevent disk failures and outages, include the TIBCO EMS server logs, configuration files and file stores on a shared SAN to be accessed by multiple nodes. 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 will allow you to restart the Messaging server from another node.

Implementing TIBCO EMS Queues

Defining Messaging Queues as Persistence

By default, TIBCO EMS uses file based stores in which persistent messages are written asynchronously to disk. There are options available to configure to work in synchronous mode. TIBCO EMS allows you to configure different types of stores namely database stores and mstores. There is also an option to configure separate stores for different queue destinations.

In Sterling Selling and Fulfillment Foundation, to protect the integration queue messages, define the integration queues as persistent.

The agent queues should be implemented as non-persistent queues. The Sterling Selling and Fulfillment Foundation agents are designed to be able to recreate the work-in-progress task messages.

High Availability using TIBCO EMS Fault Tolerance

TIBCO EMS servers provides fault tolerant mechanism which allows you to configure two servers - one being the primary server and the other as backup or standby server. The primary server accepts client connections or requests and interacts with clients to send or receive messages. If the primary server fails, the backup server takes over, it becomes the new active server and resumes the operation. It does not support more than two servers in this configuration.

TIBCO EMS provides you options to configure shared state and unshared state failover. To avoid message loss, duplication or out of order message delivery, it is always preferred to use shared state failover.

Implementing on a Shared Disk File System

To prevent disk failures and outages, include the TIBCO EMS server logs, configuration files and file stores on a shared SAN to be accessed by multiple nodes. 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 will allow you to restart the Messaging server from another node.

Networked File Systems in Sterling Selling and Fulfillment Foundation

Some customers prefer to implement Sterling Selling and Fulfillment Foundation 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 servers of Sterling Selling and Fulfillment Foundation.

Chapter 5. Architectural Patterns for High Availability

The IBM Sterling Distributed Order Management (DOM) and Sterling 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 Sterling Selling and Fulfillment Foundation-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 Sterling Selling and Fulfillment Foundation system can actually increase the availability and uptime of the solution as a whole to levels above what Sterling Selling and Fulfillment Foundation 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.

Asynchronous Integration as a Decoupling Technique

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 Sterling Selling and Fulfillment Foundation. These systems could send the order creation requests to Sterling Selling and Fulfillment Foundation:

- Synchronously using protocols such as HTTP, Web services, 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 Sterling Selling and Fulfillment Foundation 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 communication is synchronous, Sterling Selling and Fulfillment Foundation must be available for the external system to create the request. In this architecture, the availability requirements of Sterling Selling and Fulfillment Foundation have to be the greatest 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.

Caching as a Decoupling Technique

Another common decoupling technique is the use of local caching. In this pattern, the consuming application (for example, 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 Sterling Selling and Fulfillment Foundation.

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 Sterling Selling and Fulfillment Foundation typically recommends utilizing this algorithm is for caching ATP (Available to Promise) data on the Web site. In some customer environments where shopping cart abandonment rates are very high, for example 100 item lookups to one item ordered, it is better to have Sterling Selling and Fulfillment Foundation push out item availability to the Web storefront using the Sterling Selling and Fulfillment Foundation 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 Web site without any synchronous calls to Sterling Selling and Fulfillment Foundation. 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, Sterling Selling and Fulfillment Foundation 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.

Hot Deployment of Code, Configuration, and Fixes

While the methodologies and design patterns presented insulate critical areas of the solution from downtime, there are deployment techniques that are provided by IBM or are inherent within the architecture of Sterling Selling and Fulfillment Foundation 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 the dependence of Sterling Selling and Fulfillment Foundation 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.

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 fix packs and upgrades. In industry studies and based on IBM'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.

Chapter 6. Disaster Recovery

With the approaches described in the High Availability Within a Single Site topics, 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 for High Availability," on page 35, you may be able to schedule downtime with less impact to the corporation's 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.

Disaster Recovery from a Sterling Selling and Fulfillment Foundation Perspective

In the event of a data center disaster, you may have almost no option other than to re-establish Sterling Selling and Fulfillment Foundation 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.

For Testing Purposes

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. For more information, see "Key Issues in Disaster Recovery".

Cold Site Recovery

A cold site recovery can be daunting, especially for a large complex system like Sterling Selling and Fulfillment Foundation. At a minimum, you 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, Sterling Selling and Fulfillment Foundation, and so forth. It is critically important that the software version and release, and even the same fix packs be installed the same as the primary data center. Installing different software versions may result in unexpected behavior.

Next, you 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, all the of the unprocessed messages in the integration queues are lost
- Install Sterling Selling and Fulfillment Foundation 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.

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.

Key Issues in Disaster Recovery

Key issues in disaster recovery include:

- Recovery Procedures
- Database Backups and Transaction Log Files
- Integration Queue Replication
- Use of Service Names Instead of IP Addresses

Recovery Procedures

Given the extensive list of tasks to recover a system, especially for 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.

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 can not 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.

Integration Queue Replication

Integration queues are used to exchange data messages between Sterling Selling and Fulfillment Foundation 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.

Use of 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, and JMS queues. The IP address scheme at the recovery site is not the same as the primary site. If you use IP addresses, you will point to non-existent nodes.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law

IBM Japan Ltd.

1623-14, Shimotsuruma, Yamato-shi

Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be

incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

J46A/G4

555 Bailey Avenue

San Jose, CA 95141-1003

U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© IBM 2011. Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 2011.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

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

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium and the Ultrium Logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Connect Control Center®, Connect:Direct®, Connect:Enterprise®, Gentran®, Gentran®:Basic®, Gentran:Control®, Gentran:Director®, Gentran:Plus®, Gentran:Realtime®, Gentran:Server®, Gentran:Viewpoint®, Sterling Commerce™, Sterling Information Broker®, and Sterling Integrator® are trademarks or registered trademarks of Sterling Commerce™, Inc., an IBM Company.

Other company, product, and service names may be trademarks or service marks of others.

Index

Printed in USA