J2EE Application Server Performance Tuning and Capacity Planning

September 13, 2005

Srikant Subramaniam
BEA Systems

Agenda

- Understanding performance objectives
- Application Server performance tuning
- Performance monitoring tools
- Application Server capacity planning
- Q & A

Performance Objectives

- Understand the application
  - Number of users
  - Number of requests
  - Size of requests
  - Data consistency
- Performance requirements
  - Throughput
  - Response time
  - Scalability
- Constraints
  - Configuration (existing HW and SW)
  - Interoperability/legacy systems/legacy data
  - Budget and costs

Designing for Performance

- Design first for maintainability
  - Keep it simple
    Excessive generality and over-engineering hurt performance and maintainability
  - Avoid premature optimization
- Use loose coupling/coarse grained interactions
- Use local interfaces
  - Local EJB interfaces/call by reference
- Stateless design leads to scalability
- Measure early in development

Designing for Performance

- Tune all system components
  - Application Server and other platform components
  - JVM
  - Hardware, Operating System
  - Database
  - Network
- Other factors affecting performance
  - Application design
  - Inefficient coding
  - Environmental limitations (system capacity)
- Talk uses WebLogic Server to illustrate concepts

Core Server

- Thread count
  - Number of simultaneous operations that can be performed by the server
  - Default thread count
    Development mode = 15, Production mode = 25
  - Best tuned empirically
    Tune for high CPU utilization and throughput
    Effect may be constrained by pool sizes (SLSB, MDB, connection pool)
  - Self tuning
    Dynamically grow (or shrink) thread pool size

Core Server

- Default execute queue
- Application specific execute queues
  - Provides additional control for key applications
  - Configurable for servlets, JSP, EJB, RMI
- Non-configurable execute queues
  - Reserved for the administration console
  - Internal queues used for deadlock prevention, triggers and application polling

Core Server

- Chunk tuning
  - Required only for large requests/responses
  - Appropriate tuning reduces number of socket reads/writes
  - Main Factors: MTU and request size
  - Set on both client and server
  - Higher memory requirements
- Connection backlog buffering
  - Specifies how many TCP connections can be buffered before refusing additional requests
- Native I/O performance packs
  - Java I/O not necessarily slower than native I/O
  - /dev/poll vs. /sys/poll

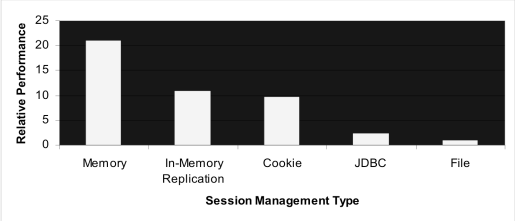
Web Application Performance

- Disable JSP page checks and servlet reloading
  - Can result in 5-8% performance improvement
- Precompile JSPs
  - Avoid recompiling JSPs every time the server restarts
  - Increases server startup time
- Assign separate execute queues for servlets and EJBs

Web Application Performance

- Manage HTTP session state carefully
  - Minimize session state
  - Avoid replicating data that does not change
  - Use aggregate objects rather than multiple single objects
  - Choose appropriate session management mechanism
    - Cookie
    - File
    - In-memory replication
    - JDBC
    - Local memory

Web Application Performance



EJB Performance

- General tips
  - Use proven session-façade pattern
  - Avoid "RequiresNew" transaction attribute
  - Use local interfaces or call-by-reference
  - Choose Stateless Session Beans over Stateful Session Beans whenever possible
  - Strict compliance vs. performance tradeoffs (check-exists-on-method, include-updates)
- Stateless session beans and message driven beans
  - Tune Max-beans-in-free-pool and Initial-beans-in-free-pool
- Stateful session beans
  - Tune Max-beans-in-cache
  - Avoid stateful session bean replication
    - Handle at web tier

EJB Performance

- CMP entity bean performance tips
  - Can be competitive with straight JDBC and can be faster for cache favorable workloads
  - Careful selection of concurrency strategy, caching, and locking behavior
    - Make use of optimistic-concurrency and cache-between-transactions where possible
  - Enable batch inserts and updates
  - Relationship caching
  - Pessimistic concurrency on a per-bean basis (bean level select-for-update)
  - Field-groups
    - Allows grouping commonly loaded CMP fields

JDBC Performance

- JDBC connection pool size
  - Have as many connections as concurrent client sessions
  - Set InitialCapacity = MaxCapacity
- Prepared statement cache
  - Cache prepared and callable statements
  - Reduces network roundtrips and preparation work in the database
  - Each connection in a connection pool has its own individual cache
- Pinned-To-Thread
  - Pin database connection to an execution thread
  - Improves performance by using the connection already held by the thread (reduces contention for connection pool)

JMS Performance

- 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)

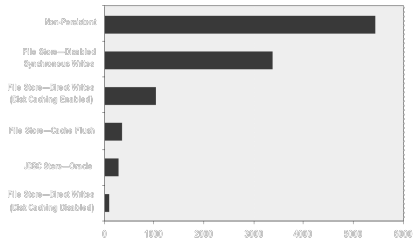
JMS Performance

- 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
  - May need to compromise on QoS

JMS Performance

- Use distributed destinations for cluster scalability
- Use automatic JMS resource pooling in EJBs and servlets
  - Enable by providing resource-ref declarations for connection factories and destinations
- Choice of delivery mode and store based on performance/reliability tradeoff
  - Primary advantage of JDBC store is ease of fail over; alternative is file store with shared disk

JMS Performance



Transaction Performance

- XA overhead
  - Due to transaction logging and inefficient XA implementation of JDBC drivers
  - Logging Last Resource (LLR) optimization to avoid the use of JDBC XA drivers and reduce XA overhead
- JMS and JDBC in a transaction
  - Requires 2PC (even if it is the same database)
  - JMS has to use an XA enabled Connection Factory
  - JDBC has to use an XA driver (slow!)
- JMS and JMS in a transaction
  - Using 2 destinations from different JMS servers in a transaction will use 2PC

Transaction Performance

- Keep transactions 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 transaction logs and JMS file stores
  - Use of Direct-Write for transaction log files may improve performance with hardware write cache enabled

Webservices Performance

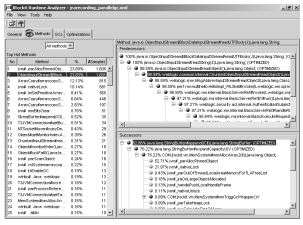
- Small simple method invocations more performant than large complex invocations or arrays
- US-ASCII character set
  - Most efficient and performant
- Security overheads
  - Data encryption and digital signatures
  - Adversely affect performance
- Literal encoding rather than SOAP encoding
  - Document-oriented instead of RPC-oriented
- Limit the use of message handlers
  - Handlers are typically executed for every operation

Performance Monitoring Tools

- CPU and Memory profilers (OptimizeIt, JProbe)
  - Good for detailed analysis but intrusive
- JRockit Runtime Analyzer provides low cost profiling
- P6Spy recommended for analysis of JDBC activity from driver perspective
- Console
  - WebLogic Server, JRockit Management
- Other analysis tools and techniques
  - Verbose logging, thread dumps, database tuning tools, etc.
  - Windows Performance Monitor, sar, prstat/top, mpstat, vmstat, netstat, iostat
- Choose the right tool for the situation

JRockit Runtime Analyzer

- Used to analyze runtime performance of JRockit and applications running on JRockit
- Low cost profiling 1-2 % overhead



- http://dev2dev.bea.com/jrockit

Capacity Planning Techniques

Capacity Planning

- What is capacity planning (CP)?
  - Estimating the hardware and software configuration required to support a given user load with acceptable performance
- Why is it important?
  - Guarantee consistent end to end performance
  - Provide guidelines for system scalability
  - Estimate future hardware requirements
- How much capacity do we need?
  - Difficult question to answer
  - CP = BMH (Buy More Hardware); not always a good idea
  - Often, too little (or too much) capacity to meet load requirements; neither of which is ideal

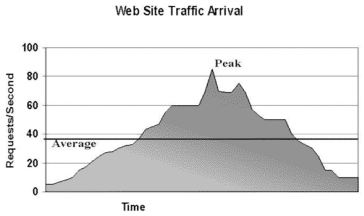
Capacity Planning Process

- Developing a capacity plan
  - Define application service levels
  - Measure current application behavior
  - Measure current system utilization
  - Identify existing system bottlenecks
  - Estimate future service level requirements
  - Workload design methodology
  - Benchmark to validate estimates
  - Continuous monitoring and analysis

Application Service Levels

- Service levels define desired application behavior
- Different service levels for each application
  - e-tailer site: High availability, support high traffic volumes with fast response times
  - B2B site: lower traffic but longer user visits
- Duration of requirements
  - 7x24x365, 9AM-5PM weekdays, November 26 – December 24
- Peak vs. average load
- Key service level metrics (under peak load)
  - Throughput
  - Response time
  - Availability

Peak vs. Average Loads



Capacity planning is all about anticipating peak user loads

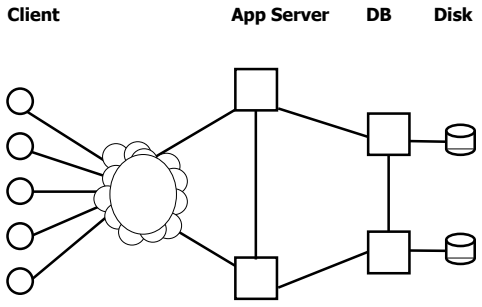
Current Application Behavior

- Metrics for recording current application behavior:
  - User load
    - Individual users logged in to the system
    - Number of HTTP sessions, servlet requests
  - Throughput
    - Requests per second
    - Identify individual components (throughput of static and dynamic content, for instance)
  - Response time
    - Factor in time spent in other tiers (HTTP servers, network infrastructure)
    - Record response time frequently to detect peaks

Current System Utilization

- Establish consistent monitoring of a few fundamental metrics
  - CPU utilization
  - Memory footprint
  - Network utilization at peak loads
  - Disk utilization at peak loads
- Monitor at several tiers
  - CPU, memory, disk: vmstat, iostat or perfmon
  - Network: netstat
  - JVM: GC performance

Existing System Bottlenecks



Every node and link is a potential bottleneck

Existing System Bottlenecks

- Analyze the data for
  - High CPU utilization
  - Excessive paging
  - Unusually high disk I/O
  - Network infrastructure capacity ceiling
- Identify components running at full utilization
  - Determine current utilization levels
  - Impediments to overall responsiveness and throughput
- Consider any existing fail over plans
  - Analyze impact of a system outage on existing traffic
  - Traffic spike preparedness
- An application is only as fast as its slowest component

Future Service Level Requirements

- How will the application usage change over time?
  - Functional changes

    Additional customization: personalization, single sign-on
    Increased availability requirements to provide guaranteed fail over of user sessions
  - Growth

    Increase in traffic volumes over time (hopefully)
    Increased functionality (and ongoing promotions) that might draw additional traffic
- Develop estimates of expected peak loads
  - Establish headroom objectives for load spikes
  - Maintain response time objective (if possible)

Estimating Peak Loads

- Trend graphs using basic performance data
  - View growth in utilization rates, estimate when existing capacity will run out and schedule additional resources to meet demand
  - Trends are often not the best estimator of the future
  - Simplistic, quick-and-dirty analysis
- Capacity planning tools
  - Tools use modeling rather than trending
  - Accuracy of predictions depends on

    Accuracy of input data on future needs
    Effect of any simplifying assumptions (does the tool model all the components?)
  - Needs thorough understanding of what parameters to model

Workload Design Methodology

- Choose content for workload
  - Result of performance and scalability study is gated by the validity of the workload under test
- Gather data on workload usage
  - Anticipate real use of website
  - Use profile based website usage
  - Model after industry developed usage models
- Identify workload patterns
  - Publish/subscribe, online shopping, B2B sites

Workload Design Methodology

- Workload validation
  - Representative

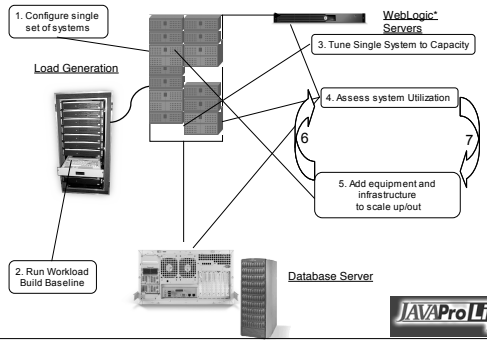
    Does it mimic typical activities?
    Is it too contrived?
  - Measurable

    What metric does the workload report?
  - Static

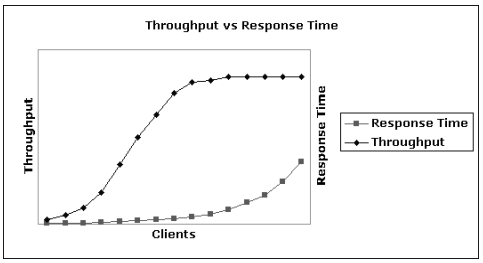
    Are transient effects factored out?
    Is the workload influenced by external factors?
  - Repeatable

    How much do any two successive runs deviate?

Validate Capacity Planning Estimates



Throughput Vs. Response Time



Capacity Planning Wrap-up

- Key Points
  - Define requirements clearly
  - Plan for peak loads
  - Make conservative estimates
  - Anticipate future requirements
  - Strive for realistic measurements