



Java™ Message Service API: Tips and Tricks for Scalable Messaging

Tom Barnes
Sunila Srivatsan
BEA Systems, Inc.
<http://bea.com>

Session TS-3797

Goal

Learn techniques for developing and architecting robust high-performance Java™ Messaging Service “JMS” API applications.

Agenda

Use Cases

Clustering

Performance

Message Delivery

Diagnosing Common Problems

JMS Tips: Use Cases

Use Cases

When to Use Messaging

When Not to Use Messaging

Clustering

Performance

Message Delivery

Diagnosing Common Problems

JMS Tips: Messaging Use Cases

When to apply JMS messaging

- Load balancing
- Batching multiple operations into one
- Store and forward
 - (a.k.a. Reliable Messaging in XML-land)
- Pub/sub broadcast (event)
- Integration
 - Between Java™ 2 Platform, Enterprise Edition “J2EE™” platform
 - J2EE platform and even non-J2EE platform vendors
 - Non-Java™
- Exactly-once (transactional) versions of the above

JMS Tips: Messaging Anti-Use Cases

When **not** to apply JMS Messaging

- Short running request/response style operations on reliable services
 - Instead: invoke servlets, EJB™ Architecture, web services directly
- Integration at message-bus “boundaries”
 - For example: between different companies
 - Use XML standards to specify message semantics and to inject “foreign” messages into the local bus

JMS Tips: Clustering

Use Cases

Clustering

What Is JMS Clustering?

Load Balancing

High Availability

Performance

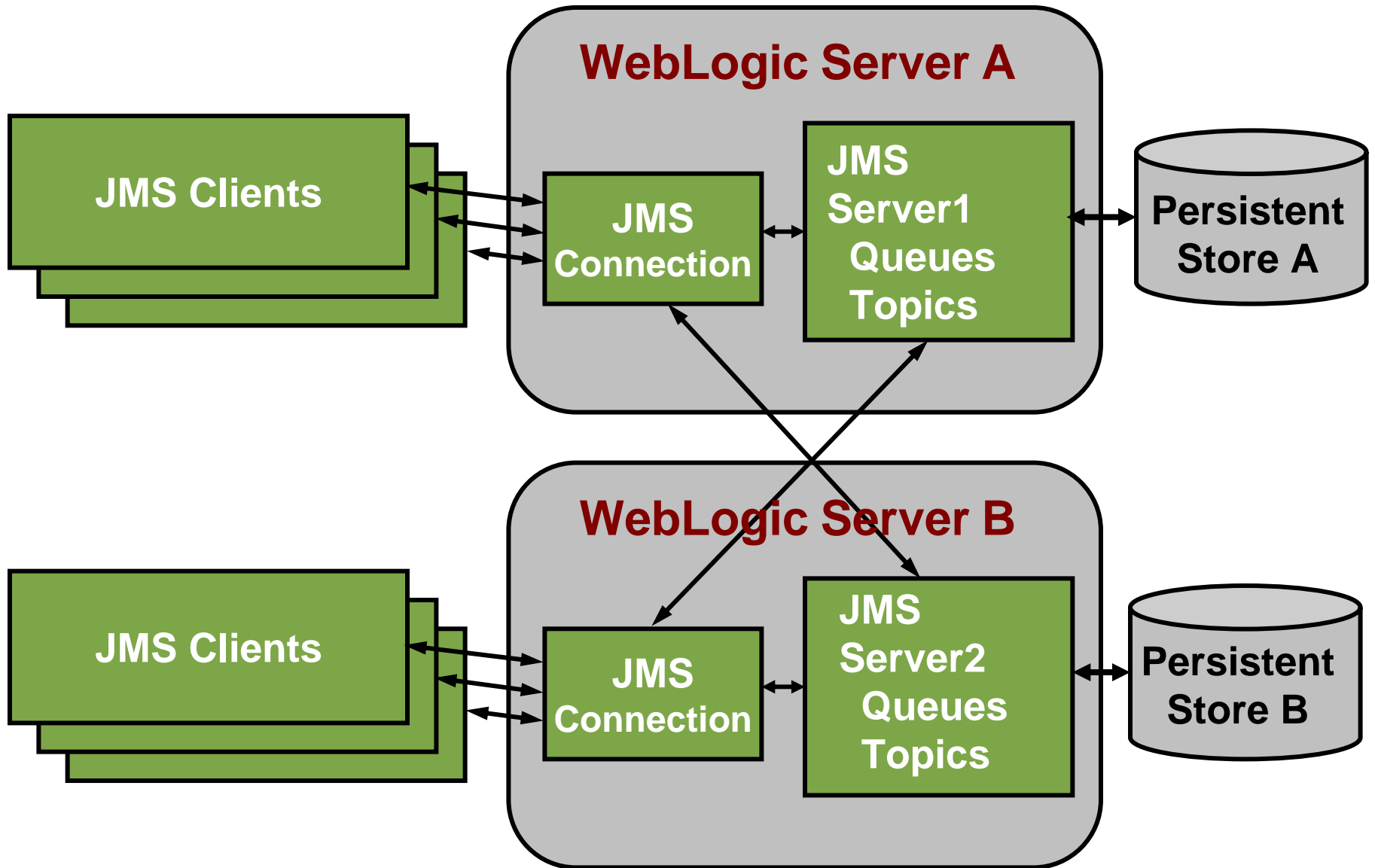
Message Delivery

Diagnosing Common Problems

JMS Tips: Clustering

What is JMS Clustering?

- Location transparency
 - Transparent access to JMS destinations and connections from any server in the cluster or any client connecting to cluster
- Connection load balancing
 - Multiple servers host JMS client connections
- Message routing
 - A client stays connected to a single server in the cluster, client requests route to the appropriate JMS server
- Distributed destinations
 - A single logical destination that represents multiple physical destinations



JMS Tips: Clustering

JMS connection factories

- Connection factories are typically clusterable RMI objects
- Create Connection() works like stateless session EJB
- Same connection factory can be used to reconnect to a cluster even as cluster membership changes

JMS Tips: Clustering

Distributed destinations

- Vendor-specific feature
- Logical destination that is a group of multiple physical destinations
- Appears and should be used as a regular destination
- Increases availability for continuous JMS service
- Responsible for distributing the load across the different physical destinations on multiple server instances
- Most common use case
 - MDBs application targeted to same server(s) that host distributed destination

JMS Tips: Clustering

Routing and load balancing

Three levels of routing and load balance:

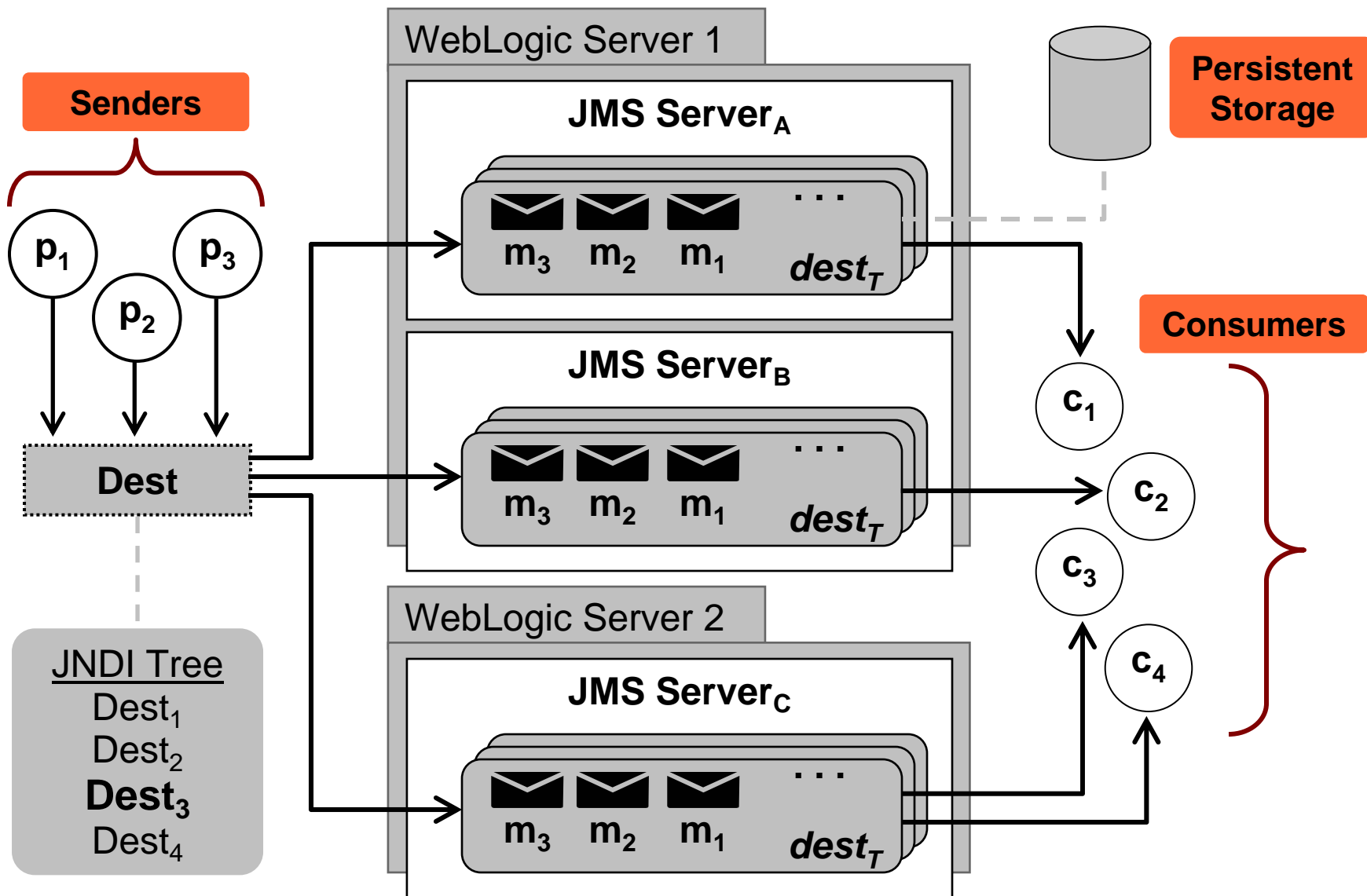
- Connection load balance which takes place on client side
 - Where is the connection?
- Message routing and load balance takes place on server side
 - Without Distributed Destination
 - Which physical destination should get the message
- Message routing and load balance takes place on server side
 - With Distributed Destination
 - Which physical destination should get the message via the distributed destinations

JMS Tips: Clustering

Distributed-destination load balancing

- Round-robin or random algorithm is used
- Producers can load balance once on first message or on every message
- Consumers load balance once when they are created
- For Distributed Queues messages are load balanced across member queues
- For Distributed Topics messages are replicated across member topics
- Persistent message goes first to the member with a configured JMS store
- Favors members already in a transaction
- Favors members on the same server as client
- Producers avoid members with no consumers and consumers look for members with no consumers

Distributed Destinations



JMS Tips: High Availability

Fail-over: client connections and sessions

- Connection level fail-over
 - For most JMS vendors, no fail-over for existing connections and the state is lost
 - ExceptionListeners are invoked and typically all connections, sessions, producers, consumers are closed by the client
 - When creating a new connection, cluster-capable connection factories will load balance among active servers in the cluster
- Session fail-over
 - Receive exceptions for consumer failures
 - Best practice to call `Session.recover()/session.rollback()/tx.rollback()`
 - Must recreate the failed consumer

JMS Tips: High Availability

Fail-over: JMS server

- JMS is an exactly-once service which means the service can only be active on one server in a cluster
- Automatic restart
 - HA framework
 - Vendor specific
- Replicated message store approaches
- Provided by some JMS vendors
 - Hardware/SAN/HA-Framework solution
 - Highly-available database-based message store
 - Also typically need to replicate, database data, J2EE platform transaction log
- If Weblogic server fails:
 - Its destinations can be migrated to a running server
 - If applications use JTA transactions, the server's transaction management also must be migrated

JMS Tips: High Availability

Store and forward

- Store requests in a local destination for eventual forwarding to a remote destination or service
- Save requests even if remote destination or service is unavailable
- Two styles of SAF:
 - Client-side (local store on client)
 - Vendor extension
 - Application coded
 - Server-side (destination on local JMS server)
 - Custom application (perhaps an MDB)
 - Bridge (a configurable forwarder)
 - Implicit (message bus automatically)
- When destination-to-destination SAF is unnecessary:
 - When remote services can pull directly from local destination (MDBs for example)
 - When messages can be sent directly to a remote destination (it is reasonably available)

JMS Tips: Performance

Use Cases

Clustering

Performance

Leveraging Aggregation

Benchmark Design

Locating Bottlenecks

Database vs. File Stores

Reducing Transaction Overhead

Throttling Techniques

Message Delivery

Diagnosing Common Problems

JMS Tips: Performance

Leverage aggregation—introduction

- The inherent scalability of network and disk I/O can be leveraged at multiple points in the messaging stack
 - Batch multiple network operations into one
 - Batch multiple disk operations into one
 - Batch multiple database operations into one
 - Batch multiple transaction operations into one
- Achieve batching through:
 - Application server tuning
 - JMS server tuning
 - Application design

JMS Tips: Performance

Leverage aggregation

- Encourage concurrent clients
 - JMS servers aggregate concurrent client requests into single I/Os
 - Use MDB pools to naturally provide concurrency
 - Use multiple senders and receiver clients
- Use a single message to hold multiple requests
 - You can get the same messages per second with 1K messages as 100-byte messages
 - Not helpful with request sizes larger than a few K
- Batch multiple messages into single transactions
 - Allows messages to be grouped into single persistent I/Os
 - Batched transactions may yield higher performance than non-transactional

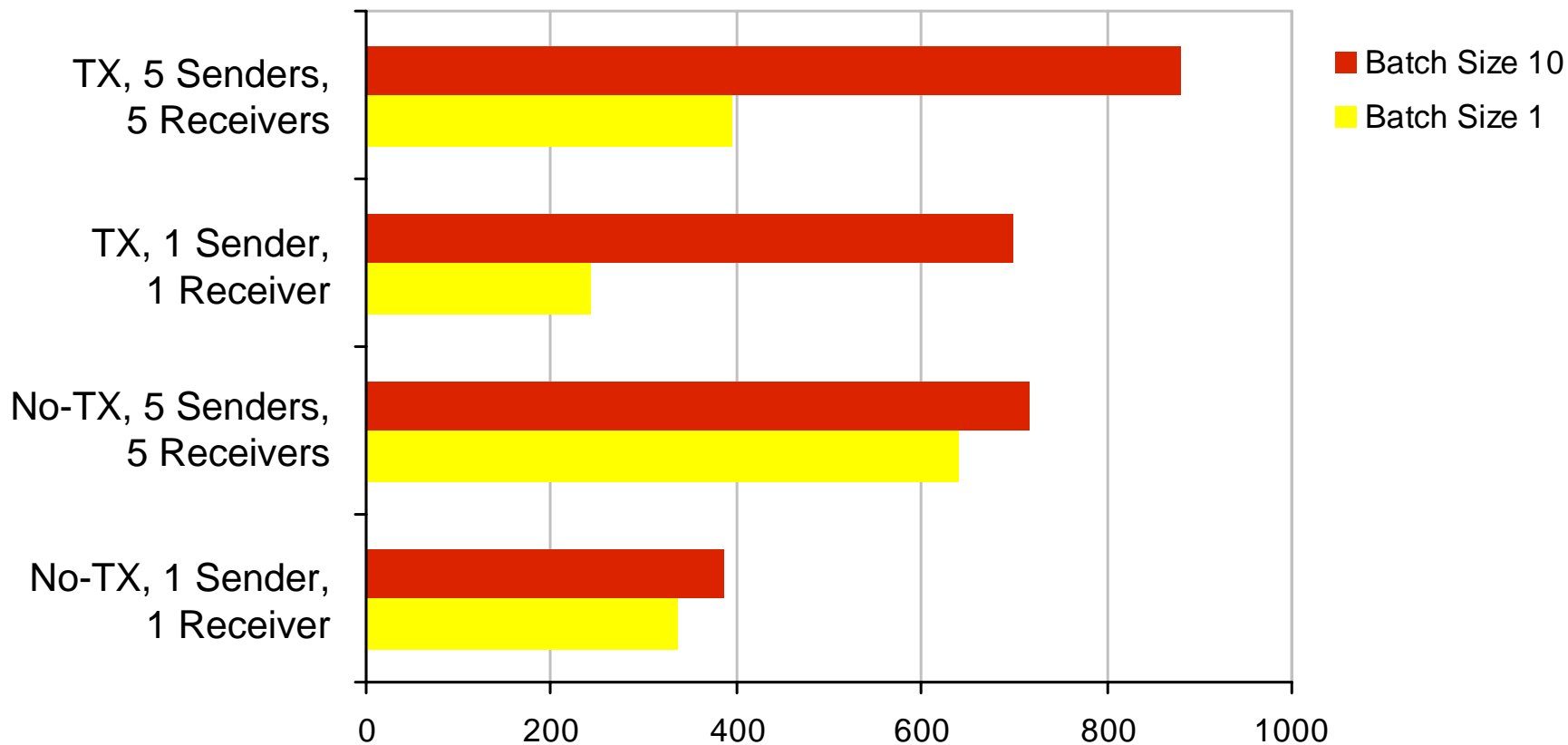
JMS Tips: Performance

Leverage aggregation—asynchronous consumers

- Use asynchronous rather than synchronous consumers
 - Allows JMS server to push messages in a “one-way” (rather than serve them via request response)
 - Leverages the “message pipe-line”
- Tune asynchronous message pipe-line
 - JMS servers typically push multiple messages at a time to asynchronous clients
 - Commonly referred to as “pipe-line” or “backlog”
 - Reduces network overhead
 - The max pipe-line size is usually configurable

JMS Tips: Performance

Leverage aggregation—comparing throughput



100-byte persistent messages, single machine with two JVMs: one for JMS server with one queue, and one shared by clients; data for relative comparisons only

JMS Tips: Performance

Benchmark design

- Best practice: develop custom benchmarks to model applications
- Generic JMS benchmark kits often (usually) don't yield representative data
 - Applications use JMS in a broad variety of ways
 - Message type, message size, concurrency, locality, filtering, etc., are all significant
 - Application overhead has a significant impact
- Designing a benchmark
 - Design benchmark to closely model app
 - Run on same hardware used in production
 - Use enough messages! (minimal 2–3 minutes worth)
 - Give system time to “warm-up” before beginning measurements

JMS Tips: Performance

Locating bottlenecks

- Bottlenecks are often outside of JMS
 - Database, application processing, XML processing, transactions
- Network can be bottleneck
 - Assume network saturation at 80% network capacity
 - Assume 512 bytes overhead per message
 - Take into account non-JMS network traffic
 - Example on next slide
- JMS persistence can be bottleneck
 - Measure by temporarily forcing all messages to be non-persistent
 - Configure JMS to treat all messages as non-persistent
 - Or switch to “asynchronous file persistence”
 - Or move file store onto a virtual (memory-based) disk

JMS Tips: Performance

Locating bottlenecks—network bottleneck example

- pub/sub
 - One publisher, 32 subscribers, 319.5K messages
- Network capacity = 105,000K per second
 - $0.8 * (1 \text{ Gbit/sec} / 8 \text{ bits/byte} / 1024 \text{ bytes/K})$
- Message overhead = 320K per message
 - 319.5K + 0.5K per message
- Messages per second per client = 10
 - $(105,000\text{K/sec}) / (320\text{K/msg}) / (1 \text{ pub} + 32 \text{ sub})$

JMS Tips: Performance

Database stores vs. file stores

- JMS vendors typically offer a choice of file- or database-based persistence
- Both have the same transaction semantics and guarantees
- Both have the same application interface (no difference in application code)
- All things being equal, file stores generally offer better throughput than a JDBC store
- File stores are easier to configure and administer
- File stores generate no network traffic
- JDBC stores may simplify fail-over

JMS Tips: Performance

Reducing transaction overhead

- Batch multiple messages into transaction
 - Sample MDB code follows
- Transactions are usually not necessary for non-persistent messaging
- Vendor optimizations
 - Mostly aimed at reducing database XA transaction overhead
 - Last-resource/last-participant optimization (not fully safe)
 - Logging last resource (feature in WebLogic)
 - Shared data source with CMP (feature in WebSphere)
- Replace transaction with “duplicate elimination”
 - Common transaction use case: exactly-once message forwarding
 - Send-retry with duplicate-elimination can replace transactions
 - Not always faster

JMS Tips: Performance

MDB batching, sample code

```
// use synchronous consumer with short timeout
// to batch multiple messages into a single tx

public void onMessage(javax.jms.Message msg) {
    long startTime = System.currentTimeMillis();
    int msgCount = 0;
    while (msg != null) {

        // process message

        if (++msgCount >= MAX_BATCH_SIZE
            || System.currentTimeMillis() - startTime
                > MAX_INTERVAL_MILLIS)
            break;
        msg = consumer.receive(MAX_IDLE_MILLIS);
    }
}
```

JMS Tips: Performance

Tuning file stores—part 1

- Hardware solutions
 - Dedicated disks
 - Multiple stores
 - Disks with battery-backed cache
- Asynchronous file store writes
 - Java-based JMS solutions typically provide an “asynchronous” disk write setting
 - Unsafe in the event of an O/S crash or power failure, but **fast**
 - Some JMS vendors use as default

JMS Tips: Performance

Tuning file stores—part 2

- Windows disk caching
 - Various MS Windows versions enable by default O/S or disk-level caching for “direct writes”
 - This is unsafe in the event of O/S crash or power failure
 - Java-based vendors typically don't use direct writes (default for WebLogic 8.1 and previous), instead they explicitly force caches to flush as needed
 - “C”-based JMS vendors typically do (as does WebLogic 9.0)
 - Configure via hardware manager settings

JMS Tips: Throttling Techniques

- Various throttling techniques exist to:
 - Smooth JMS server load
 - Prevent system overload
- Techniques include:
 - Configuring message quotas
 - Configuring sender throttling
 - Design applications using a request/response pattern
 - Tuning application concurrency

JMS Tips: Message Delivery

Use Cases

Clustering

Performance

Message Delivery

Ordered Message Delivery

Poison Message Handling

Exactly Once Messaging

Diagnosing Common Problems

JMS Tips: Ordered Message Delivery

Ensuring FIFO-ordered message delivery

- JMS guarantees ordering between a particular sender and consumer, but limits guarantee in the event of consumer rollbacks/recovers (failures)
- Generally requires single consumer per destination
 - Limits performance
- Some vendors provide “unit-of-work” or “unit-of-order” solution, which enables multiple-named FIFO “sub-orderings” within a single destination
 - Less configuration than using multiple destinations
 - Supports multiple concurrent consumers
 - Standard alternative: unique selector per consumer
- Custom message orderings (priority, etc.):
 - Vendor specific; typically configurable on the destination

JMS Tips: Poison Messages

Definition of...

- Poison messages are repeatedly redelivered
- Applications may force redelivery via:
 - Transaction rollbacks/timeouts
 - Negative acknowledgements (calling `recover()`)
 - Throwing exceptions from “`onMessage()`”
- Throwing Runtime Exceptions from MDB `onMessage`
 - Container will destroy current MDB instance
 - Container will force message redelivery
 - Container will rollback tx if there is one
- Throwing Runtime Exceptions from client `onMessage`
 - Vendor specific—in general, vendor is free to drop message after an arbitrary number of delivery attempts
 - Behavior depends on delivery mode (client, auto-ack, or transacted)

JMS Tips: Poison Messages

Handling of...

- Problem messages impact performance
- Poison message handling provided by JMS API
 - Application checks JMS API message delivery count field
 - (Optional part of JMS API)
 - Application checks redelivered flag
 - (Required part of JMS API)
 - Application sets message expiration on send
- Additional handling commonly provided by JMS API vendors
 - Automatic redelivery delay
 - Automatic delete after X redelivery attempts
 - Error/"dead-letter" queues
 - Scheduled message delivery

JMS Tips: Exactly-Once Messaging

Introducing transactions and JMS

- When to apply transactions
 - Exactly-once forwarding
 - Exactly-once message processing
 - Rarely makes sense for non-persistent messaging
- Two types of transaction: local and global
- Local transactions
 - Only JMS may participate in the transaction (limited scope)
 - Supplied by JMS API “Transacted Sessions”
 - **Won't participate in globally-scoped transactions**
- Global transactions
 - Commonly referred to as “JTA” or “XA” transactions
 - Any “XA” resource may participate in global transaction (such as a database)
 - Requires a J2EE platform transaction manager

JMS Tips: Exactly-Once Messaging

Ensuring JMS participates in a global transaction

- Use a JMS API “XAConnection” factory
- Manual enlistment
 - Programmatically enlist a JMS API session with the current JTA transaction
- Automatic enlistment
 - Vendors that provide both a JTA transaction manager and a JMS API solution automatically enlist JMS with the current transaction
- Assisted enlistment
 - J2EE platform vendors commonly supply features to wrap “foreign” JMS clients and manually enlist for you
 - Such features include Bridges, MDBs, and EJB “resource-reference” pools
- For more information, see “Integrating Remote JMS Providers” FAQ at bea.com

JMS Tips: Diagnosing Common Problems

Use Cases

Clustering

Performance

Message Delivery

Diagnosing Common Problems

Duplicate Messages

Missing Messages

Common Programming Errors

JMS Tips: Diagnosing Common Problems

Typical reasons for duplicate messages

- Normally running system
 - Poison messages (application is forcing redelivery)
 - Application is throwing exceptions from `onMessage()`
 - Transaction timeout forces redelivery
- Application error
 - Unintentional use of topics instead of queues
 - Application is failing to acknowledge/commit receives
 - (Messages show up after reboot)
 - Consumed message accidentally doesn't participate in transaction
 - Transaction commit won't delete message
 - Produced message accidentally doesn't participate in transaction
 - Message gets sent whether or not transaction commits

JMS Tips: Diagnosing Common Problems

Typical reasons for missing messages

- Normally running system
 - Message is in long-running tx, not actually missing
 - By definition, messages sent under a transaction aren't made available to consumers until the transaction commits
 - Messages are expiring
 - Non-persistent topic messages dropped by JMS server
 - Message is in a stalled async consumer's pipe-line
- Application error
 - Messages mistakenly sent as non-persistent
 - Application fails to commit send transactions
 - Consumed message not participating in a transaction
 - Transaction rollback won't force redelivery of message

JMS Tips: Diagnosing Common Problems

Common JMS API programming errors...

- Calling public setters on `javax.jms.Message`
 - Set priority, expiration, and delivery-mode via `javax.jms.Producer`
 - The equivalent `javax.jms.Message` setters are ignored
- Multi-threading `javax.jms.Session`
 - JMS sessions are used to create producers, consumers, and messages
 - JMS sessions, by definition, aren't thread safe
 - For multi-thread support, simply use multiple JMS API Sessions
- Creating a new JMS client per message
 - Instead, pool and/or cache: JNDI lookups, JMS connection, session, producer, and consumer objects...

For More Information

- <http://dev2dev.bea.com/jms/>

Q&A

Sunila Srivatsan

Tom Barnes

Submit Session Evaluations for Prizes!

Your opinions are important to Sun

- You can win a \$75.00 gift certificate to the on-site Retail Store by telling Sun what you think!
- Turn in completed forms to enter the daily drawing
- Each evaluation must be turned in the same day as the session presentation
- Five winners will be chosen each day (Sun will send the winners e-mail)
- Drop-off locations: give to the room monitors or use any of the three drop-off stations in the North and South Halls

Note: Winners on Thursday, 6/30, will receive and can redeem certificates via e-mail.



Java™ Message Service API: Tips and Tricks for Scalable Messaging

Tom Barnes
Sunila Srivatsan

BEA Systems, Inc.
<http://bea.com>

Session 3797