



WebLogic Server 8.1

Performance Tuning

Russell Raymundo
russellr@bea.com





AGENDA

Performance Objectives

Tuning Methodology

Performance Tuning

Operating System

Database

JRockit JVM

WebLogic Server

Monitoring Performance

Application Analysis

Q & A

Demo

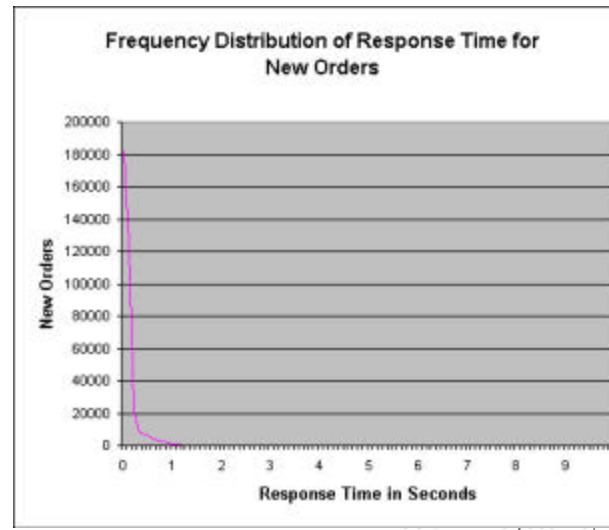
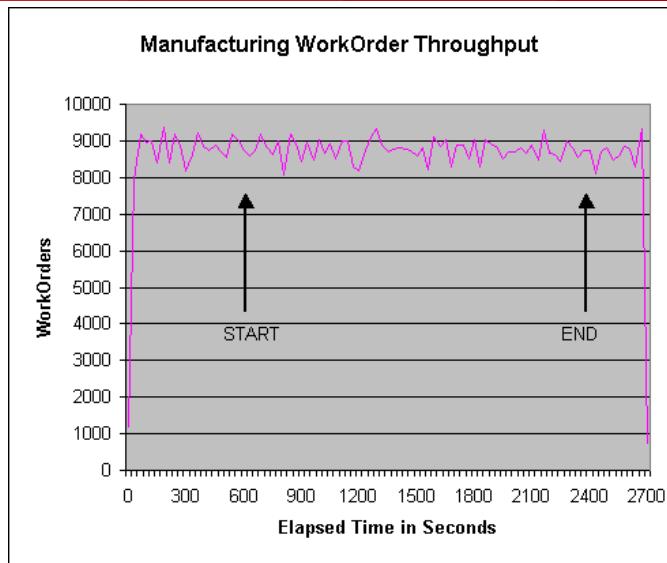


Understanding Your Performance

Objectives

Application Performance Requirements

- Performance Requirements
 - Throughput
 - Response Time
 - Reliability
 - Scalability
- Understand your Application
 - Number of Users/Requests?
 - User activity? Hotspots?
 - Amount/Size of Data?
 - Database intensive? Why?
- Constraints
 - Configuration (HW & SW)
 - Topology
 - Interoperability
 - Cost





Design for Performance

- Keep it simple
 - Overly complex or poorly designed applications hurt both performance and maintainability
- Emphasize caching
 - Look for opportunities in web server, app server, database
- Design Patterns can help performance
 - Session Facade pattern reduce external method invocation
 - Value Object pattern reduce individual get/set requests
- Measure early in development



Tuning Methodology



Systematic Approach

- Many factors can impact performance and scalability of the system.
 - Application design, system topology, database configuration and tuning, disk and network IO activity, operating system configuration, and application server controls.
- No “Silver bullet” – Every layer and subsystem matters
- Important to have process due to complexity of tuning an overall system



Iterative Process

1. Collect data: Gather performance data as the system is exercised using stress tests and performance monitoring tools to capture relevant data
2. Identify bottlenecks: Analyze the collected data to identify performance bottlenecks
3. Identify alternatives: Identify, explore, and select alternatives to address the bottlenecks
4. Apply solution: Apply the proposed solution.
5. Test: Evaluate the performance effect of the corresponding action



Techniques

- In order to measure impact of a changes, make only a single change at a time.
- Once a given bottleneck is addressed, additional bottlenecks may appear, so the process starts again by collecting performance data and initiating the cycle, until the desired level of performance is attained.



INTEGRATE



BUILD



PORTAL

Performance Tuning





Operating System Tuning

- Operation System Selection
 - Proprietary Unix, Linux, Windows...
 - Version
- Windows tuning is generally minimal
 - Since HTTP server consume many TCP sockets. Decrease the TcpTimedWaitDelay and increase MaxUserPort

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\  
    TcpTimedWaitDelay=dword:0000001e and TcpTimedWaitDelay=dword:ffff]
```
- Unix and Linux require more specific tuning
 - On Solaris, for better TCP socket performance, reduce the tcp_time_wait_interval to 30 seconds

```
ndd -set /dev/tcp tcp_time_wait_interval 30000
```
 - Increase number of file descriptors
Parameter changes in /etc/security/limits.conf (soft nofile 8192 hard nofile 8192)
 - On Linux, for better packet transfer performance, set the /sbin/ifconfig lo mtu parameter to reduce fragmentation of large packets:

```
/sbin/ifconfig lo mtu 1500
```



Operating System Tuning

- For examples of the actual tuning done on specific configurations for an end-to-end benchmark (SPECjAppServer2002) see:
 - <http://www.spec.org/osg/jAppServer2002/results/jAppServer2002.html>
- Sample OS tuning from the driver (client) for a SPECjAppServer result:
 - Parameters added to /etc/sysctl.conf

```
fs.file-max=65535  
net.ipv4.tcp_sack=0  
net.ipv4.tcp_timestamps=0
```
 - The Linux max thread limit is 1024 by default. Rebuilt libpthread.so after increasing the limit to 8192.
 - Changed max thread stack size to 256K from 2048K.



Good Database Design

- Database is common bottleneck
- Distribute the database workload across multiple disks
 - Proper organization and sizing of tables
- Reduce contention
 - Avoid table scans - Every query must hit a good, selective index.
 - Partition tables
- Careful with ORDER BY clauses – Large results sets require sorting even if only reading a few rows.



Optimize Database Disk I/O

- Use a larger block/buffer size
- Place log file on separate disk
- Increase logfile size to reduce frequency of checkpoints
- Use Raid-0 “striping”
 - To improve Read/Write data transfer performance
- Use Raid-1 “mirroring”
 - To increase redundancy and improve Read latency



JVM Tuning

- Garbage Collection is biggest factor in JVM performance
 - Automatic Memory Management – Clean up objects from heap that are no longer in use
 - Applications that create unnecessarily large numbers of objects make problem worse
 - for best performance set the minimum and maximum values to be the same
 - use 80-85% of available RAM
- Depending on application, spin locks could be beneficial
 - JRockit 1.4.2 On by default
 - XXenablefatspin
 - Sun's JDK – Off by default (Windows default On previous to 1.4.2)
 - XX:+UseSpinning -XX:PreBlockSpin=100



JVM Tuning – JRockit 1.4.2

- Garbage Collector
 - Gencon – Generational Concurrent
 - Singlecon – Single-spaced Concurrent (Default for –client option)
 - Parallel – Single-spaced Parallel (Default for –server option)
- Two Garbage collector options
 - Generational – Heap divided into two sections, an old generation and a young generation (nursery).
 - Single-spaced – All objects live out their lives in a single space on the heap
- Two Garbage Collection algorithms
 - Concurrent – Marking and sweeping “concurrently” with all other processing
 - Parallel – Stops all Java threads when heap is full and uses every CPU to perform a complete mark and sweep



JVM Tuning – JRockit 1.4.2

High Performance

- Use parallel garbage collectors (-Xgc:parallel)

High Responsiveness

- Use single-spaced or generational concurrent garbage collectors (-Xgc:singlecon or -Xgc:gencon)
- Set the size of the nursery for generational (-Xns)
 - If you are creating a lot of temporary objects you should have a large nursery
 - Larger nurseries usually result in slightly longer pauses, so, while you should try to make the nursery as large as possible, don't make it so large that pause times are unacceptable
 - You can see the nursery pause times in WebLogic JRockit JVM by starting the JVM with -Xgcpause or -Xverbose:gc



JVM Tuning – JRockit 1.4.2

- -XXaggressive:opt,memory
 - Opt – performs optimization at a higher frequency in the from the start and then gradually lower its frequency
 - Memory – uses available memory aggressively such as large pages where available, increase heap compaction, etc.
- -Xgc prio:throughput or pausetime
 - Dynamically choose between which collector and options based on goal of throughput or pausetimes.



Determine Optimal Heap Size

1. Monitor the performance of WebLogic Server under maximum load while running your application
2. Use -verbosegc (Sun JDK) / -Xverbose:gc (JRockit)

For example:

```
% java -server -ms32m -mx200m -Xverbose:gc-classpath $CLASSPATH -  
Dweblogic.Name=%SERVER_NAME% -Dbea.home="C:\bea"  
-Dweblogic.management.username=%WLS_USER%  
-Dweblogic.management.password=%WLS_PWD%  
-Dweblogic.management.server=%ADMIN_URL%  
-Dweblogic.ProductionModeEnabled=%STARTMODE%  
-Djava.security.policy="%WL_HOME%\server\lib\weblogic.policy"  
weblogic.Server  
>> logfile.txt 2>&1
```



Determine Optimal Heap Size

3. Analyze the following data points:

- How often is garbage collection taking place? In the weblogic.log file, compare the time stamps around the garbage collection.
- How long is garbage collection taking? Full garbage collection should not take longer than 3 to 5 seconds. Lower heap if major GC time is greater.
- What is your average memory footprint? In other words, what does the heap settle back down to after each full garbage collection? If the heap always settles to 85 percent free, you might set the heap size smaller.

4. Make sure that the heap size is not larger than the available free RAM on your system



Execute Queue

- Application Server Tuning focuses on execute queue and threads
- Basic operations of queues/threads
 - Socket muxer places requests on queue
 - Threads for that queue pick up and process requests from start to finish
 - Queue size will grow if incoming requests exceed throughput of system



Execute Queue

- Application Specific Execute Queue
 - Separate execute queues and pools of threads assigned to specific web applications or EJB applications
 - EJBs use the `dispatch-policy` element in `weblogic-ejb-jar.xml`
`<dispatch-policy>MyQueue</dispatch-policy>`
 - Web Apps use the `wl-dispatch-policy` in `weblogic.xml` or the `servlet param-name wl-dispatch-policy` in `web.xml`
- Non-Configurable Execute Queue
 - `Weblogic.admin.HTTP` and `weblogic.admin.RMI`
 - Reserved for the administrator console.
 - `Weblogic.kernel.System` and `weblogic.kernel.NonBlocking`
 - Internal queues used for deadlock prevention, triggers and application polling



Tuning Execute Queue

- Weblogic allows the tuning of thread count per execute queue
- Default Execute Queue is `weblogic.kernel.Default`
 - Default thread count is 15 for Development Mode and 25 for Production Mode
- Best thread count depends on application and machine
 - Ideally it will keep all processors busy without unnecessary context switching which could degrade performance.
 - I/O operations cause threads to block
- Iterative testing is best approach to determine count
 - Test with realistic environment (database contents, load, etc)
 - Application should not be the bottleneck so you can concentrate on application server



Tuning Execute Queue

▪ Guidelines

- Increase the thread count if the CPU is under utilized and queue length is high. This should make better use of the CPUs.
- Decrease the thread if CPU is high and queue is backing up. Also explore other avenues to lower CPU utilization such as JVM parameters or application issues
- Other bottlenecks such as database could prevent thread queue changes to make a performance difference
- Application-Specific Queues
 - Only affects the assigned queue during the I/O muxer operation
 - Multiple applications co-located on the same Weblogic Server instance can also use this technique to throttle less-critical requests (using a smaller thread pool)



Web

- JSP/Servlet hits can account for large percentage of the CPU time
- Always serve static content (html, img, js, ...) from web server
- Disable JSP page checks and servlet reloading
 - Set jsp-param pageCheckSeconds=-1
 - Servlet-reload-check-secs=-1 in container descriptor or ServletReloadCheckSecs=-1 in config.xml WebAppComponent



Web – Http Session

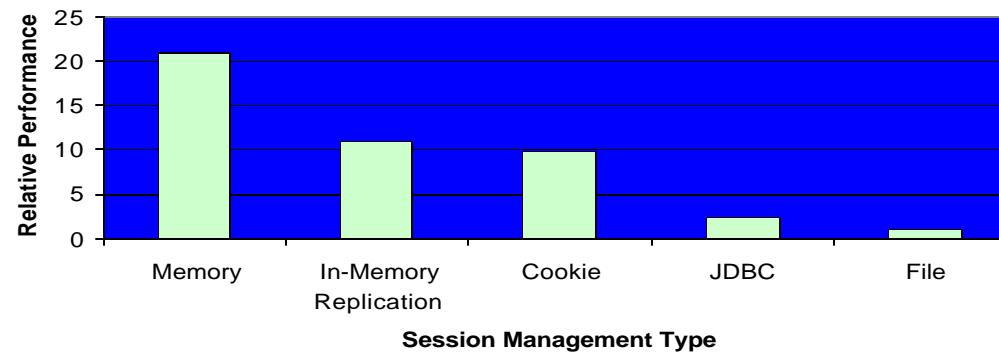
- Manage HTTP session state carefully
 - Minimize session state
 - Use multiple objects in session to optimize replication
 - Choose appropriate session persistence mechanism
 - Local memory, cookie, file, JDBC, and in-memory replication

```
weblogic.xml
<session-param>
  <param-name>PersistentStoreType</param-name>
  <param-value>memory</param-value>
</session-param>
```

- Use HttpServletRequest for data pass between webapp components during a single request (e.g. from Action class to JSP page)
- Clean up by removing session objects in a timely manner

Web – Http Session Persistence

- Five different built-in implementations of session persistence
 - Memory (local, non-replicated)
 - Cookie (session state on the client)
 - replicated (in-memory replication for clustering)
 - File (file system based)
 - Jdbc (uses database connection pool)





Server Tuning

- WebLogic Server Native I/O Performance Packs
 - Platform-optimized native socket multiplexes can improve performance
 - Increase the number simultaneous connections that can be handled
- Pure-Java socket reader implementation
 - Mainly for platforms with no performance packs
 - Can be tuned through Socket Reader threads

```
<Server ThreadPoolPercentSocketReader="30">
```
- Tuning connection backlog buffering
 - The AcceptBackLog attribute specifies how many TCP connections can be buffered before refusing additional requests, default is 50
- Tuning chunk size can improve performance with large payloads
 - -Dweblogic.Chunksize=n (The default is 4K)
 - Should be set on both the server and client JVM



JDBC Connection Pooling

- JDBC Connection pools
 - Start with a capacity of half the number of threads
 - If application uses database heavily, closer to 90-100% might help
 - Requests in excess of capacity have to queue
 - Set InitialCapacity = MaxCapacity
- Prepared statement cache size
 - WebLogic Server can cache and reuse prepared and callable statements used by applications
 - Reduces both network roundtrips and preparation work in the database
 - Each connection in a connection pool has its own individual cache
 - The default JDBC connection pool is set to 10
 - Each prepared statement cache is an open cursor in the database



JDBC Tuning

- Tuning the JDBC Connection Pool
 - PinnedToThread
 - If true, pins connections to threads
 - Avoids connection-pool contention
 - May result in more than optimal number of DB connections
 - AutoConnectionClose (new in 8.1 SP3)
 - If true, turns on JDBC connection leak detection and automatic cleanup
 - True by default, consider setting to false
 - Puts greater pressure on garbage collection, causing performance degradation
- JDBC Driver
 - Oracle 10g driver significantly faster than Oracle 9.2 driver

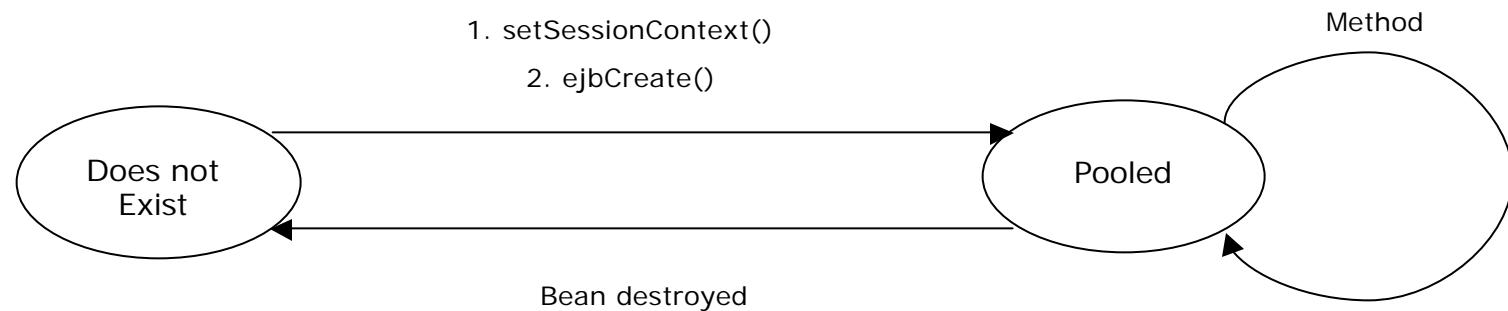


EJB

- Pooling
 - Stores anonymous beans
 - Stateless Session Beans, MDBs and Entity Beans
- Caching
 - Contains instances that have an identity
 - Stateful Session Beans
 - CMP Entity Beans
- JNDI Lookup Strategies
 - JNDI lookups are expensive. Cache home interfaces and Datasource references
- Local interfaces
 - Use for ejb-to-ejb calls within same application
 - Alternatively, use call-by-reference

Stateless Session Beans

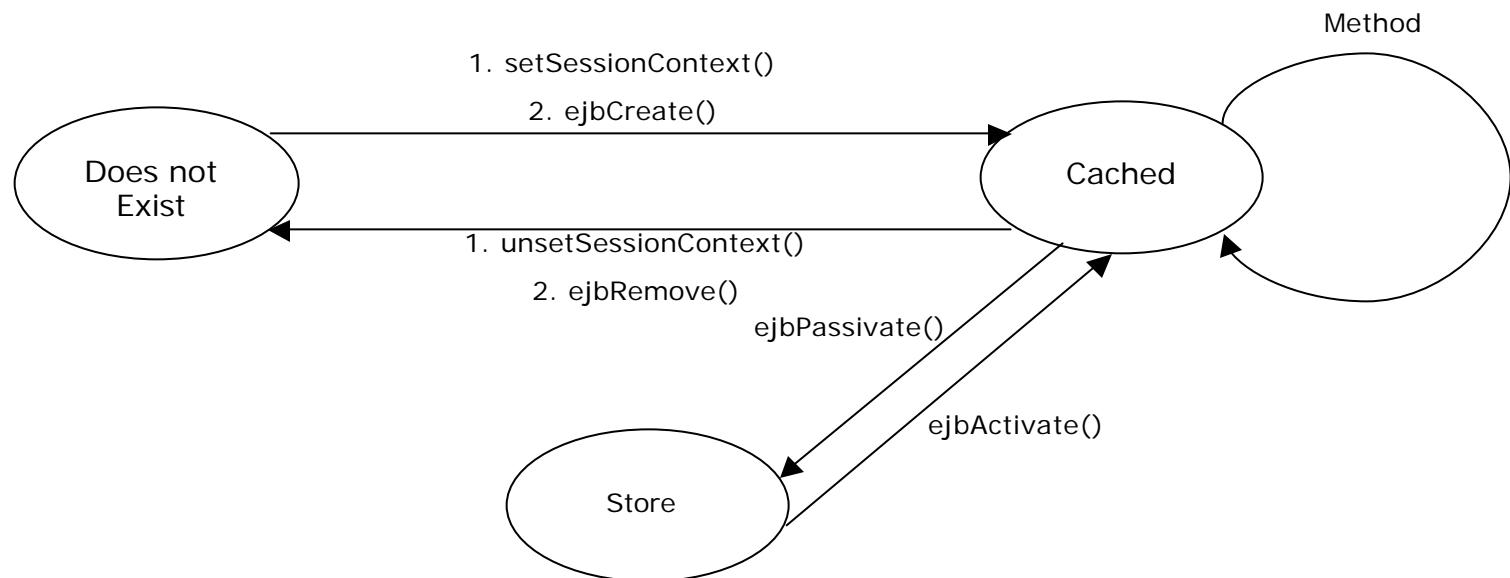
- Lifecycle



- Controls parallelism
`<max-beans-in-free-pool>`
- Can avoid expensive one-time work
`<initial-beans-in-free-pool>`

Tuning Stateful Session Beans

The Stateful Session Bean Lifecycle





Tuning Stateful Session Beans

- In general,
 - Use judiciously, specially if replicated
 - Replication is an expensive operation
 - Always call ejbRemove()
 - Unused beans occupy space in cache
 - Container could incur passivation overhead

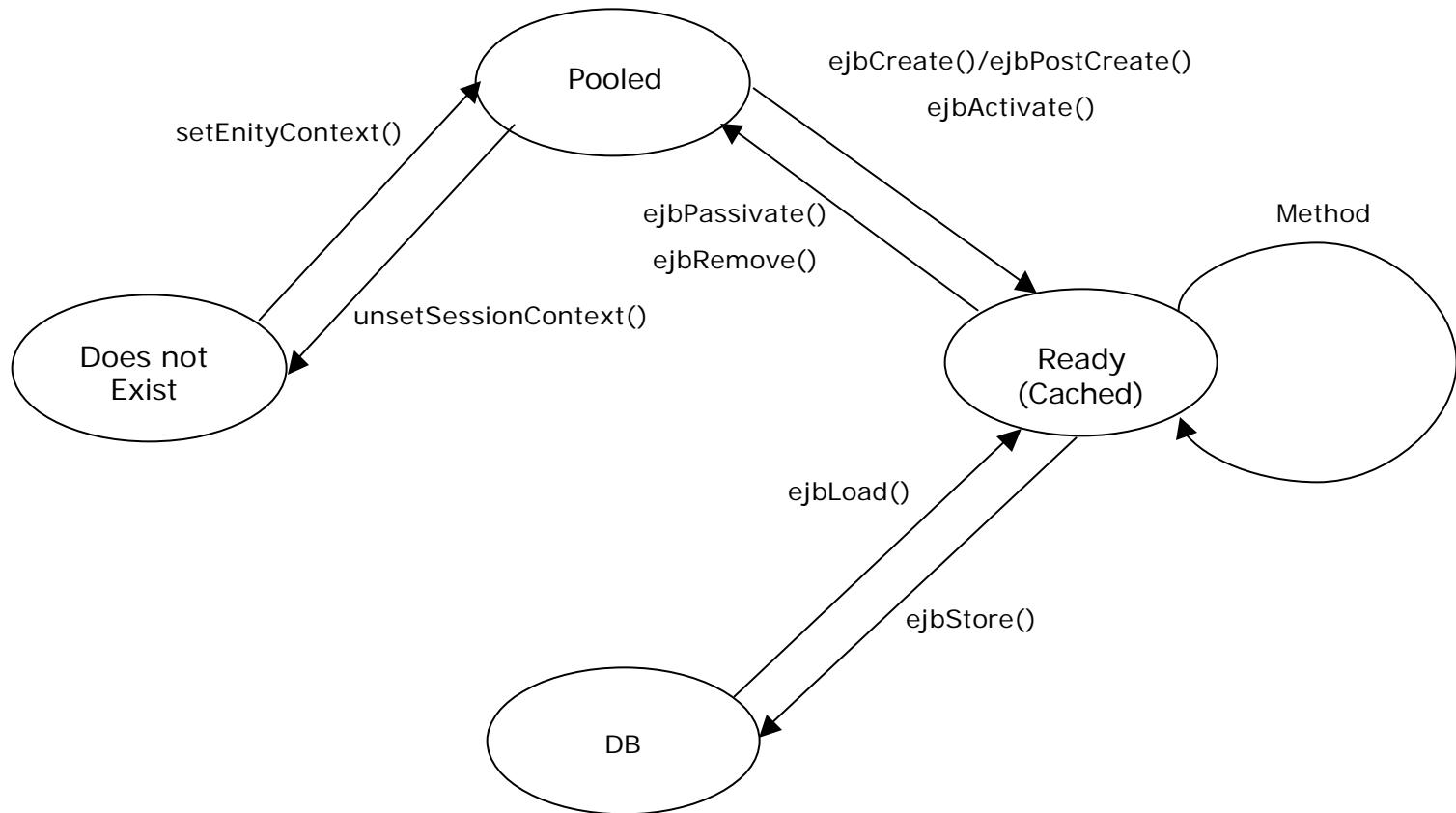


Tuning Stateful Session Beans

- The Stateful Session Bean Cache
 - <max-beans-in-cache>
 - Set to expected number of concurrent users
 - <idle-timeout-seconds>
 - Bean is passivated after this interval
 - Set to maximum expected think-time for requests
 - Also controls the session-timeout trigger interval
 - <session-timeout-seconds> (new in 8.1 SP3)
 - Bean is destroyed after this interval
 - Set to larger than <idle-timeout-seconds>

Tuning CMP Entity Beans

The Entity Bean Lifecycle





Tuning CMP Entity Beans

- Concurrency-control
 - In increasing order of cache-friendliness:
 - Exclusive, Database, Optimistic, Read-only
- 95% of CMP bean tuning is caching



Tuning CMP Entity Beans

- Tuning persistence parameters
 - Field-groups
 - Group commonly used fields and associate with finders
 - Incorrectly specified field-groups could result in additional DB access
 - Tuned-updates
 - Always on
 - Prevents unnecessary DB writes



Tuning CMP Entity Beans

- Caching

- <cache-between-transactions>
- <max-beans-in-cache>
 - Set larger values for more frequently-used beans
- The cache is not accessed for custom-finders
- <include-updates>
 - Set to true by default (false for optimistic concurrency), consider setting to false

- Pooling

- <max-beans-in-free-pool>
 - Set appropriately if setEntityContext() is expensive

Tuning CMP Entity Beans

- Tuning persistence parameters
 - <enable-batch-operations>
 - Uses JDBC 2.0 batch operations to store multiple beans using single DB operation
 - Set to true (default)
 - Implicitly defers all updates and inserts to the end of tx



Tuning CMP Entity Beans

- Tuning persistence parameters
 - CMR caching
 - Container loads related beans using single SQL when enabled
 - Can result in dramatic performance improvement
- Transaction Management
 - Transaction attribute
 - Transaction isolation
 - READ_COMMITTED, REPEATABLE_READ,
READ_COMMITTED_FOR_UPDATE, SERIALIZABLE
 - Preferable alternative to READ_COMMITTED_FOR_UPDATE is <use-select-for-update>
 - Preferable alternative to REPEATABLE_READ is optimistic-concurrency+cache-between-transactions+verify-reads
 - Caveats for alternatives: DB must support exclusive read-locks



Tuning Message Driven Beans

- MDB lifecycle similar to Stateless Session Beans
- The MDB Pool
 - Controls parallelism
 - <max-beans-in-free-pool>
 - Can avoid expensive one-time work
 - <initial-beans-in-free-pool>
- The number of receiver for MDB
 - Running in the default queue
Min ((default_pool_size) / 2 + 1, <max-beans-free-pool>)
 - Running in a separate thread queue (dispatch-policy)
Min (config-pool-size, <max-beans-free-pool>)



JMS

- Excellent reference:
 - "BEA WebLogic JMS Performance Guide" at edocs.bea.com
- Message Design Tips
 - Serialization costs vary by message type
 - Generally Byte, Stream < Object, Map < Text, XML
 - Avoid use of Strings in message properties
 - Minimize message size, consider compressing large messages
 - Message selectors are expensive (especially Xpath)
 - Use Indexed Topic Subscribers if possible
 - Use multiple destinations instead of selectors
 - Message paging degrades performance (disabled by default)

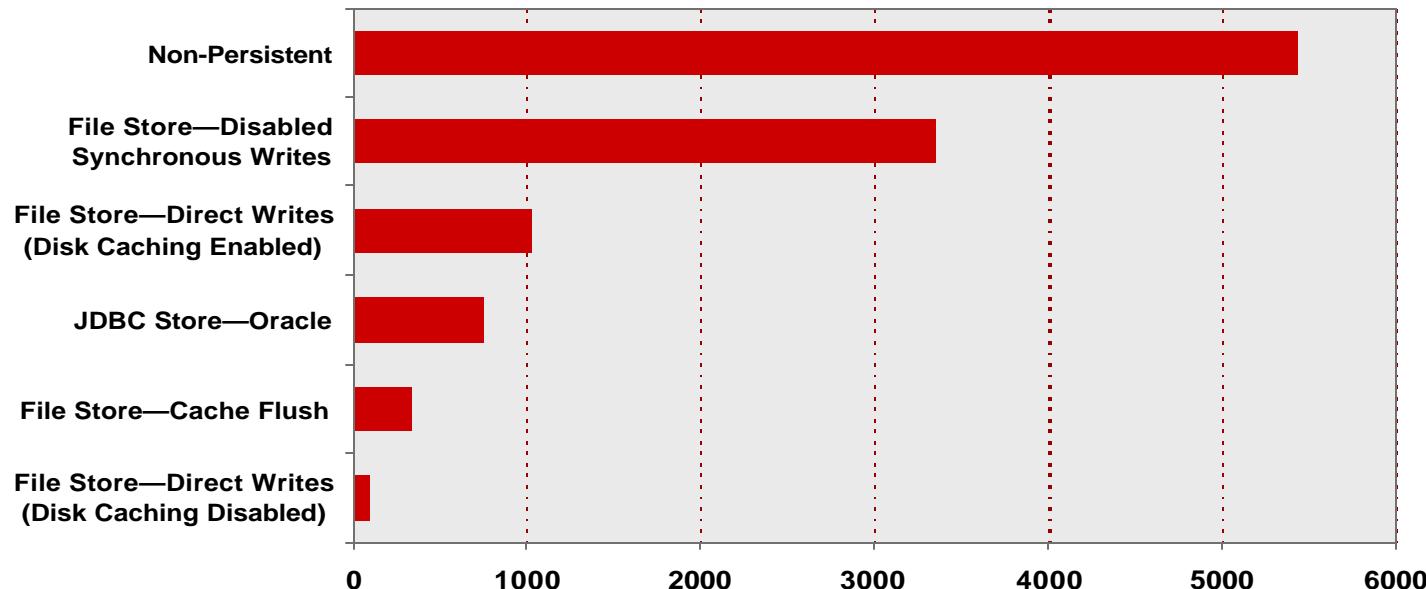


JMS

- Use flow control in cases of receiver overflow
 - Set quotas on destinations and increase blocking send timeouts
 - Tune connection factory flow control
 - Use request-response design where appropriate (receivers send periodic acknowledgements to sender side queue)
- Asynchronous consumers generally faster than synchronous
 - Tune connection factory's MessagesMaximum for asynchronous consumers (unless ordered redelivery in use)
- Choose appropriate acknowledgement mode (some affect QOS)
- Use distributed destinations for cluster scalability
- Enable automatic pooling of JMS resources in EJBs and servlets by providing resource-ref declarations for connection factories and destinations

JMS

- Choose delivery mode and store based on performance/reliability tradeoff
 - Primary advantage of JDBC store is ease of failover—alternative is file store with shared disk





Transaction Performance

- Keep transaction short
- Choose appropriate isolation level
- Minimize distributed transaction (2PC) costs
 - Use local transactions and non-XA drivers where possible
 - Experiment with various JDBC drivers for your database (XA driver performance is highly variable)
 - Make sure database server CPU is not the bottleneck
 - Use separate physical drives for WLS transaction logs and JMS file stores
 - Use of Direct-Write for transaction log file may improve performance with hardware write-cache enabled
 - XA requires more threads due to longer running transactions



Transaction Performance

- JMS and JDBC in a transaction
 - JMS and JDBC in a transaction (even if it is the same database) requires 2PC
 - JMS has to use XA enabled Connection Factory
 - JDBC has to use an XA driver
 - JDBC non-XA driver can be used with "2PC emulation" (`EnableTwoPhaseCommit="true"`). This is fake XA.
- JMS and JMS in a transaction
 - Using 2 destinations from different JMS servers in a transaction will use 2PC



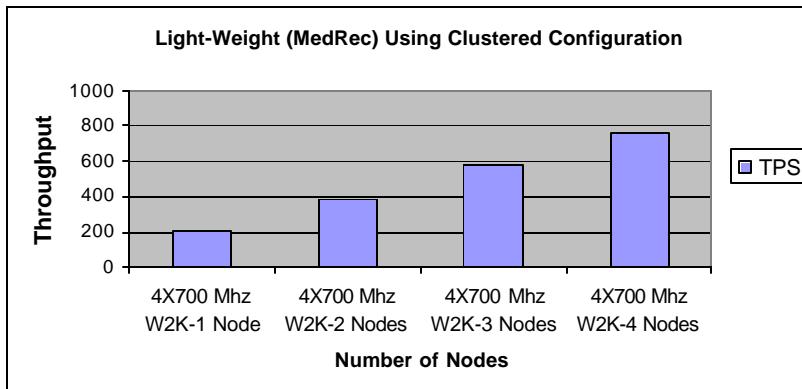
Clustering

- Use WebLogic clustering for scalability and high availability
 - WebLogic Clusterable Objects: Servlets, JSPs, EJBs , RMI , JMS connection factories and destinations, JDBC connections.
- Vertical scaling
 - More powerful machines
 - Adding managed servers to the same physical machines to efficiently utilize the machines full processing power
 - Load balancing disk I/O by having multiple managed servers use separate physical disks for transaction logs
- Horizontal Scaling
 - Adding managed servers to a separate physical machine
 - CPU bound scenarios benefit most, where adding an identical machine to the cluster should give near linear scalability

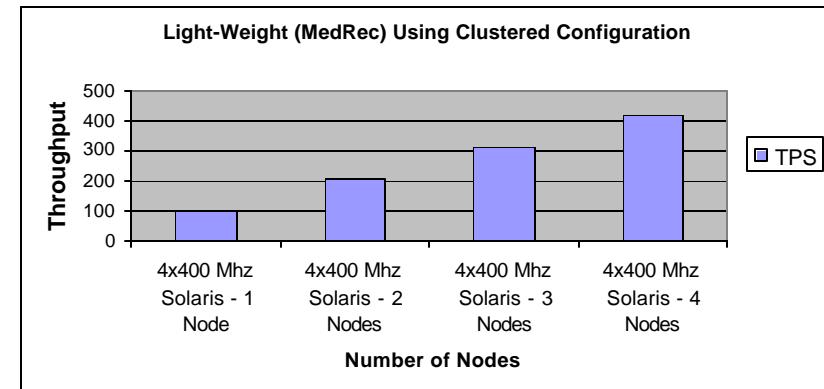
Clustering

- Linear scalability across clusters

Windows Platform



Solaris Platform



- From the WebLogic Capacity Planning Guide
<http://e-docs.bea.com/wls/docs81/capplan/>

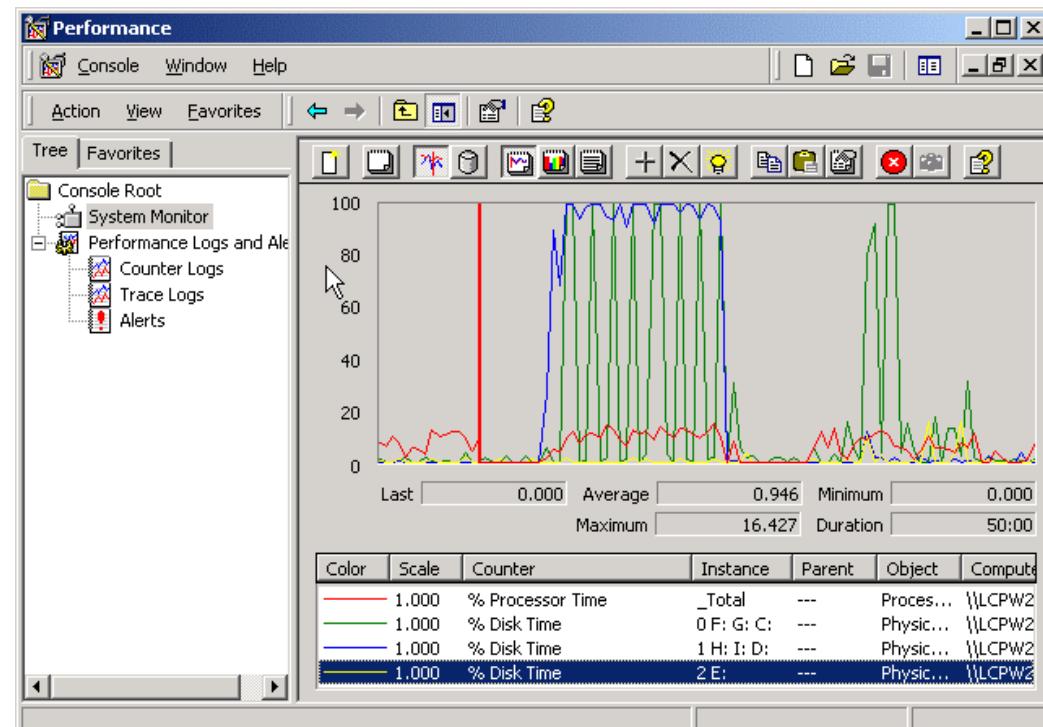


Monitoring Performance



Operating System Monitoring

- Windows Performance Monitor
 - Windows utility to monitor system characteristics
 - Displays statistics for:
 - Processors
 - Memory
 - File handles
 - Threads
 - Processes
 - Network activity





Operating System Monitoring

- Unix provides many tools for monitoring system
 - sar: system activity
 - mpstat: per-processor statistics
 - vmstat: virtual memory statistics
 - netstat: network statistics
 - iostat: Input/Output statistics

JRockit Monitoring

General | Performance | **JRockit** | Security | Compatibility Security | JMS | JTA

This page allows you to monitor runtime data about the JRockit Virtual Machine that is running the current WebLogic Server instance. This page also displays information about the memory and processors on the computer that is hosting the Virtual Machine.

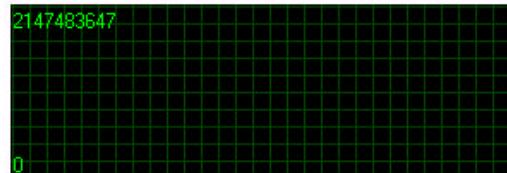
Memory

Used Heap:



The amount (in bytes) of Java heap memory that is currently being used by this Virtual Machine.

Used Physical Memory:



The amount (in bytes) of physical memory that is currently being used on the host computer. (This is memory that is being used by all processes on the computer, not just by this Virtual Machine.)

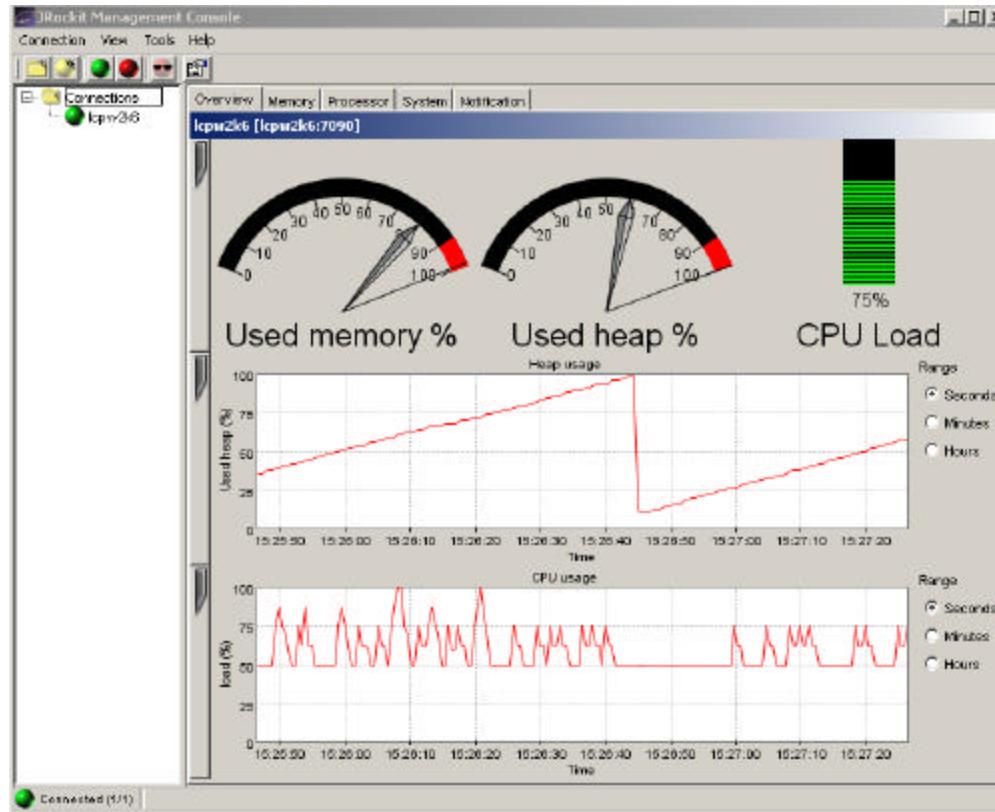
Total Nursery Size: 83886080

The amount (in bytes) of memory that is currently allocated to the nursery. If you are not using a generational garbage collector, the nursery size is 0.

Max Heap Size: 536870912

JRockit Management Console

- Enabled using JRockit –Xmanagement option
- low cost monitoring 1-2% overhead



Execute Queue

WebLogic Server Console - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost:7001/console/actions/mbean/MBeanFramesetAction?bodyFrameId=wl_console_frame_1064891097049&isNew=false&frameId=wl_console_frame_1064891097050& Go

perfDomain > Servers > perfServer > Execute Queue

Connected to : localhost:7001 | You are logged in as : system | Logout

Configuration Monitoring

Requests to a WebLogic Server instance are placed in an execute queue. Each request is assigned to a thread within the queue that performs the work. By default, a new WebLogic Server instance is configured with a default execute queue, `weblogic.kernel.default`, that contains 15 threads. In addition, WebLogic Server provides two other pre-configured queues: `weblogic.admin.HTTP` and `weblogic.admin.RMI`. Because these queues are reserved for communicating with the Administration Console and for administrative traffic, you cannot reconfigure them. Unless you configure additional execute queues and assign applications to them, Web applications and RMI objects use `weblogic.kernel.default`.

When they are available, this page displays current runtime characteristics and statistics for the server's active execute queues.

[Configure a new Execute Queue...](#)

[Customize this view...](#)

Name	Queue Length	Thread Priority	Thread Count	
MDBQueue	65536	5	64	
weblogic.kernel.Default	65536	5	15	

Discussions | | Discussions not available on http://localhost:7001/ |

Monitor Execute Queue

Configuration | Monitoring

Requests to a WebLogic Server instance are placed in an execute queue. Each request is assigned to a thread within the queue that performs the work. By default, a new WebLogic Server instance is configured with a default execute queue, `weblogic.kernel.default`, that contains 15 threads. In addition, WebLogic Server provides two other pre-configured queues: `weblogic.admin.HTTP` and `weblogic.admin.RMI`. Because these queues are reserved for communicating with the Administration Console and for administrative traffic, you cannot reconfigure them. Unless you configure additional execute queues and assign applications to them, Web applications and RMI objects use `weblogic.kernel.default`.

When they are available, this page displays current runtime characteristics and statistics for the server's active execute queues.

 [Customize this view...](#)

Name	Threads	Idle Threads	Oldest Pending Request	Queue Length	Throughput
MDBQueue	 64	0	Thu Sep 18 17:10:42 CDT 2003	0	21434
weblogic.admin.HTTP	 2	1	Thu Sep 18 17:10:42 CDT 2003	0	1181
weblogic.kernel.Non-Blocking	 3	2	Thu Sep 18 17:10:42 CDT 2003	0	6554
weblogic.admin.RMI	 3	3	Thu Sep 18 17:10:42 CDT 2003	0	576
weblogic.kernel.System	 5	5	Thu Sep 18 17:10:42 CDT 2003	0	538
default	 20	19	Thu Sep 18 17:10:42 CDT 2003	0	521491

Monitor Execute Threads

perfDomain> Servers> perfServer> Active Execute Queues>
MDBQueue> Execute Threads

Connected to : lcpw2k19:7001 | You are logged in as : system | Logout

This page displays statistics and information for all active threads in all of the server's available execute queues.

[Monitor all Execute Threads...](#)

[Customize this view...](#)

<u>Number</u>	<u>Total Requests</u>	<u>Current Request</u>	<u>Transaction</u>
0	2204	connection1427.session1609	{weblogic.transaction.internal.JTATransactionImpl: name=r7D4749A1BB6E59980579, status=Active, userProperties secondsActive=0, servers=perfServer, resourceNamesAndStatus=JMS_fstoreMDB/started+JMSI coordinatorURL=perfServer+192.168.10.149:7001+perfDomain serversAndStatus=perfDomain+perfServer/active}
1	2319	connection1427.session1505	{weblogic.transaction.internal.JTATransactionImpl: name=r7D4149A1BB6E59980579, status=Active, userProperties secondsActive=0, servers=perfServer, resourceNamesAndStatus=JMS_fstoreMDB/started+JMSI coordinatorURL=perfServer+192.168.10.149:7001+perfDomain serversAndStatus=perfDomain+perfServer/active}
2	2206	connection1427.session1601	{weblogic.transaction.internal.JTATransactionImpl: name=r7D8249A1BB6E59980579, status=Active, userProperties secondsActive=0, servers=perfServer,

Monitor Execute Threads

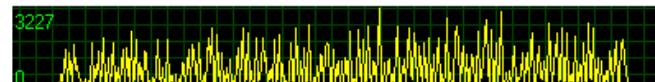
This page allows you to monitor performance information about this server.

Idle Threads: 20

The number of idle threads assigned to the queue.

Oldest Pending Request: Thu Sep 18 17:03:45 CDT 2003

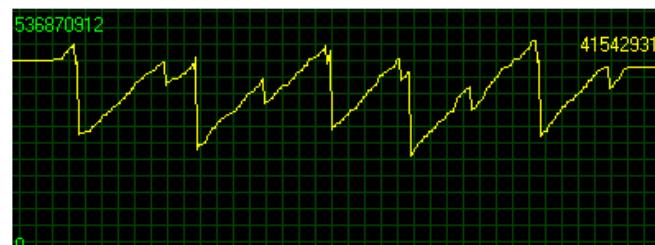
The date and time that the longest waiting request was placed in the queue.

Throughput: 

The number of requests that have been processed by the queue.

Queue Length: 

The number of waiting requests in the queue.

Memory Usage: 

The current amount of memory (in bytes) that is available in the JVM heap.

Monitor JDBC Connections

The screenshot shows the BEA WebLogic Server Administration Console interface. The left sidebar contains a navigation tree with the following structure:

- Console
- perfDomain
 - Servers
 - perfServer
 - Clusters
 - Machines
 - Deployments
 - Applications
 - messagingApp
 - mdb.jar
 - EJB Modules
 - Web Application Modules
 - Connector Modules
 - Startup & Shutdown
 - Services
 - jCOM
 - JDBC
 - Connection Pools

perfDomain> JDBC Connection Pools> JMSDBPoolXA



Connected to : lcpw2k19:7001 | You are logged in as : system | Logout

Configuration | Target and Deploy | **Monitoring** | Control | Testing | Notes

This page allows you to view runtime statistics for this JDBC connection pool. You can also customize the information that is presented by clicking the Customize this view... link.

[Customize this view...](#)

Server	State	Connections	Connections High	Connections Total	Waiters	Waiters High	Active Connections Average Count
perfServer	Running	57	64	64	0	1	0

Transaction Performance

perfDomain> Servers> perfServer> Inflight JTA Transactions 

Connected to : lcpw2k19:7001 | You are logged in as : system | [Logout](#)

When this server processes one or more transactions, this page displays statistics about each transaction currently being processed (called "inflight" transactions).



Transaction ID	Name	Status	Seconds Active	Servers	Resources
BEA1-4DD6F8B0BE3B59980579	n/a	Logging	1	perfServer	JMS_fstore=prepared JMSDBPoolXA=prepared JMS_fstoreMDB=prepared
BEA1-4DD3F8B0BE3B59980579	n/a	Logging	1	perfServer	JMS_fstore=prepared JMSDBPoolXA=prepared JMS_fstoreMDB=prepared
BEA1-4DC4F8B0BE3B59980579	n/a	Committing	1	perfServer	JMS_fstore=committed JMSDBPoolXA=prepared JMS_fstoreMDB=committed
BEA1-4DF6F8B0BE3B59980579	n/a	Preparing	1	perfServer	JMS_fstore=ended JMSDBPoolXA=ended JMS_fstoreMDB=ended
BEA1-4DF9F8B0BE3B59980579	n/a	Active	0	perfServer	JMS_fstoreMDB=started JMSDBPoolXA=started
BEA1-4DEEF8B0BE3B59980579	n/a	Preparing	1	perfServer	JMS_fstore=prepared JMSDBPoolXA=ended

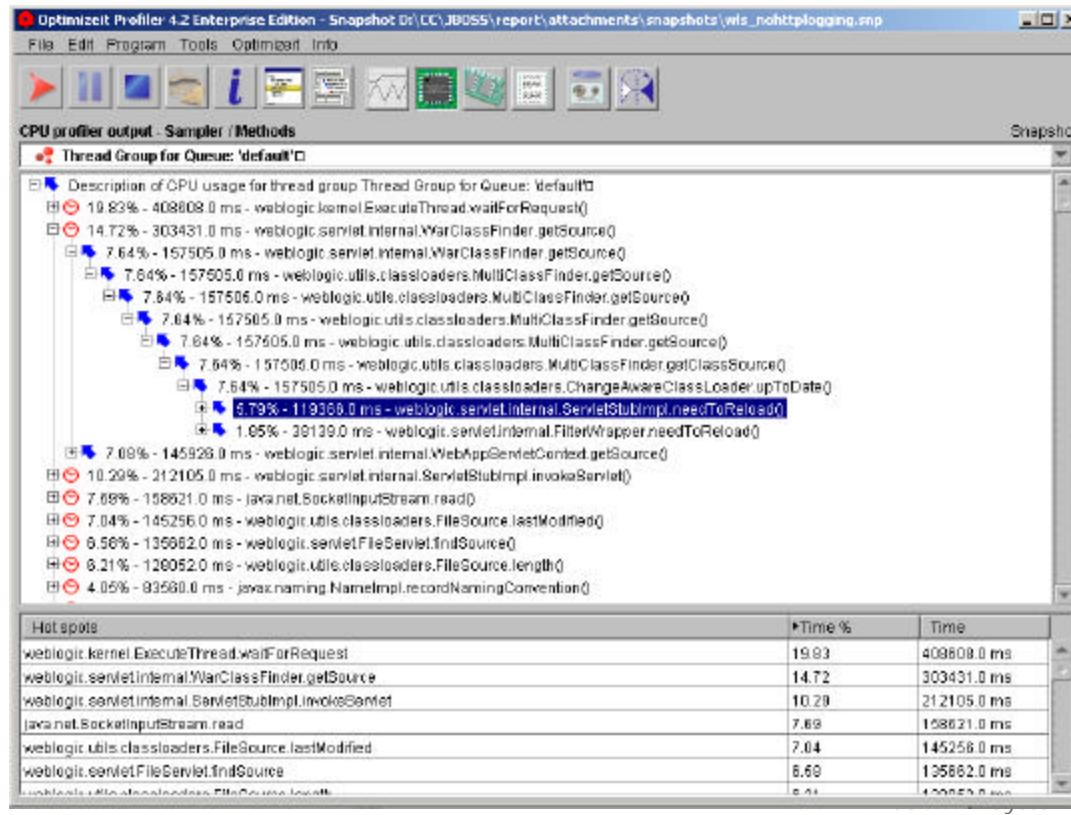


Application Analysis



Profiling Tools – Borland Optimizelt

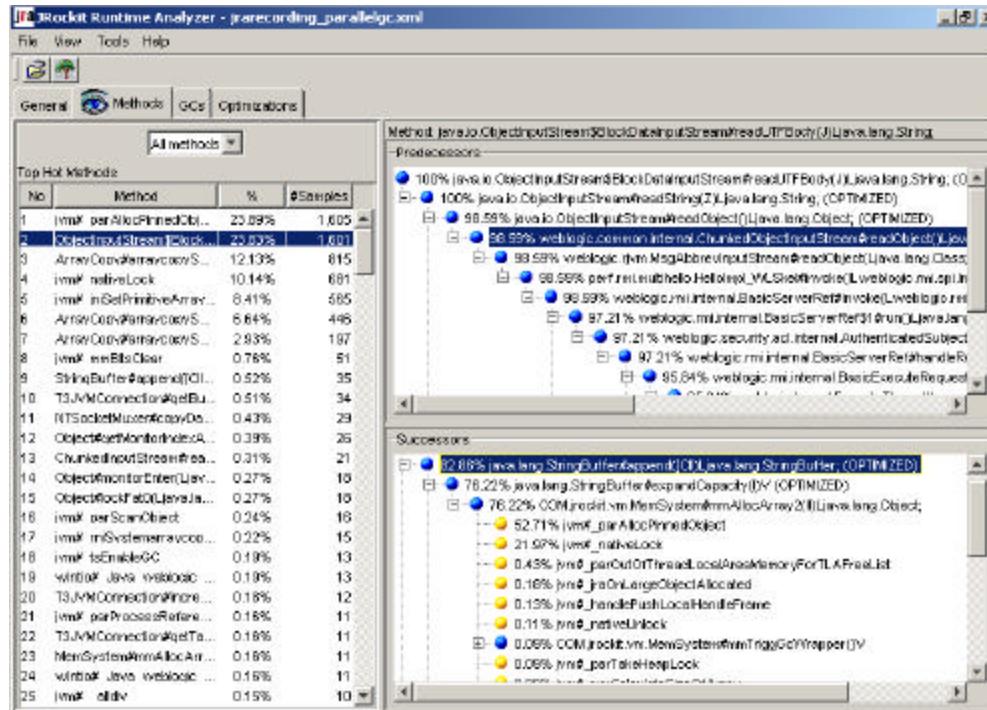
- CPU and Memory profiler for Java applicaitons
- Supports instrumentation or sampling for CPU profiling
- Heavy overhead > 10%

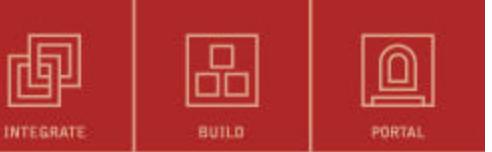


Profiling Tools

JRockit Runtime Analyzer

- Internal tool used to analyze runtime performance of applications running on JRockit
- Low cost overhead < 3%





Q & A



www.bea.com

