Sterling Selling and Fulfillment Foundation

Performance Management Guide

Release 9.1.0.18

Sterling Selling and Fulfillment Foundation

Performance Management Guide

Release 9.1.0.18

Note

Before using this information and the product it supports, read the information in “Notices” on page 147.

Copyright

This edition applies to the 9.1 Version of IBM Sterling Selling and Fulfillment Foundation and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Performance Management Overview

This document is the Performance Management Guide for the IBM® Sterling Selling and Fulfillment Foundation Release 9.1.

Performance Management is defined as all the activities one performs to ensure responsive service and processing throughput that meet the business needs at an acceptable cost.

Lifecycle

Performance Management activities occur throughout the project lifecycle. They could range from initial hardware sizing studies during the pre-sales phase, architectural trade-off studies and risk mitigation studies during the design phase, load or system tests prior to implementation, to continual system monitoring and tuning in production.

System Components and Roles

Performance Management activities are wide-ranging and affect all aspects of the system ranging from computer nodes, network, disks, application servers to Sterling Selling and Fulfillment Foundation.

One person (or role) may be responsible for one, several or all of the components. Some typical roles include:

- Hardware Engineer
- System Administrator
- Local Area Network Engineer
- Wide Area Network Engineer
- Application Server Administrator
- Database Administrator
- Sterling Selling and Fulfillment Foundation Administrator
- Capacity Planner
- Performance Analyst
- Architect/Planner

Given the diversity of interest and responsibilities, we have arranged this document into the following parts.

For example, the sections in Chapter 3, “Computer Systems,” on page 15 present the steps needed to configure the computer system nodes for Sterling Selling and Fulfillment Foundation. This section should be of interest to the Hardware Engineers, System Administrators, Local Area Network Engineers, and Wide Area Network Engineers.

The sections in “General JVM Recommendations” on page 25 explain how to configure the Java Virtual Machine (JVM). The JVM is the operating environment for Java applications which include the IBM WebSphere®, Oracle WebLogic, JBoss application server, the Sterling Selling and Fulfillment Foundation Agent/Monitor

Servers, and so on. This information should be of interest to the Application Server Administrators and Sterling Selling and Fulfillment Foundation Administrators.

The sections in “Oracle's WebLogic Tuning Recommendations” on page 51 present the steps needed to configure the IBM WebSphere, Oracle WebLogic, and JBoss application servers. This component provides the runtime environment for Sterling Selling and Fulfillment Foundation. This information should be of interest to the Application Server Administrators.

The sections in Chapter 6, “Database Management System,” on page 59 discusses the key recommendations for the IBM DB2®, Oracle, and Microsoft SQL Server database servers.

“Oracle Real Application Cluster Database” on page 78 discusses the recommendations for implementing Oracle Real Application Cluster for scalability and high availability.

“General Recommendations” on page 91 discusses how to configure Sterling Selling and Fulfillment Foundation. This section should be of interest to the Sterling Selling and Fulfillment Foundation Administrator.

The Performance Analyst or the person who is responsible for monitoring the Sterling Selling and Fulfillment Foundation system in production should read all chapters.

The Architect or Planner who is responsible for architecting and designing the entire system should read all the chapters.

Principles

When performing the performance management activities, you should keep in mind the following principles.

Balancing Performance and Architectural Attributes

Performance and scalability are critical architectural attributes. In an ideal world, we would have the luxury of configuring systems with an infinite number of the latest and fastest system components. In reality, we have to construct systems that balance performance with other architectural attributes such as availability, affordability, security, maintainability, operability, interoperability, scalability.

Take for example the following simple trade-off study between only three attributes - affordability, scalability and maintainability. If you want to configure a database with very fast I/O (maximize scalability) with a limited software budget (maximize affordability), you could implement your database files on raw devices which, to some, can be more difficult to manage. However, if you think that approach comes with unacceptable maintainability and operability burdens, you could implement the database files on the regular Unix file system. This approach would improve maintainability and operability at no additional cost but may not scale under high transaction volumes. If performance is important, you may opt to implement a specialized file system that provides raw-device performance and the maintainability of file systems. Here you would choose to maximize performance and maintainability at the expense of additional cost - you must purchase this specialized software.

The example above is a fairly simple trade-off study. Recognizing this reality, this document identifies major decision junctures, provides the context of how they fit within the overall system, provides rationale for our recommendations, and assists you in arriving at your own decision that is relevant to your organization's needs.

The planning sections are not recipe books. We do not provide a specific set of instructions that you can blindly follow to completion because we recognize that you may have unique business or operational requirements.

Keep It Simple Strategy

There are a large number of settings that can be tuned in a complex system. On HP-UX, there are over 100 TCP/IP, UDP and IP settings, over 50 HP-UX kernel parameters, close to 550 undocumented Oracle parameters, and over 250 documented Oracle parameters. The permutations and combinations of these settings are staggering. Some adjustments are beneficial - some not. Some may negate the benefit of others.

This Performance Management Principle proposes that systems be implemented with their default settings and that changes only be made when necessary.

This document identifies those adjustments that we believe are critical or beneficial. These include connection pooling, reusable SQL, Java heap settings. We identify the parameters which we believe are optional and that you can set when there is a clear need.

Your Mileage May Vary

One day, an inquisitive little girl asked her mother why she trimmed the sides of the roast before putting it into the oven. The mother said that was how her mother cooked. The little girl, still curious, asked the grandmother. After finishing laughing, the grandmother explained to the little girl that a long time ago, she had a tiny oven. She had to trim the side of the roast to prevent it from touching the side of the oven.

You should not take our recommendations (or recommendations from any book) as absolute truths. Recommendations may apply to most but not all systems. We identify those recommendations that we believe are critical. We highly recommend that you understand the context and the implications of each recommendation. We also highly recommend that you test each recommendation prior to production.

System Test Before Going Live

We cannot overemphasize the importance of system tests before going live. This recommendation can be seen throughout this document. Your system is different from other Sterling Selling and Fulfillment Foundation systems because:

- It has its own unique set of external systems that it connects to.
- It has groups of users performing work that is specific to your business, and so forth.
- It is configured differently from other systems, and so forth.
- It has different levels of customization.
- It may have some screens or processes that, although are optimized for general use, may not be optimal for your specific use.

As a result, we strongly encourage all our customers to system test the entire system, which is made up of the Sterling Selling and Fulfillment Foundation system and all the external systems, under anticipated peak transaction volumes prior to implementation into production.

Cascading Failure

A defense logistics officer, once gruffly reminded a group of young pilots that their new fighter jet was nothing more than 50,000 parts flying in tight formation.

This message has many parallels to any large computing systems. A large application system has many working components ranging from physical disk drives, operating systems, interfaces to external systems, application servers to database. All of these highly interconnected and dependent components must work well for the system to perform.

Lets us assume that a Sterling Selling and Fulfillment Foundation transaction calls out to an external system to check on item availability. If that external system is unable to scale or performs poorly, that Sterling Selling and Fulfillment Foundation transaction waits, which results in a thread being blocked. If there are many requests for that transaction, the system could become stalled when all the threads become blocked. As a result, a poorly tuned system could have a ripple effect on integrated systems.

This document presents some of these interdependencies along with approaches to monitoring them.

Measure Thrice, Check Twice, Cut Once

An old adage in carpentry is to Measure Thrice, Check Twice and Cut Once.

In the heat of a performance problem, it is very tempting to try different tuning parameters without fully understanding the root-cause of the problem. Some changes or combination of changes can have a negative impact on the system. After trying many tuning changes, it is possible that some of the non-beneficial changes are not rolled back.

From past experiences, we found that system optimization is often more effective and efficient if the problem is correctly analyzed and the root cause clearly identified. An approach that we have adopted is as follows:

- (measure) Measure the system to establish the baseline performance and throughput.
- (measure) Measure the system when performance issues arise.
- (check) Given the symptoms, formulate theories as to the root-cause of the problem and the potential tuning changes.
- (check) Ensure you can explain why certain tuning recommendations can help alleviate the problem and to formulate the expected behavior if the tuning change is applied.
- (cut) Make one (or a few) tuning change at a time - in some cases, multiple changes could negate the benefit of other changes.
- (measure) Measure the system and see if the system gained the intended benefits.

Only the Facts, Jack

This document does not attempt to rewrite the vast body of tuning knowledge found in the public domain. This document also does not serve as a substitute for third party vendor training such as IBM and Oracle. Instead, this document provides recommendations that supplement or deviate from conventional recommendations or recommendations that are specific to Sterling Selling and Fulfillment Foundation. We have liberally referenced many excellent sources of information - the Web, books, magazine articles, and so forth - that we found useful. A list of these references is found in Chapter 10, “References,” on page 145.

About the Performance Recommendations Graveyard

Technology changes rapidly. Processors double in speed every eighteen months. There are major performance enhancements every software release. As a result, recommendations that were at one time critical to a particular release of Sterling Selling and Fulfillment Foundation can become deprecated. In addition, to apply the recommendations, you may at times have to remove the obsolete recommendations.

In conjunction with the Keep It Simple Strategy and the Your Mileage May Vary Principles, you should:

- Apply tuning optimization changes when needed but only after testing.
- Capture these changes in a formal change management system.
- Document the changes from the default settings and why they were deemed necessary.
- Question their applicability as the system evolves - for example, during upgrades, operating system changes, and so forth.

We present a list of deprecated recommendations in “Performance Recommendations Graveyard” on page 14.

Chapter 2. Performance Checklist

This chapter provides a list of some of the recommendations found in this document in a checklist format. We encourage you to fully understand the rationale behind these recommendations and their implications to the overall system.

In the following tables, the columns "Dev" and "Prod" indicate whether the recommendations are Recommended (R), Critical (C) or Not Applicable (NA) in a Development or Production environment respectively.

Planning Checklist

The following are long-lead time planning elements. For example, you need to ask for a server node sizing in order to know how much computer resources to acquire. Some configurations could have more than one month lead-time.

Table 1. Planning Recommendations

Recommendation	Section	Dev	Prod	Comments
Server Node Sizing	“Pre-Sales Server Sizing” on page 15 “Capacity Plan” on page 15	NA	C	You must ensure you have sufficient computing capacity to process peak transaction volumes.
Database Disk Sizing	“Database Disk Sizing” on page 15	NA	C	You must have sufficient disk space for database server

Architectural Checklist

The following recommendations are architectural or design related.

Table 2. Planning Recommendations

Recommendation	Section	Dev	Prod	Comments
Ensure user exit or event processing times are minimal when holding on to critical locks.	“User Exits and Events” on page 111	C	C	When defining or coding user exits or events, make sure you are aware of locks held and the amount of time you could spend in the exit or event.
Record Sorting Strategy to avoid deadlocks	“Sort Order and Deadlocks” on page 126	C	C	Apply this recommendation to custom code or the manner in which records are locked to avoid deadlocks.

Computer Node Implementation Checklist

Table 3. Computer Server Node Implementation Recommendations

Recommendation	Section	Dev	Prod	Comments
OS Version and OS Kernel Parameters	“Supported Configurations” on page 15	C	C	Make sure you install the Sterling Selling and Fulfillment Foundation system on certified OS versions and levels.
Network Speed and Duplex Negotiation	“Auto-Negotiation” on page 16	C	C	Make sure your network cards are operating at the highest speeds. The network interface card and the network switch can negotiate to lower speed and duplex. When that happens, performance degrades noticeably, even under low transaction volumes.
AIX® Recommendations				
Page Space Allocation	“Page Space Allocation Policy” on page 17	C	C	The AIX default page space allocation policy does not reserve swap space when processes allocate memory allocations. This could lead to swap space overcommitment which forces AIX to kill processes when it runs out of swap space. Either: • Ensure sufficient swap space or • Set the environment variables: PSALLOC=EARLY NODISCLAIM=TRUE
AIX/Oracle Recommendations				
asynchronous I/O parameters	“Asynchronous I/O” on page 18	NA	C	The default asynchronous I/O parameters are set too low.
WebLogic / AIX Recommendations				
udp_sendspace	“Asynchronous I/O” on page 18	C	C	WebLogic's multicast packets are larger than the AIX udp_sendspace buffers. At the default level, you may get multicast errors. Set udp_sendspace to 32768

Java Virtual Machine Implementation Checklist

Table 4. JVM Implementation Recommendations

Recommendation	Section	Dev	Prod	Comments
JVM Version	“JVMs: Supported Configuration” on page 25	C	C	Make sure you install the Sterling Selling and Fulfillment Foundation system on certified JVM versions and levels.
Add `-showversion` to the JVM command line	“Java Version” on page 26	R	R	During the JVM startup, the JVM version and mode are displayed. This simple step helps eliminate cases where the wrong JVM is used.

Table 4. JVM Implementation Recommendations (continued)

Recommendation	Section	Dev	Prod	Comments
Verbose GC Statistics	"JVM Recommendations: Garbage Collection Statistics" on page 26	NA	C	Enable verbose GC statistics collection. Understanding the "health" of GCs for each JVM is critical for performance.
Defer distributed garbage collection to a long interval by setting `-Dsun.rmi.dgc.server.gcInterval`	"JVM Recommendations: Garbage Collection Statistics" on page 26	NA	C	The default distributed garbage collection setting unnecessarily forces expensive Full Garbage Collections every minute. The impact is noticeable especially for large heaps that are larger than 600MB. You should set this parameter on both the Sterling Selling and Fulfillment Foundation agents and the application servers.
Monitor for Paging	"Memory and Paging" on page 28	C	C	The JVM heap must be resident in memory. Performance degrades noticeably if the OS has to page portions of the heap out to disk.
Monitor for OutOfMemory exceptions	"OutOfMemory Exceptions" on page 28	C	C	OutOfMemory exceptions can cause unpredictable application behaviors. As a safety measure, Sterling Selling and Fulfillment Foundation stops the JVM when it catches an OutOfMemory exception.
For HotSpot JVMs				
For Sun JDK, run with -XcompressedOops if you do not need heaps up to 32GB. This will provide better memory utilization, increase performance, and reduce the overall footprint, as well.				
IBM J9 JVM				
For IBM JDK, use -Xcompressedrefs option.				
JVM VM modes	"Virtual Machine Mode" on page 30	C	C	
For HotSpot JVMs running WebLogic, set `-XX:MaxPermSize=768m`	"Permanent Generation" on page 31	C	C	The default permanent generation space setting is too small for Sun and HP JVMs. If you don't increase this setting, the JVM fails and throw a cryptic java.lang.OutOfMemory exception.
For HP JVMs, ensure amount of free space in the Old Generation is larger than the combined size of the Eden plus the occupied space in the survivor space.	"Young Generation Guarantee" on page 34	C	C	Sun and HP JVMs implement a conservative policy called the Young Generation Guarantee, specifying that the amount of free space in the Old must be larger than the eden and survivor space on the chance that every object is still alive after the GC. If the Old free is too small, the JVM reverts to Full GCs. Customers migrating from JDK 1.3.1 may have to increase their overall heap size or decrease the eden.

Table 4. JVM Implementation Recommendations (continued)

Recommendation	Section	Dev	Prod	Comments
Heap Size	“Starting Recommendations” on page 34	C	C	Configuring the JVM Heap correctly is not only critical for performance but also for availability. If the heap is sized too big, the GC pauses could be very long. If the heap is larger than physical memory, the system could "thrash". If the heap is too small, the JVM could experience outOfMemory exceptions.

Application Server Checklist

Table 5. Application Server Implementation Recommendations

Recommendation	Section	Dev	Prod	Comments
Connection Pool	“WebLogic Connection Pool” on page 52 “WebSphere Connection Pool” on page 47 “JSP Tuning: JSP Pre-Compilation” on page 56		C	Database connection establishments are very expensive operations. If connection pooling is not enabled in the application servers, transactions from application server does not scale. The Sterling Selling and Fulfillment Foundation agents are automatically started with a connection pool that is implemented in the agent infrastructure.
Assign each Sterling Selling and Fulfillment Foundation agent to its own JMS destination	“Dedicated Queues” on page 86	NA	C	For production, dedicated JMS queues are critical for performance. They are also easier to monitor. For ease of configuration and deployment in development, you can continue to use the single DefaultAgentQueue for all agents.
Assign integration- based queues to a separate JMS server	“Integration Queues” on page 85	NA	C	Put integration-based queues (e.g., queues used to receive orders from an external system) into a separate JMS server especially if the number of messages in that queue could grow to large numbers.
Precompile the JSPs	“WebLogic Tuning: JSP Pre-Compilation” on page 53 “WebSphere Tuning: JSP Pre-Compilation” on page 49	R	C	The application servers compile JSPs the first time they are used. The compilation phase can take over 30 seconds which could lead users to perceive poor user interface response times. Note: For WebLogic, we recommend the use of `weblogic.appc` over the use of `weblogic.jspc`.

Sterling Selling and Fulfillment Foundation Checklist

Table 6. Sterling Selling and Fulfillment Foundation Implementation Recommendations

Recommendation	Section	Dev	Prod	Comments
Reference Data Cache	"Performance Feature - Reference Data Caching" on page 104		C	Reference Data Caching is critical for scalability for customers who have high transaction volumes. Reference Data Caching is also critical for UI screen responsiveness. Starting in Yantra 5x 5.0 SP2, the reference data cache is enabled by default. For Development or non-production environments that are memory constrained, you can selectively enable Reference Data Caching. For example, for responsive UI, you need to, at a minimum, cache the `yfs_resource` and `yfs_resource_permissions` tables.
Search Catalog Index Memory Usage		R	C	Set JVM size to be sufficiently large to accommodate enabling an item index for use by IBM Sterling Web and IBM Sterling Call Center applications. You will need you to increase JVM heap settings. For more information, see the section on configuring cache for catalog search indexes in the *Sterling Selling and Fulfillment Foundation: Properties Guide*.
Agents and Messaging Configuration	"Agent Thread Levels" on page 102		C	For production, configure the optimum threading level, the assignment of agents to message queues or destinations, and the placement of message destinations on message servers.
Statistics	"Statistics" on page 128	NA	R	Sterling Selling and Fulfillment Foundation generates statistics for internal product use as well as use by IBM personnel. These statistics can be used to monitor throughput and to assist in performance diagnosis. We recommend leaving statistics generation on and regularly purging old statistics (e.g., greater than 3 weeks). Please be aware that the content and/or structure of the metrics can change without warning.

IBM Sterling Warehouse Management System Application Checklist

Table 7. Sterling Warehouse Management System Application Checklist

Recommendation	Section	Dev	Prod	Comments
Increase Java stack size for create wave and batch wave agents if you want to process waves with large number of shipments		NA	C	These agents need large stack sizes to perform wave optimization calculations.

Table 7. Sterling Warehouse Management System Application Checklist (continued)

Recommendation	Section	Dev	Prod	Comments
Run WMS Task Purge on a daily basis	"WMS Agents: Purge" on page 138	NA	C	Purge moves completed YFS_TASK records to the YFS_TASK_H history table.
RCP clients	"Rich Client Program Interface" on page 97	NA	C	Modify locations.ycfg to set the SSL and compression features. For remote users, we strongly recommend setting compression.

Database Checklist

Table 8. Database Checklist

Recommendation	Section	Dev	Prod	Comments
Monitor and adjust indexes	"Sterling Selling and Fulfillment Foundation Schema" on page 60	C	C	The Sterling Selling and Fulfillment Foundation schema comes with a default set of indexes for general use. In some cases, the indexes may not apply to your operational environment. Regularly monitor the resource cost of frequently used queries. See if additional indexes are needed. Also monitor if indexes can be deleted.

Oracle Database Checklist

Table 9. Oracle Database Checklist

Recommendation	Section	Dev	Prod	Comments
Set cursor_sharing=FORCE	"cursor_sharing" on page 69	C	C	This parameter makes dynamic SQL reusable, which reduces contention on the shared pool.
Oracle: Check if histograms are needed	"Index and Table Statistics" on page 74	NA	C	As you start to populate the database, check to see if there are indexed columns that have skewed data distribution - for example, most rows have the same value (e.g., space). These could include fields like `derived_from_order_header_key`, `chained_from_order_header_key`, `derived_from_order_line_key`, chained_from_order_line_key. If there are skewed columns, add a histogram. The performance impact is very noticeable. One customer saw a query that took 30 seconds drop to sub-second.

DB2 Database Checklist

Table 10. DB2 Database Checklist

Recommendation	Section	Dev	Prod	Comments
Optimizer Statistics	"Index and Table Statistics" on page 66	NA	C	Regularly run runstats to keep table and index statistics up to date to ensure the DB2 optimizer picks appropriate execution plans.

Table 10. DB2 Database Checklist (continued)

Recommendation	Section	Dev	Prod	Comments
Parameters governing DB2 locking strategy	"Recommended DB2 dbset Registry Variables" on page 61	C	C	Set DB2_EVALUNCOMMITTED, DB2_SKIPDELETED and DB2_SKIPINSERTED to reduce lock contention.
Parameters governing DB2 memory	"Recommended DB CFG Parameters" on page 63	C	C	Set parameters to manage various memory structures such as the LOCKLIST, SORTHEAP, etc. at AUTOMATIC. Set `self_tuning_mem` parameter to ON to enable the DB2 self-tuning memory features.
Volatile Table	"Volatile Tables" on page 74	NA	C	Mark tables that change significantly as volatile.

Monitoring Checklist

Table 11. Monitoring Recommendations

Recommendation	Section	Dev	Prod	Comments
Monitor CPU utilization		NA	C	Monitor CPU utilization to ensure there are no CPU contention.
Monitor Swap Usage		C	C	If there is not enough space left on the swap device (or paging file), the OS could prevent another process from starting or in some cases be forced to kill running processes.
Monitor Paging	"Memory and Paging" on page 28	C	C	The Java Virtual Machines and Database Management Systems rely on large memory buffers or heaps and are sensitive to paging. Performance degrades noticeably if there is not enough memory to keep the JVM heap in memory - even in Development. Monitor paging levels using standard operating system or third party measurement tools. For example: • On Unix and Linux, you could use SAR. • • On Windows, use System Monitor.
Monitor Heap Garbage Collection Performance			C	Monitoring heap GC performance is critical for performance and availability. For example, if the amount of heap free after a GC is continually increasing and approaching the maximum heap size, the JVM could experience outOfMemory exceptions.

Performance Recommendations Graveyard

This section lists performance recommendations that were deprecated by this release.

Table 12. Deprecated Performance Recommendations

Deprecated In	Deprecated Recommendations	Comments
Sterling Selling and Fulfillment Foundation 8.0	Removed recommendations on how to diagnose and tune JVM fragmentation.	The new IBM JDK 1.5 (J9) JVM was redesigned to eliminate the fragmentation caused by pinned and dosed objects.
Sterling Supply Chain Applications 7.9	Removed setting specific values for DB2 LOCKLIST, MAXLOCKS, PCKCACHESZ, SHEAPTHRES_SHR, SORTHEAP, NUM_IOCLEANERS, NUM_IOSERVERS, DFT_PREFETCH_SZ, MAXAPPLS and AVG_APPLS.	Rather than specifying specific values, we recommend allowing DB2 to automatically manage these parameters.
Sterling Supply Chain Applications 7.9	Oracle cursor_sharing=SIMILAR	Use cursor_sharing=FORCE.
Yantra 7x, 7.7 in Oracle10g	Oracle Parameters - shared_pool_size	Use Oracle10g/11g Automatic Memory Management (AMM) sga_max_size, sga_target and pga_aggregate_target.
Yantra 7x, 7.7 in Oracle10g	Statistics gathering	Oracle10g/11g automatically schedules statistics gathering during its maintenance window. See new recommendations in "Index and Table Statistics" on page 74
Yantra 7x, 7.5 and Oracle10g RAC	In the past, we adhered to Oracle's recommended setting for max_commit_propagation_delay of 700 centiseconds (or 7 seconds). Oracle has revised its recommendation to 0 effectively disabling the Lamport scheme.	For existing Oracle instances, reset max_commit_propagation_delay so that the default value is 0.
hpux_sched_noage		178

Chapter 3. Computer Systems

This chapter provides recommendations on how to plan, implement, configure, monitor, and tune the operating system and the computer nodes.

Generally speaking, from a performance perspective, the server nodes for Sterling Selling and Fulfillment Foundation fit into two broad categories:

- Database server node
- Mid-tier server nodes that run Sterling Selling and Fulfillment Foundation, for example, agents and application servers.

Planning for Computer Systems

The computer systems have long lead time planning elements such as developing the configuration specifications, soliciting bids, procuring the configuration. The lead time to delivery and set up could take up to a month.

The choice of hardware and vendors is typically dictated by your organization although that choice must conform to the Sterling Selling and Fulfillment Foundation-certified technology stack.

Supported Configurations

Refer to the *Sterling Selling and Fulfillment Foundation: System Requirements Guide* for a list of the support operating systems and computer servers.

Pre-Sales Server Sizing

Early in the sales phase, you can request a Sterling Selling and Fulfillment Foundation Server Sizing study to get an estimate of the processor, memory and network requirements for the standard/baseline Sterling Selling and Fulfillment Foundation.

Capacity Plan

Once you have fully developed the system, you can also engage IBM Professional Services to conduct a capacity plan study of your system. This involves measuring your system and using the measurements to forecast resource requirements at anticipated peak periods.

The benefit of this approach is that the forecasting model is based on your system which includes all your customization, and your specific configuration.

Database Disk Sizing

The size of the database disk subsystem could range widely from a few hundred gigabytes to terabytes. The size depends on the business order transaction volumes, the complexity of each order, the length of time you want to keep the orders in the active and the history database. The *Sterling Selling and Fulfillment Foundation: Installation Guide* has a section to help you estimate the space requirements for your database.

Time Synchronization

Although this is not a performance recommendation, we strongly recommend that you keep the system time synchronized across all computer nodes using a time synchronization protocol such as Network Time Protocol (NTP). Keeping the system time synchronized allows you to perform the following tasks:

- Correlate events in the database, Sterling Selling and Fulfillment Foundation, and the application server logs
- Correlate workload arrival, as recorded in the application server's access.log to system measurements (such as SAR, vmstat, and so forth)

Data Center Network

The performance of the Sterling Selling and Fulfillment Foundation system is critically dependent on the performance of the data center network. Here are some areas to consider:

- Correct auto-negotiation to the optimum bandwidth and with full duplex
- Network bandwidth

Auto-Negotiation

By design, Ethernet network interface cards (NIC) automatically negotiate the best speed and duplex with the switch that it is connected to. Generally, auto-negotiation work. We have, however, seen many cases where auto-negotiation drops the connection to sub-optimal levels (e.g., 10mps half-duplex) after a server boot.

The impact of an incorrectly negotiated network card is dramatic. For example, at one customer, application servers took over 20 minutes to start when the network card on the administration server negotiated the wrong settings.

If both the network interface card and the switch are capable of full-duplex 100mbps or 1000mbps, you can let then auto-negotiate. Alternatively, you can manually set the higher speed and duplex as described in subsequent "Auto-negotiation" sections.

An easy way to check the NIC negotiation is to FTP, SCP or RCP a large file (e.g., 256MB) file from a test node to all other nodes.

From the database server node, create a 256MB file using the following command:

```
dd if=/dev/zero of=/tmp/egg bs=16384 count=16384
```

Assuming that you have three nodes (applservernode1 to applservernode3) and you can rcp or scp into each node, issue the following:

```
export ALLHOSTS="applservernode1 applservernode2 applservernode3"
for i in $ALLHOSTS
do
time rcp /tmp/egg $i:/tmp/egg
done
```

If you cannot rcp or scp, you can issue an FTP transfer.

The time to transfer the 256MB file should be around 20 seconds for 100mbps Fast Ethernet and around 5 seconds for 1Gbps networks (for FTP). You likely have a network negotiation problem if the transfer times are much slower (for example 200 seconds).

Please see the following sections on how to monitor and set the network speed and bandwidth:

- AIX - See “Auto-Negotiation” on page 19.
- HP-UX11i - See “Auto-Negotiation” on page 21.

- Red Hat Linux - See “Auto-Negotiation” on page 23.
- Solaris - See “Auto-Negotiation” on page 20.

Network Bandwidth

The network cards on the nodes running Sterling Selling and Fulfillment Foundation must be configured with at least a 100 Mbps full-duplex link. In some cases, for example, you may have to implement gigabit network cards if you have high enough transactions going through the network. Our Pre-Sales Server Sizing (see “Pre-Sales Server Sizing” on page 15) specifies the anticipated minimum network speeds.

In production, you should monitor the network bandwidth utilization. One approach is to monitor the traffic utilization at the switch.

The application stops scaling when the network is the bottleneck.

IBM AIX

Page Space Allocation Policy

AIX, by default, implements a *late page space allocation* policy. When a program asks for a large memory allocation, AIX grants the virtual memory allocation but does not allocate the space on the backing store (or swap) until it is actually used. In contrast, *early page space allocation* first allocates the space in swap before granting the virtual memory.

With *late page space allocation*, AIX could successfully start many processes. However, as these processes use their virtual memory, AIX could run low on swap. When this happens, AIX chooses the youngest process to kill. The following message on the application server's console indicates that it was killed:

```
./startManagedWebLogic.sh[216]: 13550 Killed
```

The following error message is shown in the AIX error log if your application server instance was killed because of *late page space allocation:*

```
Date/Time:       Thu May 30 17:35:37
Sequence Number: 39
Machine Id:      000F257F4C00
Node Id:         ibm04
Class:           S
Type:            PERM
Resource Name:   SYSVMM

Description
SOFTWARE PROGRAM ABNORMALLY TERMINATED

Probable Causes
SYSTEM RUNNING OUT OF PAGING SPACE

Failure Causes
INSUFFICIENT PAGING SPACE DEFINED FOR THE SYSTEM
PROGRAM USING EXCESSIVE AMOUNT OF PAGING SPACE

        Recommended Actions
        DEFINE ADDITIONAL PAGING SPACE
        REDUCE PAGING SPACE REQUIREMENTS OF PROGRAM(S)

Detail Data
PROGRAM
java
```

```
USER'S PROCESS ID:
       19194
PROGRAM'S PAGING SPACE USE IN 1KB BLOCKS
      295388
```

You can reduce the likelihood of a *late page space allocation* kill by increasing the amount of swap space. However, the recommended approach is to selectively turn on *early page space allocation* by exporting the following environment variables:

```
export PSALLOC=early
export NODISCLAIM=true
```

The NODISCLAIM environment variable will eliminate a call to the disclaim() system routine when a free() call is issued. You will experience high CPU utilization (due to system calls) if you use early page allocation but do not set this variable.

You can increase swap and page space requirements significantly if you set these two variables for all workloads. As a result, we recommend you only set these two variables for your JVMs and not across the system for all other workloads.

For Sterling Selling and Fulfillment Foundation agents and WebLogic application servers, you can issue the commands in the startup scripts. For WebSphere, you can define the environment variables in the Environment dialog box in the administrative client.

If you make that change and not increase the swap space, the following error message is shown immediately on startup:

```
   Unable to alloc heap of requested size, perhaps the maxdata value is too
small - see README.HTML for more information.
   Unable to allocate an initial java heap of 1073741824 bytes.
   **Out of memory, aborting**

   *** panic: JVMST016: Cannot allocate memory for initial java heap
```

The exception above is actually the desired behavior because AIX is stating that it is unable to guarantee that there is enough swap space for all potential requirements.

Asynchronous I/O

AIX supports both kernelized asynchronous I/O to raw devices or Veritas Quick I/O devices and threaded asynchronous I/O to filesystem files (e.g., JFS). With KAIO, the Oracle process queues I/O requests in the kernel and are notified of I/O completion by an interrupt. In contrast, threaded asynchronous I/O uses multiple threads with each thread issuing a synchronous I/O to simulate the asynchronous I/O.

If you are going to implement your database files on filesystems like JFS or JFS2, you may have to monitor and tune the asynchronous I/O tunable parameters. By default, these parameters may be set too low for Oracle Databases on large systems with 4 CPUs or more if you are using threaded asynchronous I/O.

The asynchronous I/O parameters are too low in your configuration if you find the following messages in the DBWR, CKPT, or LGWR files in the ORACLE_BASE/admin/<dbname>/bdump directory:

```
Warning: lio_listio returned EAGAIN
Performance degradation may be seen
```

The EAGAIN from the lio_listio function indicates that "the resources necessary to queue all the I/O requests were not available" (see lio_listio subroutine documentation in the AIX Technical Reference: Base Operating System and Extensions Volume 1).

You can find out the current asynchronous I/O settings by issuing the following command:

```
> lsattr -El aio0
autoconfig available STATE to be configured at system restart True
fastpath   enable    State of fast path                       True
kprocprio  39        Server PRIORITY                          True
maxreqs    4096      Maximum number of REQUESTS               True
maxservers 10        MAXIMUM number of servers per cpu        True
minservers 1         MINIMUM number of servers                True
```

Configuring Asynchronous I/O in AIX

About this task

You can change the asynchronous I/O parameters with the following command:

```
chdev -l aio0 -a minservers=x -a maxservers=y -a maxreqs=z
```

The minservers and maxservers are at the CPU or processor level.

Monitoring Asynchronous I/O in AIX

About this task

Monitor the actual number of aioservers started during a typical workload using the following command:

```
pstat -a | grep aioserver
```

You may have to increase the number of aioservers if:

- The actual number of active aioservers is equal to the maxservers and
- The CPU utilization of all the aioservers appear to be even (indicating that all aioservers are used)
- You continue to see the `Warning: lio_listio returned EAGAIN` messages

The MetaLink Notes discusses the issue and provide recommendations for maxreqs, minservers and maxservers:

- Oracle MetaLink Note:443368.1 - *How does Oracle use AIO servers and what determines how many are used?*

Auto-Negotiation

About this task

See “Auto-Negotiation” on page 16 for a general discussion on network negotiations.

On AIX, you can check the link using SMIT. In SMIT, go to Devices > Communication > Ethernet Adapter > Adapter > Change / Show Characteristics of an Ethernet Adapter. Select the network interface. The link speed and mode configuration is displayed in the Media Speed field.

Alternatively, you can issue the following commands:

```
$ lsparent -C -k ent
ent0 Available 40-58 IBM 10/100 Mbps Ethernet PCI Adapter (23100020)

$ lsattr -E -l ent0 -a media_speed
media_speed Auto_Negotiation Media Speed True
```

To find out the actual negotiated speed and duplex, issue the following command:

```
$ netstat -v ent0 | grep Media
Media Speed Selected: Auto negotiation
Media Speed Running: 100 Mbps Full Duplex
```

If the auto-negotiation failed, you can set the NIC from SMIT or by issuing the following commands:

```
$ chdev  -l ent0 -a media_speed=100_Full_Duplex -P
ent0 changed

reboot
```

Sun Solaris

Auto-Negotiation

About this task

See "Auto-Negotiation" on page 16 for a general discussion on network negotiations.

On Solaris, you can check a gigabit (GE), Quad-Fast Ethernet (QFE) and Fast-Ethernet (HME) links with the following process:

```
#ifconfig -a
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
hme0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 10.10.10.10 netmask ffffff00 broadcast 10.10.10.255
```

In the example above, the server has a single HME interface. To query the HME settings, issue the following commands:

```
# ndd -get /dev/hme link_speed
1
# ndd -get /dev/hme link_mode
1
```

For QFE or GE, substitute in /dev/qfe or /dev/ge respectively.

The results of the commands above are as follows:

Table 13. NDD Results

NDD Variable	Results
link_speed	0 = 10mps 1 = 100mps
link_mode	0 = half-duplex 1 = full-duplex

If the auto-negotiation failed, you can manually set the NIC by issuing the following commands:

```
# ndd -set /dev/hme instance 0
# ndd -set /dev/hme adv_100T4_cap 0        disables T4 cabling
# ndd -set /dev/hme adv_100fdx_cap 1    enables 100mps full duplex
# ndd -set /dev/hme adv_100hdx_cap 0    disables 100mps half duplex
# ndd -set /dev/hme adv_10fdx_cap 0     disables 10mps full duplex
# ndd -set /dev/hme adv_10hdx_cap 0     disables 10mps half duplex
# ndd -set /dev/hme adv_autoneg_cap 0   disables autonegotiation
```

You can preserve the commands above across reboot by adding the following commands to /etc/system:

```
set hme:hme_adv_autoneg_cap=0
set hme:hme_adv_100T4_cap=0
set hme:hme_adv_100fdx_cap=1
set hme:hme_adv_100hdx_cap=0
set hme:hme_adv_10fdx_cap=0
set hme:hme_adv_10hdx_cap=0
```

You also need to manually set the switch ports to 100mps full-duplex. Please see your switch documentation.

HP HP-UX11i

This section provides recommendations on how to plan, implement, configure, monitor, and tune HP HP-UX11i server nodes running on HP PA-RISC or Itanium processors.

Auto-Negotiation

About this task

See “Auto-Negotiation” on page 16 for a general discussion on network negotiations.

On HP-UX, you can check the link by issuing the following commands:

```
$ lanscan
Hardware Station        Crd Hdw   Net-Interface  NM  MAC       HP-DLPI DLPI
Path     Address        In# State NamePPA        ID  Type      Support Mjr#
0/0/0/0  0x00306E09612B 0   UP    lan0 snap0     1   ETHER     Yes     119

$ lanadmin -x 0
Current Speed                   = 100 Full-Duplex Auto-Negotiation-ON
```

If the auto-negotiation failed, you can manually set the NIC by issuing the following commands:

```
$ lanadmin -X 100FD 0

WARNING: an incorrect setting could cause serious network problems!!!

Driver is attempting to set the new speed
Reset will take approximately 11 seconds
```

HP HP-UX11i: Asynchronous I/O

About this task

Asynchronous I/O is very important to performance especially on high transaction volume processing environments. In summary, processes that issue synchronous `read()` or `write()` I/O calls must wait for the I/O to complete before it can continue. In contrast, processes can issue multiple asynchronous (non-blocking) aio_read() or aio_write() I/O calls in parallel without waiting.

HP-UX does not enable asynchronous I/O by default. HP-UX also only supports asynchronous I/O on files that reside on raw devices and not on filesystems. If you don't enable asynchronous I/O, workloads such as Oracle will try to run multiple DBWRs processes to get a limited amount of I/O parallelism.

To enable asynchronous I/O on HP-UX, you have to:

- Create the /dev/async character device.
- Grant MLOCK privilege to the Oracle group.
- Implement asynchronous I/O to the kernel.
- To create the /dev/async character device:

  ```
  # /sbin/mknod /dev/async c 101 0x104
  # chown oracle:dba /dev/async
  # chmod 660 /dev/async

  # ls -l /dev/async
  crw-rw-rw-   1 bin        bin        101 0x000104 May 29  2011 /dev/async
  ```

  In order to use asynchronous I/O, the OS group (typically dba) that the Oracle user belongs to must be granted the MLOCK privilege. You can check if the group has the privileges by issuing the following command:

  ```
  # /usr/bin/getprivgrp dba
  dba: RTPRIO MLOCK RTSCHED
  ```

- To grant the privilege (along with RTPRIO and RTSCHED), add the following string to the /etc/privgroup file:

  ```
  dba RTPRIO RTSCHED MLOCK
  ```

- Then run the following command to enact the new privileges:

  ```
  /usr/bin/setprivgrp -f /etc/privgroup
  ```

- Next, you have to configure the asynchronous disk driver into the HP-UX kernel. To do this:
  - Launch the system administration manager (sam) and
  - Navigate to Kcweb (Kernel Configuration) > Modules
  - Select asynchdsk to be loaded into the kernel
- Next, you need to ensure the max_async_ports is at least as high as the maximum number of connections (which is dictated by processes). The max_async_ports controls how many processes can open the /dev/async device at any given time. When the maximum is reached, subsequent processes will drop down to using synchronous (blocked) I/O. To find out the current settings, issue:

  ```
  # kctune -v max_async_ports
  Tunable            max_async_ports
  Description        Maximum number of open asyncdsk ports
  Module             io
  Current Value      600
  Value at Next Boot  600
  Value at Last Boot  600
  Default Value      50
  Can Change         At Next Boot Only
  ```

- To change the max_async_ports settings, go to sam > Kernel Configuration > Tunables.

Linux

Auto-Negotiation

About this task

See “Auto-Negotiation” on page 16 for a general discussion on network negotiations.

For Linux, you can check the link speed and duplex by issuing the following command. The "FD" in the following output denotes full-duplex:

```
$ mii-tool
eth0: negotiated 100baseTx-FD flow-control, link ok
eth1: no link
```

Chapter 4. Java Virtual Machines

General JVM Recommendations

This chapter provides general recommendations on how to plan, implement, configure and tune Java Virtual Machines that is applicable to all the supported JVM families. Family-specific JVM recommendations, such as Permanent Generation settings for SunSoft JVMs and Wilderness settings for IBM JVMs are found in the following JVM-specific sections:

- “HotSpot JVM” on page 30
- “IBM J9 JVM” on page 39
- “Oracle JRockit” on page 44

The Java language is designed to be "Written Once and Run Anywhere" (WORA). When you compile a Java source, you get an intermediate Java file called the Java class. The class file is made up of bytecodes representing abstract instruction codes. These codes are not directly executable by any computer processor. In contrast, languages like C compile their source code to native instructions for a specific processor.

To run a Java program, you start a JVM and pass the class file to the JVM. The JVM provides many services including loading the class file and interpreting (executing) the byte codes. The JVM is the core technology that provides the runtime environment in which a Java application runs in.

Each Java program or application runs in its own JVM. For example, if you configured an application server cluster with ten managed server instances that is controlled by one administrative instance, your configuration runs eleven JVM processes.

Since the JVM is the underlying processing engine for Sterling Selling and Fulfillment Foundation, it is critical that the JVMs are optimally configured and running efficiently. Incorrect JVM settings could cause poor application performance. At worse, it could lead to JVM outages.

Note: The JVM recommendations provided in this guide should be used only as a starting point in the JVM tuning for your Sterling Selling and Fulfillment Foundation application. You should increase these values as and when required.

JVMs: Supported Configuration

The applications are tested and certified for use on a specific set of JVMs and JVM versions and releases. See the*Sterling Selling and Fulfillment Foundation: System Requirements Guide* for the list of supported configurations.

To find out the JVM version, issue the following command:

```
$JAVA_HOME/bin/java -version
```

Recommended JVM Command Line Options

We recommend that you specify the following command line options on all JVMs:

- JVM identifier using the -D option
- Java version using -showversion

- Garbage Collection Statistics
- Distributed Garbage Collection interval

JVM Identifier: You may have to run many JVMs in a large Sterling Selling and Fulfillment Foundation configuration. From experience, we have found it useful to tag JVMs with an identifier so that they are easy to identify and therefore to monitor and manage. One approach to tagging is to use the JVM `-D` option. The `-D` option lets you set a system property variable as a name/value pair. For example:

```
java -Dyfsag=SCHEDULE ..... <class name>     and
java -Dyfsas=server01 ..... <class name>
```

In the example above, we use the `-D` option to set a name/value pair to help identify the purpose of the JVM. The names, `yfsag` and `yfsas`, indicate the type of workload - in this case, a Sterling Selling and Fulfillment Foundation agent and an application server respectively. The values, SCHEDULE and server01, indicate the instance of the workload. If you issue the following command:

```
ps -ef | grep java | grep Dyfs
```

you will see:

```
   UID   PID  PPID  C    STIME  TTY     TIME CMD
user03  6420  6418  2 08:20:21 pts/29   0:04 java -Dyfsag=SCHEDULE -server
user03  6456  6443  2 08:23:32 pts/29   0:23 java -Dyfsas=server01 -server
```

The tagging and some simple scripting allows you to automate a lot of management tasks. For example, to generate a thread dump on all the application servers, you could issue the following command:

```
for i in 'ps -ef | grep Dyfsas | awk '{print $2}''
do
kill -3 $i
echo "Issued thread dump for pid=$i"
done

or

ps -ef | grep Dyfsas | awk '{print $2}' | xargs kill -3
```

Java Version: A common but difficult problem to diagnose is one where the wrong JVM version or level was started. You can easily spot this problem if you start the JVM with the `-showversion` option. If you ran the following command on an IBM JVM on Linux:

```
> java -showversion <class name>
```

the following output is seen in your application log:

```
java version "1.6.0"
Java(TM) SE Runtime Environment (build pap6460sr7ifix-20100220_01(
SR7+IZ70326))
IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 AIX ppc64-64 jvmap6460
sr7-20100219_54049 (JIT enabled, AOT enabled)
J9VM - 20100219_054049
JIT  - r9_20091123_13891
GC   - 20100216_AA)
JCL  - 20091202_01
```

We recommend setting the `-showversion` option for all JVMs. This simple and inexpensive step provides valuable information that can help ensure that the correct JVM version and mode are used.

JVM Recommendations: Garbage Collection Statistics: Garbage collection statistics are critical for managing and monitoring JVMs. We strongly recommend enabling garbage collection statistics in production. The statistics is the only window you have into the behavior of the JVM heap management and the efficiency of the JVM.

Please see the JVM-specific sections on recommendations on how to enable garbage collection statistics.

Optional JVM Command Line Settings: Stack Size: Each time a method is called, a stack frame is created and pushed on to the thread's stack. The stack frame contains, at a minimum, the method's local variables and the method arguments. You can get a `java.lang.StackOverflowError` exception if you reach the maximum allowable stack limit of a thread. This can happen if:

- The method call depth is very deep (for example, in the Create Wave agent, the wave optimizer call depth is roughly equal to the number of shipments assigned to a shipment group).
- The stack frame is very large.

You can set the `-Xss` option to increase the maximum stack size per thread.

See related section "Java Stack Size" on page 139 if you are running the IBM Sterling Warehouse Management System.

Hanging Threads/Deadlocks/Infinite Loops

In some rare exceptions, the JVM may have threads that are not progressing, possibly because of one of the following reasons:

- Threads are deadlocked.
- Threads are in an infinite loop.
- Threads are waiting for an external event.
- JVM bug.

You can often find these offending threads by taking several successive thread dumps and seeing if there are any threads that seem "stuck" in the same processing point. On Unix and Linux, issue the following command where pid is the process id of the JVM:

```
# kill -3 pid
```

In Windows, you have to press the CTRL+BREAK keys on the command window that started the JVM.

If you have a hanging or deadlocked thread, in the best case, all they do is tie up a number of scarce worker (execute) threads. There currently isn't any way to kill hung or deadlock threads except to schedule a restart of the JVM.

In the worst case, these offending threads hold on to crucial shared resources (such as database record locks) and are blocking other threads in this or other JVMs. This situation could lead to a system-wide slowdown as more and more threads block behind these offending threads.

If you have infinite looping threads, at best, all they do is make the server node busier. In the worst case, they start to impact the performance of transaction running in that node or they hold critical resources needed by other threads.

Recommendations:

If you suspect a JVM has a hung or looping thread:

- Take three thread dumps for that JVM. Space the thread dumps a minute apart.
- Look at the stacktrace for the Default Queue in the successive thread dumps - see if there are any threads that are active and in the same code path in each thread dump.
- The IBM Thread and Monitor Dump Analyzer for Java is an excellent tool for analyzing the thread dumps. You can download the tool from http://www.alphaworks.ibm.com/tech/jca.

  The download page contains excellent information about using the tool for analysis and troubleshooting.

If you suspect transactions are slow across many JVMs:

- Look in your database for blocking chains - specifically, what sessions are blocking whom. Find out what servers the root blockers are coming from, the types of locks that they are holding and what was the latest SQL run.
- Identify the JVMs that have the root blockers. You may have to shutdown those JVMs if the blocking sessions are spreading to a system-wide shutdown.

  Note: Since thread dumps are invaluable diagnostic tools, you should be very comfortable taking thread dumps when the need arises. For example, you should occasionally take thread dumps from all JVMs (e.g., all application server instances, all agent/monitor servers) during non-peak processing periods. This gives you a chance to find out where the thread dumps are written to and how to read the thread dumps.

Memory and Paging

Note: The JVM performs very badly, even in low transaction volume environments if the OS has to continually page the JVM heap to disk.

You must make sure paging levels are minimal. The JVM manages its heap with the assumption that the entire heap is in memory. If significant portions of the heap are on the swap devices, the node could find itself in a "thrashing" situation where it spends most of its time shuffling pages between real memory and swap. This situation could arise for many reasons including:

- Starting a JVM with a heap size that is larger than physical memory
- Starting too many JVMs and other workloads such that the combined working set size is larger than the physical memory

OutOfMemory Exceptions

Note: Sterling Selling and Fulfillment Foundation stops a JVM that throws an OutOfMemory exception.

JVMs throw OutOfMemory exceptions when they cannot find enough space for a new object allocation request. From our experience, OutOfMemory exceptions are primarily due to the following reasons:

- The JVM heap does not have enough total free space in the heap for the new object, or
- In the case of IBM JVMs, the heap may have enough total free but not enough contiguous free space for the new object.

The JVMs try to recover gracefully when OutOfMemory exceptions occur - unfortunately, the outcome of the recovery can be unpredictable. We have seen situations where threads have disappeared (they don't show up in the thread dumps), threads have gone into infinite loops, or database connections from failed threads remain opened and in some cases, hold on to record locks.

For these and many other reasons, Sterling Selling and Fulfillment Foundation deliberately stops the JVMs that encounter OutOfMemory exceptions. When that occurs, you should see the following message in the application log:

```
java.lang.OutOfMemoryError
Yantra encountered Java Virtual Machine Error, verify your JVM settings ....
Halting the system...............
```

This measure is preferable to potentially unpredictable application behaviors.

In production, you should periodically check for the occurrence of this message in the Sterling Selling and Fulfillment Foundation log and to take appropriate actions, including alerting the application administrator or restarting the JVM.

Low on Total Free Memory: A JVM may run low on total free memory when there is a memory leak, the JVM heap was sized too small or there was a temporary abnormally high memory requirement (possibly for a very large order or wave). If you encounter an outOfMemory exception, we recommend you perform the following:

- Restart the JVM with a much larger heap (for example, 1.5GB).
- Monitor the amount of space used. For Sun and HP HotSpot JVMs, you need to look at the heap used after Full GCs. If you see this value steadily growing and never shrinking, you may likely have a memory leak. If the heap used increases by a large value (larger than your original heap setting) but eventually drops down to its original level, you may have encountered a large order or wave. You may want to investigate the nature of that order to see if the order was an anomaly or if it is going to be recurring. You can use the GC statistics to set you JVM heap sizes.

If you believe you have a memory leak, you can do the following:

- For IBM JVMs, generate a heapdump and use the IBM Memory Dump Diagnostic tool to identify the memory leak. We have found this tool to be easy to use and easy to identify memory leaks. The IBM JVM automatically generates the heapdump when it runs into an OutOfMemory exception. You can also request a heapdump using a Kill -3 by first setting the following environment variables:

  ```
  export IBM_HEAPDUMP=true
  export IBM_HEAP_DUMP=true
  export IBM_HEAPDUMPDIR=<directory to store the heap dumps>
  ```
- For Sun and HP HotSpot, try generating the hprof memory dump - the IBM Memory Dump Diagnostic tool is capable of analyzing hprof dumps. Otherwise, you may have to resort to using a tool like Quest JProbe Memory Debugger.
- For Oracle JRockit, you can use the Oracle JRockit Memory Leak Detector which is part of the JRockit Management Console. The Memory Leak Detector tells you which object is growing the fastest, what percentage of the heap they are occupying, and the number of instances.

Causes of OOM: There can be many reasons why transactions use a lot of memory. Some of the typical reasons include:

- Calling APIs with the default output XML template - Sterling Selling and Fulfillment Foundation allows you to specify an output XML template to specify the amount of data to return. The effect can be dramatic. Calling the getOrderDetail API without an output template could result in a very large XML (over 40MB) depending on the order complexity. Trimming the XML could reduce the size to a few hundred bytes.

- DEBUG and VERBOSE tracing - The DEBUG and VERBOSE traces are invaluable development and debugging tools, for example printing out the API input and output XMLs. In order to print the XML, the tracing facility has to create a String representation of the XML, which for very large XML document can result in very large StringBuffer objects. As a result, you should be careful about enabling these traces in production.

HotSpot JVM

This section provides recommendations on how to plan, implement, configure, monitor, and tune the HotSpot Java Virtual Machines.

The Sun HotSpot JVM is used when you deploy Sterling Selling and Fulfillment Foundation with the WebLogic or the WebSphere application servers on a Sun Solaris operating system running on Sun UltraSPARC processor-based servers.

The HP HotSpot JVM is used when you deploy Sterling Selling and Fulfillment Foundation with the WebLogic application servers on a HP-UX11i operating system running on HP PA-RISC processor-based servers.

Note: Oracle no longer supports the use of SunSoft JVM on Linux in production.

Implementation

It is a mixed blessing that HotSpot JVMs provide many tuning parameters because tuning the JVMs can appear to be part art and part guess work. There isn't a golden set of JVM settings that apply to all customers and conditions. The settings, especially memory settings (which we discuss below) are highly dependent on the transaction mix, the amount of data cached, the complexity of the transactions, concurrency levels, and so forth.

Fortunately, the HotSpot JVMs provide good measurement feedback that allows you to measure the effectiveness of the settings.

As a starting point, we recommend you configure your JVMs with the following initial values and to review and adjust the settings as you run the JVMs under representative workloads and traffic volumes.

Starting Recommendations

As a starting point, you should configure the JVMs as follows:

- Set the JVM mode to server mode.
- Set the permanent generation to 768MB for application servers, and 256MB for agents and integration servers.

Virtual Machine Mode

The Java language is designed to be platform independent. When you compile a Java source, you get an intermediate Java class which is made up of bytecodes representing abstract instructions. In contrast, languages like C compile to native binary code that a specific hardware processor can run. To run the program, you start a JVM and pass the class file to the JVM. The JVM loads the class file and interpret (execute) the byte codes.

Interpretation and execution of bytecodes is much slower than the executing code that has been compiled to the native instruction set of the host processor.

For speed and performance, the HotSpot JVM will, on-the-fly or just-in-time (JIT), compile frequently used methods into native code. See *The Java HotSpot Performance Engine Architecture*, (Sun Microsystems, http://java.sun.com/products/hotspot/whitepaper.html) and extensive Java SE Hotspot information on http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html for more detail.

The HotSpot JVMs support two different compiler modes: ***-client*** and ***-server***. Each supports differing levels of optimization.

The **server** VM mode is designed to maximize performance of long running workloads by applying more aggressive optimizations. The **client** VM mode, in contrast, is designed to reduce application startup time and memory footprint. The client VM mode is typically better suited for applets running in browsers.

To run the JVM in the server mode, you need to add the `-server` directive when starting up the JVM. For example, to start WebLogic in server mode, issue:

```
java -server weblogic.Server
```

To run WebLogic in the client mode, issue the following command:

```
java -client weblogic.Server
```

Permanent Generation

The HotSpot JVM sets aside an area, called permanent generation, to store the JVM's reflective data such as class and method objects. The size of this area is set to 64MB by default. Due to the number of classes used by application servers, you must set the permanent generation space setting to at least 768MB. If you use the default values, an OutOfMemory exception is more likely to occur during application server initialization.

To increase the permanent generation space for application servers, issue the following command:

```
java -server -XX:MaxPermSize=768m java_class
```

To increase the permanent generation space for agents and integration servers, issue the following command:

```
java -server -XX:MaxPermSize=256m java_class
```

This recommendation applies to the application server JVM as well as the Sterling Selling and Fulfillment Foundation agents.

Note: The agent server JVM requires only 256MB of permanent space.

Heap Memory and Garbage Collection

The JVM runtime environment uses a large memory pool called the heap for object allocation. The JVM automatically invokes *garbage collections* in order to clean up the heap of unreferenced or dead objects. In contrast, memory management in legacy programming languages like C++ was left to the programmer.

If the JVM heap settings are not set correctly, the garbage collection overhead can make the system appear unresponsive. In the worst case, your transactions or the JVM could abort due to outOfMemory exceptions (please see “OutOfMemory Exceptions” on page 28).

In the past, garbage collection overhead was quite substantial and the impact to end-user response times noticeable. Many garbage collection techniques have been proposed and implemented - all with their own strengths and weaknesses. Garbage collection techniques are constantly being improved. For example, the Sun JVM supports a mainly "stop-the-world" garbage collector - all transactions have to pause in a safe point for the entire duration of the garbage collection. The Sun JVM supports a parallel concurrent collector where transactions can continue running during most of the collection.

Sun and HP-UX Generational Collectors: The Sun and HP JVM organized its heap into generations to improve the efficiency of its garbage collection, and to reduce the frequency and duration of user-perceivable garbage collection pauses. The premise behind generational collection is that memory is managed in *generations* or in pools of memory with different ages (see Figure 1).

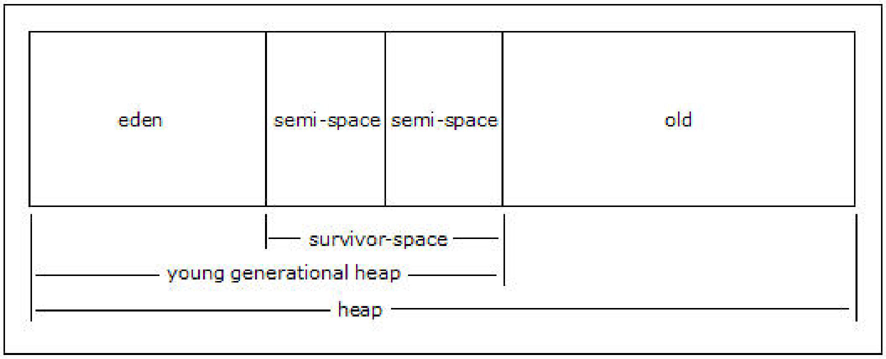


Figure 1. Heap Layout

New objects are allocated in the *eden*. When the *eden* fills up, the JVM issues a *scavenge GC* or *minor collection* to move the surviving objects into one of the two survivor or semi spaces. The JVM does this by first identifying and moving all referenced objects in the eden to one of the survivor space. At the end of the scavenge GC, the *eden* is empty (since all the referenced objects are now in the survivor space) and ready for object allocation.

The scavenge GC's efficiency depends on the amount of referenced objects it has to move to the survivor space and not on the size of the eden. The higher the amount of referenced objects, the slower the scavenge GC. Studies, however, have shown that most Java objects live a very short time. Since most objects live for a short time, one can typically create large edens.

Referenced objects in the survivor space bounce between the two survivor spaces at each scavenge GC until it either becomes unreferenced or the number of bounces has reached the *tenuring threshold*. If the tenuring threshold is reached, that object is migrated up to the old heap.

When the *old heap* fills up, the JVM issues a *Full GC* or *major collection*. In a Full GC, the JVM has to first identify all the referenced objects. When that is done, the JVM sweeps the entire heap to reclaim all free memory (for example, because the object is now dead). Finally, the JVM then moves referenced objects in order to defragment the old heap. The efficiency of the Full GC is dependent on the amount of referenced objects and the size of the heap. For more information see *The Java HotSpot Performance Engine Architecture*, (Sun Microsystems, http://java.sun.com/products/hotspot/whitepaper.html) and extensive Java SE Hotspot information on http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html.

Heap Settings

It is both a curse and a blessing that the SunSoft based JVMs provide many parameters to control the JVM heap configuration. Tuning the SunSoft generational collectors can be part art and part guess work. You may opt for the Keep It Simple Strategy Principle. In the following example, only specify the starting (-Xms) and maximum (-Xmx) heap size:

```
java -server -Xms358m -Xmx358m weblogic.Server
```

When choosing the JVM settings, you should keep the following in mind:

- Set the initial and max heap size the same - This eliminates the need of the JVM to decide when to expand or shrink the heap. This could also prevent a class of outOfMemory exceptions where there is not enough swap space when the JVM needs to expand the overall heap.
- By default, the Sterling Selling and Fulfillment Foundation caches reference data for performance. Depending on your data setup, you may have to increase the heap size or reduce the numbers of cached records. See “Performance Feature - Reference Data Caching” on page 104 for more information on the caching feature.
- Ensure that the node has enough physical memory so that portions of the heap are not paged out.

  Note: Please make sure you test your JVM heap settings with representative workloads and data under anticipated peak processing rates. In addition, you should run these tests for a number of days. Depending on your processing mix, the JVM heap settings could be different for the JVMs running the agents, the application servers and the JMS servers.

When setting the young heap, keep the following recommendations in mind:

- Set the initial and max eden size the same - This eliminates the need of the JVM to decide when to expand or shrink the eden.
- The cost of a scavenge GC is dependent on the amount of active objects that has to be moved to the survivor space and not on the size of the eden. Therefore, one can usually allocate a large eden.
- Allocate the eden large enough so that the scavenge GCs are not occurring too frequently (e.g., less than once per minute) and the collection service time is reasonably short (e.g., less than 0.3 seconds).
- Alternatively, create more JVMs to spread out the load. This reduces the amount of active objects in a JVM which, in turn, reduces the frequency and the duration of the scavenge GC.

Keep in mind the following when configuring the survivor spaces:

- The survivor spaces must be large enough to store all the active objects coming from the eden as well as the sum of active objects that have an age that is less than the tenuring threshold.

Keep in mind the following when configuring the old heap:

- The amount of free space in the old heap must be larger than both the eden size plus one of the survivor space. If the free space is less, the JVM resorts to using Full GCs (see “Young Generation Guarantee” on page 34 below).

- The cost of a Full GC is dependent on the amount of active objects as well as the size of the old heap. A Full GC is typically a lot more noticeable to the end user than a scavenge GC. A Full GC on a 256MB old heap can take up to three seconds.
- Keep in mind that Sterling Selling and Fulfillment Foundation provides the ability to cache records. If you activate this feature, you should monitor the occurrence of full GC to see if the *old generation* is large enough. See “Performance Feature - Reference Data Caching” on page 104 for more information on the caching feature.

Therefore, you should allocate the *old* heap large enough so that Full GCs are not occurring too frequently (e.g., more than once in 15 minutes) and the collection service time is less than 2 seconds

Young Generation Guarantee

Starting in JDK 1.3.1_05, the Sun/HP JDKs implemented a conservative garbage collection policy called the Young Generation Guarantee. Before starting a GC, the JVM checks if the free space in the old heap (OLD FREE) is larger than the sum of the eden. The premise is that it is possible (though highly unlikely) that every object in the eden (remains alive and uncollected) the collection and has to be promoted to the old heap. If that ever happens, the Young Generation Guarantee ensures that there is enough free space in the old heap for all the promoted objects.

Please see extensive Java SE Hotspot information on http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html for a detailed description of the Young Generation Guarantee.

Starting Recommendations

We recommend that you try the default generational settings with a 384M and a 768M heap for your agents and application servers respectively:

```
java -server -Xms768m -Xmx768m \
     -XX:MaxPermSize=256m \
     weblogic.Server
```

Another approach is to set the overall heap to 1024MB with a 200MB young generation. For Solaris, you would issue the following command:

```
java -server -Xms1024m -Xmx1024m \
     -XX:NewSize=200m -XX:MaxNewSize=200m \
     -XX:MaxPermSize=256m \
     weblogic.Server
```

For HP-UX, you would issue the following command:

```
java -server -Xms1024m -Xmx1024m \
     -Xmn200m \
     -XX:MaxPermSize=256m \
     weblogic.Server
```

You have to regularly monitor the "health" of the garbage collection and adjust accordingly. For example:

- If you notice that the amount of heap free after a Full GC is approaching 500MB (the capacity of the old heap), you could eventually get java.lang.OutOfMemory exceptions. You should investigate why your JVM is keeping that many live objects. For example, with your data, you may have large reference data caches (see “Performance Feature - Reference Data Caching” on page 104).

- Conversely, if the amount of heap free after a Full GC is much smaller than the old heap (and the load test is representative), you may consider reducing the old heap.
- Increase the overall heap size - However, make sure the Full GC takes less than 2 seconds.
- You must ensure that the node has enough physical memory so that portions of the heap are not paged out.

The optimum JVM heap setting depends on your workload characteristics, your workload concurrency levels, your workload complexity, and so forth. The JVM heap setting can be (and often is) different between the application servers and agents. In addition, the settings may be different between some agents. As a result, you must periodically check the effectiveness of each JVM's heap setting.

Garbage Collection Statistics

We recommend that you continuously collect garbage collection statistics for all JVMs even in production. The collection overhead is minor compared to the benefit. With the statistics, you can tell if:

- A JVM has or is about to run into a memory leak.
- The garbage collection is efficient.
- Your JVM heap settings are optimal.

For a Sun JVM, the following statistics are displayed if you enable `-XX:+PrintGCDetails`, `-XX:+PrintGCTimeStamps`, and `-Xloggc`:

```
   0.000: [GC 0.001: [DefNew: 32192K->511K(33152K), 0.0383176 secs]
32192K->511K(101440K), 0.0385223 secs]
   1.109: [GC 1.110: [DefNew: 32703K->198K(33152K), 0.0344874 secs]
32703K->697K(101440K), 0.0346844 secs]
```

For an HP JVM, the following statistics are shown if you enable `-Xverbosegc:file`:

```
   <GC: 1 0 13848.360276 8 16400 31 429520056 0 429522944 0 2328104 53673984
100687544 100687544 536870912 69787968 69787968 69992448 0.162748 >
   <GC: 1 0 73541.610471 9 48 31 429522944 0 429522944 2328104 9051392 53673984
100687544 100687544 536870912 70708000 70708000 70778880 0.249739 >
```

Note: Create the GC log file name with the name of the workload and the starting time. In the example below the `-XX:+PrintGCTimeStamps` directive provides relative times of the GC from the time the JVMs started for the WebLogic server. The starting time in the file name allows you to determine when the GCs occurred:

```
   WORKLOAD=SCHEDULE
   gclog_file=${WORKLOAD}_'date +%Y%m%d-%H%M%S'
   java -verbosegc -XX:+PrintGCTimeStamps -Xloggc:${gclog_file}
weblogic.Server
```

Note: For Windows, format the example appropriately.

HotSpot JVM: Garbage Collection Statistics

In our opinion, garbage collection statistics are critical and should be enabled in production. The statistics is the only window you have into the behavior of the JVM heap management and the efficiency of the JVM.

This section describes three types of garbage collection statistics:

- The first is a comprehensive set from HP’s JVM.
- The second is a terse statistics from both the HP and Sun JVM.

Comprehensive HP GC Logs: At every garbage collection, the HP JVM prints out a statistic record with 20 fields in the following format:

```
   At every garbage collection, the following 20 fields are printed:
   <GC: %1 %2 %3 %4 %5 %6 %7 %8 %9 %10 %11 %12 %13 %14 %15 %16 %17 %18 %19 _
>

     %1:  Indicates the type of the garbage collection.
           1: represents a Scavenge (GC of New Generation only)
              %2: indicates if this is a parallel scavenge.
                   0: non-parallel scavenge
                   n(>0): parallel scavenge, n represents the number of
parallel GC threads

           2: represents an Old Generation GC or a Full GC
              %2: indicates the GC reason:
                   1: Allocation failure, followed by a failed scavenge,
leading to a Full GC
                   2: Call to System.gc
                   3: Tenured Generation full
                   4: Permanent Generation full
                   5: Scavenge followed by a Train collection
                   6: CMS Generation full
                   7: Old generation expanded on last scavenge
                   8: Old generation too full to scavenge
                   9: FullGCAlot
                  10: Allocation profiler triggered

           3: represents a complete background CMS GC
              %2:  indicates the GC reason:
                   1: Occupancy > initiatingOccupancy
                   2: Expanded recently
                   3: Incremental collection will fail
                   4: Linear allocation will fail
                   5: Anticipated promotion

           4: represents an incomplete background CMS GC
                 (exited after yielding to foreground GC)
              %2:  n.m
                   n indicates the GC reason:
                     1: Occupancy > initiatingOccupancy
                     2: Expanded recently
                     3: Incremental collection will fail
                     4: Linear allocation will fail
                     5: Anticipated promotion
                     6: Incremental CMS
                   m indicates the background CMS state when yielding:
                     0: Resetting
                     1: Idling
                     2: InitialMarking
                     3: Marking
                     4: FinalMarking
                     5: Precleaning
                     6: Sweeping

     %3:  Program time at the beginning of the collection, in seconds

     %4:  Garbage collection invocation. Counts of background CMS GCs
           and other GCs are maintained separately

     %5:  Size of the object allocation request that forced the GC,
           in bytes

     %6:  Tenuring threshold - determines how long the new born object
           remains in the New Generation
```

```
The report includes the size of each space:
   Occupied before garbage collection (Before)
   Occupied after garbage collection (After)
   Current capacity (Capacity)
All values are in bytes

Eden Sub-space (within the New Generation)
  %7:  Before
  %8:  After
  %9:  Capacity

Survivor Sub-space (within the New Generation)
  %10:  Before
  %11:  After
  %12:  Capacity

Old Generation
  %13:  Before
  %14:  After
  %15:  Capacity

Permanent Generation (Storage of Reflective Objects)
  %16:  Before
  %17:  After
  %18:  Capacity

  %19:  The total stop-the-world duration, in seconds.

  _:  The total time used in collection, in seconds.
```

HP provides a graphical tool called HPjtune to help you visualize the HP JVM garbage collection activities. This tool is free and can be downloaded. See HPjtune - visualization tool for HP JVM GC activities - http://www.hp.com.

The following are additional recommendations that add on to HP's excellent documentation.

Capacity

When the JVM is in steady state, check %9 (Eden Capacity), %12 (Survivor Sub-space Capacity), %15 (Old Generation Capacity), and %18 (Permanent Generation Capacity) to make sure you have allocated the JVM heap correctly.

Things to Monitor

In a healthy heap:

- During steady state, you should see mostly Scavenge GCs (%1=1) and the occasional Full GC caused by allocation failures (%1=2, %2=1).
- The sum of the GC times (%19 and _) should not exceed 3% of the measurement interval - for example, in a 1-hour measurement interval, the time taken for all GCs should not be more than 108 seconds.
- If you see continuous Full GCs (%2=1), check to see if the free space in the old heap (%15 - %13) is less than the sum of %7 and %10. If it is, the JVM uses Full GCs even though there may be lots of free space in the heap (see “Young Generation Guarantee” on page 34). This could be due to the amount of long-lived objects in the heap (see %14 after Full GCs), the old space (%15) is too small for the amount of long-lived objects or the eden (%9) is too big.
- If the amount of long-lived objects (%14 after Full GCs) is large (e.g., greater than 350MB) and has been steadily growing, you may have a memory leak. If

%14 continues to grow, that JVM eventually fails with an outOfMemory exception (see “OutOfMemory Exceptions” on page 28).

SUN

When the following flags are set:

```
   -verbose:gc -XX:+PrintGCTimeStamps -XX:+PrintGCDetails -Xloggc:filename
```

The Sun JVM produces the following garbage collection statistics:

```
   0.000: [GC 0.001: [DefNew: 32192K->511K(33152K), 0.0383176 secs]
32192K->511K(101440K), 0.0385223 secs]
   1.109: [GC 1.110: [DefNew: 32703K->198K(33152K), 0.0344874 secs]
32703K->697K(101440K), 0.0346844 secs]
   2.408: [GC 2.409: [DefNew: 32390K->403K(33152K), 0.0227843 secs]
32889K->902K(101440K), 0.0231518 secs]
```

Please see extensive Java information on http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html for examples of how to diagnose GC problems.

SUN JVM: Potential Memory Leak: After running the JVM for a while, check the amount of objects remaining after a Full GC to see if there are potential memory leaks (please see “OutOfMemory Exceptions” on page 28).

Old Heap Too Small

If you see successive Full GCs and the "heap after GC" number is consistently larger than the size of the Old Generation, the amount of live objects is larger than the Old Generation.

GC Times

Watch for GC times that take over 2-5 seconds. Recall that all threads are paused for the duration of the GC. A transaction that normally takes 1 second grows to 3-6 seconds. More importantly, blocked threads that are holding on to database locks could start to block other threads in other JVMs.

PrintGCStats Script

Sun has developed a script, PrintGCStats, to interpret the results from the output file generated by "-Xloggc:*filename*”. The PrintGCStats script can be downloaded from http://java.sun.com/developer/technicalArticles/Programming/turbo/.

Note: The PrintGCStats script can only produce meaningful results of the -Xloggc:filename output if the JVM is started with the "-verbose:gc -XX:+PrintGCTimeStamps -XX:+PrintGCDetails" flags.

An example output of the script is as follows:

```
   what         count       total      mean      max   stddev
   gen0(s)          9       0.591   0.06563    0.297   0.0870
   gen0t(s)         9       0.600   0.06665    0.305   0.0895
   gen1t(s)         9       7.890   0.87667    1.637   0.4381
   GC(s)           18       8.490   0.47166    1.637   0.5175
   alloc(MB)        9     721.889  80.20985   80.312   0.3079
   promo(MB)        9       0.000   0.00000    0.000   0.0000

   alloc/elapsed_time    =    721.889 MB /   4045.460 s  =   0.178 MB/s
   alloc/tot_cpu_time    =    721.889 MB /  32363.680 s  =   0.022 MB/s
   alloc/mut_cpu_time    =    721.889 MB /  32295.761 s  =   0.022 MB/s
```

```
promo/elapsed_time    =      0.000 MB /   4045.460 s  =   0.000 MB/s
promo/gc0_time        =      0.000 MB /      0.600 s  =   0.000 MB/s
gc_seq_load           =     67.919 s  /  32363.680 s  =   0.210%
gc_conc_load          =      0.000 s  /  32363.680 s  =   0.000%
gc_tot_load           =     67.919 s  /  32363.680 s  =   0.210%
```

The following table describes what each of the tags in the above excerpt means.

Table 14. PrintGCStats Output Statistics

Item Name	**Description**
gen0(s)	Young generation collection time in seconds
cmsIM(s)	CMS initial mark pause in seconds
cmsRM(s)	CMS remark pause in seconds
GC(s)	All stop-the-world GC pauses in seconds
cmsCM(s)	CMS concurrent mark phase in seconds
cmsCS(s)	CMS concurrent sweep phase in seconds
alloc(MB)	Object allocation in young generation in MB
promo(MB)	Object promotion to old generation in MB
`elapsed_time`(s)	Total wall clock elapsed time for the application run in seconds
`tot_cpu_time`(s)	Total CPU time = no. of CPUs * elapsed_time
`mut_cpu_time`(s)	Total time that was available to the application in seconds
`gc0_time`(s)	Total time used by GC during young generation pauses
`alloc/elapsed_time`(MB/s)	Allocation rate per unit of elapsed time in MB/seconds
`alloc/tot_cpu_time`(MB/s)	Allocation rate per unit of total CPU time in MB/seconds
`alloc/mut_cpu_time`(MB/s)	Allocation rate per unit of total application time in MB/seconds
`promo/gc0_time`(MB/s)	Promotion rate per unit of GC time in MB/seconds
`gc_seq_load`(%)	Percentage of total time spent in stop-the-world GCs
`gc_conc_load`(%)	Percentage of total time spent in concurrent GCs
`gc_tot_load`(%)	Total percentage of GC time (sequential and concurrent)

There are two statistics generated by this script that are very useful in tuning the JVM. The statistic *gc_seq_load* generates the total stop the world GC time as a percent of total application time. The statistic *gc_tot_load* is the total GC time for both full and scavenge GCs as a percentage of total application time. When making changes to the JVM this script should be run before and after to see if there is a positive change in these numbers. It is important to note that by lowering the gc_tot_load, and increasing the gc_seq_load, there would be a degradation in performance of the application overall. The reason for this is that the gc_seq_load is the total time the application spends in "Stop the World" GCs during which all threads are stopped.

IBM J9 JVM

This section provides recommendations on how to plan, implement, configure, monitor, and tune the IBM Java Virtual Machine.

The IBM J9 JVM is used when you deploy Sterling Selling and Fulfillment Foundation with WebSphere application servers on:

- IBM AIX operating system on POWER_PC based servers or
- Red Hat Enterprise Linux operating system on Intel processor based systems

Recommendations for Implementing IBM J9 JVM

In addition to the general recommendations in "General JVM Recommendations" on page 25, you should configure the IBM JVM as follows:

- Set JIT and MMI on (by default)
- Set PSALLOC=EARLY and NODISCLAIM=TRUE (IBM AIX only)
- Set the JVM to use the Generational Concurrent Garbage Collector (-Xgcpolicy:gencon).

JIT and MMI

As we mentioned in "General JVM Recommendations" on page 25, the Java language is interpreted. Interpretation and execution of bytecodes is much slower than the executing code that has been compiled to the native instruction set of the host processor. The IBM JVM uses mixed-mode interpretation (MMI) where initially bytecodes are interpreted. When the MMI detects that bytecodes have been interpreted multiple times, it invokes a just-in-time (JIT) compiler to compile those bytecodes to native instructions.

For performance, you should ensure the JIT and MMI are enabled. The JVM performance degrades significantly if JIT is disabled. Some third party vendors may recommend disabling certain portions of the JIT compiler. In those specific situations, we recommend you run controlled performance tests with and without that specific JIT option to understand the impact to performance.

The JDK 1.5 Diagnostic Guide (see Sumit Chawla, *Fine-Tuning Java Garbage Collection Performance, How to detect and troubleshoot garbage collection issues with the IBM Java Virtual Machine*, IBM). This provides an excellent description of the JIT compiler and the MMI.

PSALLOC and NODISCLAIM (AIX only)

IBM AIX implements, by default, a late page allocation policy. When you start a JVM with a large heap, AIX does not guarantee that there is sufficient page space to back the heap. AIX only allocates space on the page device when you use the heap. In some cases, AIX may have to kill JVMs when it is low on free space on the page devices.

To prevent this, we recommend setting the following environment variables prior to starting the JVM:

```
PSALLOC=EARLY
NODISCLAIM=TRUE
```

Please see "Page Space Allocation Policy" on page 17 for more information.

Generational Concurrent Garbage Collector

The Generational Concurrent Garbage Collector can be activated using the -Xgcpolicy:gencon command line option. By doing so, the garbage collector behaves the same as that for the SUN JDK, as described in "Sun and HP-UX Generational Collectors" on page 32. The Generational Concurrent Garbage Collector helps when application allocates many short-lived objects. Because of the transactional nature of the Sterling Selling and Fulfillment Foundation application, a large majority of the objects in the transaction do not survive beyond the transaction commit and can be garbage collected quickly.

During internal tests, we have noticed a 30% improvement in elapsed time with a 1-5% increase in CPU consumption when using the gencon GC policy over the default GC policy (-Xgcpolicy.optthruput).

Heap Memory and Garbage Collection

The JVM runtime environment uses a large memory pool called the heap for object allocation. The JVM automatically invokes *garbage collections* in order to clean up the heap of unreferenced or dead objects. In contrast, memory management in legacy programming languages like C++ was left to the programmer.

JVM is the foundation or engine on which Sterling Selling and Fulfillment Foundation and the WebLogic or WebSphere application server runs. If the JVM heap settings are not set correctly, the garbage collection overhead can make the system appear unresponsive. In the worst case, your transactions or the JVM could abort due to outOfMemory exceptions (please see "OutOfMemory Exceptions" on page 28).

The optimal JVM heap settings depends on many factors such as the type of processing. For example, a JVM dedicated to servicing short lived transactions has different demands than one that services a few but very large transactions.

Heap Settings

The default heap settings are appropriate for small applications. By default, the heap on AIX starts at 4MB and can grow to 64MB. You must adjust the heap settings for your environment.

Fortunately, the IBM JVM was designed to work with most scenarios with little tuning. From past experiences, we generally only set the initial and maximum heap size. The IBM JVM also provides good statistics to help monitor and tune the JVMs. See Mattias Persson, Holly Cummins, *Java technology, IBM Style, Part 2, Garbage collection with the Extensible Verbose Toolkit,* IBM for an excellent article on monitoring the JVM performance.

Starting Recommendations

We recommend you configure the IBM JVMs with the following starting recommendations and test the JVMs under representative workloads and traffic volumes prior to going live in production.

As a starting point, you should configure the JVMs running the Sterling Selling and Fulfillment Foundation agents and application servers with a 384MB, 1024MB, or larger heap respectively. The following are two sample configurations to start the Sterling Selling and Fulfillment Foundation Schedule agent and a WebLogic application server:

```
export PSALLOC=EARLY          # AIX Only
export NODISCLAIM=TRUE        # AIX Only

java -Xms384m -Xmx384m \
-Dsun.rmi.dgc.server.gcInterval=3600000 \
-verbosegc \
com.yantra.integration.adapter.IntegrationAdapter

java -Xms1024m -Xmx1024m \
-Dsun.rmi.dgc.server.gcInterval=3600000 \
-verbosegc \
weblogic.Server
```

The -Xms and -Xmx set the initial and maximum heap size. When setting these values, keep the following guidelines in mind:

- Set the initial (-Xms) and maximum (-Xmx) heap size the same - This eliminates the need of the JVM to decide when to expand or shrink the heap. This could also prevent the situation where there is not enough swap space when the JVM needs to expand the overall heap.
- By default, Sterling Selling and Fulfillment Foundation caches reference data for performance. Depending on your data setup, you may have to increase the heap size or reduce the numbers of cached records. See “Performance Feature - Reference Data Caching” on page 104 for more information on the caching feature.
- You must ensure that the node has enough physical memory so that portions of the heap are not paged out.
- You must ensure there is enough swap space to back the virtual address space requirement for all concurrently running JVMs or, if you are on AIX, ensure the JVMs are started with PSALLOC=EARLY and NODISCLAIM=TRUE (see “Page Space Allocation Policy” on page 17).

Garbage Collection Statistics

We recommend that you continuously collect garbage collection statistics for all JVMs especially in production. The collection overhead is minor compared to the benefit. With the statistics, you can tell if:

- A JVM has or is about to run into a memory leak.
- The garbage collection is efficient.
- Your JVM heap settings are optimal.

To enable GC statistics, set the -verbosegc option. The JDK 1.5 Diagnostic Guide (IBM Developer Kit and Runtime Environment, Java Technology Edition, Diagnostic Guide, Version 5.0, SC34-6650) provides very good guidance on how to interpret the garbage collection statistics.

Garbage Collection Statistics

Garbage collection statistics are critical and should be enabled in production. The statistics can be used to understand the behavior of the JVM heap management and the efficiency of the JVM.

This section describes the garbage collection statistics.

IBM provides excellent documentation on their garbage collector and how to interpret their GC statistics (see IBM Developer Kit and Runtime Environment, Java Technology Edition, Diagnostic Guide, Version 5.0, SC34-6650).

Here is a sample of the GC log:

```
   <af type="tenured" id="100" timestamp="Sun Nov 25 15:56:09 2007"
intervalms="120245.593">
     <minimum requested_bytes="10016" />
     <time exclusiveaccessms="0.045" />
     <tenured freebytes="2704" totalbytes="1073741824" percent="0" >
       <soa freebytes="2704" totalbytes="1073741824" percent="0" />
       <loa freebytes="0" totalbytes="0" percent="0" />
     </tenured>
     <gc type="global" id="100" totalid="100" intervalms="120245.689">
       <refs_cleared soft="0" threshold="32" weak="0" phantom="0" />
       <finalization objectsqueued="0" />
```

```
    <timesms mark="35.301" sweep="5.074" compact="0.000" total="40.426" />
    <tenured freebytes="808526296" totalbytes="1073741824" percent="75" >
      <soa freebytes="808526296" totalbytes="1073741824" percent="75" />
      <loa freebytes="0" totalbytes="0" percent="0" />
    </tenured>
  </gc>
  <tenured freebytes="808516280" totalbytes="1073741824" percent="75" >
    <soa freebytes="808516280" totalbytes="1073741824" percent="75" />
    <loa freebytes="0" totalbytes="0" percent="0" />
  </tenured>
  <time totalms="40.569" />
</af>
```

In the example above, `<af type="tenured" id="100"` indicates that this is the 100th time an attempt to allocate memory failed and as a result, a GC was initiated. An allocation failure is not an error in the system or code. When there is not enough free space in the heap, the JVM automatically initiate a garbage collection. The last time an allocation failure occurred was 120245.593 ms ago (or 120.245 seconds).

The lines starting with `<gc type="global" id="100"` provides information on the collection process. In the example above, garbage collection initiated the mark and sweep phases which completed in 35.301 and 5.074 milliseconds respectively. The JVM determined that the heap was not fragmented and hence did not require to compact the heap. At the end of the GC, the heap became 808,516,280 bytes free.

Frequency of GC Health Check

You should check how often GCs are occurring by looking at the time between allocation failures.

GC Times

You should monitor the amount of time the JVM spends in GC. Typically, your JVM should:

- Spend less than 0.5 seconds in each GC cycle.
- The percentage of time in garbage collection should be less than 3% - This percentage can be calculated by dividing the sum of the garbage collection times over an interval by the interval. The interval could be a fixed 20 minutes or the last 20 GCs.

Potential Memory Leak

If the JVM is running for a while and the percentage free is continually decreasing with each successive GC, that JVM could be heading to an outOfMemory condition. This could indicate that either the Java application is keeping a lot of active objects (e.g., reference data caching) or there is a memory leak.

By default, the IBM JVM produces a HeapDump when it runs out of memory. You can also configure the IBM JVM to produce a HeapDump on a signal. See “Heapdump” on page 44 below.

IBM Pattern Modeling and Analysis Tool for Java Garbage Collector

IBM provides an excellent tool called the IBM Pattern Modeling and Analysis Tool for Java Garbage Collector (pmat) to help you visualize the GC statistics such as pause times, amount of space requested, etc. The pmat is available for download from IBM's alphaworks website at http://www.alphaworks.ibm.com/tech/pmat. The download page contains excellent information about using the tool for analysis and troubleshooting.

Heapdump

For IBM JVMs, generate a heapdump and use the IBM Memory Dump Diagnostic tool to identify the memory leak. We have found this tool to be easy to use and easy to identify memory leaks. The IBM JVM automatically generates the heapdump when it runs into an OutOfMemory exception. You can also request a heapdump using a Kill -3 by first setting the following environment variables:

```
IBM_HEAPDUMP=true
IBM_HEAPDUMP_OUTOFMEMORY=true
IBM_HEAPDUMPDIR=<directory to store the heap dumps>
```

The IBM Memory Dump Diagnostic is an excellent tool for analyzing the heapdump. You can download the Memory Dump Diagnostic from http://www.alphaworks.ibm.com/tech/heapanalyzer.

You can read up on IBM heapdumps (and a lot more) in the IBM JDK 5.0/6.0 Diagnostic Guides (see IBM Developer Kit and Runtime Environment, Java Technology Edition, Diagnostic Guide, Version 5.0, SC34-6650).

Oracle JRockit

This section provides recommendations on how to plan, implement, configure, monitor, and tune the Oracle JRockit Java Virtual Machine.

The JRockit JVM is used when you deploy Sterling Selling and Fulfillment Foundation with WebLogic application servers on either the Red Hat Enterprise Linux or Windows operating system on Intel processor-based systems.

Implementation

The Oracle JRockit was designed for server-side applications. One distinguishing feature of JRockit is its adaptability. During the life of the JVM, JRockit could change the type of garbage collector used or the size of the heap.

Starting Recommendations

To exploit the adaptivity of the JRockit JVM, we recommend the simple starting JVM settings:

```
java -Xms384m -Xmx384m \
-Dsun.rmi.dgc.server.gcInterval=3600000 \
-Xverbosetimestamp -verbosegc \
com.yantra.integration.adapter.IntegrationAdapter

java -Xms768m -Xmx768m \
-Dsun.rmi.dgc.server.gcInterval=3600000 \
-Xverbosetimestamp -verbosegc \
weblogic.Server
```

Heap Settings

The -Xms and -Xmx sets the initial and maximum heap size. When setting these values, keep the following guidelines in mind:

- Set the initial (-Xms) and maximum (-Xmx) heap size the same - This eliminates the need of the JVM to decide when to expand or shrink the heap. This could also prevent the situation where there is not enough swap space when the JVM needs to expand the overall heap.
- By default, Sterling Selling and Fulfillment Foundation caches reference data for performance. Depending on your data setup, you may have to increase the heap size or reduce the numbers of cached records. See “Performance Feature - Reference Data Caching” on page 104 for more information on the caching feature.
- You must ensure that the node has enough physical memory so that portions of the heap are not paged out.

Garbage Collection Statistics

We recommend that you continuously collect garbage collection statistics for all JVMs even in production. The collection overhead is minor compared to the benefit. With the statistics, you can tell if:

- A JVM has or is about to run into a memory leak.
- The garbage collection is efficient.
- Your JVM heap settings are optimal.

To enable GC statistics, set the `-verbosegc` option.

Chapter 5. Application Servers

WebSphere Tuning

This section assumes that you:

- Are familiar with and have installed WebSphere application server in a network deployment
- Have read the *IBM WebSphere Application Server Tuning Guide*.

This section elaborates on the recommendations found in the *IBM WebSphere Application Server Tuning Guide*.

WebSphere Queuing Network

The *IBM WebSphere Application Server Tuning Guide* describes transactions to WebSphere as being processed in a network of interconnected queues that includes the network, Web Server, Web Container, EJB container, data source, connection pool, and the database sessions.

IBM then recommends that the queues closest to the client be large (e.g, the network) and that downstream queues (e.g., EJB container, data source) grow progressively smaller as it gets further from the client. One of the reasons offered is that an application that spends 90% of its time in a complex servlet and only 10% of its time making short JDBC queries would require a significantly smaller database connection queue than the Web Container queue.

We agree with the first statement that the network queue can be large because it is preferable to queue in the network and not in the application server. However, from our experience, the downstream queues should be set to the same size if not larger.

In database-intensive applications, such as Sterling Selling and Fulfillment Foundation, when a transaction enters the Web Container, the APIs almost always require a database connection. If one were to allow 20 concurrent transactions (by setting `Maximum Thread Size` in the Web Container Services) to run against a connection pool of 12, there is a possibility that at peak processing periods, 8 transactions would either be forced to wait for a connection or throw an exception because it can't get a connection.

Using the same argument, the database instance should be able to create more sessions than the sum of all the connections possible from all the WebSphere instances combined. You also need additional database sessions for standalone Java applications that need database services (for example, the Sterling Selling and Fulfillment Foundation agents or monitors), real-time performance monitors, database utilities and so forth.

WebSphere Connection Pool

Creating database connections are very expensive operations. For performance, Sterling Selling and Fulfillment Foundation takes advantage of the WebSphere connection pool.

To enable connection pooling, you need to:

- Define a data source to Sterling Selling and Fulfillment Foundation.
- Define a connection pool in WebSphere and then associate the connection pool to the datasource name in Sterling Selling and Fulfillment Foundation.
- IBM recommends that data sources do not participate in container managed persistence (CMP). For configuration, go to the data sources setup in WebSphere and clear the **Use this data source in container managed persistence (CMP)** field while creating the data source.

WebSphere Connection Pool: Define Data Source in Sterling Selling and Fulfillment Foundation

To define the datasource name to Sterling Selling and Fulfillment Foundation, add the following entry in the `<INSTALL_DIR>/properties/customer_overrides.properties` file (see "Property File" on page 114):

```
yfs.yfs.dblogin.datasource.name=yfsdbsourceperf
```

For additional information about overriding properties using the `customer_overrides.properties file`, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*.

At initialization, Sterling Selling and Fulfillment Foundation uses the datasource name to find the connection pool in WebSphere. In the example above, the datasource name is `yfsdbsourceperf.`

IBM recommends that you benchmark your application before migration to production to ensure that these values are set optimally. IBM also recommends that you continually monitor the connection pool usage levels to ensure that these parameters are set optimally.

Define a Connection Pool in WebSphere

In the WebSphere administrative console, create a new connection pool with the following attributes.

Table 15. Connection Pool - Connection Settings

Attribute	Value
Minimum pool size	Initial number of connections to create for the connection pool. If the pool is allowed to shrink, it does not shrink below this number. See below for recommendations.
Maximum pool size	Maximum number of connections that can be created for this pool. See below for recommendations.
Statement cache size	The maximum number of prepared statements to cache for the data source.

IBM recommends that you benchmark your application before migration to production to ensure that these values are set optimally. IBM also recommends that you continually monitor the connection pool usage levels to ensure that these parameters are set optimally.

Minimum Pool Size

Bear in mind the following guidelines when setting the minimum pool size attribute:

- You should set the minimum pool size to satisfy your daily average connection requirements. This level can be derived by monitoring your actual pool usage levels.

- You may want to set the minimum pool size to a higher number if your system experiences frequent traffic bursts.
- You may not want to set minimum pool size to a very high number because both WebSphere and the database server needs to maintain a high number of connections.

Maximum Pool Size

This attribute sets the maximum number of connections the pool can grow to within a single WebSphere instance. If you set this value to 20 and you have ten WebSphere instances, in theory, WebSphere could create 200 database connections.

Bear in mind the following guidelines when setting the maximum pool size attribute:

- If you are not using the multi-schema feature, each Sterling Selling and Fulfillment Foundation transaction generally requires one connection. Therefore, you need one connection per active thread. In practice, set the maximum capacity to be around the active thread count plus a small number (for example, 2 or 5) for a safety buffer.
- If you are using the multi-schema feature, each Sterling Selling and Fulfillment Foundation transaction requires up to four connections. These are required for the metadata schema, configuration schema, transaction schema and the statistics schema. You will need to establish separate pools for each one of these schemas. Therefore, you need one connection per pool per active thread. In practice, it is recommended to set the maximum capacity to be equal to the active thread count plus a small number (for example, 2 or 5) for a safety buffer.
- Monitor your application in production to confirm that the ratio of connection usage follows the guidelines mentioned.
- Benchmark your application to see if custom code, user exits, and so forth require additional connections.

Statement Cache Size

This attribute tells WebSphere to create a cache at the data source level to store prepared statements up to the value specified.

Prepared statements are precompiled SQL statements that can be repeatedly invoked with different parameter values. Prepared statements reduce the need to compile the SQL statements.

To disable prepared statement caching, set the prepared statement cache size to 0. To use the cache, you can set a value to a higher value.

WebSphere Tuning: JSP Pre-Compilation

When users call a JSP page the first time, WebSphere automatically compiles that JSP. JSP compilations, however, can take a long time, which could lead to the perception that the system is slow. Some JSPs can take over a minute to compile. You can avoid this problem by precompiling the JSPs before or after deployment.

To do this, use the JspBatchCompiler script from your WebSphere install's `${WAS_ROOT}/bin`.

For precompiling prior to deployment, use:

```
./JspBatchCompiler.sh -ear.path <ear location>/<EAR name> -forceCompilation true
-compileToDir <path>
```

You can then deploy this precompiled EAR as usual through the WebSphere Administration console. It is possible that WebSphere may need more memory when **deploying** this EAR. The advantage in precompiling before deployment is that you will not have to precompile on every single node after deployment.

For precompiling after deployment, use:

```
./JspBatchCompiler.sh  \
-enterpriseapp.name ${appName} \
-cell.name <customer cell name> \
-node.name <customer node name> \
-server.name <customer server name> \
-profileName <profile name> \
-keepgenerated true
```

Note: For Windows, format the example appropriately.

Note: The customer cell name, customer node name, customer server name, and customer profile name parameters refer to the WebSphere instance where the EAR was deployed.

For additional information about how to use the JspBatchCompiler, use the -help option to the command or refer to WebSphere's Infocenter documentation.

WebSphere: HTTP Load-Balancing

The Sterling Selling and Fulfillment Foundation HTTP screens are stateful in the sense that a screen preserves state information for subsequent screens. As a result, you have to set up proxy servers or load-balancers to load-balance HTTP requests with a “sticky” load-balancing policy. This ensures HTTP requests go back to the server that have the session states.

Load-balancing can improve performance for large number of HTTP users because the user population is serviced by multiple application servers that are managed as a cluster. Load-balancing can be implemented with a variety of technologies ranging from the Apache proxy servers to hardware-based load balancers.

HTTP Session Replication

Sterling Selling and Fulfilment Foundation supports HTTP session replication with WebSphere 7.0. See the application server documentation (info center) for the OS version to set up session replication. Testing has been performed with memory-to-memoryreplication for peer-to-peer mode.

WebSphere: Monitoring

You should monitor the following on a regular basis:

- Execute Thread Count
  - Track the average and maximum number of active execute threads through third-party tools or custom-developed JMX-based programs.
  - Correlate that number to the workload level issued to the application servers.
  - Either using mathematical projections or system tests, estimate the number of concurrent threads expected during your peak operational periods. As a general rule, you should plan to keep the average active threads to 15 or less.
- Garbage Collection

  - Monitor the frequency and health of the JVM's heap management. Please see the section relevant to your JVM in "General JVM Recommendations" on page 25.
- Connection Pool Usage
  - Check if Connection High is equal to the JDBC pool size - The Connection High is the highest number of connections ever reached. Recall, the JDBC pool to be equal to the maximum possible transaction concurrency level plus a safety buffer of two. The Connection High should not be the same as the JDBC pool size.

Oracle's WebLogic Tuning Recommendations

This section assumes that you:

- Are familiar with and have installed WebLogic application server in a clustered mode
- Have read Oracle's *WebLogic Server Performance and Tuning* which can be found in http://www.oracle.com/technology/documentation/index.html

In keeping with our performance management principles, this section augments the recommendations found in Oracle's *WebLogic Server Performance and Tuning* when needed. You should carefully review Oracle's recommendations because not all apply. As always, you should test any changes from their default settings to see if the changes benefit your configuration. Some of our recommendations that deviate from Oracle's include:

- Do not use Oracle's one-way message send performance feature that was introduced in WebLogic 9.2. The one-way send feature allow message producers to send messages to queues without waiting for a corresponding response or acknowledgement. This feature improves performance at the cost of lowering reliability. The Sterling Selling and Fulfillment Foundation workloads, JMS interactions are transactional and require an acknowledgement.

Work Manager

The Sterling Selling and Fulfillment Foundation workloads run in WebLogic's default work manager. You should monitor the default work manager's pool size over time to see if the pool size is sufficient.

If the pool size climbs to 25 or more threads, you may want to consider = configuring additional WebLogic Server instances instead of increasing the thread count.

Application Server Instances

Please keep in mind the following when determining the number of WebLogic instances:

- Configure at most one WebLogic Server instance per processor - If you have a 4-way node (4 processors), then configure at most 4 WebLogic Server instances on that node.
- Plan for a single WebLogic Server instance to be able to use at most 4 (preferably 2) processors. If you have an 8-way node, you should not implement one WebLogic Server instance and expect it to effectively use all processors. With this rule, to fully utilize the node, you should plan for two to four WebLogic Server instances.

WebLogic Connection Pool

Creating database connections are very expensive operations. For performance, Sterling Selling and Fulfillment Foundation takes advantage of the WebLogic connection pool.

To enable connection pooling, you need to:

- Define a data source to Sterling Selling and Fulfillment Foundation.
- Define a connection pool in WebLogic.
- Define a data source in WebLogic that ties the data source in Sterling Selling and Fulfillment Foundation to the connection pool in WebLogic.

WebLogic Connection Pool: Define Data Source in Sterling Selling and Fulfillment Foundation

To define the data source name to Sterling Selling and Fulfillment Foundation, add the following entry in the `<INSTALL_DIR>/properties/customer_overrides.properties` file (see "Property File" on page 114):

```
yfs.yfs.dblogin.datasource.name=yfsdbsourceperf
```

For additional information about overriding properties using the `customer_overrides.properties file`, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*.

At initialization, Sterling Selling and Fulfillment Foundation uses the datasource name to find the connection pool in WebLogic. In the example above, the datasource name is `yfsdbsourceperf.`

IBM recommends that you benchmark your application before migration to production to ensure that these values are set optimally. IBM also recommends that you continually monitor the connection pool usage levels to ensure that these parameters are set optimally.

Initial Capacity

Bear in mind the following guidelines when setting the initial capacity attribute:

- You should set the initial capacity to satisfy your daily average connection requirements. This level can be derived by monitoring your actual pool usage levels.
- You may want to set the initial capacity to a higher number if your system experiences frequent traffic bursts.
- You may not want to set initial capacity to a very high number because both WebLogic and database server need to maintain a high number of connections. For example, assume you have 8 managed server instances, each with 15 execute threads. If you set initial and maximum connection at 17, WebLogic creates and maintain 136 database connections.

Note: The Oracle WebLogic Server Performance and Tuning manual (see Tune the Number of Database Connections) recommends setting the initial connection pool capacity equal to the maximum capacity. Unfortunately, if you to follow that recommendation, you can not determine the current pool usage levels since the pool usage would be equal to the initial and the maximum. As a result:

- You can not determine if there is a connection leak - for example, if the current connection pool usage is higher than your work manager thread utilization.

- You can not know if your current connection pool usage is close to the maximum capacity.

For that reason, we prefer to keep the initial capacity lower than the maximum capacity.

Maximum Capacity

This attribute sets the maximum number of connections your pool can grow to within a single WebLogic Server instance. If you set this value to 27 and you have eight WebLogic Server instances, in theory, WebLogic could create up to 216 database connections.

Bear in mind the following guidelines when setting the maximum capacity attribute:

- If you are not using the multi-schema feature, each Sterling Selling and Fulfillment Foundation transaction generally requires one connection. Therefore, you need one connection per active thread. In practice, set the maximum capacity to be around the active thread count plus a small number (for example, 2 or 5) for a safety buffer.
- If you are using the multi-schema feature, each Sterling Selling and Fulfillment Foundation transaction requires up to four connections. These are required for the metadata schema, configuration schema, transaction schema and the statistics schema. You will need to establish separate pools for each one of these schemas. Therefore, you need one connection per pool per active thread. In practice, it is recommended to set the maximum capacity to be equal to the active thread count plus a small number (for example, 2 or 5) for a safety buffer.
- Monitor your application in production to confirm that the ratio of connection usage follows the guidelines mentioned.
- Benchmark your application to see if custom code, user exits, and so forth require additional connections.

Allow Shrinking and Shrink Frequency

This attribute pair informs WebLogic to release inactive connections if they have been idle for the period as specified by "shrink frequency". This has the advantage of releasing resources both at the WebLogic and database server end.

Prepared Statement Cache Size

This attribute tells WebLogic to create a cache for each database connection that can store prepared statements up to the value specified.

Prepared statements are precompiled SQL statements that can be repeatedly invoked with different parameter values. Prepared statements reduce the need to compile the SQL statements.

We recommend disabling the cache. To disable prepared statement caching, set the prepared statement cache size to 0.

WebLogic Tuning: JSP Pre-Compilation

When users call up a JSP page the first time, WebLogic automatically translates the JSP file into a servlet and then compiles that servlet. This process can take over 30 seconds, which could lead to user dissatisfaction. Further, this process is performed serially even on a multiprocessor node - if you have multiple users hitting five different pages, WebLogic compiles these pages one at a time. As a result, we strongly recommend precompiling the JSP pages prior to deployment into production.

To precompile, you need to build the Sterling Selling and Fulfillment Foundation enterprise archive file (smcfs.ear). See the *Sterling Selling and Fulfillment Foundation: Installation Guide*.

When you have the ear, issue the following:

```
   WL_HOME=The Weblogic server directory
   INSTALL_DIR=Sterling Selling and
Fulfillment Foundation
installation directory
   WLS_JARS=${JAVA_HOME}/lib/tools.jar:\
   ${WL_HOME}/lib/weblogic_sp.jar:\
   ${WL_HOME}/lib/weblogic.jar:\
   ${WL_HOME}/lib/ojdbc6.jar:\
   ${JAVA_HOME}/jre/lib/rt.jar:\
   ${WL_HOME}/lib/webservices.jar
   STERLING_CLASSPATH="${WLS_JARS}:${CLASSPATH}"
   APPC_CLASSPATH=${WL_HOME}/lib/weblogic.jar:\
   ${JAVA_HOME}/lib/tools.jar:\
   ${JAVA_HOME}/bin/java -Xms1024m -Xmx1024m \
   -classpath ${APPC_CLASSPATH} weblogic.appc \
   -output ${INSTALL_DIR}/external_deployments/smcfs.ear \
   -forceGeneration \
   -O \
   -verbose \
   -classpath ${STERLING_CLASSPATH} \
   ${INSTALL_DIR}/external_deployments/smcfs.ear
```

Note: For Windows, format the example appropriately.

The precompiled JSPs are stored back into the `smcfs.ear` file.

WebLogic Server Cluster

For operational simplicity, you could implement the WebLogic Server as a single (non-clustered) instance. However, for availability and performance, many installations implement a WebLogic cluster over separate physical nodes. When creating the cluster, consider the following:

- For performance, multiple WebLogic instances in a cluster generally out-perform a single WebLogic instance on a large SMP node.
- For availability, multiple WebLogic instances spread over multiple physical nodes reduce the impact of losing a node.
- For performance, multiple WebLogic instances spread over multiple physical nodes reduce the impact of garbage collection on response times to users.
- For maintainability, set aside a bank of consecutive IP addresses for the cluster so that you can multi-home the network cards. Each WebLogic instance requires a unique IP address.

WebLogic: HTTP Load-Balancing

The Sterling Selling and Fulfillment Foundation HTTP screens are stateful, in the sense that a screen preserves state information for subsequent screens. As a result, you have to set up proxy servers or load-balancers to load-balance HTTP requests with a “sticky” load-balancing policy. This ensures HTTP requests go back to the server that have the session states.

Load-balancing can improve performance for large number of HTTP users because the user population is serviced by multiple application servers that are managed as a cluster. Load-balancing can be implemented with a variety of technologies ranging from the Apache proxy servers to hardware-based load balancers.

HTTP Session Replication

Sterling Selling and Fulfillment Foundation supports HTTP in-memory session replication in the following configuration:

- WebLogic
- Apache 2.0.44 with the WebLogic plug-in as the proxy server with idempotent set to 'OFF'

See the *Sterling Selling and Fulfillment Foundation: Installation Guide* for instructions on how to set up WebLogic in-memory session replication.

Note: The Apache or load-balancer idempotent flag must be set to OFF. In rare cases, for example, when a transaction completes and commits but was unable to post the response to the proxy server, the proxy server could retransmit the transaction. For some update transactions, this could result in duplicate update entries.

WebLogic: Monitoring

You should monitor the following on a regular basis:

- Work Manager Thread Utilization
  - Track the average and maximum number of active execute threads through third-party tools or custom-developed JMX-based programs.
  - Correlate that number to the workload level issued to the application servers.
  - Either using mathematical projections or system tests, estimate the number of concurrent threads expected during your peak operational periods. As a general rule, you should plan to keep the average active threads to 15 or less.
- Garbage Collection
  - Monitor the frequency and health of the JVM's heap management. Please see the section relevant to your JVM in “General JVM Recommendations” on page 25.
- Connection Pool Usage
  - Check if Connection High is equal to the JDBC pool size - the Connection High is the highest number of connections ever reached. Recall, the JDBC pool to be equal to the maximum possible transaction concurrency level plus a safety buffer of two. The Connection High should not be the same as the JDBC pool size.

JBoss Tuning

This section assumes that you:

- Are familiar with and have installed the JBoss application server in a clustered mode.
- Have read the Jboss admin and development guide (*JBoss Enterprise Application Platform Documentation Library* – http://www.redhat.com/docs/en-US/JBoss_Enterprise_Application_Platform/EAP_4_3_Portal.html#EAP4_3_Rel).

This section elaborates on the recommendations found in *JBoss Enterprise Application Platform Documentation Library* (http://www.redhat.com/docs/en-US/JBoss_Enterprise_Application_Platform/EAP_4_3_Portal.html#EAP4_3_Rel).

JBoss Tuning Considerations

- Disable scanning for hot deployments.
- Set up prepared statement cache and connection pooling.
- Precompile the EAR (See section "JSP Tuning: JSP Pre-Compilation) and disable compilation checks in production.
- Set Web container thread pools accordingly. Watch the `currentThreadsBusy` parameter. If this increases, watch the threadpool and the connectionPool.
- Trim log4j settings (turn off console appender; reduce severity from WARN to ERROR)
- Turn off development mode in your `server/slim/jbossweb-tomcat50.sar/conf/web.xml`.
- Disable mail service.
- Disable hypersonic database (not to be used in production).
- If you are not using JBossMQ or JBoss JMS, disable the same.
- Disable XA datasource (Sterling Supply Chain applications do not use XA datasources).
- Disable scheduler service if not needed.

JBOSS Connection Pooling

Creating database connections are very expensive operations. For performance, Sterling Selling and Fulfillment Foundation takes advantage of the connection pooling feature in JBoss.

To enable connection pooling, you need to:

- Define a data source to Sterling Selling and Fulfillment Foundation.
- Define a connection pool in JBoss.
- Define a data source in JBoss that ties the data source in Sterling Selling and Fulfillment Foundation to the connection pool in JBoss.

JSP Tuning: JSP Pre-Compilation

When users invoke a JSP page for the first time, JBoss automatically compiles the JSP. The JSP compilation may take a long time. Some JSPs may take more than one minute to finish compilation. Due to this, it appears as though the system performance is slow. To improve the system performance, ensure that you precompile JSPs before building the Sterling Selling and Fulfillment Foundation enterprise archive file (`smcfs<application_name>.ear`).

To precompile JSPs:

- Add the following lines to the properties/sandbox.cfg file to request precompilation. By default, the JSPs for JBoss are not precompiled:

  ```
  JBOSS_PRECOMPILE_JSP=TRUE
  JBOSS_DIR=<JBOSS_HOME>
  ```

- From <INSTALL_DIR>/bin, run the command to rebuild your property files:

  ```
  setupfiles.sh
  ```

- Build the Sterling Selling and Fulfillment Foundation enterprise archive file (`smcfs<application_name>`.ear) using instructions from the *Sterling Selling and Fulfillment Foundation: Installation Guide*.

The precompile scripts expects the JBoss server to be the default name of "all". See the *Sterling Selling and Fulfillment Foundation: Installation Guide*.

You can increase the amount of JVM heap that the JSP compiler runs in by changing the following parameters in the <INSTALL_DIR>/bin/build.properties:

```
jboss.PRECOMPILE_MAX_MEMORY=2048m
jboss.PRECOMPILE_MIN_MEMORY=512m
```

JBoss: HTTP Load-Balancing

The Sterling Selling and Fulfillment Foundation HTTP screens are stateful in the sense that a screen preserves state information for subsequent screens. As a result, you have to set up proxy servers or load-balancers to load-balance HTTP requests with a “sticky” load-balancing policy. This ensures HTTP requests go back to the server that have the session states.

Load-balancing can improve performance for large number of HTTP users because the user population is serviced by multiple application servers that are managed as a cluster. Load-balancing can be implemented with a variety of technologies ranging from the Apache proxy servers to hardware-based load balancers.

HTTP Session Replication

Please refer to the JBoss application server documentation (http://www.redhat.com/docs/en-US/JBoss_Enterprise_Application_Platform/EAP_4_3_Portal.html#EAP4_3_Rel) for details to set up HTTP Session replication. Testing was performed with CacheMode set to REPL_ASYNC (the default value) in `jboss-service.xml` under the `~/deploy/jbossweb-cluster.sar/META-INF/` directory.

Chapter 6. Database Management System

Sterling Selling and Fulfillment Foundation uses a database server as a repository for the transactional, reference, and history data generated and used by Sterling Selling and Fulfillment Foundation.

This chapter provides planning elements that have to be completed prior to the implementation phase. The key planning tasks include, at a minimum:

- Selecting a certified database management server software and version.
- Determining the size and configuration of the database server node
- Determining the size and configuration of the database disk subsystem
- Determining the disk technology

Supported Configurations

See the *Sterling Selling and Fulfillment Foundation: System Requirements Guide* for a list of the supported configurations.

Server Sizing

At appropriate times in the project lifecycle, you can request a Server Sizing study from your Professional Services Project Manager or a IBM Sales Executive. This study starts with the Sterling Selling and Fulfillment Foundation Server Sizing Questionnaire. IBM Performance Engineering creates a sizing document that provides an estimated processor, memory, and network requirement for the standard/baseline Sterling Selling and Fulfillment Foundation. You need to factor in additional requirements such as other workloads on the same node (for example, additional third party software, customization, performance monitors, and so forth).

Disk Technology

Your disk capacity requirement is a very important input to the disk configuration planning process. This is not a simple process that involves many other factors including:

- Survivability
  - Configure the disks with the ability to survive single or multiple disk failures (e.g., RAID-1 or RAID-10).
  - Configure the disk array with multiple I/O paths to the server to survive I/O path failures.
  - Configure the disks to be accessible from multiple server nodes to tolerate a single node failure.
- Manageability
  - If you have very short time windows to backup the database, select disk arrays that allow you to take logical backups (e.g., array snapshots).
- Scalability/performance
  - Configure the disk array with many small disks instead of a few large disks so that you can increase the number of I/O paths.
  - Configure the disk array with large NVRAM cache to improve read and write performance.

– Configure the disks with stripping (e.g., RAID-0 or RAID-10).

Let's take for example that you need 900GB and you have disk arrays or storage area networks (SAN) that are made up of 93GB disks. The following table summarizes the trade-off choices for the common disk organizations. Let's further assume that the database is implemented over ninety 10GB data files.

Table 16. Disk Organization - Trade-Off

Tech	Scalability	Survivability	Maintainability	Num Disks
JBOD	Poor - Subject to throughput of individual disks	Poor - Single disk failure creates outage and require database recovery	Poor - High disk utilization skew	10
RAID-0	Excellent - Striping N disks provides read/write throughput at N times a single disk	Poor - Single disk failure creates outage and require database recovery	Excellent - expect near uniform disk utilization within a logical unit. Potential LUN utilization skew.	10
RAID-1	Poor - similar performance to JBOD	Better - Could survive multiple disk failures in different mirrored sets	Poor - High disk utilization skew	20
RAID-5	Excellent for read - similar to RAID-0. Potentially poor for write performance.	Better - Able to survive a single disk failure. Multiple disk failures creates an outage and require database recovery.	Excellent - low disk util skew. Possible LUN utilization skew.	11
RAID-01	Excellent read/write performance.	Could tolerate up to two disk failures as long as both failures are not in the same mirrored set.	Excellent - low disk util skew. Possible LUN utilization skew.	20
RAID-10	Excellent read/write performance.	Could tolerate up to N disk failures as long as there isn't two failures in a mirrored set	Excellent - low disk util skew. Possible LUN utilization skew.	20

Sterling Selling and Fulfillment Foundation Schema

The *Sterling Selling and Fulfillment Foundation: Installation Guide* provides directions on how to create the Sterling Selling and Fulfillment Foundation database, the Sterling Selling and Fulfillment Foundation tables, and their associated indexes.

These DDL statements are intended to allow you to create a simple schema that is suitable for general use. You need to review and possibly modify these statements for production. Specifically:

- The DDL statements create a minimal set of indexes for general Sterling Selling and Fulfillment Foundation use. You may need to create additional or modify existing indexes for your business practice.
- The DDL statements create a single tablespace with a large data file. You may want to create additional tablespaces for manageability as well as additional data files for I/O load balancing.

Indexes

Most customers use a subset of the broad functionality in Sterling Selling and Fulfillment Foundation. As a result, the base Sterling Selling and Fulfillment Foundation database schema with the default or starting set of indexes may have to be adjusted for your specific use. Therefore, you should validate the starting index set to see if they support your workloads.

As a suggestion, prior to production, you should conduct a system test where all the key screens, agents and APIs run against a copy of the production database. During this test, you can check if additional indexes are required and if there are any unused indexes you can disable or drop.

Please see the following sections on how to enable index monitoring:

- For Oracle, see "Sterling Selling and Fulfillment Foundation Schema: Oracle Index Monitoring and Tuning" on page 72.
- For DB2, see "Sterling Selling and Fulfillment Foundation Schema: DB2 Index Monitoring and Tuning" on page 66.
- For Microsoft SQL Server, see "Microsoft SQL Server Index Monitoring and Tuning" on page 77.

Custom Indexes

Please follow the following convention when you create a new index:

First, make sure the index name does not end with the following suffix:

- _PK" - This suffix is reserved for indexes that are the primary key for the underlying table. For example, the index, `yfs_order_header_pk`, is the primary key index for the `yfs_order_header` table.
- _I*nn*" where *nn* is an integer value from 0 to 99 - This suffix is reserved for secondary or alternate indexes for the underlying table. For example, the index, `yfs_order_header_i1`, is a secondary index for the `yfs_order_header` table.

The convention above prevents situations where new base Sterling Selling and Fulfillment Foundation indexes have the same name as one of your custom index.

Secondly, to further differentiate custom indexes from the base, the custom index should start with `EXTN_` as a prefix and X*nn* in the index name. For example, if you add two indexes to the YFS_ORDER_HEADER table, the index names should be EXTN_ORDER_HEADER_X1 and EXTN_ORDER_HEADER_X2.

DB2: Implementation

This section provides guidelines on the implementation, configuration and tuning of DB2 9.7.

Recommended DB2 dbset Registry Variables

The following registry variables are recommended for performance.

Table 17. db2set registry variables

db2set registry variables	Value
DB2_USE_ALTERNATE_PAGE_CLEANING	ON
DB2_EVALUNCOMMITTED	ON

Table 17. db2set registry variables (continued)

db2set registry variables	Value
DB2_SKIPDELETED	ON
DB2_SKIPINSERTED	ON
DB2_PARALLEL_IO	See below for recommendations
DB2_SELECTIVITY	ALL
DB2LOCK_TO_RB	STATEMENT

DB2_EVALUNCOMMITTED

Enabling this variable can reduce the amount of unneeded lock contention from read share and next-key share. By default, DB2 requests share locks on the index or record before it checks if the record satisfies the query predicate. Queries that scan a set of records in tables with high frequency of inserts or updates can unnecessarily block on records that do not belong to its result set.

When you set `DB2_EVALUNCOMMITTED=ON`, DB2 performs an uncommitted read on the record to perform the predicate check. If the record satisfies the predicate, DB2 then requests a share lock on that record.

DB2_SKIPDELETED

Enabling this variable allows index-range or table-scan queries to skip over records that are in an uncommitted delete state. This reduces the amount of lock contention from read share and next-key share locks from range queries in tables with a high frequency of deletes.

When enabled, allows, where possible, table or index access scans to defer or avoid row locking until a data record is known to satisfy predicate evaluation. With this variable enabled, predicate evaluation may occur on uncommitted data.

It is applicable only to statements using either Cursor Stability or Read Stability isolation levels. For index scans, the index must be a type-2 index. Furthermore, deleted rows are skipped unconditionally on table scan access while deleted keys are not skipped for type-2 index scans unless the registry variable DB2_SKIPDELETED is also set.

DB2_SKIPINSERTED

Enabling this parameter allows SELECTs with cursor stability or read stability isolation levels to skip over uncommitted inserted rows. This parameter setting can reduce record lock contention on tables with heavy insert rates.

DB2_PARALLEL_IO

Enabling this variable changes the way in which DB2 calculates I/O parallelism to the tablespace. By default, DB2 sets I/O parallelism to a tablespace to be the number of containers in that tablespace. For example, if the tablespace has four containers, prefetches to that tablespace are performed as four extent-sized prefetch requests.

You should set the DB2_PARALLEL_IO variable if you implement containers on stripped devices (e.g., RAID-5, RAID-10 or RAID-01).

If you set DB2_PARALLEL_IO=ON or DB2_PARALLEL_IO=*, DB2 assumes that containers are implemented on a RAID 5 (6+1) configuration - six data disks plus 1 parity disk. Using the example above, prefetches to the four-container tablespace above are performed in 24 extent-sized prefetch requests.

You should monitor the unread_prefetch_pages and prefetch_wait_time monitor element from the snapshot_database monitor to assess the effectiveness of your prefetch parallel I/O settings. The unread_prefetch_pages monitor element tracks the number of prefetch pages that were evicted from the buffer pool before it was used. A continually growing number could indicate that the prefetch requests are too large either because the prefetch size is larger than the pages needed or that the prefetch activities are bringing in too many pages for the capacity of the buffer pool. In either case, you may want to consider reducing the prefetch size.

The application could be waiting for pages if you have high prefetch_wait_time values.

DB2_SELECTIVITY

Enabling this variable allows the selectivity clause to be used in the where clause. Without setting DB2_SELECTIVITY=ALL, Db2only allows the selectivity clause to be used for User Defined Functions (UDFs).

DB2LOCK_TO_RB

This registry variable specifies whether lock timeouts cause an entire transaction or only the current statement to be rolled back. If DB2LOCK_TO_RB is set to STATEMENT, locked timeouts cause only the current statement to be rolled back. Any other setting results in transaction rollback.

Recommended DB CFG Parameters

For performance, we recommend setting the following parameters.

Table 18. db cfg Parameters

db cfg parameters	Value
SELF_TUNING_MEM	ON
DATABASE_MEMORY	AUTOMATIC (for Windows and AIX) COMPUTED (for Linux, HP-UX and Solaris)
LOCKLIST	AUTOMATIC
MAXLOCKS	AUTOMATIC
PCKCACHESZ	AUTOMATIC
SHEAPTHRES_SHR	AUTOMATIC
SORTHEAP	AUTOMATIC
NUM_IOCLEANERS	AUTOMATIC
NUM_IOSERVERS	AUTOMATIC
DFT_PREFETCH_SZ	AUTOMATIC
MAXAPPLS	AUTOMATIC
AVG_APPLS	AUTOMATIC
DBHEAP	5,000 or higher
LOGFILSIZ	262144

Table 18. db cfg Parameters (continued)

db cfg parameters	Value
LOGPRIMARY	more than 15
LOGSECOND	0
NUM_LOG_SPAN	LOGPRIMARY - safety buffer
DFT_DEGREE	1
CUR_COMMIT	ON
STMT_CONC	LITERALS

SELF_TUNING_MEM

Setting this parameter to ON enables the DB2 self-tuning memory manager (STMM) to automatically and dynamically set memory allocations to the memory consumers such as buffer pools, lock lists, package cache and sort heap.

DATABASE_MEMORY

Setting DATABASE_MEMORY to AUTOMATIC (for AIX or Windows) or COMPUTED (for Linux, HP-UX or Solaris) allows DB2 to adjust the amount of database memory depending on load, memory pressures, etc.

LOCKLIST, MAXLOCKS, PCKCACHESZ, SHEAPTHRES_SHR, SORTHEAP

Setting these parameters to AUTOMATIC allows STMM to dynamically manage their memory allocations.

DBHEAP

The default DBHEAP is too small. You should set it anywhere from 5,000 or higher depending on the amount of memory available and the traffic volume.

LOGFILSIZ, LOGPRIMARY, LOGSECOND

You should size your transaction logs based on your transaction volumes. As a starting point, you could configure four transaction logs (LOGPRIMARY=4) of 1GB (LOGFILSIZ=262144 4K-pages) for high transaction volume. You may want to increase this value to 15 or higher.

You should adjust these settings as needed.

You should track the following monitor element to assess the effectiveness of these settings:

- `total_log_used` and `tot_log_used_top` to see how much of the logs are used. You should investigate which workloads are consuming or holding the transaction logs when this value approaches the total primary log capacity. If needed, you may have to adjust the LOGPRIMARY higher.

NUM_LOG_SPAN

Setting this parameter limits the number of logs a transaction can span, which prevents situations where DB2 cannot switch transaction logs because all transaction logs are active. For example:

- Somebody could update a record in Control Center but forget to commit the change.
- There could be software bug that updates one or more database records but not commit the work.

This parameter should be set to at least 3 so that valid long running transactions (e.g., WMS Create Wave agent) are not prematurely forced. This parameter should be set to at most `LOGPRIMARY` minus a safety buffer (e.g., 2). For example, if you have set `LOGPRIMARY=10`, then set `NUM_LOG_SPAN=8`.

DFT_DEGREE

This parameter sets the default degree of parallelism for intra-partition parallelism. In general, online transactional applications such as Sterling Selling and Fulfillment Foundation, typically experiences high volume of short queries that do not benefit from parallel queries. As a result, we recommend setting DFT_DEGREE=1 which disables intra-partition parallelism.

Parallelism can benefit long running, resource-intensive operations such as creating indexes on a large table. To enable parallelism, you need to:

- Enable INTRA_PARALLEL (see "Recommended DB CFG Parameters" on page 63
- Override the default degree of parallelism in the application (connection) prior to performing the operation. For example, issue the following command to set the degree of parallelism to 8:

  ```
  db2 set current degree = '8'
  ```

DB2 Event Monitors

Sterling Selling and Fulfillment Foundation is written to minimize the occurrence of deadlocks. For example, critical database records such as inventory records are always obtained in the same order. However, deadlocks can still happen for many reasons including:

- Custom code may obtain records in a different order.
- DB2 may choose an access plan that retrieves records in a different order.

To help diagnose deadlocks, we recommend setting the following event monitor:

```
MON=<monitor name - e.g., DLMON>
OUTDIR=<directory to store deadlock information>

db2 -v create event monitor $MON for deadlocks with details \
         write to file $OUTDIR buffersize 64 nonblocked
db2 -v set event monitor $MON state = 1
```

When a deadlock occurs, issue the following:

```
db2 flush event monitor $MON
db2evmon -path $OUTDIR
```

The flush ensures deadlock records in the buffers are written out. The db2evmon formats the deadlock information.

CLI Packages

If you configure a service as a string of API calls, all performing under a single transaction commit boundary, that service may fail with a SQL0805N error with a package NULLID.SYSLN203.

This happens when the number of cursors opened by a given SQL statement goes beyond the capacity defined for the small and large CLI packages (which are used by JDBC). In such a case, you should first check to see if there was a cursor leak. If you are certain there isn't a cursor leak, you can then either break up the service into smaller chunks or increase the number of packages bound.

The syntax for adding extra packages is:

db2 bind ../sqllib/bnd/db2clipk.bnd clipkg 10

The number of packages required can differ depending on individual situations. You should test your application under expected peak concurrency levels and transaction rates to ensure you have sufficient packages.

Sterling Selling and Fulfillment Foundation Schema: DB2 Index Monitoring and Tuning

As we mentioned in "Indexes" on page 61, you may have to adjust the base starting index set to suit your operational environment. You can find out what indexes are used (and by corollary, which ones are not used) You can use DB2's Design Advisor to monitor index usage. The Design Advisor recommends additional indexes as well as indexes that are not used.

Index and Table Statistics

The database optimizers rely on up-to-date accurate table and index statistics to generate optimal access plans.

In addition, columns, such as `enterprise_key` in the `yfs_order_header`, can exhibit high skew - for example, there could be many orders for one enterprise and a few orders for another enterprise. Columns such as `derived_from_order_header_key` in the `yfs_order_header` table could have very high skew, which results in low cardinality because they only contain spaces. This can happen when customers have small numbers of derived orders.

Queries, such as the one below, against columns with high skew and/or low cardinality:

```
select *
from yfs_order_header
where derived_from_order_header_key = '2011012412213801928344';
```

can result in table scans even if the columns are indexed. The example above was from an actual case (see below).

From the optimizer's perspective, the queries against these columns either return a small or a very large result set. To err on the side of caution, the optimizer generally chooses a table scan over an index range scan.

You can get the optimizer to choose a more optimal access plan by providing additional statistics in the form of histograms.

Issue the following command to create histograms in DB2:

```
   db2 runstats on table <table name> on key columns with distribution on key
columns and sampled detailed indexes all allow read access
```

Oracle: Implementation

This section assumes that you have read the following Oracle documents that are applicable to your platform:

- Release notes specific to your platform
- Quick Installation Guide for your platform
- Installation Guide for your platform
- Performance Tuning Guide

These documents can be found in the Oracle Technology Network site at http://otn.oracle.com.

This section provides the recommendations that we found useful or critical to the Sterling Selling and Fulfillment Foundation performance.

Recommended Oracle Parameters

The following table summarizes the recommended choices:

Table 19. init.ora Parameters

Parameters	Oracle
db_block_size	8KB
processes	Must be greater than the number of connections needed by the (a) application servers, (b) the agents/monitors and (c) operational management tools.
compatible	10.2.0.1 (or the appropriate Oracle Rel 2 level) 11.2.0.1 (or the appropriate Oracle Rel 1 level)
sga_max_size, sga_target, pga_aggregate_target	1GB to 4GB depending on the amount of physical memory on your database node
cursor_sharing	FORCE
timed_statistics	True
optimizer_mode	ALL_ROWS
open_cursors	Default (higher if prepared statement caching used)
query_rewrite_enabled	True (if using the **Sterling Warehouse Management System** application)
query_rewrite_integrity	Trusted (if using the **Sterling Warehouse Management System** application)
HP-UX Only	
disk_asynch_io	True
memory_target	4GB minimum. It can be increased depending on the amount of physical memory on your database node

processes

This parameter sets the limit on the number of database connections. You have to pick a reasonably high enough number so that the combined connection requirements from the application servers, agents, and so forth do not exceed the connection limit during peak processing periods. If you do, you must restart the Oracle instance to increase this limit.

Fortunately, with the use of connection pooling in application servers, the number of database connections is less than the number of users logged in to Sterling Selling and Fulfillment Foundation. Depending on your anticipated peak workload traffic, this parameter could range from a small number like 25 to a large number in the thousands.

You must regularly monitor the number of concurrent connections in production (and especially during peak periods) to ensure that it does not reach the maximum. When the maximum session is reached, Oracle refuses to establish new connection requests.

See WebLogic connection pooling discussions in “WebLogic Connection Pool” on page 52.

See WebSphere connection pooling discussions in “WebSphere Connection Pool” on page 47.

Guidelines for Estimating Number of Connections.

You can roughly estimate the number of concurrent users required by Sterling Selling and Fulfillment Foundation with the following formula:

$$\text{concurrentOracleConnections} = A + B + C + D$$

where:

- A = Maximum number of agents transaction threads that run concurrently (see “Agent Thread Levels” on page 102.
- B = Maximum application server connection pool size times the number of application server instances. See Maximum Capacity.
- C = Any additional connections that are opened by customized code or user exits that do not go through the application server connection pool. This connection requirement is specific to your implementation.
- D = Number of asynchronous adapters (Service Definition Framework) times the number of connections per adapter.

The Sterling Selling and Fulfillment Foundation agents and monitors are long-running Java applications that open and use one Oracle connection per thread.

Example: Lets assume that you plan to configure a system with the following characteristics:

- Six application server instances where each application server instance can run up to a maximum of twenty-five (25) transactions concurrently.
- Twelve agent threads
- Four asynchronous adapters where each could have up to four threads

- Maximum ten connections for operational tools such as Oracle OEM or Quest SpotLight

Lets further assume that each transaction in the application server only requires one database connection. Specifically, user exits do not open their own database connection. As a result, for the example above, you need:

- In the worst case, 25 x 6 or 150 database connections from the application servers during the peak period if there is a possibility that all application server threads become active. This of course would not be a desirable state - if there ever is such a possibility, you should configure more application server instances.
- 12 database connections for the agents/monitors
- 4 x 4 or 16 database connections from the asynchronous adapters
- 10 database connections from your operational monitors

As a result, you should plan for at least 150 + 12 + 16 + 10 or 188 database connections.

As always, we strongly recommend that you benchmark your system to validate these assumptions and estimates prior to a production implementation. During the test, you should monitor the connection pool usage levels in each of the WebLogic Server instances, the number of agents that you need to run in order to meet your processing and service levels and the actual Oracle connections established.

compatible

You should set the `compatible` parameter to the four level release number that your Oracle software is running at in order to take advantage of the latest optimizer features. An example of the release number is 10.2.0.3 or 11.2.0.1.

sga_max_size, sga_target, pga_aggregate_target

In Oracle, setting `sga_target` allows Automatic Storage Memory Management to manage the memory inside the System Global Area (SGA). You can dynamically change the `sga_target` up to the value specified by `sga_max_size`.

As a result, you could either set `sga_target` to be equal to or less than the value of `sga_max_size`.

cursor_sharing

With cursor_sharing enabled, Oracle converts dynamic (non-reusable) SQL into reusable SQL by changing literal values into bind variables. Enabling cursor sharing significantly reduces shared pool and library cache contention.

Setting cursor_sharing to FORCE will also enable adaptive cursor sharing in Oracle, which allows the optimizer to peek at bind variables and choose optimal execution plans for queries that are bind sensitive.

For optimal performance, you must set `cursor_sharing=FORCE`.

optimizer_mode

Starting in Oracle10g, the optimizer mode of CHOOSE has been deprecated. You should set the `optimizer_mode` to the default of `ALL_ROWS`.

open_cursors

This parameter limits the number of cursors an Oracle session can keep open at any time. Generally, the default is sufficient unless you set a high prepared statement cache size (see Prepared Statement Cache Size in "WebLogic Connection Pool: Define Data Source in Sterling Selling and Fulfillment Foundation" on page 52).

To find out the number of cursors opened by sessions, issue the following query:

```
select a.value, s.username, s.sid, s.serial# from v$sesstat a, v$statname b,
v$session s where a.statistic# = b.statistic# and s.sid=a.sid and b.name =
'opened cursors current';
```

For additional details regarding monitoring open database cursors, please refer to metalink note ID 753605.1.

query_rewrite_enabled and query_rewrite_integrity

If you are using the **Sterling Warehouse Management System** application, you have to create at least one function-based index as part of the application installation. As a result, the Sterling Selling and Fulfillment Foundation schema must have QUERY REWRITE privilege. In addition, in order for Oracle to use the function-based indexes in queries, you have to set the Oracle parameters:

- QUERY_REWRITE_ENABLED to true and
- QUERY_REWRITE_INTEGRITY to trusted.

max_async_ports, disk_asynch_io

Asynchronous I/O is very important to performance especially on high transaction volume processing environments. In summary, processes that issue synchronous `read()` or `write()` I/O calls must wait for the I/O to complete before it can continue. In contrast, processes can issue multiple asynchronous (non-blocking) `aio_read()` or `aio_write()` I/O calls in parallel without waiting.

HP-UX does not enable asynchronous I/O by default. HP-UX also only supports asynchronous I/O on files that reside on raw devices and not on filesystems. If you don't enable asynchronous I/O, you will have to run multiple DBWRs (up to 20) to get a limited amount of I/O parallelism.

Please see:

- "Asynchronous I/O" on page 18 to enable asynchronous I/O on AIX.
- "HP HP-UX11i: Asynchronous I/O" on page 21 to enable asynchronous I/O on HP-UX.

After enabling asynch I/O in HP-UX, you need to set the Oracle parameter, `disk_asynch_io`, to true in spfile.

memory_target introduced in Oracle 11g

In Oracle 11g, the memory_target and memory_max_target parameters specify the amount of memory that the Automatic Memory Management can dynamically allocate to the PGA and SGA. The AMM can reduce or increase the SGA and PGA, up to the memory_target, as needed. You can only change the memory_target value to the value specified in memory_max_target. As a result, you can set the memory_target to be equal or less than the value of memory_max_target.

If the sga_target and pga_aggregate_target is also set, the AMM uses these values as the minimum size for their respective areas. To allow Oracle to take full control of the memory management, these parameters should be set to zero.

Automatic Storage Management (ASM)

We recommend managing your database storage in Oracle's Automatic Storage Management. Some of the benefits of using ASM include:

- Improved I/O performance and scalability
- Simpler database administration tasks
- Automatic I/O tuning
- Reduction in number of objects to manage

We strongly encourage you to read the many ASM whitepapers and documents on Oracle's web-site. In addition, most storage vendors have written best practice papers on how to configure ASM for their storage products.

The following are specific ASM recommendations that we have found to be critical for performance:

- For HP-UX, we recommend importing only raw-devices into ASM. As we discussed in “max_async_ports, disk_asynch_io” on page 70 above, HP-UX only supports asynchronous I/O on files that are on raw devices and not filesystems.
- For high volume processing environments, ensure ASM is configured with "disk" devices from high-performance disk storage arrays. Some characteristics that you should look for include large NVRAM caches to buffer disk reads and writes, efficient RAID implementation, etc.
- Configure ASM with "External Redundancy" so that redundancy is provided by your storage array instead of being implemented by Oracle. This setting will eliminate the extra overhead in Oracle to maintain redundancy.

Redo Logs

Redo logs are critical for database and instance recovery. Proper redo log configuration is also critical for performance. Some recommendations for redo logs configuration include:

- Implement redo logs in Automatic Storage Management (ASM).
- We strongly discourage implementing redo logs on file systems. If you prefer file systems, you should implement redo logs on file systems that are configured to perform I/O in small data blocks to avoid partial block writes.
  Redo log buffers are typically small. If redo logs are implemented on file systems that are configured as 8KB blocks, some redo log writes requires the file system to read in the block, append the log buffer to that block and then write out the block to disk.
  If you are using a file system on Solaris or AIX, the redo log file system should be configured for 512 byte blocks. For HP-UX, the file system block size should be 1024 bytes. See http://www.ixora.com.au/tips/use_raw_log_files.htm.
- If the redo logs are placed on file systems, enable direct I/O - specifically avoid the situation where the writes are buffered in the file system cache before written out to disk.
- Consider implementing redo logs on dedicated disk devices.
- Consider implementing redo log group log files on alternating disks.

Redo File Size

Your choice of redo file size depends on your trade-off between performance and availability, specifically the time needed to recover the Oracle instance in the event of a failure. For performance, some installations opt to create large redo logs to reduce the frequency of log switches. However, doing so means that there are potentially more transactions in the redo logs that have to be replayed during recovery. The general rule for sizing redo log files is to look at the time it takes to switch log files. By issuing the following query you can see how often the redo log files are changing. As a general rule the logs should not be switching more that once every twenty to thirty minutes:

```
select * from v$loghist
order by first_time desc
```

The following is an example of the output:

```
THREAD#   SEQUENCE#   FIRST_CHANGE#             FIRST_TIME   SWITCH_CHANGE#
      1          97         7132082  10/20/2011 11:47:53 PM         7155874
      1          96         7086715  10/20/2011 11:32:04 PM         7132082
      1          95         7043684  10/20/2011 11:15:07 PM         7086715
      1          94         6998984  10/20/2011 11:00:57 PM         7043684
      1          93         6950799  10/20/2011 10:48:03 PM         6998984
```

In the example above, the logs switched every fifteen minutes.

Server Mode

You should create Oracle to use dedicated servers (instead of shared servers). Shared servers can be useful in two-tier client/server configurations where a large number of users need to access the database directly.

In Sterling Selling and Fulfillment Foundation, the WebLogic or WebSphere Application Server serves as a transaction manager to multiplex large number of users into a finite number of connections to the Oracle instance. Long running processes such as Agent Servers, by design, maintain a single, dedicated connection to Oracle. As a result, in both cases, dedicated servers are recommended.

Sterling Selling and Fulfillment Foundation Schema: Oracle Index Monitoring and Tuning

As we mentioned in “Indexes” on page 61, you may have to adjust the base starting index set to suit your operational environment. You can find out what indexes are used (and by corollary, which ones are not used) through index monitoring. To enable index monitoring, issue the following commands, one for each index:

```
...
alter index yfs_order_header_pk monitoring usage;
alter index yfs_order_header_i1 monitoring usage;
alter index yfs_order_header_i2 monitoring usage;
...
```

You can generate the command above by issuing the following query:

```
select 'alter index ' || index_name || ' monitoring usage;'
from user_indexes;
```

Periodically, as you run your functionality and system test, you can run the following query to see if which indexes have been used and which have not yet been used:

```
select index_name, monitoring, used, start_monitoring
from v$object_usage;

INDEX_NAME          MONITORING  USED  START_MONITORING
-------------------  ----------  ----  -------------------
YFS_ORDER_HEADER_I1  YES         YES   01/29/2011 01:23:03
```

To turn off index monitoring, issue the following command:

```
alter index yfs_order_header_i1 nomonitoring usage;
```

Oracle Table Partitioning

You can use Oracle partitioning to aid the maintainability of large tables. You should not view partitioning as a performance tool to achieve higher throughput; under certain circumstances it may increase throughput, but these circumstances are rare in Sterling Selling and Fulfillment FoundationApplication. However, as a tool to improve the maintainability of the largest tables, partitioning can be valuable. Before implementing any partitions in a production environment, it is essential that you test the changes with the expected production workflows. Sterling has tested and developed the following points with regard to Oracle partitioning and the Sterling Selling and Fulfillment Foundation application:

- With the careful selection of tables based on workflow analysis, and using Global Indexes, table partitioning did not cause an appreciable degradation of throughput compared to non-partitioned tables.
- Converting the Global Indexes to Local (non-prefixed) Indexes showed a minimal I/O increase. Application throughput dropped minimally. These tables were accessed only by the purge agents and at low access volumes. General industry consensus is to use Global Indexes for high query volumes, though local indexes on low access volumes may maintain acceptable performance.
- Table partition compression can save up to 85% of the disk space used. We recommend this only on low volume access tables such as the history tables.

Table partitions should ideally be set up on the initial installation. Tables may be partitioned once loaded. We have used and recommend testing Oracle's "*dbms.redefinition*" package. This package is well documented by Oracle in the **Oracle® Database PL/SQL Packages and Types Reference** and the **Oracle® Database Administrator's Guide**.

Oracle does not support LONG columns in table partitions. Any LONG columns need to be converted to CLOB before attempting to partition a table. The LONG to CLOB conversion can be performed by the "dbms.redefinition" package at the same time as the table partitioning.

Oracle Table Partition Compression

As mentioned in “Oracle Table Partitioning,” partition compression can save up to 85% of the disk space usage.

We recommend that only low access volume table partitions are compressed after testing with production-like workflows and loads.

Tablespaces

Prior to production, you should plan the overall storage strategy.

Since there are strong preferences, the DDLs to create temporary tablespaces and data tablespaces are left to the discretion of the customer.

We instead provide the following recommendations for your consideration:

- You should implement tablespaces with automatic space management by specifying segment space management auto.
- With LMTs, you may want to consider creating tablespaces that store small reference tables with the autoallocate extent allocation model:

```
create tablespace yt1
    datafile '/u03/dbs/pyantradb/yt1_001.dbf' size 2047m
    extent management local autoallocate
    segment space management auto;
```

- If you have very large tables, you may want to consider putting those tables into their own tablespace and to use the uniform extent allocation model:

```
create tablespace yfs_order_header_t1
    datafile '/u03/dbs/pyantradb/y_order_header_t1_001.dbf' size 2002m
    extent management local uniform size 1000m
    segment space management auto;
```

- You should create your temporary tablespace as a temporary data file (temp files). Temp files are used to store intermediate results (e.g., from large sort operations). Changes to temp files are not recorded in the redo logs:

```
create temporary tablespace yfs_temp
    tempfile '/u03/dbs/pyantradb/yfs_temp_01.dbf' size 1024m
    extent management local
    uniform size 1m;
```

Tables

After creating the tablespaces, you can create a set of DDL scripts using the dbverify.sh script and the -applyddl parameter set to false, as follows:

```
dbverify.sh (or .cmd) -applyddl false
```

This creates all the necessary CREATE TABLE and CREATE INDEX statements. You may then edit these scripts to map to your tablespaces.

Index and Table Statistics

Database optimizers rely on "relatively" up-to-date table and index statistics to generate optimal access plans. Oracle does not need the statistics to be absolutely correct or current, just relatively correct and representative. As a result, you don't have to gather statistics every day for every table especially if your database is already large (in the terabyte range).

Starting in Oracle10g, Oracle by default automatically gathered statistics during its maintenance window for tables that underwent sufficient changes. Oracle bypasses statistics generation for tables that have not changed significantly.

Volatile Tables

The following tables change significantly during the day and are not candidates for automatic statistics gathering:

- YFS_TASK_Q
- YFS_TASK
- YFS_EXPORT
- YFS_IMPORT

For example, the YFS_TASK_Q table represents task that are in different state of processing. That table grows and shrinks throughout the day. At night, when order processing has completed, the table will have few in-progress records. When automatic statistics gathering run during the maintenance window, the statistics will incorrectly present this table as a small table.

We recommend either one of the two options below for these tables:

- Delete statistics for these table and then lock down the statistics.
- Manually collect statistics during the day when the table is large and then lock down the statistics.

In the first option, in the absence of statistics, Oracle will assume a large table. The commands for the first options are:

```
exec dbms_stats.delete_table_stats(<schema owner>,'YFS_TASK_Q')

exec dbms_stats.lock_table_stats(<schema owner>,'YFS_TASK_Q')
```

The commands for the second option are:

```
exec dbms_stats.gather_table_stats (ownname => 'YANTRA', -
tabname=>'YFS_TASK_Q', -
estimate_percent => dbms_stats.auto_sample_size)

exec dbms_stats.lock_table_stats(<schema owner>,'YFS_TASK_Q')
```

Skewed Columns and Histograms

As part of generating the statistics, Oracle generates histograms for skewed columns.

Skewed columns are columns that have a non-uniform distribution of values. For example, the enterprise_key column in the YFS_ORDER_HEADER table may be made up of a few values where one value may be more prevalent. In contrast, columns such as the order_no is more uniformly distributed.

Given basic statistics such as number of rows and the number of distinct column values, Oracle tends to choose a full table scan when faced with a query, such as the one below, against columns with high skew and/or low cardinality:

```
select *
from yfs_order_header
where derived_from_order_header_key = '2011012412213801928344';
```

can result in table scans even if the columns are indexed. The example above was from an actual case (see below).

Note: Customer reported that the Order Detail screen took over 4 minutes. The query that checks if the order is a derived order resulted in a table scan of the YFS_ORDER_HEADER table. When customer ran dbms_stats to create histograms, Order Detail screen dropped to 1 second.

From the optimizer's perspective, the queries against these columns either return a small or a very large result set. To err on the side of caution, the optimizer generally chooses a table scan over an index range scan.

You can get the optimizer to choose a more optimal access plan by providing additional statistics in the form of histograms.

By default, Oracle creates histograms as part of the statistics generation. You can verify if a column has histograms by issuing the following command:

```
   select table_name,column_name,histogram
   from user_tab_columns

   TABLE_NAME      COLUMN_NAME                     HISTOGRAM
   YFS_ORDER_LINE  CHAINED_FROM_ORDER_LINE_KEY     NONE
   YFS_ORDER_LINE  CHAINED_FROM_ORDER_HEADER_KEY   NONE
   YFS_ORDER_LINE  DERIVED_FROM_ORDER_LINE_KEY     FREQUENCY
   YFS_ORDER_LINE  DERIVED_FROM_ORDER_HEADER_KEY   FREQUENCY
```

In the example above, Oracle created histograms for the two DERIVED_FROM columns but not the CHAINED_FROM columns. To manually create the histograms for the CHAINED_FROM columns, issue the following command:

```
   exec dbms_stats.gather_table_stats (ownname => 'YANTRA', -
   tabname=>'YFS_ORDER_LINE', -
   estimate_percent => dbms_stats.auto_sample_size, -
   method_opt=>'for columns size auto CHAINED_FROM_ORDER_LINE_KEY,
CHAINED_FROM_ORDER_HEADER_KEY');
```

When you rerun the histogram query, you should now get:

```
   TABLE_NAME      COLUMN_NAME                     HISTOGRAM
   YFS_ORDER_LINE  CHAINED_FROM_ORDER_LINE_KEY     FREQUENCY
   YFS_ORDER_LINE  CHAINED_FROM_ORDER_HEADER_KEY   FREQUENCY
   YFS_ORDER_LINE  DERIVED_FROM_ORDER_LINE_KEY     FREQUENCY
   YFS_ORDER_LINE  DERIVED_FROM_ORDER_HEADER_KEY   FREQUENCY
```

In the example above, the `method_opt` with the `auto` parameter lets Oracle decide whether histograms are to be created based on the column's data distribution and the way the columns are being used by the application.

Identifying Skewed Columns

The following query helps you identify columns with skewed data distribution:

```
   select ui.table_name,ui.index_name, column_name, column_position, num_rows,
distinct_keys as dist_keys
   from user_indexes ui, user_ind_columns uic
   where ui.table_name = uic.table_name and
       ui.index_name = uic.index_name and
       ui.num_rows > 0 and
       ui.distinct_keys/ui.num_rows < 0.1
   order by table_name, index_name, column_position

TABLE_NAME      INDEX_NAME         COLUMN_NAME                    NUM_      DIST
                                                                  ROWS     _KEYS
YFS_ORDER_LINE  YFS_ORDER_LINE_I3  CHAINED_FROM_ORDER_HEADER_KEY  6552586     1
YFS_ORDER_LINE  YFS_ORDER_LINE_I4  DERIVED_FROM_ORDER_HEADER_KEY  6357590     1
YFS_ORDER_LINE  YFS_ORDER_LINE_I5  DERIVED_FROM_ORDER_LINE_KEY    6624191     1
YFS_ORDER_LINE  YFS_ORDER_LINE_I6  CHAINED_FROM_ORDER_LINE_KEY    6534969     1
YFS_ORDER_LINE  YFS_ORDER_LINE_I7  DEPENDENT_ON_LINE_KEY          6457481     1
```

In the example above, the customer does not use derived or chained orders.

You should ensure that histograms are added to indexed columns if the absence of histograms causes Oracle to choose an inefficient plan.

Microsoft SQL Server

Microsoft SQL Server Implementation: Parameters

Set the following collation setting to support case-insensitive sorts needed by Sterling Selling and Fulfillment Foundation.

Table 20. Microsoft SQL Server Installation Decision Points

Description	Recommendation
Collation Settings	SQL Collation: Dictionary order, case-insensitive, for use with any Character Set

Microsoft SQL Server Index Monitoring and Tuning

In Microsoft SQL Server, you can use the Index Tuning Wizard to recommend new indexes as well as indexes that are not used.

Statistics

By default, Microsoft SQL Server automatically creates statistics on indexed fields when the index is created. If deemed beneficial, Microsoft SQL Server also creates statistics on non-indexed fields that are used in joins or filter criteria. The information on the column's data distribution could help the Query Optimizer choose the optimum execution plan. Although this feature can be disabled, we recommend leaving it on. To check if automatic statistics creation is enabled, issue the following query:

```
select databasepropertyex('ywinss01','IsAutoCreateStatistics')
```

The result is 1 if enabled and 0 if disabled.

Microsoft SQL Server also automatically manages the statistics based on the amount of changes to the table. When the number of changes exceed a threshold, Microsoft SQL Server automatically generates new statistics. Although this feature can be disabled, we recommend leaving it on. To check if automatic statistics update is enabled, issue the following query:

select databasepropertyex('ywinss01','IsAutoUpdateStatistics')

The result is 1 if enabled and 0 if disabled.

Applications Manager: The Applications Manager is a Java applet that is used to configure the Sterling Selling and Fulfillment Foundation rules. You may have to start the applet with more memory if you are modifying a large or complex configuration. You can change the memory settings in the JVM/JRE plug-in control panel in Microsoft Windows. Go to Start > Control Panel > Java Plug-In > Advanced. In the Java Runtime Properties, put in "-mx356m". In this example, the JRE's heap is allowed to grow to 356MB.

DB2 PureScale

DB2 pureScale is a powerful database technology that allows customers to cluster multiple DB2 members together for performance, scalability, and availability.

This topic provides DB2 pureScale tuning or configuration information that is specific or applicable to the Sterling Selling and Fulfillment Foundation. For general installation and tuning recommendations, refer to IBM DB2 pureScale documentation, training, and support.

DB2 pureScale: Supported DB Platforms

The Sterling Selling and Fulfillment Foundation, Release 9.1 is certified with DB2 pureScale on, AIX 6 running on IBM Power 7 processors.

DB2 pureScale: Support Limitations

The Sterling Selling and Fulfillment Foundation is limited to 4-member DB2 pureScale configurations.

To seek any clarification, you can contact Technical Support.

Sequences

Sterling Selling and Fulfillment Foundation uses the `create sequence <sequence-name>cycle cache 10000 no order` sequence number to ensure that the generated primary key values are unique.

The cache option tells DB2 to pre-allocate a set of sequences in memory for better?? performance. With a cache of 10000, the first request fetches the first sequence number from the disk. The next 9999 requests are satisfied in memory. Without the cache option, all requests are goes to disk.

Default NO ORDER option, allows each DB2 pureScale member to cache their own set of sequence numbers.

High Availability

For detailed instructions, refer the *Sterling Selling and Fulfillment Foundation: High Availability Guide*.

Implicit Rollback

For implicit rollback, set the jdbcService.db2Pool.prop_implicitRollbackOption=2.

Oracle Real Application Cluster Database

This section provides limited guidelines on implementing, configuring and tuning Oracle Real Application Cluster (RAC). RAC is a powerful technology offered by Oracle. This section presents information that is specific or applicable to Sterling Selling and Fulfillment Foundation. Installation and tuning RAC in general is beyond the scope of this document.

Consult Oracle documentation, technical support, and training for all questions pertaining to RAC and the associated technologies such as HP ServiceGuard that are integral to the RAC solution.

Oracle RAC is a technology that allows you to cluster multiple Oracle instances to acts as one Oracle instance.

Supported DB Platforms

The Sterling Selling and Fulfillment Foundation Release 9.1 has been tested with Oracle RAC on the following database server platforms:

- Red Hat Enterprise Linux 5.5 Server running on Intel Xeon EM64T/AMD64 (64-bit) processors.

Supported Filesystems

The Oracle RAC database data files can be implemented on clustered filesystems or raw devices managed in Oracle Automatic Storage Management (ASM).

The filesystem is an important operating system component that is critical for both performance and data integrity. IBM requires that Sterling Selling and Fulfillment Foundation be configured and deployed only on filesystems that are approved and certified for use with RAC by the Oracle Corporation.

Sterling Selling and Fulfillment Foundation has been tested with Oracle RAC running with:

- Raw devices
- Oracle Cluster File System (running on Red Hat Enterprise Linux)

Sterling Selling and Fulfillment Foundation does not support systems that run on non-Oracle RAC-supported filesystems such as the Sistina GFS.

Support Limitations

There are practical limits to any technology. One should not expect every technology to scale infinitely. At this time, Sterling Selling and Fulfillment Foundation supports up to 4-node RAC configurations.

Sterling Selling and Fulfillment Foundation has, till date, been either tested or deployed on:

- 2-node (HP RP7410 eight-way processors each for a total of 16 processors)
- 2-node (HP Itanium2-based servers)
- 4-node (DELL PowerEdge M600 two quad core processors each for a total of 32 processors)

Call Technical Support if you have questions.

OLTP Applications and Oracle RAC Concerns

For Online Transaction Processing (OLTP) applications, including Sterling Selling and Fulfillment Foundation, one common concern is high insertion rates and the effect on index maintenance. In high volume OLTP applications, index leaf blocks have to be maintained and passed among the multiple nodes to keep them all in sync. Generally, when new records, like orders, are being indexed, they are being written to the right most part of the index. In very high transaction volumes, concurrent insertions could wait while a similar request is handled by a different node. The index leaf block for the right most part of the index cannot be released to another node until the request is completed. This forces more sequenced rather than simultaneous processing and is likely to drag significantly on performance.

Another example is the frequency with which inventory records are being accessed and updated.

Research suggests that other OLTP app vendors are generally aware of these issues — some only certifying for a maximum number of nodes and other articles suggesting optimal node / CPU configurations for Oracle RAC.

Some industry literature suggest using hash partition or reverse indexes to reduce or eliminate contention to enable OLTP applications for RAC. What isn't stated is that these techniques can negatively affect application performance which could slow down query response times. At this point of time, Sterling Selling and Fulfillment Foundation has not been tested for, and nor do we support the use of reverse indexes or hash partitions for Oracle RAC enablement. Check with Technical Support for the latest information.

Database Sequence Numbers

Sterling Selling and Fulfillment Foundation uses Oracle sequence numbers to quickly generate unique numbers. If you are upgrading from pre Yantra 5x 5.0 SP2 versions, ensure the `seq_yfs_task_key` sequence is created with the NOORDER parameter. If the ORDER option is enabled, RAC disables the CACHE option.

The SQL command to create sequence is as follows:

```
create sequence seq_yfs_table_key
increment by 1 start with 1
maxvalue 9999999999
minvalue 1
cycle
cache 500 noorder ;
```

In the example above, the CACHE option pre-allocates and stores 500 sequence numbers in the instance's SGA for fast access. When those sequence numbers are used up, Oracle pre-allocates another group of sequence numbers. The CACHE option should be set to a value so that sequence requests for one to two seconds during the peak period can be satisfied if memory is critical for performance (see *Oracle 11g SQL Reference* and *Oracle 11g Real Application Cluster (RAC) Administration*.)

For example, if the sequence cache is set to 500, the last_sequence_number in user_sequences table should not grow by more than 500 every two seconds or 30,000 every minute. You should monitor this value periodically during the peak hour.

The NOORDER option allows each RAC instance to preallocate its own group of sequence numbers. The NOORDER option is enabled by default. If the NOORDER option is disabled (or if the ORDER option is selected), Oracle disables the CACHE option.

Enabling the CACHE option with a sufficiently high value and the NOORDER option are critical for Oracle RAC performance.

You can issue the following command to check whether the ORDER option is disabled:

```
select sequence_name, order_flag, cache_size,last_number
from user_sequences
where sequence_name = 'SEQ_YFS_TABLE_KEY'
```

If the ORDER_FLAG is set to "N", the NOORDER option is enabled:

```
SEQUENCE_NAME                  ORDER_FLAG CACHE_SIZE LAST_NUMBER
------------------------------ ---------- ---------- -----------
SEQ_YFS_TABLE_KEY                       N        500   422838694
```

High Availability

Refer to the *Sterling Selling and Fulfillment Foundation: High Availability Guide* for more detailed instructions.

From a performance perspective, you need to configure the Sterling Selling and Fulfillment Foundation system so that it can quickly discover the Oracle failure and to quickly recover the connections.

The Sterling Selling and Fulfillment Foundation system is made up of client programs that connect to the Oracle instance. These include:

- Oracle WebLogic, IBM WebSphere, or JBoss AS application servers

- Sterling Selling and Fulfillment Foundation agents or monitors

WebLogic Connection Pool Properties

In WebLogic, we recommend setting the following Connection Pool properties so that WebLogic can detect stale or dead connections faster.

Table 21. WebLogic Connection Pool Properties

Parameter	Value
Test Frequency	120
Test Table Name	SQL SELECT 1 FROM DUAL
Test Reserved Connections	Enable
Initial Capacity	3
Maximum Capacity	See “WebLogic Connection Pool: Define Data Source in Sterling Selling and Fulfillment Foundation” on page 52
Shrink Frequency	Leave at the default of 900 seconds

With the settings above, the WebLogic connection pool manager tests idle connections every `Test Frequency` seconds by issuing a Select statement to `Test Table Name`. Connections that do not pass the test are closed and a new connection reestablished. This setting helps the connection pool manager to get rid of dead or stale connections.

Additionally, when you set `Test Connections On Reserve` to true, the connection pool manager tests connections before the pool manager gives the connection to transactions. This test adds a small delay to each connection request.

You must set `Test Table Name`. The settings above are invalid without the `Test Table Name` setting.

The `Initial Capacity` and `Maximum Capacity` settings should be set to your operational requirements (see “WebLogic Connection Pool: Define Data Source in Sterling Selling and Fulfillment Foundation” on page 52). You should not set `Initial Capacity` to zero - when WebLogic shrinks the connection pool (at every ShrinkPeriodMinutes minutes), it aggressively shrinks all currently unused connections, even connections that were recently active.

You may not want to set `Test Created Connections` and `Test Released Connection` especially if you already have enabled `Test Connections On Reserve`. The probability that a connection has died after it was created or after it was released should be very low.

Using a SCAN address in a datasource is not supported in WebLogic server prior to 10.3.4. You must configure a MultiDataSource for your RAC database pool in Weblogic Server 10.3.2.

TCP/IP

The default time for a connection request to an unavailable node to timeout is deliberately set to a high value. This value allows connection requests (e.g., telnet connections) the opportunity to find the node on the Internet. This setting is less applicable in a high-speed switched network.

On Solaris, a telnet to a nonexistent node takes about 2.75 minutes to timeout. On HPUX11, the timeout is around 75 seconds.

The connection timeout value can be tuned down by issuing the following ndd commands:

```
ndd -set /dev/tcp tcp_ip_abort_cinterval 1000
ndd -set /dev/tcp tcp_rexmit_interval_initial 200
ndd -set /dev/tcp tcp_rexmit_interval_max 5000
```

This known phenomenon is described in the SunSolve article found in Document 01363, *How to reduce the time-out period for telnet connections* (http://sunsolve.sun.com/search/document.do?assetkey=1-30-01363-1.) The settings are applicable to both Solaris and HP-UX.

Fast Application Notification Support

Fast Application Notification (FAN) provides RAC the ability to inform the client programs the status of the cluster. With FAN, the client programs, especially those with connection pools, can drop stale connections to failed nodes.

Sterling Selling and Fulfillment Foundation does not support Fast Application Notification.

Sterling Selling and Fulfillment Foundation Specific Tuning Using Key Randomizer

The Sterling Selling and Fulfillment Foundation applications, by default, generates primary key with one of the following formats:

- YYYYMMDDHHmmSSSSSSqqqqqqqq (non multi-schema)
- ccYYMMDDHHmmSSSSSSqqqqqqqq (multi-schema)

Where;

- YYYY or YY is the four or two digit year
- cc represents a two digit schema id (for multi-schema)
- MM is the two digit month
- DD is the two digit day of the month
- HH is the two digit hour (24 hour)
- mm is the two digit minute of the hour
- SSSSSS is the seconds and milliseconds
- qqqqqqqq is a sequence number

Although, the key format is easy to use, the fact that the key values are only ever increasing can be a performance bottleneck. As key values are added to the primary key index, the database stores the value in the right sequential order. Since, the key values are ever increasing the index values will be added to the high end of the index, which is also commonly known as the right hand block. At very high transactional rates, that index block can become a bottleneck as transactions contend to insert values into that block.

If you expect sufficiently high transaction volumes, you can use the Sterling key randomizer to reduce the amount of traffic between node members. To do this, add the following attribute (keyType) to the `entity.xml` and regenerate the entities. For example, to add random digits to the primary key for the YFS_ORDER_RELEASE_STATUS table, add the

```
<Entity
Description="This table stores status-quantity break up for
  each line-schedule." EntityType="TRANSACTION" HasHistory="Y"
  Module="omp" KeyType="Random" Name="Order_Release_Status"
  Prefix="YFS_" TableName="YFS_ORDER_RELEASE_STATUS"
  TableType="TRANSACTION" GenerateIndexedListMethods="true"
  XMLName="OrderStatus">
<Attributes>
<Attribute ColumnName="ORDER_RELEASE_STATUS_KEY"
      DataType="Key" DefaultValue="' ' "
      Description="The primary key for the
        YFS_ORDER_RELEASE_STATUS table."
      Name="Order_Release_Status_Key" Nullable="false"
      XMLName="OrderReleaseStatusKey"/>
,Attribute ColumnName="ORDER_RELEASE_KEY"
      DataType="Key"
      Description="The order release that is associated with this
        order release status record."
      Name="Order_Release_Key" Nullable="true"
      XMLName="OrderReleaseKey"/>
...
</Attributes>
</Entity>
```

After extending the appropriate entity XML files, you can rebuild your `entities.jar` by running the `deployer.sh`. For more information about how to run the `deployer.sh`, see the *Sterling Selling and Fulfillment Foundation: Customization Basics* Guide.

Currently, you can randomize the primary keys for the following tables:

- yfs_order_release_status
- yfs_order_line_schedule

Chapter 7. Java Message Services

Java Message Services Overview

This chapter provides guidelines on implementing, configuring and tuning for Java Message Services (JMS) in general.

Sterling Selling and Fulfillment Foundation uses JMS extensively. For example:

- The Sterling Selling and Fulfillment Foundation agents use JMS as a source of work.
- The Sterling Selling and Fulfillment Foundation integration servers use JMS as a means to communicate with external systems.

Agent Queues

The Sterling Selling and Fulfillment Foundation agents are designed to issue a "getter" to look for work that needs to be processed and to create the appropriate messages into a queue. For example, the Schedule agent on startup checks the Schedule JMS queue. If that queue is empty, it automatically fires a "getter" query against the YFS_TASK_Q table looking for tasks that need to be processed by the Schedule transaction. A JMS message is created for each eligible task. Similarly, the Sterling Selling and Fulfillment Foundation order or inventory monitors fire "getters" to look for orders or inventory items in a particular state (for which they are being monitored for). As above, the appropriate messages are put into the JMS queue.

By default, the getter creates up to 5,000 messages even when there are more eligible work. The default is generally sufficient. You can change the limit if you find that the agent is spending more time retrieving work and creating the messages than in processing. This could happen if you have a high number of processing threads or if the retrieval cost is high. You can change the limit by changing the "Number of Records to Buffer" in the agent's Transaction Detail (in IBM Sterling Application Platform > Process Modeling) > Agent Criteria Definition > Agent Criteria Details > Criteria Parameter. See the *Sterling Selling and Fulfillment Foundation: Application Platform Configuration Guide* for more information.

Integration Queues

In contrast, integration queues are used for external communication. For example, one could architect the system where multiple sales channels capture orders. These orders are passed to Sterling Selling and Fulfillment Foundation through an integration queue.

Similarly, Sterling Selling and Fulfillment Foundation can pass messages to external systems when transactions are processed.

Persistence

You can define queues as being persistent or nonpersistent. Messages in nonpersistent queues are lost after the queue is restarted. For example, if you have 100 messages in the queue, all those messages are lost when the WebLogic JMS server or the WebSphere MQ queue manager is restarted. In contrast, messages in persistent queues are preserved after a restart. Using the same example from above, the same 100 messages are in the queue after a restart.

In general, the following recommendations apply:

- All queues used by the Sterling Selling and Fulfillment Foundation agent should be defined as nonpersistent. As we described above, the agents can easily recreate the messages if lost.
- All integration queues used for external communications, either for messages coming from external systems to Sterling Selling and Fulfillment Foundation or for messages going from Sterling Selling and Fulfillment Foundation to external systems, must be defined as persistent. In most cases, recreating integration messages can be difficult especially when the information in two or more systems have to be re-synchronized.

Dedicated Queues

We strongly recommend you define a dedicated queue for each agent and service that uses JMS for work because of:

- Performance
- Monitoring

For both the WebLogic JMS and IBM WebSphere MQ, the cost of pulling up a message is proportional to the number of messages the JMS server or queue managers have to interrogate.

In the current WebLogic JMS implementation, a request for a message with a certain selector results in a sequential search through the JMS queue until a message with the specified selector is found. The JMS manager could use a lot of CPU searching for messages if there are lots of messages in the queue. Putting high volume messages into a separate JMS destination eliminates the search - the JMS manager either finds that there are no messages in that destination or it finds the message immediately.

Similarly, in the current IBM MQSeries® JMS implementation, the consumer (client) uses the supplied mq.jar to connect to the MQSeries queue manager. When the client asks for a message, the client code in com.ibm.mq.jar retrieves messages from the queue and checks whether the message has the specific selector. The mq.jar continues to do this until it has found the appropriate message or there are no more messages in the queue. When there are no more messages, the mq.jar sleeps for 5 seconds and repeats the polling cycle. Putting messages into its own JMS destination means that the mq.jar either finds the message immediately or sleep for 5 seconds.

In some extreme cases, the performance and cost is very noticeable. Take the case of a queue with messages for multiple agents and 100,000 integration messages. When a message for the Schedule transaction is created, that message is added after the existing 100,000 messages. When the Schedule transaction getter runs, the getter needs to walk through the entire queue looking for Schedule messages.

An exception to the above is development and possibly test environments. In those cases, to ease configuration and management overhead, it may be acceptable to put all the JMS destinations into a single JMS queue.

JMS Queue File Placement: Performance

The WebSphere MQ logs and files and the WebLogic JMS file and paging stores can be implemented on an internal disk. Message queues on a single internal disk should be able to provide from 150K to 200K messages per hour. Obviously, many factors can affect the message throughput including the size of the message content, the burstiness of the traffic, and so forth).

For high transaction systems, for example, a nightly upload of global inventory visibility messages or the import of point-of-sales orders, you should consider placing the WebSphere MQ logs and files and the Oracle WebLogic JMS file and paging stores on a SAN RAID-10 LUN, possibly with a large NVRAM cache. The striping component in the RAID-10 spreads the message I/Os over multiple disks. The NVRAM cache could reduce the number of physical disk I/Os.

In extremely high transaction volume scenarios, you may have to consider implementing multiple WebLogic JMS servers or MQ queue managers. This is applicable to solutions where the message order is not important.

Availability

For failover and high availability, you should consider placing the WebSphere MQ logs and files and the WebLogic JMS file stores for persistent queues on an external SAN. In the event of a node failure, a standby node could attach to the SAN to access the files. In addition, you could replicate the content of the SAN to prevent message loss in the event of a data center disaster. See the *Sterling Selling and Fulfillment Foundation: High Availability Guide* for more information.

Parameters

Please see the following chapters for specific recommendations:

"WebLogic JMS Recommendations"

"WebSphere MQ Parameters" on page 88

WebLogic JMS Recommendations

This section provides guidelines on implementing, configuring and tuning the WebLogic JMS.

Dedicated JMS Server

You should consider running the JMS server on one or more dedicated WebLogic servers that is outside the Sterling Selling and Fulfillment Foundation WebLogic cluster. These server instances should only provide JMS services. The benefits of isolating the JMS server on its own server include:

- Easier to monitor and manage
- Easier to diagnose issues - Issues that arise, such as OutOfMemory exceptions, must be related to JMS services or JMS messages.

Integration Queues

In addition, you should consider putting integration queues into their own dedicated WebLogic JMS servers running on separate JVMs especially if these queues can grow unbounded or at a fast rate.

These integration queues should be configured as persistent so that messages can be recovered after JMS failures. Recovering integration messages can be difficult especially if they involve reconciling when there are many systems or applications involved in processing the messages.

You should consider implementing controls so that producers cannot significantly create messages faster than consumers can process messages. In extreme cases, high number of messages in the queue could consume most of the JMS servers's JVM heap resulting in degraded or loss of service.

The benefits of implementing dedicated JMS servers for integration queues include:

- Isolating integration-based message queues that could grow unbounded from the more predicable queues used by the Sterling Selling and Fulfillment Foundation agents.
- The ability to configure, manage, and monitor the queues to the expected message traffic - for example, you may want to create JVMs with 1GB heap for integration-based JMS servers and smaller heaps for the Sterling Selling and Fulfillment Foundation agents.

WebLogic JMS: Message and Byte Paging

For WebLogic JMS, you should enable message and/or byte paging on JMS queues that could grow unbounded (for example, integration-based queues. With this facility, the message bodies (not the message headers) are paged out of the JVM memory on to the local file system when the paging thresholds are exceeded. This can reduce the amount of JVM heap space used, which could prevent service degradation or loss.

Note: In extreme cases, excessively high number of message headers can still lead to outOfMemory exceptions.

WebSphere MQ Parameters

This section provides specific guidelines on implementing, configuring and tuning WebSphere MQ.

Depending on your processing volumes and the number of MQ queue consumers and producers you expect to start, you may have to change the log and channel parameters in the qm.ini or mqs.ini file.

Channel

Each thread started that reads from or writes to the MQ queues requires a channel. If you were to start 20 JVMs with 5 threads each, you need at least 100 channels (which is the default). You may also have to increase the number of channels if you have workloads that open and close the JMS connections rapidly.

If you experience messages indicating that the max channels have been reached, do the following:

- Check to see if there is a connection or channel leak. Run the following command to see how many active channels are used:

  ```
  echo "dis chs(*)" | runmqsc | grep RUNNING | wc -l
  ```
- You may have to run each workload at peak production loads in your test environment to diagnose channel leaks.

- If you suspect that channels are not getting reclaimed fast enough or if your TCP/IP connection is not reliable, you should set the following parameters. The KeepAlive parameter tells the queue manager to check the existence of the client. If the client is not there, the queue manager reclaims the channel. The MaxChannels defaults to 100. In production settings, that parameter could grow to a much higher number like 300 or 500:

```
   TCP:
      KeepAlive=YES

Channels:
      MaxChannels=300
      MaxActiveChannels=100
```

Log Files

MQ uses log files to maintain message integrity in the event of a queue manager restart or a media failure.

The number of log files depends on your configuration, the size of the messages, the logging type, and the message volumes. You should performance test your application at or above peak production loads to see if the default MQ log settings are sufficient. If you are using CIRCULAR logging, the following may be reasonable starting values:

```
Log:
   LogPrimaryFiles=4
   LogSecondaryFiles=2
   LogFilePages=65535
   LogType=CIRCULAR
   LogBufferPages=0
```

If you use LINEAR logging (for example, to be able to survive media failure), you must set LogPrimaryFiles higher.

Placement of MQ Log and Data Files

If your system has to be able to process a high message rate, you may should consider placing your MQ log and data files on a fast SAN, preferably configured with a large NVRAM and RAID-10. A single internal disk should have sufficient capacity to allow up to 150K to 200K messages per second. Files on a RAID-10 LUN should be able to get up to around 1.5M to 2.0M messages per hour. Beyond that message rate, you may want to consider implementing multiple queue managers with separate data and log files.

Chapter 8. Selling and Fulfillment Foundation

General Recommendations

This chapter provides recommendations on how to plan, implement, configure, monitor, and tune Sterling Selling and Fulfillment Foundation in general, that transcends both the Sterling Distributed Order Management and the Sterling Warehouse Management System applications.

Sterling Selling and Fulfillment Foundation with the default factory (data) settings provides a simple configuration that is suitable for development, training, or product familiarization. That configuration is not suitable for production except for customer with very low transaction volumes. This chapter guides you through the components that you have to configure for higher transaction volumes.

This chapter assumes that you:

- Are familiar with the installation of Sterling Selling and Fulfillment Foundation and have read the *Sterling Selling and Fulfillment Foundation: Installation Guide*
- Are familiar with the basic functionality of Sterling Selling and Fulfillment Foundation
- Have read the *Sterling Selling and Fulfillment Foundation: Release Notes -- Known Issues*

Scalability Requirements

An important aspect of planning the implementation and configuration of your Sterling Selling and Fulfillment Foundation system for production is determining your workload and business processing characteristics, and your performance requirements. This includes (at a minimum) the following:

- Identifying the key or high transaction volume use case scenarios - for example, in retail environments, you may have an order capture, order returns, order modification and order authorization use case scenarios. For each use case scenario, you should determine:
  - The workloads (both custom-developed and IBM-supplied) that are carried out
  - The forecasted peak transaction volumes
  - When the peak processing periods occur during the year
  - The external systems the Sterling Selling and Fulfillment Foundation system is integrated with

When choosing use case scenarios, you should include:

- Workloads with anticipated high transaction volumes
- Workloads that are complex (for example, orders with large number of order lines)
- Workloads/transactions that have to traverse long network distances (for example, user and data center in different continents)
- High volume transactions that are integrated with external systems

For each use case scenario, you should:

- Perform load testing to at least the anticipated peak workload volumes.

- Measure the computing resource cost at different workload traffic volumes.
- Estimate the computing cost per unit work.
- Identify and tune expensive workloads - This could include ensuring all SQL are supported by appropriate indexes ("Custom Indexes" on page 61), custom code, and so forth.
- Incorporate the cost per unit work into a resource capacity forecasting or planning model.
- Project out the resource requirements for the peak periods.

If you have remote users, you have to test use case scenarios that involve screens or network based transactions across a real or simulated wide-area network. These include:

- The use of the Sterling Selling and Fulfillment Foundation screens (for example, to enter or modify orders)
- RF transactions (for example, users at a warehouse in Asia interacting with the WMS application in North America)

The answers to the questions above are critical to how you configure Sterling Selling and Fulfillment Foundation.

System Test

We strongly advise that you schedule time and resources to test Sterling Selling and Fulfillment Foundation system (including all custom code, integrated external systems, and so forth) prior to implementation. IBM tests Sterling Selling and Fulfillment Foundation to common or general usage patterns. Your configuration may differ greatly:

- Custom code - Need to ensure your custom code scales and does not have longevity issues. These are issues that show up after running the system for many days - for example, memory or connection leaks.
- Integration to external systems - Need to ensure that external systems can scale along with Sterling Selling and Fulfillment Foundation. In the right conditions, slow external systems could tie up the Sterling Selling and Fulfillment Foundation resource and could lead to a system slow down.
- Configuration - Need to test the Sterling Selling and Fulfillment Foundation system with representative data. For example, your configuration may have much larger catalogs and ship nodes than most customers.
- User locations - Need to ensure users get responsive service. For example, you may have large customer groups located in a different continent from Sterling Selling and Fulfillment Foundation. You may also have customers who dial in to access Sterling Selling and Fulfillment Foundation. You need to ensure that all users get appropriate screen response times.

User Interfaces

Out of the box, Sterling Selling and Fulfillment Foundation provides the following graphical interfaces to allow users to interact with the application:

- Application Console - a thin HTML-based client to view and manage orders
- Applications Manager and System Management Console - a thick client built on Java Swing to manage the application configuration
- Rich Client Platform - this is a Java/SWT thick client built on the Eclipse RCP framework

The user interface provides the means for users to interact with Sterling Selling and Fulfillment Foundation Applicationto view, create, modify and delete information.

Application Console

An Application Console is an HTML-based interface. Microsoft Internet Explorer is used to render the screens of the Application Console.

Customization: Sterling Selling and Fulfillment Foundation allows you to create or customize the screens of the Application Console. You may want to do so for the following reasons:

- You want to reduce processing or the screen size - From the usage scenario studies above, you may find that your users need a subset of a *detail* screen (e.g., order detail). Further, you expect a very large number of users to be located at a remote call center in a different continent. To reduce server processing and the amount of bytes send across the network, you can create a new screen that only has the information needed.
- In conjunction with simplifying the screens, you can also customize an APIs output XML using templates. This not only reduces the number of bytes returned but can also reduce server and database processing. See "API Output XML Files" on page 111.
- You may want to control the types of searches that the general users can issue. The default search facility allows users to build up searches by picking different criteria. For example, the order search allows users to look for orders based on many criteria including status, enterprise code, and so forth. Some permutation of criteria can result in queries that require a lot of resources. You can create a search screen for general use that has a list of searches that you and the DBA have tested and have deemed to be "safe" for general use. You can further develop a screen with greater search capabilities for supervisors or application administrators. This search screen could mandate entering the enterprise code and a date range for the search to limit the number of records returned. Additionally, you may want to remove certain searches such as looking for orders in a particular status.

See the *Sterling Selling and Fulfillment Foundation: Customizing the Swing Interface* for more information on how to extend or customize screens.

HTML Compression: You should consider HTML compression if you have users who are connected to Sterling Selling and Fulfillment Foundation over high latency or low bandwidth network links. HTML compression can reduce the size of the HTML pages by up to 85%.

Currently, the Sterling Selling and Fulfillment Foundation Console UIs have been tested with F5 Big-IP v9 as an off-board HTML compression engine.

If you were to use a Big-IP, you should be aware of the following Big-IP specific configuration requirements in the HTTP Profile configuration tab:

- You need to set the **response chunking** parameter to **rechunk**. The default is **preserve**. The reason is that Sterling Selling and Fulfillment Foundation does not set the content length in the HTTP headers when sending out the response. With the default setting of preserve, the Big-IP does not attempt to compress HTML pages that do not have content lengths set. With the **rechunk** setting, Big-IP compresses the response chunks as they are received. More importantly, the Big-IP can forward along the compressed chunks without waiting for the entire HTML page to be compressed.

- Big-IP allows you to specify the amount of compression processing that it attempts. The setting can range from Level 1 which tries a minimal compression in favor of processing speed to Level 9 which tries to find the most compression. We found that Level 1 compression was able to get up to 85% compression and that the benefits from Level 9 undetectable. As a result, we defaulted to testing with Level 1 compression.
- We set the compression buffer size to 128KB instead of the default of 4KB. The general thought is that the buffer size should be able to store the entire compressed response in order to set the content header length. In our testing, we didn't see any appreciable differences between 4KB and 128KB. This may be due to the fact that we had already set response chunking to rechunk - as a result, the Big-IP does not have to set the content length on the compressed response. However, we were advised to set the buffer size to at least 128KB.

We recognize that there are other HTML compression technologies available including Apache deflate module and Juniper DX application acceleration devices. Please keep in mind that we have, to date, only tested against the F5 Big-IP v9.

Please also keep in mind that the compression is only certified for the Application Console. The Applications Manager and the System Management Console do not support compression. The nWMS radio frequency and VT220 terminal screens are small and should not require compression.

Temporary Internet Files: You can reduce the number of hits against the application servers for static content by enabling temporary Internet file cache in Microsoft Internet Explorer. This improves your UI response times. To enable the cache:

- Go to the Internet Options dialog box.
- In Microsoft Internet Explorer, go to Tools > Internet Option:
  - Click on the Settings button in the Temporary Internet Files panel.
  - Enable the “Check for newer version of stored pages” radio button to Automatically.
  - Make sure there is sufficient disk space to store temporary Internet files (e.g., 500MB or higher).

SSL Acceleration: If you have many users and are planning on encrypting the Sterling Selling and Fulfillment Foundation screens with SSL, you should consider the use of off-board hardware-based SSL accelerators. SSL encryption/decryption are expensive operations and can reduce application server throughput by over 30%.

Currently, we have tested the use of an F5 Big-IP v9 as an off-board SSL acceleration engine and as an SSL Proxy. As an SSL Proxy, all page requests going to the F5 are sent as HTTPS. The Big-IP performs all the SSL processing and forwards all the requests to the applications servers in the "clear".

If you plan to use a Big-IP, you should be aware of the following Big-IP specific configuration requirements in the HTTP Profile configuration tab:

- If you use WebLogic application servers, you have to set the **header insert** parameter to **WL-Proxy-SSL: true**. This header directive informs WebLogic that there is an SSL Proxy sitting in front of the application server.
- If you use WebSphere application servers, you need to configure the **redirect rewrite** parameter to **ALL**.

We recognize that there are other SSL acceleration technologies available. Please keep in mind that we have, to date, only tested against the F5 Big-IP v9.

Also keep in mind that SSL acceleration is targeted towards the Sterling Selling and Fulfillment Foundation Console. The Applications Manager and the System Management Console do not require SSL.

Search Screens: Sterling Selling and Fulfillment Foundation provides a flexible search facility that allows users to look for orders, shipments and audit records with a wide range of criteria. Some search combinations are more expensive than others.

We recommend that you work with the user community to identify search combinations that are likely to be used in production. Each search combination should be tested to ensure they are optimized and acceptable in a production setting. When testing these searches, you need to make sure the tables searched are sufficiently large (e.g., over 1 million records). Inefficient queries may not be evident in small databases. In addition, ensure the tables are populated with an appropriate data mix. For example, if the query is looking for orders with certain attributes in the closed state, you should ensure that these attributes and the number of closed orders are representative. Database optimizers picks search paths that it believe are optimal for the data distribution.

It is likely that some search combinations require indexes to be created (see "Indexes" on page 61).

Case-insensitive Search

The Sterling Selling and Fulfillment Foundation Search feature supports case-insensitive searches against the YFS_PERSON_INFO table on the following columns:

- FIRST_NAME
- LAST_NAME
- EMAILID
- ADDRESS_LINE1
- ADDRESS_LINE2
- CITY
- STATE
- ZIP_CODE
- COUNTRY

The data continues to be stored in the database in mixed-case (mixture of upper and lower case).

Oracle

To support case-insensitive searches in Oracle, you must add function-based indexes on the searched columns. To create a function-based index that supports case-insensitive searches on the emailid column, issue the following:

```
create index yfs_person_info_cust1 on yfs_person_info(upper(emailid))
```

Note: Creating an index does not result in case-insensitive searches. The index only helps in the speedy retrieval of records when a query is fired on email id. For information about enabling case-insensitive searches, refer to the *Sterling Selling and Fulfillment Foundation: Extending the Database*.

DB2

For DB2, you have to add a generated column for each searched column and an index on that generated column. For example, as in the example above, you need to perform the following:

```
set integrity for yfs_person_info off

alter table yfs_person_info
   add column emailid_up generated always as (upper(emailid))

set integrity for yfs_person_info
   immediate checked force generated

create index extn_per_info_i1
   on yfs_person_info(emailid_up)

select *
from yfs_person_info
where upper(emailid) = 'SMITH'
```

In the example above, a generated column (emailid_up) was defined as a generated column and indexed.

Microsoft SQL Server

In Microsoft SQL Server, searches are case-insensitive by default so there are no changes needed.

JSP Precompilation: Precompiling the JSPs when you "build" the application is very important for user interface response times. If the JSPs are not precompiled, the application servers compile the JSP on-the-fly the first time it is used. These compiles can take up to 30 seconds or more and could lead to the perceptions of a badly performing system.

Please see “WebLogic Tuning: JSP Pre-Compilation” on page 53 on precompiling JSPs in WebLogic application servers.

Please see “WebSphere Tuning: JSP Pre-Compilation” on page 49 on precompiling JSPs in WebSphere application servers.

HTML Limitations: The view screens (for example, the Order Detail and Shipment Detail screens) present the order or shipment entity and related records in their entirety. The HTML page for large orders or shipments can be very large and could take a long time to display especially over a wide area network. For example, the view screen for a 200 line order could be up to 500KB. Displaying this screen over a 128kbps line could take 30 seconds or more.

If your enterprise regularly process large orders, you may want to consider the following:

- As we mentioned above, consider customizing the screens to only return the data needed, implementing HTML/HTTP compression, using output templates to reduce the size of the output, and so forth.

Applications Manager

The Applications Manager is a Java applet that is used to configure the Sterling Selling and Fulfillment Foundation rules. You may have to start the applet with more memory if you are modifying a large or complex configuration. You can change the memory settings in the JVM/JRE plug-in control panel in Microsoft Windows. Go to Start > Control Panel > Java Plug-In > Advanced. In the Java Runtime Properties, put in "-mx356m". In this example, the JRE's heap is allowed to grow to 356MB.

Rich Client Program Interface

The Rich Client Platform is a Java/SWT thick client built on the Eclipse RCP framework. A number of Sterling Selling and Fulfillment Foundation Packaged Composite Application (PCAs), such as IBM Sterling Call Center and IBM Sterling Store and Store Operations, use this graphical interface.

Enabling Content Compression

The Rich Client Platform client supports content compression for both the request and the response to and from the application server. The compression will reduce the application bytes by around 85%.

You can enable compression in the `locations.ycfg` parameter file. See the *Sterling Selling and Fulfillment Foundation: Installation Guide* for more information on how to configure the locations.ycfg file.

You may want to consider developing multiple locations.ycfg files with different settings. For example, you can define a `locations.ycfg` file for local users without content compression and a `locations.ycfg` file for remote users with compression enabled. This gives you the flexibility to deploy the appropriate `locations.ycfg` file to different user groups.

In the following example, the `REMOTE` location has compression enabled by setting the `CompressionEnabled` attribute to Y:

```
   <Locations>
     <Location id = "DEFAULT" proxyServer="yourproxyserver.com"
       proxyPort="8080" updateType ="pull">
       <Config Name = "DEFAULT" Protocol = "https" BaseUrl = "localhost"
         PortNumber = "7001" ApiUrl
="/smcfs<application_name>/RcpServlet"
         CompressionEnabled = "N"
       </Config>
     </Location>
     <Location id = "REMOTE" proxyServer="yourproxyserver.com"
       proxyPort="8080" updateType ="client">
       <Config Name = "DEFAULT" Protocol = "https"
         BaseUrl = "localhost" PortNumber = "7001"
         ApiUrl ="/smcfs<application_name>/RcpServlet"
         CompressionEnabled = "Y"
       </Config>
     </Location>
   </Locations>
```

Images

To improve the performance of the screens, the RCP client has the ability to retrieve and display images in a separate background thread. This can be beneficial when you are displaying large orders. For example, when you display the order detail screen in COM PCA and you display small images of the item at the order line level, the main thread will paint the order detail screen and its content. A separate thread will paint the icons.

To improve the performance further, the images are cached on your local drive after they have been retrieved. The cache could eliminate requests back to the application server for images. The cache is deleted when you restart the RCP client to ensure that you have the latest images.

For performance, you might want to consider the following:

- You can specify an image server that is separate from the application server. For example, in the following example, the images come from http://yantraimg.acme.com. This will steer static image retrieval to specialized image servers:

```
<Locations>
  ...
  <Location id = "REMOTE" proxyServer="yourproxyserver.com"
    proxyPort="8080" updateType ="client">
    <Config Name = "DEFAULT" Protocol = "https"
      BaseUrl = "localhost" PortNumber = "7001"
      ApiUrl ="/smcfs<application_name>/RcpServlet"
      CompressionEnabled = "Y"
    </Config>
    <Config Name = "IMAGE" Protocol = "http"
      BaseUrl = "yantraimg.acme.com" PortNumber = "7001"
      ApiUrl="/smcfs<application_name>/icons/rcp/$param1$.gif"
      CompressionEnabled = "N"
    <Config Name = "IMAGE_SMALL" Protocol = "http"
      BaseUrl = "yantraimg.acme.com" PortNumber = "7001"
      ApiUrl="/smcfs<application_name>/icons/rcp/$param1$_small.gif"
      DefaultApiUrl="/smcfs<application_name>/icons/rcp/404.gif"
      CompressionEnabled = "N"
    </Config>
    ...
  </Location>
</Locations>
```

- You can disable images for users on limited or slow networks by removing the IMAGE, IMAGE_SMALL and IMAGE_BIG Config elements from your `locations.ycfg` file.
- You can specify the same icon for items or groups of items. Subsequent request for this icon will be served from cache.

Guidelines for Processing Large Orders

This section describes the best practices and system design considerations that should be taken into account when you have to process orders or shipments that have a large number of order lines.

That number varies depending on many factors, such as order complexity. As a general guideline, we recommend that you consult Sterling Support if you have to process orders that have over 200 lines. These could, for example, be large, planned purchase orders in B2B associations.

Best Practices

Customers have used numerous strategies to handle large orders. Some of these strategies are described here for your consideration. Their applicability will, however, depend on your business requirements:

- Build a custom screen to view large orders.

You can develop a custom, HTML-based user interface to display the order in smaller sections.

For example, you can create a custom screen that calls the getOrderDetails API with a selective output template to display only the order header information (and not the order line information). This will significantly reduce the amount of data that is to be retrieved from the database, and the size of information to be displayed on the HTML screen. This HTML screen can have a button to allow the user to get a list of the order lines. When the user clicks this button, the getOrderLineList API is called to get a list of order lines. Again, through the use of the output template, you can control the amount of data retrieved and displayed. Finally, when the user clicks a specific line, the getOrderLineDetails API is called to get all the information relating to that line.

There are many variations to this technique. The actual implementation is dependent on your use case scenarios.

- Break up large orders into smaller, but manageable orders.

  Some customers have opted to break their large orders into smaller chunks for manageability. One approach is to develop a front end process to split the orders based on business rules, prior to the orders entering the Sterling Distributed Order Management application. The procurement analyst can use the Master Order field, for example, to find all the related orders.

- Decouple heavy processing from the user interaction.

  Decoupling long-running processes from user interaction can result in a faster response.

  For example, let us assume that you have to receive a large shipment of serialized items on a single pallet. This could be, for example, a pallet of 10,000 gift cards, each having its own serial number. As part of the receipt process, each item is assigned to one serial number.

  In this case, you may want to consider implementing custom screens to allow users to view the shipment, serial numbers, and a button to trigger the receipt and serial number registration in the background. This can be achieved by having a button, which, when clicked, will send a message to a background agent to process the receipt and serial numbers.

  Thus, the user will be able to continue working when the receipt and serial number processing is being performed in the background.

- Test the application with large orders.

  Ensure that you include the largest anticipated orders into your testing. Plan the test with orders larger than your largest orders. For example, if you expect 1,000-line orders, test with orders containing 1,200 lines.

Other Architectural Considerations

The system design or architectural approaches described in this section typically yield sub-optimal performance when working with large orders:

- Using the HTML-based Order Detail screen to display a large order.

  As described earlier, the standard, product Order Detail screen retrieves both the order header and order line information. This approach is not optimum for large orders because the screen has to retrieve and display all the details of the order.

- Displaying large amount of data across slow networks.

  Customers who are connected to low-bandwidth connections (for example, 56 kbps modems) or high-latency connections (for example, connected through satellite links) should impose a constraint on the amount of data that is to be displayed.

A general industry rule is to keep the actual network payload to 50 KB. Customers who have to send a lot of data should compress the output stream. Devices such as F5 Big-IP load balancers have built-in compression that can reduce the payload by 80%. With compression, you can generate screens that are up to 250 KB in size because the resulting compressed screen will be about 50 KB.

In practice, you should, as mentioned earlier, understand your use case scenarios. Displaying the data in smaller chunks may be more appropriate because it is unlikely that you will need all the data from all the lines.

- Test with representative conditions.

  Your testing environment should reflect the actual production conditions. For example, as mentioned earlier, if users are located at remote locations on high-latency networks with low bandwidth connections, you should not test the user interfaces on fast, local area networks. You should, at a minimum, run tests from the actual remote locations. If this is not feasible, you should consider simulated WANs. For example, you can run your tests through Shunra devices to assess the impact of slower networks. The URL for Shunra Software Ltd. is given below:

  http://www.shunra.com

  Similarly, you should test the application with representative, production database sizes.

- Performing too much processing per order line.

  When retrieving and processing the list of order lines, be aware of the processing required for each order line. Although the processing required may be low for each line, the processing can be significant when there are many lines.

  For example, assume that you want to calculate the cost of the items associated with each line. Further, assume that you need to call an external system to get the price, and the pricing call takes 200 milliseconds. If you have to display a list with 600 lines, the pricing calls could take 120 seconds. In some cases, it may be better to call the pricing engine once with 600 items, especially if the cost of the call represents the majority of the time. Alternatively, you can consider providing a button to allow the user to reprice the lines if repricing is needed.

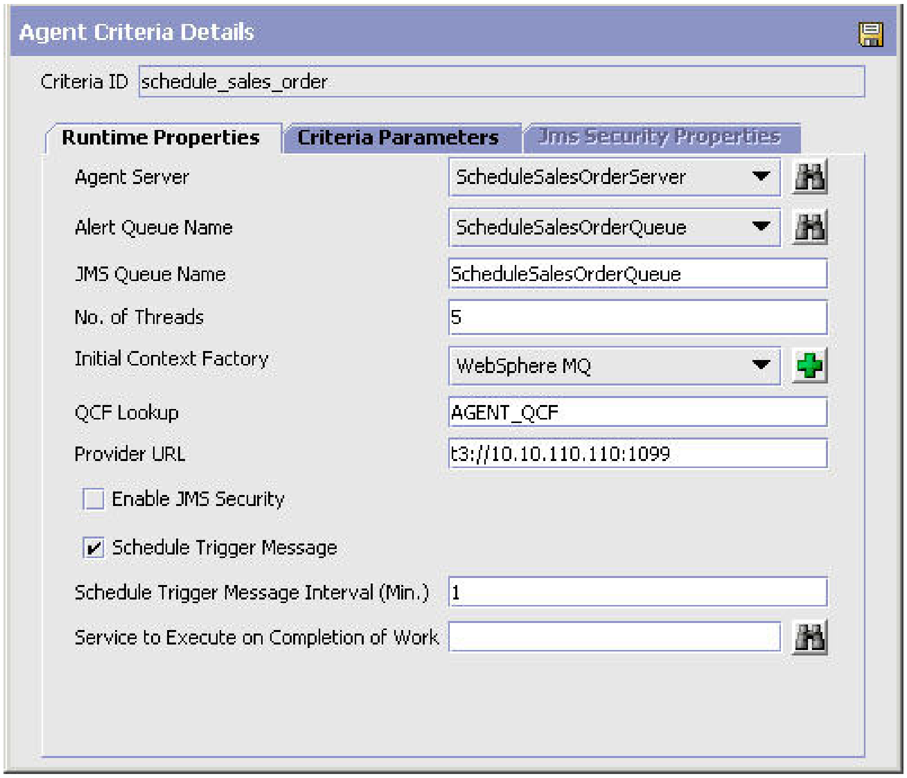
Integration Adapters/Agents

Agent Servers or Integration Adapters are Java applications that run time-triggered (agent) transactions (see the *Sterling Selling and Fulfillment Foundation: Distributed Order Management Configuration Guide*). Transactions process orders or shipments such as moving orders from one state to another.

Out of the box, all time-triggered (agent) transactions are configured to run in a single Agent Server (called the DefaultAgent). This simple setup is convenient for training, development or product demos. This setup is not suitable for production because all the processing threads will run in one JVM.

Agent Criteria

The Applications Manager allows you to configure your transactions. See the *Sterling Selling and Fulfillment Foundation: Application Platform Configuration Guide* for detailed instructions on how to configure the Agent Criteria.



Criteria ID: schedule_sales_order

Agent Server	ScheduleSalesOrderServer
Alert Queue Name	ScheduleSalesOrderQueue
JMS Queue Name	ScheduleSalesOrderQueue
No. of Threads	5
Initial Context Factory	WebSphere MQ
QCF Lookup	AGENT_QCF
Provider URL	t3://10.10.110.110:1099
□ Enable JMS Security	
☑ Schedule Trigger Message	
Schedule Trigger Message Interval (Min.)	1
Service to Execute on Completion of Work	

The example above defines the agent criteria for the Schedule time-triggered (agent) transaction. The Schedule transactions run in an Agent Server which we have called the `ScheduleSalesOrderServer` server. When you start an Agent Server with the Criteria ID of `schedule_sales_order`, that server is instructed to run five threads of the Schedule transaction. If you need more processing threads, you can either increase the number of threads or run more instances of this Agent Server. Please see “Agent Thread Levels” on page 102 for recommendations on how to estimate your threading levels.

All time-triggered (agent) transactions are driven by tasks in their message queue. A queue may serve one or more transactions. We strongly recommend you configure one queue for each high volume transaction.

Agent Getters

Work tasks are placed into the messaging queue by a *getter*. A getter’s job is to put qualified orders (in this example, orders that are in the Scheduled state) into the `ScheduleSalesOrderQueue` queue. You can specify how many orders the getter picks up each time it runs. By default, the getter picks up 5,000 orders.

Like the time-triggered (agent) transactions, a getter is also driven by work tasks in the queue - in this case, by a *getter work task* instead of a transaction work task. The getter work task is created by a trigger server.

You should consider the following recommendations when configuring the Agent Servers:

- Agent thread levels
- Getters that can accept enterprise code as an additional parameter
- For JMS Servers:
  - Dedicated JMS Servers
  - Excessive agent scheduling

  - Dedicated JMS Destinations
  - Running JMS servers in client VM mode
  - Enabling message and byte paging

Agent Thread Levels

You should derive the optimum number of Agent Servers to run and the number of transaction threads for each Agent Server. The Agent Server's throughput depends on many factors such as the amount of customization or user exits, the amount of data contention, the size and capacity of the agent servers, and so forth.

One approach you can use to derive your agent's effective throughput is:

- Allow work to queue up. Make sure there are at least one to two hours worth of work queued up.
- Run a single transaction thread and record the total (running) elapse time.
- Determine the total amount of work performed by the transaction thread for sample monitoring scripts).
- Calculate the effective throughput of that agent thread by dividing total amount of work by elapse time. The throughput rate is specified in terms of work per unit time (e.g., order lines per hour or order lines per minute).

During the test, you should make sure there are no significant system bottlenecks impeding the Agent Server's performance. Some of the performance indicators you should watch for include:

- Excessive JVM garbage collection activities (especially Full GCs)
- Excessive database waits (e.g., I/O, latches, and so forth)
- Inefficient queries (e.g., missing indexes)
- Data lock contention
- Excessive thread synchronization

Make sure the Agent Server is running optimally before calculating its potential throughput rate.

You can schedule multiple agent threads if your average processing level is greater than the effective throughput for a single agent thread. For the reasons mentioned above, more threads (beyond a reasonable level) does not always mean higher throughput.

Excessive Agent Scheduling

You should not over-aggressively schedule the time-triggered (agent) transactions - for example, configuring a time-triggered transaction to run on many Agent Servers with high threading levels when you expect to a low traffic volume for that transaction. If you schedule the agents too aggressively, you could end up with a situation where the agents (consumers) are outpacing the producers. As a result, the queue typically has a few transactions which are quickly processed. When processed, the Agent Server schedules another getter -- the frequent getter tasks could cause unnecessary overheads as it looks for work to do.

In this case, "more does not necessarily mean more".

Java Message Service

Sterling Selling and Fulfillment Foundation uses JMS extensively. For example:

- The Sterling Selling and Fulfillment Foundation agents use JMS as a source of work.
- The Sterling Selling and Fulfillment Foundation integration servers use JMS as a means to communicate with external systems.

Integration Queues

Integration-based queues are queues for inbound external messages (such as orders from external partners or inventory adjustments from external warehouse management systems) or outbound external messages (such as alert messages to an e-mail system).

You should consider putting these queues into one or more dedicated JMS servers especially if these queues can grow unbounded. In addition, these JMS destinations should be configured as persistent so that messages can be recovered after JMS failures.

You should consider implementing controls so that producers cannot significantly create messages faster than consumers can process messages. In extreme cases, high number of messages in the queue could consume most of the JMS servers's JVM heap resulting in degraded or loss of service.

The benefits of implementing dedicated JMS servers for integration queues include:

- Isolating integration-based message queues that could grow unbounded from the more predicable queues used by the Sterling Selling and Fulfillment Foundation agents
- The ability to configure, manage and monitor the queues to the expected message traffic - for example, you may want to create JVMs with 1GB heap for integration-based JMS servers and smaller heaps for the Sterling Selling and Fulfillment Foundation agents

Dedicated JMS Destination

You should configure a dedicated JMS Destination for each time-triggered (agent) transaction for the following reasons:

- Ease of monitoring - With dedicated destinations, it is easier to see the number of messages coming into a destination, the number of messages that require processing, the maximum number of messages that ever existed in that destination. With that information, you can also calculate the messaging inflow and outflow rates.
- Performance - With dedicated JMS destinations, the selector is able to quickly find the message with the specified selector/filter.

In a common JMS destination with lots of messages (e.g., greater than 20K messages), the selector could take several seconds to find the appropriate message.

JMS Persistence

Many of the Sterling Selling and Fulfillment Foundation agents find work to process from message queues. These work requests are kept in nonpersistent message queues. These messages are recreated, either when an external or internal agent trigger is issued.

Integration messages (e.g., createOrder messages from external systems) must be kept in persistent message queues. JMS reads the messages back into memory from the persistent store when the JMS server is restarted.

You should implement persistent JMS queues on a RAID-10 or RAID-5 disk array for performance and availability. These RAID disk arrays, especially for RAID-5, should be supported by a non-volatile cache to ensure fast I/O write operations. For high persistent message volumes, local disk queues can become an I/O bottleneck.

Performance Feature - Reference Data Caching

Reference Data Caching is critical for performance and scalability. From experience, UI login time could jump to 30 seconds if the YFS_RESOURCE and YFS_RESOURCE_PERMISSION tables are not cached correctly. The application's overall throughput will drop significantly if caching is not enabled.

Reference Data Caching Overview

So, what is reference data caching? Simplistically, when a transaction issues a database SELECT and returns ten records, the ten records are cached in the JVM. We will go into further details below.

Starting in Yantra 5x 5.0 SP2, caching is enabled by default. The cached records are stored in the JVM heap. Typically, around 350MB will be used by caching during steady-state. Of course, the actual amount could depend on your operations. As a result, with caching enabled, you should monitor the health of the JVM heap garbage collections. For memory constrained environments, you may want to enable caching on specific tables.

Cache Management

The Sterling Selling and Fulfillment Foundation reference data caching is implemented by a *local*, *simple*, *lazy-loading*, *asynchronous-refresh* cache manager. The cache manager is a *lazy-loader* in the sense that it does not read in the cacheable reference tables at start up but would instead only cache records as they are being read. The benefit of the lazy-loading strategy is that data is only cached where they are needed.

The cache manager implements a *simple* cache management policy. Data that is cached remains in the cache until the cache manager is instructed to flush the cache. This could happen because the cache has reached a certain size limit or a reference data record was changed from a standard Sterling Selling and Fulfillment Foundation API. The cache manager does not implement cache management policies, such as record flushing using a least recently used algorithm, in order to avoid cache management overheads. In our controlled test, this *simple* cache manager provides significant performance benefits with little management overhead.

In keeping with the simple cache strategy, when a reference data record is changed by a Sterling Selling and Fulfillment Foundation API, the local cache manager notifies all the other cache managers to flush the reference data table. There is a small time-lag between when the reference data is changed to when the last cache manager is notified.

When the cache managers receive the change notification, the cache managers flushes all the cached entries for the affected table. As a result, you should cache tables that are infrequently changed. More importantly, this notification comes from the Sterling Selling and Fulfillment Foundation APIs. As a result, you should ensure that reference data is never changed via database tools like SQL*Plus.

Recommendations:

- You should enable reference data caching when you need the extra performance boost.
- You should ensure that the reference data is not subject to frequent updates.

Caching Strategies

As we stated above, with caching, you introduce the possibility of data consistency issues. This data inconsistency may occur when an API changes a reference data record in one JVM while another transaction is using another copy of that reference data in another JVM.

That said, caching is a widely used technique that favors scalability, performance and affordability against possibly maintainability, data consistency, and accuracy.

In this section, we describe strategies you can use to mitigate the data consistency issues.

Strategy 1 - Trade-off Performance and Affordability against Data Consistency

In this strategy, you may ask yourself the question. First, does the possibility of data consistency exist? Since the refreshes are done asynchronously, the answer is yes. The next question is, what is the probability of a data consistency? One of the factors that this answer depends on is the transaction volume. There may be more. For example, if you were to make the reference data changes at night when transaction volumes are low, you may decide that the probability of data consistency is potentially low. The last question you need to ask is, what is the impact of an inconsistent data? If you determine that the impact is insignificant, then you may decide to go with this strategy. The decision is yours to make.

Strategy 2 - Trade-off Performance and Affordability against Maintainability while keeping Data Consistency

In this strategy, you control updates against the cacheable reference data to eliminate any possibility of data consistency. One approach is to place the cacheable reference tables into a separate tablespace.

In addition, with Oracle, using the following command, you alter the tablespace to only allow reads:

```
alter tablespace <tablespace name> read only;
```

Oracle ensures that these tables are not modified without your knowledge. To modify the cached reference data, you then alter the tablespace back to read/write and modify the reference data through the Applications Manager. To be safe, you would probably do this when there is very little transactional activity on the system. When you are done, you can then mark that tablespace as read only with the following command:

```
alter tablespace <tablespace name> read write;
```

Automatically Refreshing Data Cache

When a record of a cached table is modified by a Sterling Selling and Fulfillment Foundation API, the local cache manager sends change notification messages to all the other cache managers in the Sterling Selling and Fulfillment Foundation system. These messages are sent sequentially - going from one cache manager to the next. The time to notify all cache managers is dependent on the number of cache managers - the more managers, the longer the notification process.

Manually Refreshing Data Cache

You can use the System Management Console to manually refresh the Sterling Selling and Fulfillment Foundation cache for an application server. Go to the Details page for each application server or the Sterling Selling and Fulfillment Foundation agent instance and press the "Clear Cache" icon. The cache for the application server is cleared.

List of Cache Managers

The list of cache managers are dynamically maintained in the YFS_HEARTBEAT database table. Sterling Selling and Fulfillment Foundation servers, integration servers or agents automatically register themselves into this table when they start and deregister themselves when they stop. In addition, they also update their status in the YFS_HEARTBEAT table on a regular basis. At any time, the heartbeat table has a record for every running Sterling Selling and Fulfillment Foundation server instances and integration server/agents.

The "Cache Clear Count" column in the System Management Console > Table Level Cache List screen provides statistics on the number of times the cache was cleared at the table level.

Cleaning Up the Cache Managers List

A JVM may not be able to deregister its entry from the YFS_HEARTBEAT table if it died abruptly. This could lead to stale entries that point to non-existent JVMs. You can clean up these stale entries by running the Health Monitor agent (see *Sterling Selling and Fulfillment Foundation: System Management and Administration Guide*) for more detail.

Enabling Reference Data Caching

By default, Sterling Selling and Fulfillment Foundation enables reference data caching for:

- Application server instances
- Sterling Selling and Fulfillment Foundation agents and monitors

The caching feature and the tables that are cached are governed by the `dbclassCache.properties` file that is located in the `<INSTALL_DIR>/properties` directory. You can change the cache settings by adding override parameters into the `customer_overrides.properties` file. For example, you can disable caching for the YFS_ACTION table by adding the following line to the `customer_overrides.properties` file:

```
YFS_ACTION.enabled=false
```

For additional information about overriding properties using the `customer_overrides.properties file`, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*.

Currently, there are about 140 reference tables that are cacheable.

The System Management Console allows you to confirm that tables are cached.

Controlling the Size of the Cache

When a transaction issues a SELECT against a cacheable table, the cache manager saves the retrieved records as well as the SELECT WHERE clause. The WHERE-clause is used as a hash key to quickly determine the existence of cached records.

The cache manager stores four distinct components:

- The cached record (which the cache manager calls "OBJECT")
- The query WHERE clauses that returned one or zero database record (which the cache manager refers to as "SELECT")
- The query WHERE clauses that returned zero or more database records (which the cache manager calls "LIST") and
- The results of COUNT queries and their WHERE clause (which the cache manager calls "COUNT").

  Note: In case you are wondering, the use of the terms "SELECT", "LIST" and "COUNT" is historical and refers to the fact that the WHERE clauses were used by the selectWithWhere(), listWithWhere and the countWithWhere() database methods. Knowing these terms will help you set the cache limits later in this document.

Take for example the following queries against the cacheable YFS_ORGANIZATION table. The first query (using the selectWithWhere() method) returns one record (call this record ORG-3):

```
select *
from yfs_organization
where organization_code = 'ORG-3'
```

At the end of the query, the cache manager stores the ORG-3 record into a Java Map which the cache manager refers to as OBJECT. Next, the cache manager stores the WHERE clause ("`where organization_code = 'ORG-3'`") in the SELECT Java Map (see diagram below). The SELECT Map associates the WHERE clause (which is a hash key) to the cached record.

The second query (using the listWithWhere() method) to the same table returns six records (ORG-1 and ORG-6):

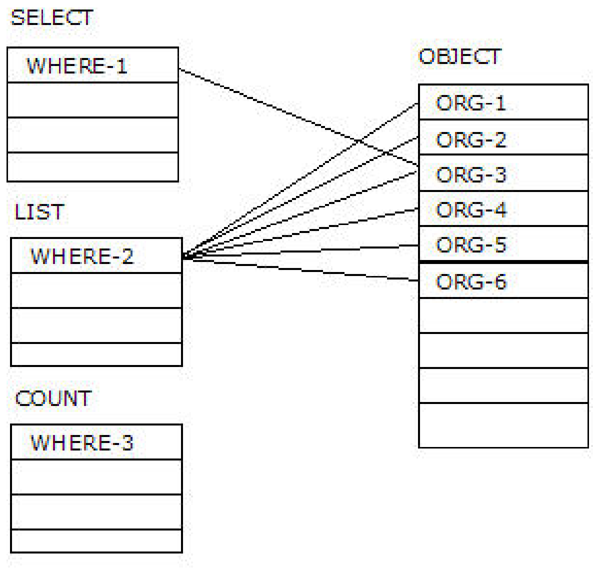
```
select *
from yfs_organization
where catalog_organization_code = 'ACME'
```

At the end of that query, the cache manager stores the second WHERE clause into the LIST Java Map and the ORG-1, ORG-2, ORG-4, ORG-5 and ORG-6 records into OBJECT along with a structure that associates the six cached record to the WHERE clause. The cache manager does not add ORG-3 because it was already added to the OBJECT Map from the first query.

Finally, the WHERE clause and the count results from the third query (using the countWithWhere() method) is stored in the COUNT Map:

```
select count(*)
from yfs_organization
where catalog_organization_code = 'ACME'
```

The following diagram depicts a simplified version of the cache structure:



By default, the cache manager uses the following parameters from the `dbclassCache.properties` file to control how many COUNT results, SELECT WHERE clause, LIST WHERE clauses and OBJECT (cached records) can be stored for each table:

```
sci.globalcache.count.size=10000
sci.globalcache.select.size=10000
sci.globalcache.list.size=10000
sci.globalcache.object.size=10000
```

As a result, a cacheable table can at most store in the JVM:

- Up to 10,000 cached records (in the OBJECT Map)
- Up to 10,000 SELECT WHERE clauses
- Up to 10,000 LIST WHERE clauses and
- Up to 10,000 COUNT results and their WHERE clauses.

You can use the `customer_overrides.properties` file to override the settings. For example, you can increase the cache limit for the number of YFS_RESOURCE records to 30,000 with the following parameter:

```
YFS_RESOURCE.objects=30000
```

For additional information about overriding properties using the `customer_overrides.properties file`, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*.

The "Objects Cached" column in the System Management Console > Table Level Cache List provides the number of records cached for each table.

Please keep the following in mind if you change the default settings:

- Each cached record occupies space in the JVM heap. If you increase the number of records cached, you must ensure garbage collections are still effective and "healthy".
- Conversely, do not set the cache limit too low such that the Sterling Selling and Fulfillment Foundation cache has to continually flush the cached tables

The UI login process takes over 30 seconds if you set the cache limit for the YFS_RESOURCE and YFS_RESOURCE_PERMISSION tables too low (e.g., 1,000). These two tables have over 3 thousand records which are read as part of the UI login process. By setting a low cache limit (less than the number of records in these two tables), the Sterling Selling and Fulfillment Foundation cache must flush out earlier cache records when the cache fills up. As a result, the next login must read the records again.

To minimize the amount of cache management overhead, the caching mechanism implements a simple space management strategy - when the number of cached records for a table hits the limit specified above, the cache manager initializes (or refresh) that table's cache to being empty.

Strategies for Enabling Reference Data Caching

The following are some suggestions to consider if you decide to cache some or all of the reference tables:

- Cache tables that have very low write or update activities. When a record is changed, the local cache manager has to notify the other active cache manager to flush that table.
- Monitor the frequency at which the cache tables are flushed:
  - If a table is being flushed frequently because the records are being changed, you may want to consider not caching these tables. For example, your process may involve updating records in that table en mass. If that is the case, the cost of the large number of cache flush notifications could out weigh the benefits of caching that table.
  - If a table is being flushed frequently because the number of OBJECTS is hitting the cache limit, you should study the number of records cached and the cache hit ratio. You may, for example, not want to cache the table if the table has a very large number of cacheable records (e.g., during the day, transactions will range through all the records) and the potential cache hit ratio is low. Conversely, you may want to increase the OBJECTS limit if the potential cacheable records is just slightly over the limit.
- Cache queries that are expensive.
- Monitor heap garbage collection to make sure that the garbage collection overhead is not significant. We recommend you keep the garbage collection overheads (which we define as the amount of time spent in garbage collection over an interval) to less than 3%.

Monitoring Cache: The number of records a JVM caches depend on many factors including the type of transaction, the data that it retrieves, the breadth of functionality used, and possibly seasonality.

For example, an agent that is configured to Schedule Orders only caches records that is used by the Schedule transaction. An application server, in contrast, serves a broad range of transactions and typically requires more memory for the cache. An application server that services both Sterling Distributed Order Management (DOM) and Sterling Warehouse Management System (WMS) likely caches more records than one that only services DOM.

You can monitor cache usage from the System Management Console. Go to the Detail page for an application server or the Sterling Selling and Fulfillment Foundation agent. Press the "Table Level Cache" button.

Cache Drop Messages

The cache manager produces the following message, at the log4j WARN level, to report cache flushes:

```
2011-02-11 13:10:44,753:WARN   :main: Clearing cache. Number
 cached=7787,Lists cached=2,Singletons cached=2: YFS_ResourceDBCacheHome
```

The "number cached" refers to the cached records (OBJECTS). The "Lists cached" refers to the LIST WHERE clauses. The "Singletons cached" refers to the "SELECT WHERE clauses".

YFS_HEARTBEAT

Sterling Selling and Fulfillment Foundation records an entry into the YFS_HEARTBEAT table for each application server and each Sterling Selling and Fulfillment Foundation server that starts up. These entries enable Sterling Selling and Fulfillment Foundation to manage servers and to broadcast cached data updates to them. When a Sterling Selling and Fulfillment Foundation server is shut down normally, the corresponding YFS_HEARTBEAT record is removed.

When a Sterling Selling and Fulfillment Foundation server ends abnormally (or whenever an application server ends) the corresponding record can remain in the YFS_HEARTBEAT table even though it no longer points to a valid running server. These pointers to servers that are no longer running are known as "stale entries." Large number of stale entries could slow down the management of the servers. For example, the cache refresh broadcast will have to try to notify the servers pointed by the stale entries.

Periodically, each JVM updates its status in its YFS_HEARTBEAT record. By default, that refresh interval is set to `yantra.statistics.persist.interval` / 2 or 5 minutes.

To eliminate stale entries from the JNDI tree, you should run the Health Monitor agent (see *Sterling Selling and Fulfillment Foundation: System Management and Administration Guide*) for more detail.

Services

Sterling Selling and Fulfillment Foundation provides certain standard out-of-the-box services, which could be used on actions configured from events. These services have been provided in synchronous mode. Some of these services like Receipt Closure, may require to be changed to asynchronous mode to maximize performance.

To make them asynchronous, you would need to copy the current supplied service to another service flow, and change the starting point to one of the asynchronous transports like WebLogic JMS, MQ, and so forth.

In events where the originally supplied service flows are configured to call synchronously, you would need to create a custom service which would publish the Input XML to an asynchronous transport component like WebLogic JMS, MSMQ, and so forth.

For more information on defining service definitions, see the *Sterling Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

APIs

Sterling Selling and Fulfillment Foundation provides an extensive list of APIs that client programs can invoke. Here are some recommendations for you to consider.

API Output XML Files

The Sterling Selling and Fulfillment Foundation APIs, such as *getOrderDetails*, return data based on the specification of the following two XMLs files:

- Output XML file
- Template XML file

The output XML file defines all the possible elements and attributes that the API is capable of returning. The template XML file allows you to specify a subset of the elements and attributes that the API returns.

For performance, especially for high volume APIs, you should optimize the template XML file. Refer to the *Sterling Selling and Fulfillment Foundation: Extending the Database* for recommendations.

In the template XML, strive not to use the `TotalNumberOfRecords` attribute. Coding this attribute makes the application issue a separate count query against the database. The count query can be expensive if there is a large number of records that qualify.

List APIs

List APIs allow you to retrieve sets of data from the Sterling Selling and Fulfillment Foundation database. These APIs are typically labeled getXXXList - for example, getItemList, getLocationList, and getOrganizationList. In some cases, the list API could "find" a large number of records which would cause the API to return a very large output XML. If unchecked, the output XML could consume a large portion of the Java heap.

Developers can, and should, limit the number of records returned by setting the attribute `MaximumRecords` in the list XML. For example, the following input XML returns at most 2 records:

```
<Item MaximumRecords="2" />
```

You can also enforce a system-wide limit by setting `yantra.app.maxrecords` (see "API Control" on page 115).

User Exits and Events

APIs give you the ability to add your own custom code in user exits or events at well-defined points in the processing. The user exits and events are defined in the *Sterling Selling and Fulfillment Foundation API Javadocs* and in the *Sterling Selling and Fulfillment Foundation: Extending Transactions*.

When using user exits and events, keep the following in mind:

- Ensure the processing time in the exit and events are short. Long exit or event processing times increases transaction response time which could result in lowered throughput.
- Ensure that call-out (requests) to external systems can scale beyond anticipated peak processing rates. Unscalable or degraded call-outs can significantly elongate user exit or event processing times.

- Ensure that you do not hold critical record locks during the call out. Critical record locks are defined as those records, such as the YFS_INVENTORY_ITEM, YFS_LOCATION, YFS_LOCATION_INVENTORY, YFS_ORDER_HEADER, and so forth that are potentially needed by other transactions. Please see below.

You need to be aware of whether you are holding record locks when invoking user exits or events, especially exits or events that could take a long time to process. For example, if your transaction is holding YFS_INVENTORY_ITEM locks and your exit takes a minute to process, you could potentially block other inventory processing transactions that requires that lock.

You can find out whether you are holding on to locks during a call out by review VERBOSE traces. Look for any SELECT... FOR UPDATE statements issued prior to the call out.

Calling a user exit while holding on to locks may not be an issue if you are certain the user exit or event completes quickly (e.g., less than 100ms). For example, you may have coded a user exit to publish an ON_SUCCESS message to a message queue. The call out response time is less certain if your exit calls out to an external system. We have seen many cases at customer sites where external systems call outs either failed to return or have taken over two minutes.

Wildcard Characters

Oracle, DB2 and Microsoft SQL Server use the underscore character ("_") as a single character wildcard and the percent character ("%") as a wildcard character that can match zero or more characters. If possible, you should avoid using these two characters in indexed fields. Take for example the case where you have a record with ORDER_NO equal to E1_DIV01_03215466.

The following query is fast because only records with 'E1_DIV01_03215466' qualifies:

```
select order_header_key
from yfs_order_header
where order_no = 'E1_DIV01_03215466';
```

But the following query can be very slow, especially if you have millions of records that start with "E1%":

```
select order_header_key
from yfs_order_header
where order_no like 'E1_DIV01_0321546%';
```

In the example above, records with ORDER_NO equal to E11DIV0110321546, E11DIV01A0321546 and so on qualifies. As a result, the database server has to find every qualifying record with ORDER_NOs ranging from E1<*low value*>DIV01<*low value*>0321546% to E1<*high value*>DIV01<*high value*>0321546%.

If you use wildcards as part of the column value, you can escape the wildcards as shown in the following example:

```
select order_header_key
from yfs_order_header
where order_no like 'E1\_DIV01\_0321546%'
escape '\';
```

log4j Logging

The *Sterling Selling and Fulfillment Foundation: Installation Guide* provides more detail on how to configure log4j. Logs are important because they provide information to help you detect:

- Application problems - for example, application errors during development
- Order processing exceptions - for example, the inventory levels of an item are low and is causing orders to backorder

Logging Level

The Sterling Selling and Fulfillment Foundation's implementation of logging provides the following four application logging levels:
- ERRORDTL
- ERROR
- WARN
- INFO

and the following four diagnostic logging levels:
- TIMER
- SQLDEBUG
- DEBUG
- VERBOSE

You can turn on all or a combination of some of these levels. You can also designate different log destinations.

For production, you should enable either the INFO or WARN logging level. The application logging levels are cumulative. If you enable INFO, you get all four levels from INFO to ERRORDTL. If you enable WARN, you get three levels from WARN to ERRORDTL.

When needed, you can enable diagnostic logging levels for short periods of time in production. The DEBUG and VERBOSE consume large amount of computing resource and generate large amount of log entries. Enabling VERBOSE logging also enables all diagnostic logging levels from VERBOSE to TIMER as well as all application logging from INFO to ERRORDTL. VERBOSE logs prints out lots of information including the input, intermediate and resulting XMLs, debug information, and so forth.

The TIMER logging level produces a one-line trace entry to record when certain processing sections are entered and exited. This diagnostic logging level is useful for identifying areas for tuning.

The SQLDEBUG diagnostic logging level produces log entries for each SQL statement processed. In addition, SQLDEBUG also enables the TIMER logging level. This logging level is useful if you suspect that there are slow SQL statements.

You can control the log4j logging levels by using any one of the following methods:
- `log4jconfig.xml`.
- System Management Console or modifyTraces API call.

The logging level defined in the `log4jconfig.xml` is restrictive. Therefore, you can define the maximum logging levels allowed for a particular class using the `log4jconfig.xml`.

You can also set the logging level criteria either through the System Management Console or by calling the modifyTraces API.

If you do not define a logging level criteria through the System Management Console, the logging level is set to 'INFO' or the value defined in the`log4jconfig.xml` is considered. However, the system will consider the lowest value.

Log Destinations

By default, the `log4jconfig.xml.sample` file defines a `ROLLINGFILE_APPENDER` with a hard coded destination of `/application_path/log/sci.log`. If you were to start multiple JVMs (e.g., multiple agents and/or application servers), they all write to the same file. In some cases, the log messages from multiple JVMs could be interleaved.

To avoid this, you can use a parameter to define a separate log file for each JVM. This can be accomplished as follows:

- In the log4jconfig.xml, set the ROLLINGFILE_APPENDER as follows:

  ```
     <appender name="ROLLINGFILE_APPENDER"
  class="org.apache.log4j.RollingFileAppender">
     <param name="MaxFileSize" value="2048KB" />
     <param name="MaxBackupIndex" value="2" />
     <param name="File" value="${LOGFILE}" />
     <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d:%-7p:%t: %-60m: %-25c{1}%n"/>
     </layout>
     </appender>
  ```

  Note: For Windows, format the example appropriately.
- Pass in the LOGFILE parameter when starting up the JVM. In the following example, the log file has the agent or application server name followed by the node name and a date and time:

  ```
  AGENT_LOGFILE=${LOG_DIR}/${AGENT_NAME}_{$HOSTNAME}-'date +%Y%m%d%-H%M%S`.log
  java -DLOGFILE=${AGENT_LOGFILE} \
  com.yantra.integration.adapter.IntegrationAdapter
  ```

  Note: For Windows, format the example appropriately.

Property File

For additional information about overriding properties using the `customer_overrides.properties file`, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*.

Application Server Connection Pool Parameters

Sterling Selling and Fulfillment Foundation components (e.g., EJB, servlets) that run in the application servers use the Application Server Connection Pool parameters to find the connection pool. Refer to “WebLogic Connection Pool” on page 52 if you are using WebLogic or “WebSphere Connection Pool” on page 47 if you are using WebSphere for more detail.

The Application Server Connection Pool parameters are not applicable to agents and the asynchronous adapters because they do not run in the application server.

Note: If Sterling Selling and Fulfillment Foundation running in an application server cannot get a connection through the connection pool, it tries to establish a connection through the Direct Connection parameters. If you do not like this behavior, you can comment out the Direct Connection parameters in the application server-specific `yfs.properties` file. The application server transactions that cannot get a connection aborts with an exception.

Reference Data Cache Parameters

Reference data caching is critical for performance and scalability. By default, the cache is enabled. Please see “Performance Feature - Reference Data Caching” on page 104 for more detail, recommendations and strategies.

User Interface Control

The UI Control parameters are only applicable to the screen workloads (e.g., JSPs) running in the application servers. They provide system level controls to the application administrators.

yfs.ui.maxRecords

For more information about user interface control, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*. In addition to this control, the List screens have a Maximum Record field which is currently defaults to 30. Therefore, out-of-the-box, if the user issues a search that has 1,000 records, only 30 are displayed. The user can, at their discretion, change the value of Maximum Record up to the value specified by `yfs.ui.MaxRecords`.

There is an important points that you need to be aware of:

The `yfs.ui.MaxRecords` only controls the number of records (e.g., orders or items) that can be displayed in a list. It does not control the amount of work the database has to perform to get those records. For example, a user can issue a very inefficient query by asking for all orders that "contains" the letter "Z" in the order number or in the customer's e-mail id. Those queries typically result in a full table scan of potentially large tables.

yfs.ui.queryTimeout

For more information about this parameter, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*.

API Control

yantra.app.maxrecords

This parameter serves as a safe guard to limit records returned by LIST APIs to 5,000 records. Please see “List APIs” on page 111.

Statistics

By default, Sterling Selling and Fulfillment Foundation generates application-level statistics every 10 minutes. The statistics generation is governed by the following parameters:

```
yantra.statistics.collect=y
yantra.statistics.persist.interval=10m
```

For more information, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*. The System Management Console (see “System Management Console and Health Monitor Agent” on page 127) relies on these statistics.

Please see “Statistics” on page 128 for more information.

Caution: The time that JVMs refresh their YFS_HEARTBEAT status is set to `yantra.statistics.persist.interval` / 2. Therefore, by default, the YFS_HEARTBEAT refresh interval is set to 5 minutes. We recommend that you do not increase the `yantra.statistics.persist.interval` parameter because of its secondary effect on the YFS_HEARTBEAT refresh interval.

Hot SKU Feature

Please see “Hot SKU Controls” on page 123 for information on the Hot SKU control parameters.

yfs.inventory.sortandlock

To prevent deadlocks, Sterling Selling and Fulfillment Foundation sorts the order or shipment lines by the items at the line level (see “Sort Order and Deadlocks” on page 126) prior to processing. As the application processes the line, it locks the YFS_INVENTORY_ITEM record. For example, given the following four line order where:

- Line 1, item A
- Line 2, item G
- Line 3, item F
- Line 4, item E

Sterling Selling and Fulfillment Foundation locks the items and process the lines in the following sequence:

- Lock item A, process line 1
- Lock item E, process line 4
- Lock item F, process line 3
- Lock item G, process line 2

Transactions that follow this convention reduces the likelihood of deadlocks. The exception is when orders has kits. Using the example above, assume that item G in line 2 is a kit that is made up of the following kit items D, B, and C. Since the application sorts the item at the line level, the application still processes lines 1, 4, 3, and 2 as above. However, when the transaction processes item G, it potentially locks the kit items out of sequence. Using the example above, the locking sequence is as follows:

- Lock item A, process line 1
- Lock item E, process line 4
- Lock item F, process line 3
- Lock items B, C, and D, process line 2

If you are processing kits and are experiencing deadlocks, you can set the `yfs.inventory.sortandlock` parameter to Y. With the parameter enabled, the application first sorts and locks all the line item and kit items prior to processing the transaction. Using the example above, if you enable `yfs.inventory.sortandlock`, the application performs the following:

- Lock item A, B, C, D, E and F first
- Process line 1
- Process line 4
- Process line 3
- Process line 2

Note: Setting the `yfs.inventory.sortandlock` increases the amount of time the YFS_INVENTORY_ITEM locks are held. That increase may not be noticeable in small orders (for example, five line orders). However, that increase could be noticeable if the number of lines is large (for example, over 100 or 200 lines).

Warning: You should not set this parameter if you do not process kits. Setting this parameter does not add any value to non-kit orders.

See Related Sections:
- "Sort Order and Deadlocks" on page 126

Performance Feature - Hot SKU

Sterling Selling and Fulfillment Foundation locks the inventory item record for an item before manipulating that item's supply or demand information. That inventory item lock is held until the transaction is finished.

Determining The Amount Of Inventory Lock Contention: Transactions that hold inventory item record locks can block other transactions that need the same record. A certain amount of lock contention is acceptable especially if transactions are blocked infrequently or for short periods of time and if there is no material impact on processing throughput or end-user response times.

Determining Level of Lock Contention in Oracle

You can determine the level of inventory lock contention with the following techniques. In Oracle:
- Use AWR to calculate the amount of lock contention.
- In Oracle, query the v$session table to understand the extent of the lock contention.

AWR reports provide a measure of the total amount of time (in seconds) all transactions waited for record locks. This metric is found in the "Wait Events for DB" section (page 2) of a AWR report. In the following example, transactions waited for enqueues for a total of 741 seconds in that 30-minute measurement interval:

```
Wait Events for DB: YRAC05  Instance: YRAC051  Snaps: 15202 -15203
                                                                  Avg
                                                     Total Wait   wait    Waits
Event                               Waits   Timeouts   Time (s)   (ms)     /txn
---------------------------- ------------ ---------- ---------- ------ --------
db file sequential read           903,826          0      6,246      7      3.0
db file scattered read            879,659          0      4,281      5      2.9
enqueue                             3,542          6        741    209      0.0
library cache pin                     375        231        719   1918      0.0
buffer busy waits                 116,687          0        449      4      0.4
log file sync                     129,571          0        134      1      0.4
```

Dividing that number of enqueue wait times (741 seconds) by the measurement interval (30 minutes) shows that the enqueue contention was on average 0.41 blocked seconds per second. From a statistical point of view, one transaction was blocked 41% of the time every second. If you have ten concurrently running transactions, at one extreme, this statistic could be interpreted as all transaction was blocked 4.1%. At the other extreme, one transaction could have been completely blocked for 719 seconds.

In the example above, the lock contention is minimal. As a guideline, high lock contention situations are characterized as:

- Enqueue wait seconds per second is greater than 5 second per second or
- Enqueue wait is the top wait

If enqueue wait times are significant, run the following query to identify the sessions that are blocked, the amount of time that they were blocked for, and the objects they are blocked on:

```
select sid,last_call_et, sql_text
from v$session vs, v$sqlarea sa
where last_call_et > 0 and
     vs.sql_hash_value = sa.hash_value and
     vs.lockwait > ' '
order by last_call_et desc;

SID      LAST_CALL_ET       SQL_TEXT
13               1      SELECT  /*YANTRA*/   YFS_ORDER_HEADER.*
                      FROM YFS_ORDER_HEADER YFS_ORDER_HEADER
                      WHERE ENTERPRISE_KEY =:"SYS_B_0" AND
                            ORDER_NO = :"SYS_B_1"  FOR UPDATE
```

In the example above, session (SID=31) blocked for 1 second while trying to lock a YFS_ORDER_HEADER record.

We suggest you look at the following:

- Determine the objects that transactions are blocked on (e.g., are transactions blocked on YFS_INVENTORY_ITEM or some other table).
- Determine the amount of time these transactions block for - If the blocks are for a few seconds (e.g., 1-2 seconds) and the number of order lines per order are small, the level of contention may be acceptable.

This query, along with the contention level derived from AWR, lets you determine the extent of the lock contention.

Determining the Level of Lock Contention in DB2

For UDB, check the following monitor elements:

- `lock_wait_time` to determine the amount of lock contention. If you divide this number by the measurement interval, you get the average lock wait (in milliseconds) per second.
- Check the `table_name` monitor element in the `snapshot_lockwait` monitor to see where most of the lock contention are coming from.
- For each blocked agent, check the `stmt_text` and `uow_lock_wait_time` monitor elements in the snapshot_statement monitor.

We suggest you look at the following:

- Determine the objects that transactions are blocked on (e.g., are transactions blocked on YFS_INVENTORY_ITEM or some other table).

- Determine the amount of time these transactions block for - if the blocks are for a few seconds (e.g., 1-2 seconds) and the number of order lines per order are small, the level of contention may be acceptable.

Conditions For Inventory Lock Contention: The following three conditions must exist together for high inventory lock contention:

- Sufficiently high number of concurrent transactions that require inventory locks
- Sufficiently long inventory lock-holding times
- Presence of a few common inventory (SKU) in the orders of the concurrently running transactions

If the transaction volume is low and only one transaction is running, this transaction does not experience any inventory lock contention. The likelihood of inventory lock contention grows as the number of concurrently running inventory processing transactions (e.g., createOrder, schedule, release, and so forth) increases.

The impact of lock contention may be minimal if the lock-holding times are very short. Blocked transactions eventually get and lock the inventory item they need to process.

If there are no common SKUs, all the concurrently running transactions are able to process without blocking.

Optimization: If the inventory lock contention level is high, relative to your processing concurrency levels, or if you feel that your processing throughput or end-user response times are impacted, we suggest the following course of action.

Look at the lock-holding times. Run each inventory processing transaction with SQLDEBUG traces or possibly VERBOSE traces. VERBOSE traces provides more data but can be more intrusive than SQLDEBUG:

- See how long, on average, the transactions take.
- See when the first inventory locks are obtained (and as a result, how long they are held) within that transaction boundary. The goal is to keep lock-holding times short.
- See if there are places in the transaction that take a long time to process and the processing occurs when inventory item locks are held. For example:
  - The transaction may have a user-exit that calls out to external systems. If that external system slows down or is unable to scale, the user-exit time increases. This elongates the lock-holding times.
  - There may be SQL statements that run for a long time and can be optimized with better database statistics or an additional index.
  - Look at the GC logs - Make sure the transaction is not slowed down by long costly garbage collection pauses.

Reducing lock-holding times can have compounding effects - as lock-holding times decreases, transactions finish faster and, as a result, lower concurrency levels.

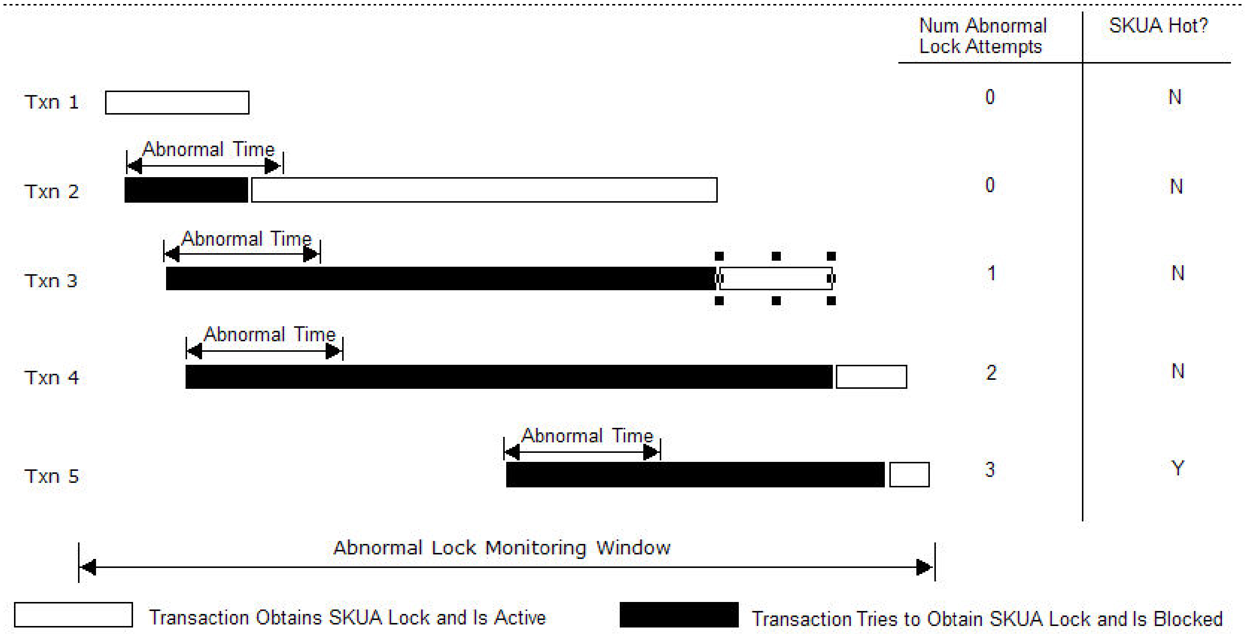
Hot SKU Feature: If inventory lock contention is still unacceptably high after you have applied the optimization from above, you have two options that can potentially reduce the level of inventory item lock:

- Hot SKU Feature
- Hot SKU Feature (without lock request timeout)

Hot SKU Feature (without lock request timeout)

In a nutshell, the Hot SKU feature tracks the time to lock inventory item records. If a lock request for an item (called SKUA) is longer than the `yfs.hotsku.secondsToClassifyAsAbnormalTime` threshold, the Hot SKU feature increments the number of "abnormal" lock attempts for SKUA. If the number of "abnormal" attempts in a monitoring window is more than the `yfs.hotsku.secondsToClassifyAsAbnormalTime` threshold, the Hot SKU feature considers SKUA a Hot SKU.

In the example below, the first transaction was able to obtain the SKUA lock immediately.


Num Abnormal Lock Attempts
SKUA Hot?
Txn 1
0
N
Abnormal Time
Txn 2
0
N
Abnormal Time
Txn 3
1
N
Abnormal Time
Txn 4
2
N
Abnormal Time
Txn 5
3
Y
Abnormal Lock Monitoring Window
Transaction Obtains SKUA Lock and Is Active
Transaction Tries to Obtain SKUA Lock and Is Blocked

As a result, the number of abnormal lock attempts (which we assume to have started at zero) stays at zero. The second transaction blocked but eventually got the lock within the abnormal lock time period. Since the request completed within the "abnormal" time, the request was not considered abnormal - as a result, the number of abnormal lock attempts is left at zero and SKUA is not considered a Hot SKU.

The next three transactions try to obtain the lock and were blocked for longer than the abnormal lock time window. Each "abnormal lock" attempt increases the abnormal lock count within that monitoring window. SKUA turns hot when the count reaches the `yfs.hotsku.numRequestsInTrackingWindowToKeepAsHotSku` threshold (which defaults to 3).

A Hot SKU is downgraded to a normal SKU if the number of references to that SKU is less than the `yfs.hotsku.numRequestsInTrackingWindowToKeepAsHotSku` threshold in a monitoring window.

Hot SKU detection and enablement occurs at each JVM. For example, a sudden high burst of demand for a single SKU could result in the createOrder adapter to consider that SKU hot. Later, as the downstream agents process those orders, they independently detect and enable those SKUs as Hot SKUs if they encounter sufficient number of "abnormal" locks.

When an inventory item is upgraded to Hot SKU status, transactions do not lock that inventory item before manipulating its demand or supply information. Instead, the transactions insert the demand or supply information into two new tables, YFS_INVENTORY_DEMAND_ADDNL or YFS_INVENTORY_SUPPLY_ADDNL respectively. As a result, demand or supply information can be recorded in a non-blocking manner because inserts do not block other transactions from proceeding. Transactions continue in this mode until the inventory items have been downgraded to the normal status.

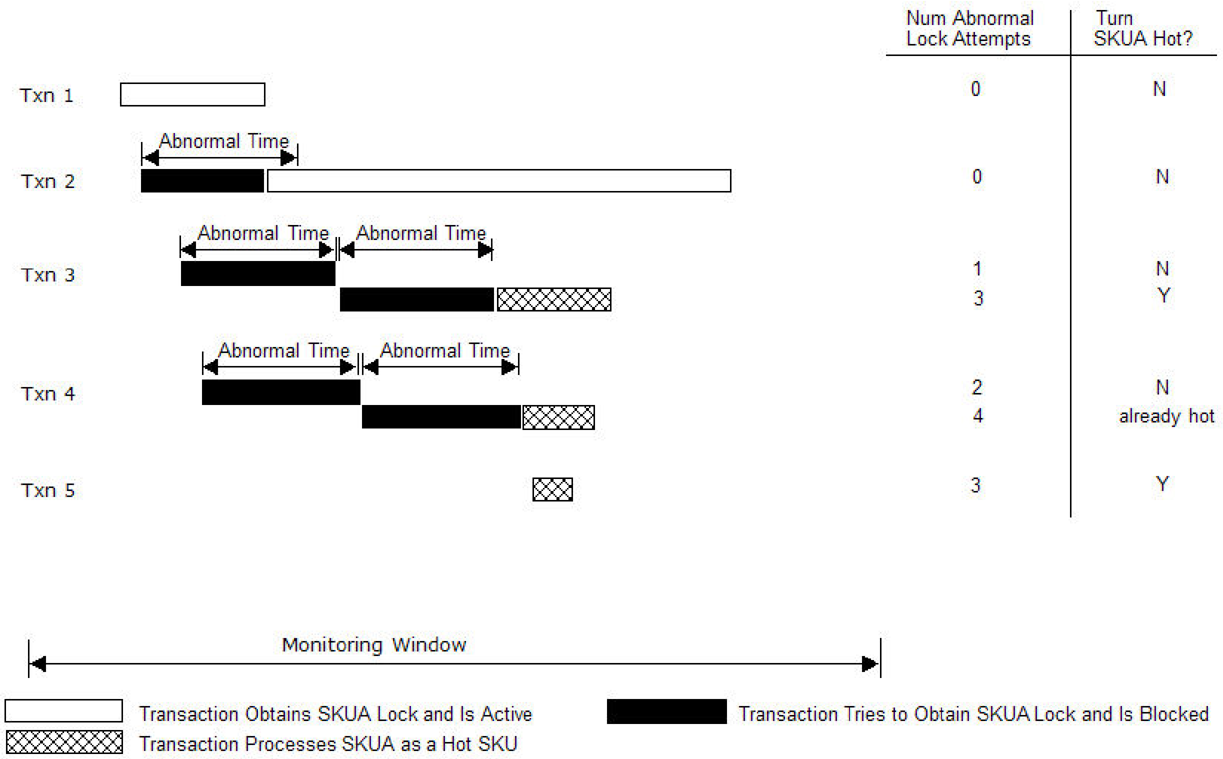
Inventory demand queries automatically check both the YFS_INVENTORY_DEMAND and YFS_INVENTORY_DEMAND_ADDNL tables. Similarly, inventory supply queries checks the YFS_INVENTORY_SUPPLY and YFS_INVENTORY_SUPPLY_ADDNL tables.

The quantities in the inventory additional records are consolidated back to their base inventory tables by the Consolidate Additional Inventory agent.

Hot SKU Feature with Lock Request Timeout option

The lock request timeout option in the Hot SKU feature may be useful in reducing the amount of lock holding time by limiting the time the lock request blocks. This can be useful when you are primarily processing large orders (for example, over 50 lines per order) where the transactions could block for a very long time.

Using the previous example, with the same transaction arrival times and processing times, we see that Transaction 1 immediately obtains SKUA's record lock and processes SKUA without the Hot SKU feature. Transaction 2 comes in and blocks on the Transaction 1 but eventually gets the lock within the abnormal time. As a result, Transaction 2 also processes SKUA without the Hot SKU feature.


Num Abnormal Lock Attempts
Turn SKUA Hot?
Txn 1
0
N
Abnormal Time
Txn 2
0
N
Abnormal Time
Abnormal Time
Txn 3
1
N
3
Y
Abnormal Time
Abnormal Time
Txn 4
2
N
4
already hot
Txn 5
3
Y
Monitoring Window
Transaction Obtains SKUA Lock and Is Active
Transaction Processes SKUA as a Hot SKU
Transaction Tries to Obtain SKUA Lock and Is Blocked

Transaction 3 starts and is blocked by Transaction 1. Eventually, Transaction 3's lock request times out. When that happens, the transaction increments SKUA's abnormal lock attempt count to 1 and reissues the lock request.

Similarly, Transaction 4 is blocked by Transaction 1 until its lock request times out. This transaction increments the abnormal lock attempt count to 2.

When Transaction 3 times out its second lock request, the Hot SKU feature upgrades SKUA to Hot SKU status. When that has happened, Transaction 3 can process SKUA even though Transaction 2 still has the SKUA record lock.

Similarly, when Transaction 4 times out, it is able to process SKUA.

More importantly, when Transaction 5 comes in, it sees that SKUA is already a Hot SKU and does not attempt to lock the SKUA record.

One important benefit to lock request timeout is in situations where the blocker could be holding on to locks for a long time. With the timeout option enabled, the blocked transactions has to wait up to a maximum of abnormal lock count (`yfs.hotsku. numberOfAbnormal LocksForSwitchTo HotSKU`) times the abnormal lock timeout (`yfs.hotsku. secondsToClassifyAs AbnormalTime`) before the transaction starts to process the SKU as a Hot SKU.

Consolidate Additional Inventory Agent: If you enable the Hot SKU feature, you must run the Consolidate Additional Inventory agent (see the *Sterling Selling and Fulfillment Foundation: Application Platform Configuration Guide*). This agent consolidates the quantity in the demand and supply additional records back into the base YFS_INVENTORY_DEMAND or YFS_INVENTORY_SUPPLY tables. The additional demand and supply records are deleted after the quantities are consolidated.

Typically, you should configure this agent to run continuously with one thread. You don't need to overly aggressively schedule this agent - if you do, the agent consolidates a small number of additional records. At the same time, you do not want to run for long periods without this agent - if you do, you could accumulate a large number of inventory additional records which can slow down inventory queries.

If you are using the DB2 database server, you need to set the parameter DB2_SKIPINSERTED to ON and mark the YFS_INVENTORY_SUPPLY_ADDNL and YFS_INVENTORY_DEMAND_ADDNL tables as volatile. These settings reduce lock contention. Please see the following sections for more information:

- "Recommended DB2 dbset Registry Variables" on page 61
- "Volatile Tables" on page 74

Hot SKU Activity Monitoring: The following messages are logged by the Hot SKU feature:

```
   Thread-8:2011-06-02 13:50:06,336:INFO   : Turning/retaining Item:
[Acme:SKUA:EACH:A] into a hot sku as abnormal lock count has increased now
to:4: YFSAvailHotSKUItem

   Thread-7:2011-06-02 16:14:44,871:INFO   : Turning Item: [Acme:SKUA:EACH:A]
into a cold SKU as total requests in last 2 windows were :2 and 2:
YFSAvailHotSKUItem
```

In the example above, the inventory item, SKUA, for the Acme organization, with UOM EACH and product class A, was upgraded to Hot SKU status because the Hot SKU feature encountered four abnormal lock attempts in a monitoring period. Later, SKUA was downgraded to normal status when there was only 2 lock requests in the last two monitoring windows.

If you see multiple "turning hot" messages for a particular SKU (for example SKUA in the following example), you ran into a situation where multiple threads tried to lock an inventory item, which was at that time not considered a hot SKU, and was blocked. When those threads eventually get the lock, it prints the message indicating that it encountered an "abnormal" lock and has decided to turn that item hot:

```
   Thread-8:2011-06-02 13:50:06,336:INFO   : Turning/retaining Item:
[Acme:SKUA:EACH:A] into a hot sku as abnormal lock count has increased now
to:4: YFSAvailHotSKUItem
   Thread-10:2011-06-02 13:50:06,417:INFO   : Turning/retaining Item:
[Acme:SKUA:EACH:A] into a hot sku as abnormal lock count has increased now
to:5: YFSAvailHotSKUItem
   Thread-7:2011-06-02 13:50:06,423:INFO   : Turning/retaining Item:
[Acme:SKUA:EACH:A] into a hot sku as abnormal lock count has increased now
to:6: YFSAvailHotSKUItem
```

That item was likely successfully converted to a hot SKU if you do not see any more subsequent "turning hot" messages for that item.

On the other hand, if you continue to see "turning hot" messages for the same SKU in the same window, you may have a "Hot" SKU that has low inventory. When a SKU's inventory level is below a safety level, the Hot SKU feature continues to lock that inventory item to calculate the items availability (see "Limitations" on page 125 below).

Hot SKU Controls:

Note: The Hot SKU component is a very powerful feature that you can deploy if your organization is experiencing true high Hot SKU contention. These parameters may cause performance problems if set incorrectly. Read this document carefully. Verify that you have true Hot SKU contention before enabling this feature or changing any settings. If in doubt, call Technical Support.

In your `customer_overrides.properties` file, specify the Hot SKU Control parameters. For additional information about overriding properties using the `customer_overrides.properties file`, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*.

Table 22. Hot SKU Control

Parameter	**Description**
In the customer_overrides.properties file, add the following entries:	
yfs.yfs.hotsku. useHotSKUFeature	Control used to enable/disable the Hot SKU feature. By default, this parameter is set to "N".
yfs.yfs.hotsku.useTimeOutLocking	Control used to enable or disable the lock request timeout option in the Hot SKU feature. By default, the parameter is set to "N".
yfs.yfs.hotsku. secondsToClassifyAs AbnormalTime	Threshold used to determine when a lock request is "abnormal". If a lock request exceeded this threshold, that lock request is counted as an "abnormal" lock request. Defaults to 0.5 seconds.
yfs.yfs.hotsku.windowTimeInMinutes	Interval of one tracking window during which "abnormal" lock requests are tracked. A subsequent window begins once the first window ends. Default interval is 10 minutes.
yfs.yfs.hotsku. numberOfAbnormalLocks ForSwitchTo HotSKU	Threshold used to determine when to promote an inventory item to Hot SKU status. If the number of "abnormal" lock requests for an item exceeds this value within the tracking window, that item is promoted to Hot SKU status. see "Three Usage Scenarios" on page 125 for recommended settings.
yfs.yfs.hotsku.numRequestsInTrackingWindow ToKeepAsHotSku	Minimum number of requests needed within the tracking window to keep the item in Hot SKU status.
yfs.yfs.hotsku.showExtraMessagesAsInfo	Enables extra messages to be displayed at info logging level. Default value is "N".
yfs.yfs.hotsku.itemMapPurgeLeadTime	This parameter is used when the number of Hot SKU items in the cache has reached its limit (specified by yfs.yfs.hotsku.maxItem MapSizeInMemory). When cache is full, eject Hot SKU if it has not been accessed in yfs.hotsku.itemMapPurgeLeadTime minutes. Default value is 10.
For Sterling Selling and Fulfillment Foundation Internal Use Only	
yfs.yfs.hotsku.qtyMultiplier	Defaults to 30. Do not modify without IBM guidance.
yfs.yfs.hotsku.highRequest QuantityMultiplier	Defaults to 2. Do not modify without IBM guidance.

Table 22. Hot SKU Control (continued)

Parameter	Description
yfs.yfs.hotsku.maxItem MapSizeInMemory	Defaults to 1000. Do not modify without IBM guidance.

Three Usage Scenarios:

Currently we envisage three scenarios for the Hot SKU feature.

Table 23. Hot SKU Usage Scenarios

Scenario	Description	Parameters
Not Enabled	Inventory lock contention is minimal. The Hot SKU feature is not needed.	`yfs.hotsku.useHotSKUFeature=N`

Small Orders	Inventory lock contention is sufficiently high and is caused by a high volume of small orders (around 1-5 order lines per order)	`yfs.hotsku.useHotSKUFeature=Y`
		`yfs.hotsku.numberOfAbnormalLocksForSwitchTo HotSKU=3`
		yfs.hotsku.windowTimeInMinutes=20.0
		yfs.hotsku.numRequestsInTrackingWindowToKeepAs HotSku=3

Large Orders	Inventory lock contention is sufficiently high and is caused by a high volume of large orders (more than 50 order lines per order)	`yfs.hotsku.useHotSKUFeature=Y`
		`yfs.hotsku.useTimeOutLocking=Y`
		`yfs.hotsku.numberOfAbnormalLocksForSwitchTo HotSKU=3`
		yfs.hotsku.windowTimeInMinutes=5.0
		yfs.hotsku.numRequestsInTrackingWindowToKeepAs HotSku=5

Limitations: There are four situations where Sterling Selling and Fulfillment Foundation continues to lock the YFS_INVENTORY_ITEM records even when the item is considered a Hot SKU.

First, the following APIs always lock the YFS_INVENTORY_ITEM records during supply adjustment:

- updateFutureInventory
- getInventoryMismatch

Second, Sterling Selling and Fulfillment Foundation always locks YFS_INVENTORY_ITEM records for tag-controlled items if the request is for specific tag criteria.

Third, Sterling Selling and Fulfillment Foundation locks the YFS_INVENTORY_ITEM records for an item that is currently a Hot SKU in order to calculate availability if the inventory for that item is below a safety level.

Note: The reserveItemInventory API checks the safety level availability of Hot SKU items, however, it does not load an availability picture for Hot SKU items. A prior call to one of the promising APIs such as findInventory is required to load the availability picture for Hot SKU items.

Fourth, if the ‘Summarize and Maintain Total Supply and Demand Values For Tag Controlled Items' Installation Rule is enabled, the Hot SKU logic is not used, and the values of the TotalOnhandSupply, TotalOtherSupply and TotalDemand fields are updated accordingly. For more information on defining additional inventory rules, refer to the *Sterling Selling and Fulfillment Foundation: Global Inventory Visibility Configuration Guide*.

Sort Order and Deadlocks

Deadlocks occur when two or more sessions mutually block each other to the point where neither session can progress. As a result, these sessions continue to block until the database management system kills one of the deadlocked sessions in order for the others to continue.

Deadlocks occur when two or more sessions obtain resource locks in an arbitrary fashion. For example, the following is a classic example:

```
Txn 1                                 Txn 2
Locks Record A                     Locks Record B
Tries to Lock Record B    (blocked)    Tries to Lock Record A (blocked)
```

In the example above, Txn 1 holds the lock for Record A and Txn 2 holds the lock for Record B. When Txn 1 tries to lock Record B, it becomes blocked. When Txn 2 tries to lock Record A, it also becomes blocked. Now, neither session can progress unless one of the transaction is killed.

If the resource locks were obtained in a consistent order, the deadlock does not occur. For example, all transactions agree to lock the records in ascending order (Record A then Record B).

Replaying the example above, we now have:

```
Txn 1                                 Txn 2
Locks Record A                     Tries to Lock Record A (blocked)
Locks Record B
commits
                                   Locks Record A
                                   Locks Record B
                                   commits
```

In the example above, Txn 2 is delayed but not deadlocked. Both transactions eventually complete.

Sort Order

When you develop custom code, you should be aware that Sterling Selling and Fulfillment Foundation obtains YFS_INVENTORY_ITEM locks in the following sort order:

Item ID, Product Class and UOM

If you adopt this sort order, you should greatly minimize the chance of deadlocks.

Precompiling JSPs

The Sterling Selling and Fulfillment Foundation UI is made up Java Server Pages (JSPs). When users call up a JSP the first time, the application server automatically translates and compiles the JSP file. This process can over 30 seconds, which could lead to the perception of an unresponsive system. Further, this process is performed serially even on a multiprocessor node - if you have multiple users hitting five different pages, WebLogic compiles these pages one at a time. As a result, we strongly recommend precompiling the JSP pages prior to deployment into production.

You should ensure your application server administrator precompiles the JSPs as part of the application deployment.

Please see “WebLogic Tuning: JSP Pre-Compilation” on page 53 on precompiling JSPs in WebLogic application servers.

Please see “WebSphere Tuning: JSP Pre-Compilation” on page 49 on precompiling JSPs in WebSphere application servers.

MS Internet Explorer: Temporary Internet Files

About this task

You can reduce the number of hits against the application servers for static content by enabling temporary Internet file cache in Microsoft Internet Explorer. This improves your UI response times. To enable the cache:

- Go to the Internet Options dialog box.
- In Microsoft Internet Explorer, go to Tools > Internet Option:
  - Click on the Settings button in the Temporary Internet Files panel.
  - Enable the “Check for newer version of stored pages” radio button to Automatically.
  - Make sure there is sufficient disk space to store temporary Internet files (e.g., 500MB or higher).

Monitoring

You can monitor the status and progress of Sterling Selling and Fulfillment Foundation with the following tools or techniques:

- System Management Console
- Throughput Queries
- Sterling Selling and Fulfillment Foundation Statistics
- YFS_INBOX
- Application Logs

System Management Console and Health Monitor Agent

The System Management Console is an application monitor. Some of the areas you can monitor include:

- The processing throughput, response time, the amount of pending work, and the number of errors generated at the API and agent level
- The status of the application servers
- The number of messages in JMS queues

In addition, the System Management Console allows you to:

- Shut down, suspend, or resume agent and integration servers.
- Clear reference data cache for a single or all cached tables.
- Enable/disable API, agents, user exits, services, and the application consoles application traces.

The companion Health Monitor agent can be configured to alert system administrators when problems occur such as when an application server crashes or agent servers are not processing tasks.

The System Management Console's functionality, screens, and related tasks are documented in detail in the *Sterling Selling and Fulfillment Foundation: System Management and Administration Guide*.

Statistics

The System Management Console gets most of its measurements from data found in the YFS_STATISTICS_DETAIL table. These statistics are described in the *Sterling Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

Sterling Selling and Fulfillment Foundation Statistics records, by default, are generated every 10 minutes for each active API and transaction running in each application server or Integration Adapter. For example, if the Schedule transaction was active in 3 Integration Adapters, you have 3 sets of statistics for each measurement interval.

Time-triggered transactions, at a minimum, generate the following four metrics:

- The `GetJobsProcessed` metric indicates how many times Get Jobs were issued to look for work for this transaction.
- The `ExecuteMessageCreated` metric indicates how many records were selected for processing.
- The `ExecuteMessageSuccess` metric indicates how many messages were successfully processed.
- The `ExecuteMessageFailure` metric conversely indicates how many messages were not successfully processed.

With these four metrics, you could:

- Track `ExecuteMessageSuccess` to see how much work the application is processing throughout the day.
- Track the ratio of `ExecuteMessageSuccess` divided by `ExecuteMessageCreated` to get an idea of the effectiveness. For example, a ratio of 0.8 means that only 80% of the orders are successfully processed. If the effectiveness ratio is consistently low, it could indicate that the application is encountering a large number of work (or orders) that repeatedly fail. This could lead to extra processing overhead.
- Calculate the resource cost per unit work processed by correlating the number of worked processed against the CPU consumed. You could track this to see if the cost per unit work is changing. This metric is useful for identifying areas to optimize. It is also the basis for computing resource capacity forecasting or planning.

In addition, some transactions produce transaction specific statistics. For example, some of the metrics the Schedule transaction generates includes `NumOrdersBackordered`, `NumWorkOrdersCreated`, and so forth.

Inbox

You should monitor the number of active alerts in the YFS_INBOX table. Sterling Selling and Fulfillment Foundation alerts come from the following source:

- Transactions that are configured to raise alerts
- Monitor (such as the order monitor)

Users subscribed to queues with large number of open alerts can experience slow logins. Very large YFS_INBOX tables can impact login times for all users.

You can find out the number of active and non-active alerts by issuing the following query:

```
select active_flag,count(*)
from yfs_inbox
group by active_flag
```

You can find out the distribution of alerts by queue name and inbox type by issuing the following query:

```
select queue_name, inbox_type, active_flag, count(*)
from yfs_inbox inb,yfs_queue q
where inb.queue_key = q.queue_key
group by queue_name, inbox_type, active_flag
```

You can find out the hourly rate of alert creation for July 4, 2011 by issuing the following query:

```
select substr(inbox_key,1,10),count(*)
from yfs_inbox
where inbox_key > '20110704000000' and
      inbox_key < '20110704999999'
group by substr(inbox_key,1,10);
```

Application Logs

You should regularly monitor Sterling Selling and Fulfillment Foundation and application server logs for, at a minimum, the following:

- Application errors or business exception conditions - for example, invalid input XML to APIs, and so forth.
- System errors - e.g., Java OutOfMemory or NullPointer exceptions

Audits

In the entity XMLs for a table, the out-of-the-box attribute “AuditRequired” is turned on for many entities, such as Pricing and Entitlement rule related entities. This flag results in maintaining YFS_AUDIT records for all changes that are made to these tables.

It is recommended that you turn this flag to off for the initial data load of entitlement rules. If the flag is left on, there is a performance overhead of 2ms per record YFS_ENTITLE_RULE_DTL_ITEM entity.

It is also recommended that you evaluate whether you really need this audit for all entities for which you will be performing large data updates, for example, copyPriceList and manageEntitlementRule. Unless absolutely required, the audit of pricing and entitlement entities should be turned off.

For information about how to turn off the AuditRequired attribute, refer to the "Generating Audit References for Entities" section of the *Sterling Selling and Fulfillment Foundation: Extending the Database*.

Sterling Distributed Order Management

This section provides recommendations on how to plan, implement, configure, monitor, and tune the IBM Sterling Distributed Order Management system with the default factory (data) settings, which provide a simple configuration that is suitable for development, training or product familiarization. That configuration is not suitable for production except for customer with very low transaction volumes. This section guides you through the components that you have to configure for higher transaction volumes.

This section assumes that you:

- Are familiar with the basic functionality of the Sterling Distributed Order Management
- Have read the common Sterling Selling and Fulfillment Foundation performance concepts in “General Recommendations” on page 91
- Have read and followed all the instructions found in the *Sterling Selling and Fulfillment Foundation: Installation Guide*
- Have read the *Sterling Selling and Fulfillment Foundation: Release Notes -- Known Issues*

Schedule Agent for Backorder Efficiency

The SCHEDULE transaction schedules orders to specific ship nodes making sure that the scheduled ship nodes have enough inventory to process the order. An order line is put into a backordered state when there is insufficient inventory to be retried at a later time. By default, the retry interval is set to five hours. This setting should be applicable to most customers. In some cases, your SCHEDULE agent may be spending a lot of time reprocessing backorders. For example, your warehouses may only restock once a week and you have a large number of backorders. If you have a large number of backordered orders and the backorders can last a few days, you may want to consider one or both of the following:

- Increase the backorder reprocess interval (possibly to a day) in the Scheduling Rule Details screen (see the *Sterling Selling and Fulfillment Foundation: Distributed Order Management Configuration Guide*)
- Create a separate SCHEDULE agent to only process backorders. The benefit of this approach is you can reduce the frequency when you trigger the backorder Schedule agent. For example, you could continue to automatically trigger your main SCHEDULE agent once a minute (if you have very strict end-to-end order processing service level agreements) and your backorder SCHEDULE agent every two hours or after inventory supply updates.

Real-Time Inventory Availability Monitor for ATP Efficiency

The Real-Time Inventory Availability Monitor is used to alert external systems when inventory availability crosses predefined thresholds. When items are flagged for real-time availability monitoring, a record is inserted into the YFS_INVENTORY_ACTIVITY table by inventory transactions that update supply or demand information.

This monitor checks inventory availability based on information in the YFS_INVENTORY_ACTIVITY table. The activity records associated with an item are deleted after the inventory check.

If you plan to use the real-time inventory availability monitor, we suggest you start with 5 threads and monitor the number of records in the YFS_INVENTORY_ACTIVITY table. It is highly recommended that you aggressively monitor the YFS_INVENTORY_ACTIVITY table. Additionally, we also recommend that you set this agent to be auto triggered with an interval of 5 minutes.

The following query can be used to monitor build up in the YFS_INVENTORY_ACTIVITY table. This query tells you the oldest activity record in the table:

```
select sysdate, min(inventory_activity_key) "Min Datetime"
from yfs_inventory_activity

SYSDATE                     Min Datetime
12/22/2011 3:57:57 PM          20111222155659187360102
```

In the example above, the query was issued at 15:57:57 and the oldest inventory activity record was created at 15:56:59. Therefore, the monitor is keeping up by about 1 minute.

If the time gap between the current time and the oldest record keeps increasing over time, we recommend starting additional JVMs of this agent.

Note: Although described as 'real-time', availability changes may not be triggered immediately as inventory changes occur if the agent has a backlog of messages to process. Furthermore, this monitor exists as a time-triggered transaction, and thus monitors availability of inventory items only when the monitor is triggered based on the configured runtime properties.

Getters with Enterprise Code

The getters for the following time-triggered (agent) transactions can take enterprise code as an additional parameter:

- Order monitor
- Shipment monitor
- Negotiation monitor
- Payment collection

When the Agent Server processes a default getter task (a getter task that picks up work for any enterprise), the server turns around and creates a getter message for each enterprise. Each of these getters by default pick up their own 5,000 orders. Therefore, if you have four defined enterprises, the first getter message results in the creation of four enterprise-specific getter messages. Those four getter messages could potentially create up to 20,000 task messages. If you have many enterprises, you may want to consider:

- Lowering the number of orders a getter puts into the message queue or
- Explicitly scheduling getters with enterprise codes (instead of using the default getter which gets orders for all enterprises).

Sort Order and Deadlocks

Deadlocks occur when two or more sessions mutually block each other to the point where neither session can progress. As a result, these sessions continue to block until the database management system kills one of the deadlocked sessions in order for the others to continue.

Deadlocks occur when two or more sessions obtain resource locks in an arbitrary fashion. For example, the following is a classic example:

```
Txn 1                                  Txn 2
Locks Record A                      Locks Record B
Tries to Lock Record B    (blocked)    Tries to Lock Record A (blocked)
```

In the example above, Txn 1 holds the lock for Record A and Txn 2 holds the lock for Record B. When Txn 1 tries to lock Record B, it becomes blocked. When Txn 2 tries to lock Record A, it also becomes blocked. Now, neither session can progress unless one of the transaction is killed.

If the resource locks were obtained in a consistent order, the deadlock does not occur. For example, all transactions agree to lock the records in ascending order (Record A then Record B).

Replaying the example above, we now have:

```
Txn 1                                Txn 2
Locks Record A                    Tries to Lock Record A (blocked)
Locks Record B
commits
                                  Locks Record A
                                  Locks Record B
                                  commits
```

In the example above, Txn 2 is delayed but not deadlocked. Both transactions eventually complete.

Sort Order

When you develop custom code, you should be aware that Sterling Selling and Fulfillment Foundation obtains YFS_INVENTORY_ITEM locks in the following sort order:

Item ID, Product Class and UOM

If you adopt this sort order, you should greatly minimize the chance of deadlocks.

Agent Throughput

In addition to the data provided by the System Management Console and the Sterling Selling and Fulfillment Foundation Statistics, you can also get application processing statistics by data mining the Sterling Selling and Fulfillment Foundation database. This technique takes advantage of the following application characteristics:

- A record is created in yfs_order_header for every new order.
- A record is created in yfs_order_line for every order line.
- A record is created in yfs_order_release_status each time the order line moves through the various states in its lifecycle.
- An audit record is created in yfs_order_audit each time an order or order line is modified.
- An audit record is created in yfs_inventory_audit each time an inventory item is modified.
- Each record has a primary key whose value is made up of two parts:
  - A date/time component in the form of year, month, date, hours, minutes, and seconds. For example, a record that was created on September 21, 2011 at 4:20:14 pm has 20110921162014 as the first part of the key).
  - A monotonically-increasing sequence number.

Order Creation Throughput

For example, in Oracle, to calculate the rate at which orders were created on a specific date (e.g., July 4, 2011), you can issue the following query:

```
select substr(order_header_key,1,10) time, count(*) as count
from yfs_order_header
where order_header_key > '20110704000000' and
      order_header_key < '20110704999999'
group by substr(order_header_key,1,10);
```

This query produces a listing like this:

```
   TIME                          COUNT
   ------------------------ ----------
   2011070406                     3333
   2011070407                     3366
   2011070408                     3333
```

Note: For DB2, you should issue throughput queries as uncommitted reads. By default, queries run at the cursor stability level. As a result, DB2 has to obtain a read share lock on the record it is reading. Queries against tables with high insert or update activities block behind records with update or exclusive locks.

For DB2 to issue the query above at the uncommitted read lock level, issue the query with the "WITH UR" option:

```
   select substr(order_header_key,1,10) time, count(*) as count
   from yfs_order_header
   where order_header_key > '20110704000000' and
         order_header_key < '20110704999999'
   group by substr(order_header_key,1,10)
   with UR;
```

In Microsoft SQL Server, issue the following:

```
   -- number of order headers
   select substring(order_header_key,1,12) "Orders", count(*) "Meas. Minute
Rate"
   from yfs_order_header
   where order_header_key like '20110704%'
   group by substring(order_header_key,1,12);

   -- number of order lines created
   select substring(order_line_key,1,12) "Order Lines", count(*) "Meas. Minute
Rate"
   from yfs_order_line
   where order_line_key like '20110704%'
   group by substring(order_line_key,1,12);
```

Order LifeCycle Throughput

Similarly, you can calculate the throughput of orders going through its various lifecycle states with the following example:

```
   select pipeline_key, status, substr(order_release_status_key,1,10) time,
          count(*) count
   from yantra.yfs_order_release_status
   where order_release_status_key > '20110704000000' and
         order_release_status_key < '20110704999999'
   group by pipeline_key,
          status,
          substr(order_release_status_key,1,10);

   PIPELINE_KEY             STATUS          TIME                          COUNT
------------------------ --------------- ------------------------ ----------
   2011070409425525425230   1100            2011102906                    13333
   2011070409425525425230   1100            2011102907                    13464
   2011070409425525425230   1100            2001102908                    13333
   2011070409425525425230   1300            2011102906                       50
   2011070409425525425230   1300            2011102907                       23
   2011070409425525425230   1300            2011102908                       50
   2011070409425525425230   3200            2011102906                    13234
   2011070409425525425230   3200            2011102907                    13477
   2011070409425525425230   3200            2011102908                    13290
```

The definition of the STATUS is found in YFS_STATUS and PIPELINE_KEY in YFS_PIPELINE. For example, status of 1100 indicates order lines being created. In the example above, there were 13,333 order lines created for one pipeline and another 4,333 order lines created in another pipeline.

Note: You should baseline the throughput of individual agents and key APIs to get an idea of the potential throughput. You should then monitor the agents in production against the baseline. This continual monitoring may reveal issues - for example, a credit authorization agency providing slower response times, issues with the database, and so forth.

Note: You can monitor the flow of the orders on an hourly basis by pivoting the data so that STATUS is in the column and TIME is in the row. For example, the data above can be displayed as follows:

```
Time              1100      1300      3200
2011070406       13333        50     13234
2011070407       13464        23     13447
2011070408       13333        50     13290
```

In the pivot example above, 13,333 order lines were created on 2011/07/04 at 06am. At that same time period, 50 order lines went to Backorder and 13,234 were Released. More importantly, one may conclude that the flow of the orders through the pipeline is good because order releases are keeping pace with order creation.

There are many ways to create pivot tables including Microsoft Excel (use Data > PivotTable and PivotChart Report...).

Order Kit Line Creation Throughput

To calculate Kit Line creation, issue the following example:

```
select substr(order_kit_line_key,1,10) time,
       count(*) count
from yfs_order_kit_line
where order_kit_line_key > '20110704000000' and
      order_kit_line_key < '20110704999999'
group by substr(order_kit_line_key,1,10);
```

This query produces a listing like this:

```
TIME                          COUNT
------------------------ ----------
2011070406                     3333
2011070407                     3366
2011070408                     3333
```

Throughput Query Limitations

As we discussed above, the throughput queries provides processing rates by counting the number of records created. If you run the throughput query against the YFS_ORDER_RELEASE_STATUS table, you get rates at which order lines move through the pipeline statuses.

Reprocessing

The throughput queries do not report "unsuccessful" work and as a result can appear skewed if you have a lot of order reprocessing. You can, however, augment these throughput queries with data from the Sterling Selling and Fulfillment Foundation Statistics (see “Statistics” on page 128).

For example, assume there are 10,000 orders available for scheduling. When the Schedule agent processes the 10,000 orders, it finds that 9,000 orders cannot be scheduled because they are either awaiting authorization or items are backordered. The throughput query reports that the Schedule agent successfully scheduled 1,000 orders but does not indicate that it tried to but was unable to schedule the other 9,000 orders. In these extreme cases, the Schedule agents appear to consume a lot of computing resources for the amount of work (as reported by the throughput query) performed.

In addition to tracking the order flow, you should also track the number of exceptions using the exception query below (see "Inbox" on page 128).

Maximum Potential Throughput

The throughput query reports actual work done within each measurement or reporting period. The rates can be less than the maximum throughput when there are idle agent threads during the reporting period - for example, when there is not enough work for the agents to process.

To calculate your agent configuration's maximum throughput, you need to create a queue of work so that all agent threads are busy the entire reporting period and the amount of reprocessing is normal or representative of your peak day.

Sterling Warehouse Management System

This section provides recommendations on how to plan, implement, configure, monitor, and tune the Sterling Warehouse Management System. Sterling Warehouse Management System' default factory (data) settings provide a simple configuration that is suitable for development, training or product familiarization. That configuration is not suitable for production except for customer with very low transaction volumes. This section guides you through the components that you have to configure for higher transaction volumes.

This section assumes that you:

- Are familiar with the basic functionality of Sterling Warehouse Management System
- Have read the common Sterling Selling and Fulfillment Foundation performance concepts in "General Recommendations" on page 91
- Have read and followed all the instructions found in the *Sterling Selling and Fulfillment Foundation: Installation Guide*
- Have read the *Sterling Selling and Fulfillment Foundation: Release Notes -- Known Issues*

Property File

The following parameters are used to influence the Sterling Warehouse Management System processing:

Table 24. yfs.properties

Parameters	Value
In the customer_overrides.properties file, add the following entries:	
yfs.yfs.solver.iterations.wavecreate	1
yfs.yfs.containerization.maxshipmentsinoneround	75

The Create Wave transaction uses an efficient constraint-based optimization engine to assign shipments to waves. This engine iteratively assigns shipments to waves and recalculate cost. Suboptimal wave assignments are discarded and another solution set attempted. You can limit the number of iterations by editing `<INSTALL_DIR>/properties/customer_overrides.properties` file and the following entry:

```
yfs.yfs.solver.iterations.wavecreate=< number of iterations >
```

For additional information about overriding properties using the `customer_overrides.properties file`, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*.

We recommend testing the Create Wave transaction for your warehouse at the default iteration level (1) and at a higher level (e.g., iteration of 5) to see if there is an appreciable difference in processing times and shipment wave assignment. In some cases, setting `yfs.solver.iterations.wavecreate` to a lower number results in marginal differences in the wave assignment but a significant reduction in processing times.

One way to test the efficacy of the `yfs.solver.iterations.wavecreate` setting is to run controlled tests in your QA environment. For example, an approach is to:

- Create a reasonable number of shipments that are ready for the Create Wave processing.
- Take a backup so that you can repeat the test.
- Run the Create Wave with increasing values and assess the resulting waves.
- Restore the database and repeat above.

The `yfs.containerization.maxshipmentsinoneround` parameter sets the number of shipments considered for containerization per iteration. The default is 75. With that setting, 75 shipments at a time are containerized and committed. This process reduces the number of record locks held.

WMS Agents

This section describes the runtime or performance characteristics of the Sterling Warehouse Management System agents or transactions.

Scheduling Using Agent Criteria Group

About this task

Sterling Warehouse Management System customers with a large number of small warehouses that require wave planning may want to use the "agent criteria group" wave scheduling feature. By default, the wave processing agents (e.g., Create Wave, Release Wave) are triggered individually for each node.

You could use utilities such as CRON on Unix to automatically trigger each of the nodes at some interval. However, if you have a 100 nodes and you would like to issue the triggers every hour, you would need to set up CRON for the 100 nodes for each agent.

An alternative is to use agent criteria groups. This can be accomplished in the following steps:

- First define an agent criteria group in the Sterling Application Platform > System Administration > Agent Criteria Groups.
- Next, assign one or more nodes to the appropriate agent criteria group in the Sterling Application Platform > Participant Modeling. For each node, go to the Organization Details > Roles & Participation dialog box. Select the appropriate agent criteria group.

- Next, in the Sterling Application Platform > Process Modeling > Wave > Outbound Picking process model, select Transaction on the left screen. Select the appropriate transaction (e.g., Create Wave, Release Wave, and so forth).
- Create a new Agent Criteria Definition. In that Agent Criteria Details > Criteria Parameter, assign the agent criteria group to the appropriate parameter.

When you start the agent and trigger for this agent criteria, you start the transaction for the agent criteria group. This in turn starts the transaction for each of the nodes assigned to that agent criteria group.

For example, you may define a node group by time zones or regional groups.

You may want to continue scheduling the wave agents for large warehouse nodes (those with high shipment volumes) individually. Putting large nodes in node groups causes nodes could result in nodes waiting for the large nodes to complete their processing.

See the *Sterling Selling and Fulfillment Foundation: Application Platform Configuration Guide* for more information.

WMS Agents: Processing Concurrency

For scalability, the Sterling Selling and Fulfillment Foundation agents are designed to run in multiple parallel threads. Some of the Sterling Warehouse Management System agents, by design, run single threaded for a given warehouse node. These agents include the:

- Create Wave
- Release Wave

WMS Agents: Create Wave: The Create Wave agent assigns eligible shipments and shipment lines for a warehouse node into optimum waves. By design, only one Create Wave transaction can concurrently run for a warehouse node.

You can, however, run multiple Create Wave transactions concurrently if you have multiple warehouse nodes - provided, as stated above, only one Create Wave transaction is running per warehouse node. This restriction is enforced by the application.

You can specify the number of threads in the Applications Manager (see "Integration Adapters/Agents" on page 100).

WMS Agents: Release Wave: The Release Wave transaction creates pick tasks from shipment lines in a wave. As part of the processing, this transaction serializes access to inventory item records for that node by locking YFS_TRANSACTION_LOCK records to prevent concurrent updates to inventory items during Release Wave processing. There is a YFS_TRANSACTION_LOCK record for each inventory item/node combination.

As a result, you should only run one Release Wave thread per warehouse node.

Allocate Task Agent

You may want to use the Allocate Task agent if you process large waves (for example, over 500 line waves). The Release Wave acquires locks on the YFS_LOCATION_INVENTORY record before managing the inventory at those locations. For large waves, the locks held by the Release Wave agent could impact other transactions, such as picks, moves, etc., that also need YFS_LOCATION_INVENTORY record locks.

You can direct the Release Wave agent to defer inventory location updates. This allows the Release Wave agent to complete its processing without acquiring these locks. A subsequent agent, the Allocate Task, acquires the YFS_LOCATION_INVENTORY record locks and update the inventory at the location on a task basis. The amount of time that the record lock is held is much shorter (essentially for the duration of processing that task).

For more information about AllocateTask agent, refer to the *Sterling Selling and Fulfillment Foundation: Warehouse Management System Configuration Guide*.

Agents Between Create Wave to Release Wave: In general, all the agents from Create Wave through to Release Wave inclusive, including all custom agents, should be run in a single threaded fashion for each agent criteria group (see "Scheduling Using Agent Criteria Group" on page 136) or for each node if you want to ensure the waves are released in the order that they are created.

For example, assume you have 10 nodes - N01 to N10. Assume also that:

- Nodes N01 to N03 are assigned to agent criteria group 1.
- Nodes N04 to N08 are assigned to agent criteria group 2.
- N09 and N10 are scheduled individually.

Then for a given agent (say 'Assign Lane') you should run 4 JVMs (one for each agent criteria group and one each for nodes N09 and N10). Each of these JVMs have to be configured to run with only one thread per transaction. These agents and transactions can run in parallel.

As we mentioned above, this is only necessary if you need your waves to be released in the order in which the waves were created. If the ordering is unnecessary, you can run these transactions in parallel.

WMS Agents: Purge

We strongly recommend running the WMS Task Purge agent on a daily basis. This agent is used to keep the YFS_TASK table small by moving completed YFS_TASK records to the YFS_TASK_H history table. YFS_TASK table that grows unchecked could affect the performance of WMS task-based transactions, such as next task suggestion.

Long Running Transactions in DB2

The WMS application is made of both short and long running transactions. Short transactions are characterized by a small number of database records read and possibly updated within a short processing time under a single unit of work. At the end of the processing (or unit of work), the workload commits the transaction. Database locks are released.

In contrast, some workloads, by their nature, are long running transactions. For example, the Create Wave transaction groups eligible shipments and shipment lines into optimum waves based on customer-specified wave constraints. The length of the processing time depends on many factors such as the number of shipments, the complexity of the optimization, the wave constraints, and so forth.

You should consider the following when configuring a DB2 database:

- Monitor the amount of transaction log usage - specifically monitor TOTAL_LOG_USED, TOTAL_LOG_USED_TOP, SEC_LOG_USED_TOP, and SEC_LOGS_ALLOCATED monitor elements. You should ensure that the amount of log used does not approach the capacity of the primary logs and that DB2 is not spilling over to secondary logs.
- Monitor the APPL_ID_OLDEST_XACT monitor element - see which transaction holds the oldest transaction log entry.
- Enable NUM_LOG_SPAN parameter to safeguard against a long running transaction holding too many logs that could result in a situation where all the transaction logs are full. Please see NUM_LOG_SPAN discussion in “Recommended DB CFG Parameters” on page 63.

Java Stack Size

You have to increase the Java stack size if you plan to create waves or batch waves with more than 4,000 shipment lines that are assigned to a single shipment group. You can use the following table as a guideline.

Table 25. Stack Size Recommendations for Create Wave/Batch Wave

Shipment Lines per Shipment Group	Stack Size
4,000	2MB
10,000	4MB
15,000	6MB
20,000	8MB
25,000	10MB

Note: Since there are various factors that govern the JVM's stack size, you should increase the JVM stack size argument when you encounter a java.lang.StackOverflowError exception.

Please see “Optional JVM Command Line Settings: Stack Size” on page 27 for instructions on how to set the Java thread stack size.

Sterling Selling and Fulfillment Foundation UI Console: Asynchronous Manifest Closure: Sterling Warehouse Management System allows you to close manifests from the Sterling Selling and Fulfillment Foundation UI synchronously or asynchronously. By default, in synchronous mode, the user has to wait for the request to complete. Depending on the number of shipments in a manifest, the manifest close operation can take a long time and may result in users believing the UI is "locked up".

Sterling Warehouse Management System allows you to configure the system so that manifests are closed asynchronously. In this mode, the request from the UI creates a message for the CLOSE_MANIFEST agent. The screen is released to the user after the message is created. To change to the asynchronous manifest close mode, edit <INSTALL_DIR>/properties/customer_overrides.properties file and the following entry:

```
yfs.yfs.closemanifest.online=N
```

If this property is set, the user needs to configure the CLOSE_MANIFEST agent for processing manifest closures requests. Users also need to check for alerts/errors in the Alert Console. The manifest status "Closure Failed" indicates occurrence of errors while closing a manifest. For additional information about overriding properties using the `customer_overrides.properties file`, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*.

WMS User Interfaces: Asynchronous Batch Confirmation

Sterling Warehouse Management System allows you to confirm batch sheets from the Sterling Selling and Fulfillment Foundation UI synchronously or asynchronously. By default, in the synchronous mode, you have to wait for the request to complete. Depending on the number of tasks in the batch, the batch confirmation operation can take a long time and may result in users believing the UI is "locked up".

Sterling Warehouse Management System allows you to configure the system so that batches are confirmed asynchronously. In this mode, the request from the UI creates a message for the REQ_BATCH_COMPLETION agent. The screen is released to the user after the message is created. To change to the asynchronous confirm batch mode, edit `<INSTALL_DIR>/properties/customer_overrides.properties` file and the following entry:

```
yfs.yfs.confirmbatch.online=N
```

If this property is set, the user needs to configure the REQ_BATCH_COMPLETION agent for processing the batch confirmation requests. The users also need to check for alerts/errors in the Alert Console. The batch status "Completion Failed" indicates occurrence of errors while confirming a batch. For additional information about overriding properties using the `customer_overrides.properties file`, see the *Sterling Selling and Fulfillment Foundation: Properties Guide*.

Mobile Devices

The Sterling Warehouse Management System application supports two mobile device displays - a VT100 character-based display and a Microsoft PocketPC Graphical UI display. The PocketPC display interacts with the Sterling Warehouse Management System with HTML. The VT100 display sends VT100 characters.

You may want to consider using the VT100 RF display if you have limited network bandwidth.

IBM Sterling Web

Optimizing Static Content for Sterling Web

To optimize the application server performance, you may want to allow redirection to your HTTP server to serve up static content such as images. Sterling Selling and Fulfillment Foundation contains the ability to configure items with an alternate URL to point to the image file.

To configure item images in Sterling Selling and Fulfillment Foundation, you can use the IBM Sterling Business Center module. Follow the steps in the *IBM Sterling Business Center Item Administration Guide* to specify the location of the server and the path to the image file.

Enabling Compression for Sterling Web Pages

IBM Sterling Web pages may be up to 1 MB in size, due to extensive JavaScript libraries needed to support EXT-JS. You can deploy Apache HTTP proxy with compression enabled in order to accelerate the page loading time of the application. Apache proxy server may be configured to perform gzip compression, providing an inexpensive software alternative to onboard compression. For more information on software compression, consult Apache's documentation on the mod_deflate module.

Alternately, hardware compression may be deployed to reduce the size of HTTP and HTTPS content. For more details on setting up hardware compression using the F5 Big-IP device, see “HTML Compression” on page 93.

API Security for Sterling Web

IBM Sterling Web may require an increased number of JDBC connections when API Security is enabled. By default, API security is disabled. If enabled, monitor the maximum usage of JDBC connections and resize the connection pool accordingly.

Application Server Heap Sizing

For Sterling Web, follow the guidelines given in the “Starting Recommendations” section of the respective application server JVM section, and tune the heap parameters for your maximum expected application load. See the *Sterling Selling and Fulfillment Foundation: Catalog Management Concepts Guide* for further information. As a guideline, IBM has used 4 GB heaps for its tests with item index caching and default reference data caching turned on.

Chapter 9. Performance Tuning Considerations for BI (Business Intelligence)

This chapter will be provided in the next release of the product.

Chapter 10. References

Some of the books referenced in this document and that we strongly recommend include the following:

Oracle

- *Oracle11g SQL Reference*, Oracle
- *Oracle11g Real Application Cluster (RAC) Administration*, Oracle

IBM DB2

Refer to the IBM Information Center for DB2 planning, administrative, performance, and implementation information:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp

Sun Java Virtual Machine

- *The Java HotSpot Performance Engine Architecture*, Sun Microsystems, http://java.sun.com/products/hotspot/whitepaper.html
- Refer to extensive Java SE Hotspot information on http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html

HP Java Virtual Machine

- HPjtune - visualization tool for HP JVM GC activities - http://www.hp.com

IBM Java Virtual Machine

- Mattias Persson, *Java technology, IBM Style, Garbage Collection Policies, Part 1,* IBM
- Mattias Persson, Holly Cummins, *Java technology, IBM Style, Part 2, Garbage collection with the Extensible Verbose Toolkit,* IBM
- Sumit Chawla, *Fine-Tuning Java Garbage Collection Performance, How to detect and troubleshoot garbage collection issues with the IBM Java Virtual Machine*, IBM
- IBM Developer Kit and Runtime Environment, Java Technology Edition, Diagnostic Guide, Version 5.0, SC34-6650

Oracle WebLogic

- Oracle WebLogic Server Performance and Tuning 10*g* Release 3 (10.3) http://download.oracle.com/docs/cd/E12840_01/wls/docs103/perform/index.html
- System Administration for Oracle WebLogic Server 10*g* Release 3 (10.3) - http://download.oracle.com/docs/cd/E12840_01/wls/docs103/admin.html

IBM WebSphere

- *IBM WebSphere Application Server Tuning Guide*, IBM
- Mark Endrei, *IBM WebSphere V7 Handbook*, IBM Redbook
- WebSphere InfoCenter for Version 7.0

JBoss

- JBoss Enterprise Application Platform Documentation Library http://www.redhat.com/docs/en-US/JBoss_Enterprise_Application_Platform/EAP_4_3_Portal.html#EAP4_3_Rel

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