WebSphere Application Server - Best Practices: Performance Tuning

presented by:

Michael S. Pallos, MBA
Senior Solution Architect
IBM Certified Systems Expert: WebSphere MQ 5.2
e-Business Solution Designer/Technologist

Candle Corporation
Agenda

- Presentation Logistics
- WAS Architecture
- Best Practices
  - Performance Analysis
  - Tuning techniques
    - Hardware
    - IBM HTTP Server
    - Application Server
    - Database Server
Architecture

Infrastructure and Web Applications
General Architecture

- WebSphere Application Server is based on the IBM Enterprise Server for Java Specification
- Three logical tiers within the application server:
  - Core HTTP engine
  - Web Container (Servlet engine)
  - EJB Container (Enterprise Java Bean engine)
- Communication between tiers implemented as plug-ins
  - This makes performance tuning a combinatorial problem
  - Configuration changes propagate and side effects cascade
- WAS 5.0 is J2EE 1.3
- WAS 4.x is J2EE 1.2 – no JMS or JCA
General Architecture

[Diagram showing the architecture of a web server, including components such as Core HTTP engine, Web Server, Native Web server plug-in, Connection manager, Web Container (Servlet engine), EJB Container, EJB Container Interface, JTS, IIOP, JNDI, JBDC, and tools to create, test, deploy, and manage Web applications.]
Core HTTP Engine

- Handles requests for static resources
  - HTML
  - CGI programs
  - Plug-ins applications
  - Graphics
- Sends servlet requests to the Web Container
  - Logs into the servlet engine
Core HTTP Engine

- First line of scalability by using an HTTP “sprayer” (IND) with multiple servers running in a cluster
- Can use the round-robin method or intelligent routing based on real-time workload metrics
Core HTTP Engine

Interactive Network Dispatcher

HTTP Engine Instance 1

HTTP Engine Instance 2

HTTP Engine Instance n

Shared or Replicated Resources, Shared Database, Shared Files, Shared Sessions
Core HTTP Engine

- Measuring performance:
  - Response time
    - Turnaround time for handling a single HTTP request
  - Throughput
    - Total number of simple HTTP requests that can be handled by the engine in one second
Web Container

- Run-time environment plugs into supported Web servers via ISAPI and NSAPI
- Passes servlet requests to servlet manager
- Servlet manager processes request and sends data back to client
Web Container

- Contains most tuning points
- Very sensitive to topology choices
- Can be cloned
  - Sometimes it helps boost performance depending on the platform
Servlet Engine

- Servlet Service Manager
  - SSI
  - JNDI/LDAP
  - IIOP
  - Session State
  - User Profile
  - Enhanced Servlet API
  - Java SSL

- Various WebSphere App. Server Plug-Ins
- GWAPI Plug-Ins

- Servlets
- JSPs
- EJB Container With JTS
- Connector Beans and Connection Manager

- Isolated in its own Process(es)
- Java connectors to various back ends, CICS, etc

- Configuration requests from common Web-based UI
- Configuration Repository (property files or LDAP)

Source: www.developer.ibm.com
EJB Container

- Handles lifecycle and persistence for EJBs
- Performance can be affected by the fact that it accepts IIOP connections and it has to ensure transactional integrity
EJB Container

- Inherently slow
  - Name resolution is an expensive task
  - EJBs are heavy
  - Database connections are also expensive

- Performance tuning is critical
Best Practices

Performance Analysis & Tuning Techniques
Notation

- Property
  - Type of setting being described
- Measurement
  - Defines the actual settings
- Context
  - Unit of work for the measurement
- Subject
  - Object types within the context

\[ \alpha \beta \lambda(\sigma) \]
Notation

- This is how you read it
  - P: Preferred
  - L: Limit
    - “for a”
  - c: Cache
    - “of”
  - i: Instance(s)

- The actual setting on WAS is called:
  - Instance Cache
  - Preferred limit
\[ \text{AL}_{c(l)} = (N_{ab(x)} \times N_x) + N_{asb} \]

where

- \(N_{ab(lx)}\): total number of active beans on longest transaction
- \(N_x\): total number of concurrent transactions
- \(N_{asb}\): number of active session bean instances
Appendix Includes

- Hardware Capacity
- Operating System
  - AIX
  - Solaris
- IBM HTTP Server
- Application Server
- DB2
Final Thoughts

♦ Why Candle
  – Business
  – Technical
  – IBM Certification 811 / 812

♦ Value Proposition
  – ROI
  – COE
  – TCO

♦ Identifying additional opportunity
  – 21st Century
  – BCBS of Florida
  – Suntrust
Questions
Thank you for your participation

- For more information, go to: www.candle.com/websphere
- For a free whitepaper, go to: www.candle.com/websphereoffer
- For a free 30-day monitor trial, go to:
  - www.candle.com/www1/webspheretrial

E-mail: Michael_Pallos@candle.com
Phone: +1 (863) 709-1611 (USA)
Appendix

Performance Analysis & Tuning Techniques Calculations
Hardware

- You must run on the right hardware
- Hardware alone cannot solve performance problems
- Hardware should get out of the way and let the application run without interference
- This is the intended role
Hardware

- No CPU under 300 MHz should be considered
  - The more the merrier
- Your can never have enough memory
  - Consider at least 512 MB per CPU
  - Maximize Tx and Rx queues on your network adapter to handle traffic
There are a few things you need to modify on AIX if you want to run WebSphere.

Some setting must be modified at installation time.

Some other settings need to be changed as a result of the application behavior on WebSphere.
TCP/IP Settings on AIX

- **thewall**
  - Default is 1 GB or ½ of real memory (smallest one)
  - Increase in 4KN increments

- **sb_max**
  - Increase size in increments of 4096 bytes

- **somaxconn**
  - Specifies maximum listen backlog
  - Default is 1024 Bytes
  - Increase in 1KB increments
TCP/IP Settings on AIX

- **Tcp_sendspace**
  - Sets the value for the tcp socket send buffer size
  - Increase in multiples of 4096 Bytes
  - Changes take effect immediately

- **Tcp_recvspace**
  - Sets the value for the tcp socket receive buffer size
  - Increase in multiples of 4096 Bytes
  - Changes take effect immediately
  - It’s never higher than the major network adapter transmit queue limit
TCP/IP Settings on AIX

- **Udp_sendspace**
  - Increase in multiples of 4KB
  - Changes take effect immediately

- **Udp_recvspace**
  - Set starting value at 10*udp_sendspace
  - Increase size in 4KB increments
  - Changes take effect immediately
TCP/IP Settings on AIX

- Rfc1323
  - Set to 1
    - This means tcp_sendspace and tcp_recvspace can grow about 64 KB
- You’ll need to set rfc1323 = 1 on both sides if you want good performance
- If rfx1323 <> 1 on both sides then tcp_sendspace and tcp_recvspace will be limited to 65536
TCP/IP Settings on AIX

- **Tcp_timewait**
  - Default is 1 second
  - Increase in intervals of 15 seconds

- **Tcp_mssdflt**
  - Sets the maximum packet size when talking to remote networks
    - IN this case “size” pertains only to data in the packet (no overhead)
  - Set to MTU - 52 to ensure transmission of full packets only
  - Changes take effect immediately
TCP/IP Settings on AIX

- **Ncb_max_cache**
  - Defines the maximum size of the Network Buffer Cache
  - Default is 128 KB
    - Objects larger than 128 KB will not be cached
  - Adjust depending on network speed and average object size for the application
    - Consider serialized copies of persistent objects
TCP/IP on AIX

- **MTU**
  - Defines the packet size
  - Default value depends on the adapter installed on the machine
    - Range goes for 512 Bytes to 65536 Bytes
  - Use “isattr” command to see current settings
  - Use “chdev” to change the value
    - All MTUs must be equal across the network to avoid bottlenecks
    - Changes are persistent
File Descriptions on AIX

- **Ulimit**
  - Default is 2000
    - Usually good enough for small machines
  - If “Memory allocation error” occurs, set it to “-unlimited”
    - This was observed on a 24-way RS/6000 S80 running 4 clones
    - Usually better for large SMP machines with clones
Solaris

- Performance tuning on Solaris is a little similar to AIX
- There are a few differences in the way AIX and Solaris manage TCP/IP communications
- The following tuning parameters apply to Solaris 2.6 and 2.7
TCP/IP Settings on Solaris

- **Tcp_close_wait_interval**
- The default value is 2400000 ms but it’s usually too high
- Set it to 30000 ms as a starting point and measure connection delays
- Increase or decrease according the pattern of behavior exhibited by the application
TCP/IP Settings on Solaris

Some customers report performance gains by tuning other settings within the TCP/IP stack:

- Tcp_conn_req_max_q
- Tcp_comm_hash_size
- Tcp_xmit_hiwat
File Descriptors on Solaris

- **Ulimit**
  - Set this parameter to be at least 1024
    - Use “ulimit -n 1024” to set the parameter
    - Use “ulimit -a” to display current values and system limitations
- If you see a “Too many files open” error on WebSphere’s stderr.log file you will need to increase the value above 1024
- Increase or decrease according to the pattern of behavior exhibited by your application
Windows NT

- Windows NT has fewer tuning points than other supported operating systems
- This is not necessarily a good thing
OS Level and Virtual Memory

- Operating system level
  - Version 4, Service Pack 5 or later
  - Version 5 (windows 2000), Service Pack 2

- Virtual memory
  - Set initial size to 300 MB, maximum size to 400 MB
  - Increase or decrease according to the pattern of behavior exhibited by the application
  - Do not let Windows manage it for you
IBM HTTP Server
The Web Server

- First queue in the WebSphere Queuing Network
- Needs to be tuned in combination with the servlet engine queue in most cases
- Additional considerations must be taken if using SSL
IBM HTTP Server

- The IBM HTTP Server for AIX holds the world record for the performance benchmark result by SPECweb96.
- The record was set on January 2000 on a RS/6000 S80.
- www.spec.org
Servicing HTTP Requests on UNIX

- A main process spawning several child (httpd) processes
- Main process assigns requests to each httpd process
Servicing HTTP Requests on Windows

- Windows
  - A parent process and a child process
  - Child process handles workload as a multithreaded process (one request per thread)
MaxClients on UNIX

- Maximum number of simultaneous HTTP requests the IBM HTTP Server can server
- Default is 150, absolute minimum is 2048
- Use low values (5-150) for CPU-intensive applications
- Let it go above 150 for database-intensive applications
  - If your applications uses EJBs MaxClients will be high in most cases
  - The more your go to back-end resources the higher the impact MaxClients will have on both, capacity and performance.
ThreadsPerChild on Windows

- Equivalent to MaxClients on Unix
  - It defines the maximum number of simultaneous requests the child process should service
- Default is 50
- If not using connection pooling it becomes directly proportional to the absolute maximum number of concurrent users allowed
- ThreadsPerChild (or MaxClients) is the regulating factor for the Web Server within the WebSphere Queuing Network
Finding a Tuning Pattern

- Start with a mid-range value
  - MaxClients = 50 for CPU-intensive applications
  - MaxClients = 150 for database-intensive applications

- Explore the effects of small and big deltas
  - Try MaxClients += 10
  - Then try MaxClients += 1000

- Watch for CPU utilization while testing
  - If the CPU reaches 100% utilization you will need to refine your increments
Fine-Tuning Incoming Workload

- **StartServers**
  - Default is 5
  - Let `StartServers = MaxClients` to free the CPU from having to create new child process

- **MinSpareServers**
  - Default is 5
  - Keep it low (high values are bad for performance)

- **MaxSpareServers**
  - Default is 10
  - Let `MaxSpareServers = MaxClients`
Fine-Tuning Incoming Workload

- MaxRequestsPerChild
  - Default is 10000
  - Keep it high

- ListenBacklog
  - Default is 512
  - No tuning is needed in most cases
  - Increase if under a TCP SYN flood attack
HTTP Connections

- The IBM HTTP Server complies with the HTTP/1.1 Specification when it comes to connections
- HTTP connections can be persistent in the sense that they are reusable
  - Just like database connections
Managing Persistent Connections

- **KeepAlive**
  - Default is ON
  - Turn off if you don’t want persistent connections

- **KeepAlive Timeout**
  - Default is 15 sec
  - Decrease as load increases to avoid gridlock

- **MaxKeepAliveRequests**
  - Default is 100
  - Increase for better performance
  - Set to 0 for unlimited
Timeout

- Number of seconds it will take to receive
  - GET requests
  - TCP packets on POST requests
  - Acknowledgements on TCP packet transmissions
- Default is 300
- Increase if your application
  - Goes to slow back-end systems
  - Wraps legacy code
  - Relies on applets to implement business logic
  - Will be accesses by users over slow lines (dialup)
Application Server
Notation

- **Property**
  - Type of setting being described
- **Measurement**
  - Defines the actual settings
- **Context**
  - Unit of work for the measurement
- **Subject**
  - Object types within the context
Notation

• This is how you read it
  – P: Preferred
  – L: Limit
   • “for a”
  – c: Cache
   • “of”
  – i: Instance(s)

• The actual setting on WAS is called:
  – Instance Cache
    Preferred limit
# Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute</td>
<td>A</td>
</tr>
<tr>
<td>Effective</td>
<td>E</td>
</tr>
<tr>
<td>Dynamic</td>
<td>D</td>
</tr>
<tr>
<td>Idle</td>
<td>I</td>
</tr>
<tr>
<td>Keep-Alive</td>
<td>K</td>
</tr>
<tr>
<td>Minimum</td>
<td>m</td>
</tr>
<tr>
<td>Maximum</td>
<td>M</td>
</tr>
<tr>
<td>Total Number</td>
<td>N</td>
</tr>
<tr>
<td>Orphan</td>
<td>O</td>
</tr>
<tr>
<td>Preferred</td>
<td>P</td>
</tr>
<tr>
<td>Clean-Up</td>
<td>U</td>
</tr>
<tr>
<td>Invalidation</td>
<td>V</td>
</tr>
<tr>
<td>Write</td>
<td>W</td>
</tr>
</tbody>
</table>
# Measurements

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backlog</td>
<td>B</td>
</tr>
<tr>
<td>Capacity</td>
<td>C</td>
</tr>
<tr>
<td>Interval</td>
<td>I</td>
</tr>
<tr>
<td>Limit</td>
<td>L</td>
</tr>
<tr>
<td>Request</td>
<td>R</td>
</tr>
<tr>
<td>Size</td>
<td>S</td>
</tr>
</tbody>
</table>
## Contexts

<table>
<thead>
<tr>
<th>Context</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>c</td>
</tr>
<tr>
<td>Heap</td>
<td>h</td>
</tr>
<tr>
<td>I/O</td>
<td>i/o</td>
</tr>
<tr>
<td>Pool</td>
<td>p</td>
</tr>
<tr>
<td>Subject</td>
<td>Symbol</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Active Bean</td>
<td>ab</td>
</tr>
<tr>
<td>Active Session Bean</td>
<td>asb</td>
</tr>
<tr>
<td>Bean</td>
<td>b</td>
</tr>
<tr>
<td>Data Source Connection</td>
<td>DSc</td>
</tr>
<tr>
<td>Data Source Prepared Statement</td>
<td>DSps</td>
</tr>
<tr>
<td>Instance</td>
<td>i</td>
</tr>
<tr>
<td>Java Virtual Machine</td>
<td>JVM</td>
</tr>
<tr>
<td>ORB Connection</td>
<td>ORBc</td>
</tr>
<tr>
<td>ORB Thread</td>
<td>ORBt</td>
</tr>
<tr>
<td>ORB LocateRequest</td>
<td>ORBlr</td>
</tr>
</tbody>
</table>
## Subjects (con’t)

<table>
<thead>
<tr>
<th>Subject</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORB Connection</td>
<td>ORBc</td>
</tr>
<tr>
<td>ORB LocateRequest</td>
<td>ORBlr</td>
</tr>
<tr>
<td>ORB Request</td>
<td>ORBr</td>
</tr>
<tr>
<td>ORB Thread</td>
<td>ORBt</td>
</tr>
<tr>
<td>Session (HttpServletRequest)</td>
<td>s</td>
</tr>
<tr>
<td>Servlet/JSP</td>
<td>sj</td>
</tr>
<tr>
<td>Web Container Connections</td>
<td>WCc</td>
</tr>
<tr>
<td>Web Container Thread</td>
<td>WCt</td>
</tr>
<tr>
<td>Transaction</td>
<td>x</td>
</tr>
<tr>
<td>Web Container (as a whole)</td>
<td>WC</td>
</tr>
</tbody>
</table>
Absolute Cache Limit ($AL_{c(l)}$)

- $AL_{c(l)} = (N_{ab(x)} \times N_x) + N_{asb}$ where
  - $N_{ab(lx)}$: total number of active beans on longest transaction
  - $N_x$: total number of concurrent transactions
  - $N_{asb}$: number of active session bean instances

- Use Resource Analyzer to monitor caching and change accordingly
Tuning ($S_{c(i)}$)

- $S_{c(i)}$ Determines the number of buckets in the cache has table
  - Number of EJB instances in cache
- $S_{c(i)} < AL_{c(i)}$
  - Normally you would want these two values to be equal
  - This makes the entire cache available to the EJB Container service
Tuning ($I_{c(i)}$)

- Sets the cache clean-up schedule
  - Value in ms
- It reduces cache size to the Preferred Limit ($PL_{c(i)}$) settings
  - The idea is to free up hash-table entries
  - Done via a background thread
- Increases if $S_{c(i)}$ increases
Tuning \( (mS_{p(WCt)}) \)

- The Web Container assigns servlet and JDSP calls to threads
- Threads are created at startup time and are kept in a pool
  - \( (mS_{p(WCt)}) \) regulates how many will be available at any given time
- Increase for memory-bound applications
  - This will speed them up
  - The added CPU workload will be manageable
- Decrease for CPU-bound applications
  - You would rather create threads as needed than maintain unused ones
Tuning \( \text{MS}_{p(WCt)} \)

- Increase \( \text{MS}_{p(WCt)} \) for CPU-bound applications
  - buy watch for CPU saturation
  - Decrease Thread Inactivity Timeout to lower CPU utilization
  - You don’t want unused threads occupying valuable space

- Decrease for memory-bound applications
  - This will reduce CPU overhead
  - Creating new threads is not a problem for memory-bound applications
Transport Settings

- Maximum Keep-Alive Connections \( (MKA_p(WCc)) \)
- Maximum Requests per Live Connection \( (MR_{WCC}) \)
- Keep-Alive Timeout \( (KAT_{WCC}) \)
- I/O timeout \( (T_{i/o}) \)
- Connection Backlog \( (B_{WCC}) \)
Tuning \( \text{MKA}_p(WCc) \)

- Maximum number of concurrent persistent connections across all HTTP transports
- Global scope
- In general
  - \( \text{MKA}_p(WCc) \leq \text{MS}_p(WCt) \)
  - \( \text{MKA}_p(WCc) \leq \text{WS:MaxClients} \)
  - \( \text{MKA}_p(WCc) \leq \text{WS:threadsPerChild} \)
- Default
  - \( \text{MKA}_p(WCc) = \text{MS}_p(WCt) \)
    - Keep it that way for know
Tuning ($\text{MR}_{WC}$)

- Defines a connection’s bandwidth
  - Maximum number of requests over a connections
- Global scope
- Default is 100
- Keep in mind that the actual capacity ($\text{EC}_{WC}$) of the Web Container is determined by MRpC as follows:
  - $\text{EC}_{WC} = \text{MKA}_{p(WC)} \times \text{MRS}_C$
- A capacity test for $x$ virtual users, each producing $y$ requests overall will be successful if and only if:
  - $\text{EC}_{WC} \geq (x \times y)$
$\text{MKA}_{p(WCc)}$ and $\text{MR}_{WCc}$

- $\text{MKA}_{p(WCc)}$ represents how many pipes are going to be available
- $\text{MR}_{WCc}$ represents the diameter of each pipe
KAT\textsubscript{WCC} and T\textsubscript{i/o}

- **Keep-Alive Timeout**
  - Maximum amount of time (in seconds) the connection will wait for the next HTTP request
  - Global scope
  - Default is 5 sec
  - Increase if connecting to slow legacy applications
  - Decrease if CPU utilization is too high

- **I/O Timeout**
  - Maximum amount of time (in seconds) dedicated to read/write while processing a request
  - Global scope
  - Default is 5 sec
  - Increase if using object wrappers or connecting to CORBA objects
Tuning $B_{WCc}$

- Maximum number of connections in buffer
- Discrete scope
- Default is 511
- Decrease if CPU utilization is too high
  - This may increase response time
- Increase to get more speed
Servlet Caching Settings

- Enables Servlet/JSP Dynamic Caching
  - Servlet/JSP Cache Size $S_{c(sj)}$
  - Default Priority
  - External Cache Groups
Tuning $S_{c(sj)}$

- Maximum number of entries in the cache
  - Entries refer to servlet/JSP instances created for individual clients
    - Normally individual instances die once the client leaves
    - Not when Dynamic Servlet/JSP Cache ($DS_{c(sj)}$) is enabled
  - Set value to be very large
    - Default is 1000
    - The more you have the better the cache will function
- Watch for memory usage
Advanced Settings

- Maximum In-Memory Count ($\text{MIM}_{p(s)}$)
- Invalidation Timeout ($VT_s$)
Maximum In-Memory Session Count

- For non-persistent sessions
  - It determines the number of sessions in the base session table
    - Enable Allow Overflow to allow creating secondary tables
  - Make it at least equal to WS:MaxClients (WS:ThreadsPerChild on Windows)

- For persistent sessions
  - It specifies the size of the general cache
    - Number of session updates to be cached
    - Make it at least half the total number of session updates in the longest transaction
Tuning VTs

- Specifies the maximum session idle time
  - Value must be at least two minutes
    - Timer is accurate to within two minutes
  - When using time-based Write Frequency:
    - $VT_s > 2 \times W_s$

- Global scope when not overridden by WAR deployment descriptors
  - Always invalidate sessions explicitly
ORB General Properties

- Request Timeout ($T_{\text{ORBr}}$)
- Locate Request Timeout ($T_{\text{ORBlr}}$)
- Connection Cache Maximum ($MS_{c(\text{ORBc})}$)
- Connection Cache minimum ($mS_{c(\text{ORBc})}$)
- Thread Pool Size ($S_{p(\text{ORBt})}$)
Tuning $T_{ORBr}$

- Maximum waiting time for a request message
- Default is 180 sec
- Range goes from 0 to 2147483647 (maximum Java Int)
- Increase for CPU-bound applications
  - Except if the EJB Container and the Web Container are deployed on the same machine
- Increase for database-centered applications and CORBA-intensive applications
  - Use default value as a starting point
Tuning $T_{ORBlr}$

- Maximum waiting time for a LocateRequest message
- Default is 180 sec
- Range goes from 0 to 2147483647 (maximum Java Int)
- Use same criteria as for Request Timeout
Tuning $mS_{c(ORB_c)}$

- Number of TCP connections that must be in the connection cache in order for the ORB to clean up and connections that are not busy
  - Default is 100 with a range from 0 to 255
- The ORB uses TCP connections as transports for RMI calls
  - EJBs with several short remotely exposed methods
    - $mS_{c(ORB_c)} = 0.1 \times \bar{m}$
      - $\bar{m}$ = average number of remotely exposed methods for all EJBs deployed on the server
  - EJBs with few long remotely exposed methods
    - $mS_{c(ORB_c)} = 0.25 \times \bar{m}$
Tuning $MS_c(ORBc)$

- Maximum number of active TCP connections allowed in the connection cache table at any given time
- The default value is 240, with a range from 0 to 256.
- EJBs with several short remotely exposed methods
  - $2 \times u < MS_c(ORBc) < 256$
- EJBs with a few long remotely exposed methods
  - $4 \times u < MS_c(ORBc) < 256$
Tuning $S_{p(ORBt)}$

- The new ORB is also thread-configurable
  - One thread per RMI call
- $S_{p(ORBt)}$ specifies the starting size of the thread pool for the application server
- One thread pool and cache per application server
  - Threads should be picked from the pool and returned afterwards
  - Creating and destroying threads is more expensive than pooling
    - It makes the server CPU-bound
- Use same criteria as $MS_{c(ORBc)}$ depending on how your EJBs are architected
Tuning $S_p(ORBt)$

- Consider selective EJB deployment if CPU utilization on the machine goes above 80%
- Set the value for $S_p(ORBt)$ on each machine using the same rules as for $MS_c(ORBc)$
Data Source Settings

- Control connection pools
  - Minimum Connection Pool Size ($mS_{p(DSc)}$)
  - Maximum Connection Pool Size ($MS_{p(DSc)}$)
  - Connection Timeout ($T_{DSc}$)
  - Idle Timeout ($IT_{DSc}$)
  - Orphan Timeout ($OT_{DSc}$)
  - Prepared Statement Cache Size ($S_{c(DSps)}$)
  - Auto Connection Cleanup
Tuning Data Sources

- Creating connections is expensive
  - One process per DB2 connection in UNIX

- \( (mS_{p(DSc)}) \) settings:
  - WS:MaxClient > MKA_{pWCC} < MS_{p(DSc)}
    - Same for WS:ThreadsPerChild
  - Valid for session databases and data repositories

- \( mS_{p(DSc)} \) settings:
  - Default is 1
Tuning Data Sources

- Low settings for $MS_{p(DSc)}$ have shown to work better

- In UNIX environments:
  - For session databases
    $mS_{p(DSc)} < 0.2 \times MS_{p(DSc)}$
  - For data repositories
    $mS_{p(DSc)} < 0.05 \times MS_{p(DSc)}$
Tuning $S_{c(DSps)}$

- Experimental data on Windows 2000 shows performance increases when
  - $S_{c(DSps)} > N_{DSps} \times \gamma$
    - Where $\gamma = \min(WS:ThreadPerChild, MS_{p(DSc)})$
- This rule may not hold in UNIX
  - WS:MaxClients represents processes instead of threads
  - $MS_{p(DSc)}$ represents also processes (one per connection)
- You will have to experiment with your applications to find the appropriate rule
Auto Connection Cleanup

- New in WAS 4.0
- An algorithm that runs as a separate process that scans Data Source connections and tries to identify
  - Idle connections
  - Orphan connections
- Similar to the JVM’s garbage collections mechanism
- Enabling it ensures healthy connection pools
  - It increases CPU utilization
- Disable it if you know you’re cleaning up connections explicitly
JVM
JVM Heap Size

- Probably the most critical element affecting JVM performance
  - It determines garbage-collection heuristics
- Two settings
  - Starting Heap Size ($mS_{h(JVM)}$)
  - Maximum Heap Size ($MS_{h(JVM)}$)
- In general, $mS_{h(JVM)} = 0.25 \times MS_{h(JVM)}$
Tuning $MS_{h(JVM)}$ and $mS_{h(JVM)}$

- A small heap will force the JVM to collect garbage more often
  - Garbage collection won’t take long
- The opposite effect occurs with large heap sizes
  - Large heap sizes improve throughput
    - Consider $MS_{h(JVM)} = 256$ for 4-way machines
- $mS_{h(JVM)}$ should stay within physical memory
  - You can specify a minimum size and let the heap grow as needed
    - No maximum value
  - This doesn’t work on all applications
Tuning the JIT Compiler and Garbage Collection

- **Defaults**
  - JIT = on
  - Asynchronous garbage collection = on
  - Class garbage collection = on
  - Changes expressed as command-line arguments

- Requesting garbage collection (System.gc()) at the appropriate times may improve performance up to 15% on AIX

- Turning off asynchronous garbage collection (-noasyncgc) may improve performance for EJB-based applications on Solaris
DB2
Rules for Data Repositories

- **Instance**
  - maxagents = \( \Sigma (\text{maxappls}) \)

- **Databases**
  - maxappls = \( (\text{MKA}_{p(WCc)} + \text{MS}_{p(DSc)}) \times \text{Clones} \)
    - Default is 40
  - buffpage = 2 * maxappls (minimum value)
Rules for Session Databases

- Independent sessions database and Data Source
- No $c$
  - $\text{maxappls} < \text{MS}_{p(Dsc)}$
- When using clones
  - $\text{maxappls} < \text{MS}_{p(Dsc)} \times \text{Clones}$
The End