JBoss Messaging

Ovidiu Feodorov, Project Lead
Tim Fox, Core Developer

June 13, 2006

Agenda

- Why do we replace JBossMQ?
- Project Goals
- JBoss Messaging 1.0 – Current Situation
  - ✓ Features
  - ✓ Architecture
  - ✓ Installation and configuration
- Roadmap

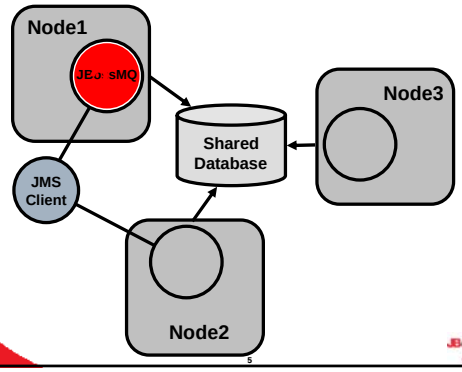
Reasons to replace JBossMQ

- JEMS (JBoss Enterprise Middleware System) already contains a production-quality JMS provider, JBossMQ
- JBossMQ
  - ✓ Evolved from SpyderMQ
  - ✓ Mature implementation
  - ✓ Used in production in numerous environments
  - ✓ However, it has several fundamental limitations ...

JBossMQ limitations

- Performance problems in certain high-load configurations
  - ✓ we will see later performance comparison charts
- Originally designed without HA support
- HA features added later in the form of a HA Singleton
  - ✓ single JMS provider instance in a cluster
  - ✓ recoverability for PERSISTENT messages
  - ✓ in-flight NON-PERSISTENT messages are lost in case of failure.
  - ✓ non-transparent client fail-over

JBossMQ High Availability



JBoss Messaging Project Goals

- Fully compliant JMS 1.1 implementation
  - ✓ Compatibility tested with Sun's TCK
  - ✓ Implementation available standalone as well as AS-integrated
- Goal achieved in 1.0

JBoss Messaging Project Goals (2)

- Improved performance over JBossMQ
  - ✓ Completely new architecture based on Channels (more about this later)
  - ✓ New threading model
  - ✓ Optimized persistence
  - ✓ Better transaction handling
  - ✓ Less serialization
  - ✓ Support for large queues or subscriptions
- Goal achieved in 1.0

JBoss Messaging Project Goals (3)

- A completely new clustering model
- Greatly improved HA and load balancing features
  - ✓ Not a simple HA Singleton anymore
  - ✓ Distributed and replicated destinations available
  - ✓ Transparent client fail-over
- Goal planned to be fully achieved in 1.2

JBoss Messaging Project Goals (4)

- Backbone of JBoss ESB
  - ✓ Messaging Core planned to be used as asynchronous messaging foundation for JBoss ESB
  - ✓ Integration with JBoss ESB sometime in Q3 2006
- In progress

JBoss Messaging Project Goals (5)

- JMS interface to JGroups
- In progress

Features

- JBoss Messaging 1.0 is a fully compliant JMS 1.1 provider
- JMS 1.1 compliance is tested with Sun's J2EE CTS (Compatibility Test Suite)
- Current pass rate is 100%
- No clustering for 1.0
  - ✓ JMS clustering will be available in the 1.2 release

Architecture

- A Messaging sever instance consists of two major (and independent) layers
  - ✓ The Messaging Core
  - ✓ The JMS Façade

The Messaging Core

- Messaging Core is a generic, reliable and distributed messaging transport system
- Does just two things and does them well
  - ✓ Guarantees the *reliability* of a message submission, for those messages that have been configured to be reliable
  - ✓ It is *distributed* by nature, so it can reliably send messages between different address spaces
- Supports generic messages (not necessarily JMS)
- Has a proprietary API

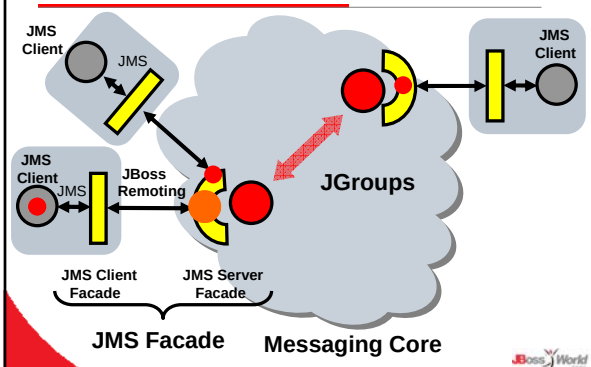
13

The JMS Facade

- Gives JMS "personality" to the Messaging Core
- Implements the JMS API
- Its inner workings are built in top of JBoss AOP
- Stack of aspects plus a set of services

14

The JMS Facade



15

Messaging Core Internals

- The Messaging Core is an aggregation of
  - ✓ Receivers
  - ✓ Channels
  - ✓ Routers

16

Receiver

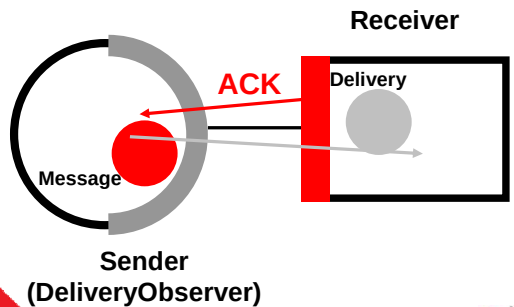
- A Receiver is a the basic message handling component, that

  1) Receives messages for consumption or forwarding
  2) Returns a Delivery object instance for each message it receives

- The receiver then uses the Delivery instance to acknowledge the message, immediately or later
- The sender (implementing DeliveryObserver) hangs on Delivery, and implicitly on message until acknowledgment arrives

17

Receiver



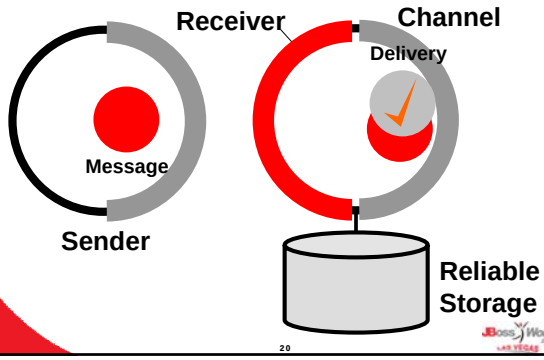
18

Channels and reliability

- In Core, messages flow from senders to receivers and acknowledgments flow back
- As long as a Delivery's corresponding message is reliably stored, the message IS NOT LOST even if both the sender and the receiver crash
- NOT losing messages is the Channel's job

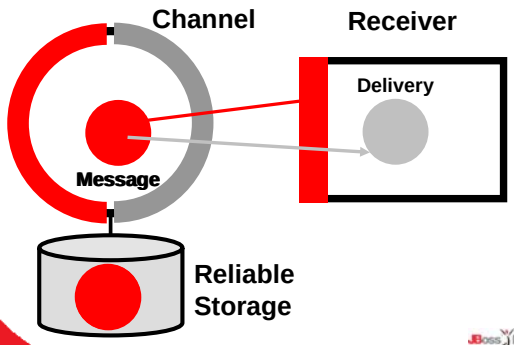
19

A Reliable Channel, part one



20

A Reliable Channel, part two



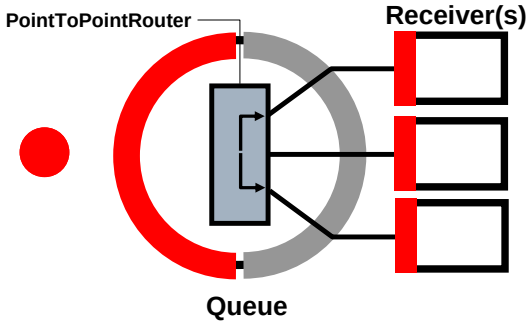
21

All together now ...

- Receivers, Channels and Routers are just the basic building blocks
- They are used to assemble more complex structures, as queues and topics ...

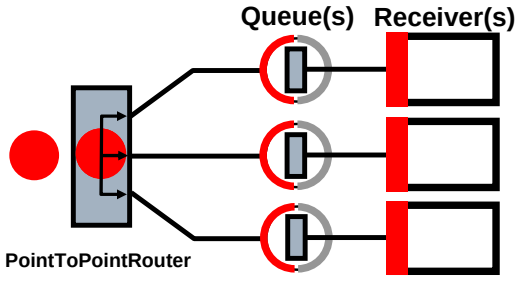
22

A core Queue



23

A core Topic

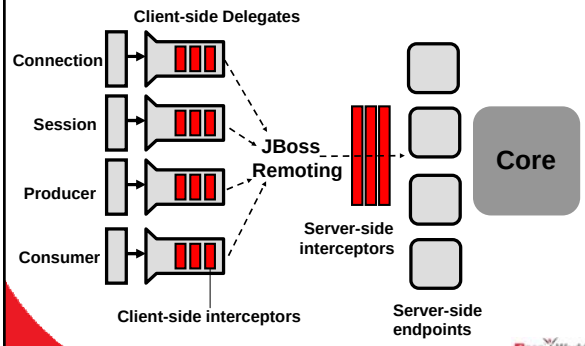


24

The JMS Facade

- The JMS façade gives the JMS "personality" to the Messaging server
- Other different façades could be implemented
- The current implementation is built in top of the JBossAOP framework
- A client-side and server-side AOP aspect stack plus a set of services (MBeans)

The JMS Façade continued



JBoss Messaging internals

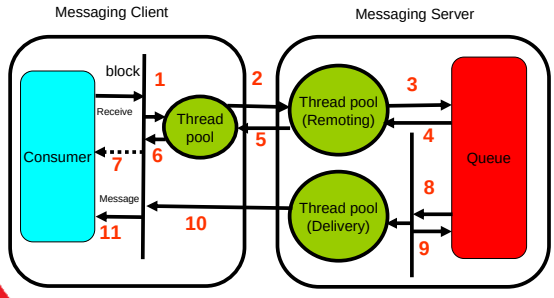
Focus on a few areas of current and soon to be available features

- Threading model
- Transactions
- Persistence
- Serialization and copying
- Large numbers of messages and large messages
- Distributed destinations

Threading

- Threads are a precious commodity
- Minimise threads blocking on server for messages.
- Minimise number of threads used for delivery.

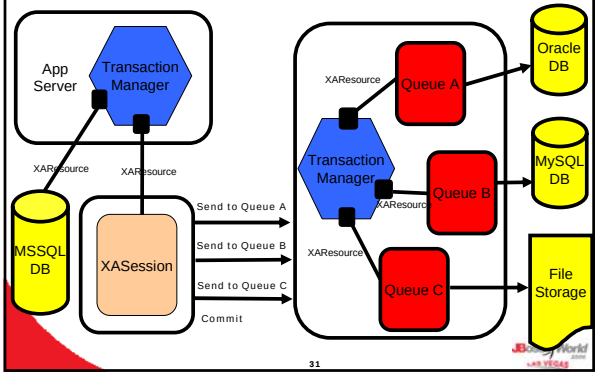
Threading – Message receive



Transactions

- Efficient use of JDBC transactions – batch updates
- JBoss Messaging provides XAResource instances
- Coming soon - Destinations can be enlisted as separate XAResource instances –allowing different stores, allows system to scale.
- Coming soon - with JBoss Transactions will give full XA recoverability
- Coming soon - Separate file based transaction log

Transactions



31

Optimized Persistence

- Many persistence optimizations over and above JBoss MQ
- Currently ships with JDBC Persistence support for MSSQL, MySQL, Oracle, Sybase, PostgreSQL, HSQL
- Local file based persistence support on the way – likely to leverage functionality in JBoss Transactions.

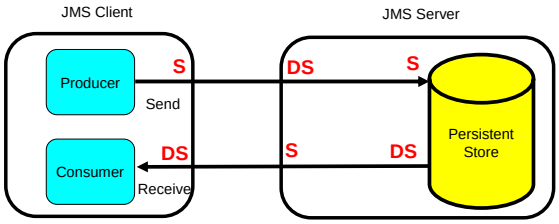
32

Minimise Serialization and copying

- Serialization is expensive – both in *time* and *space*– keep it to a minimum.
- Don't serialize the whole message in the db.
- Don't serialize across the wire if it can be avoided.
- Minimise copying of messages – in some INVM cases no copying is necessary – passing by reference enables high performance
- Pluggable serialization library

33

Serialization – the naïve way

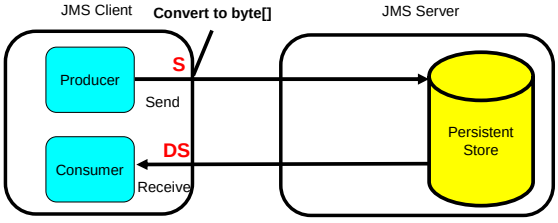


S = Serialize, DS = De-serialize

3 x Serialization + 3 x De-serialization

34

Serialization – a better way



S = Serialize, DS = De-serialize

1 x Serialization + 1 x De-serialization

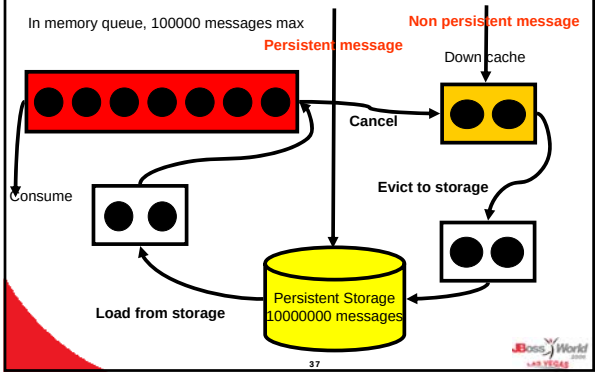
35

Very large queues / subscriptions

- Queues/subs may need to hold many millions of messages.
- Cannot store in memory at once
- Page messages to and from storage as necessary
- JBoss Messaging can handle very large queues
- Coming soon - Support for very large messages – streams, chunks, compression.

36

Page-able Queues/subscriptions



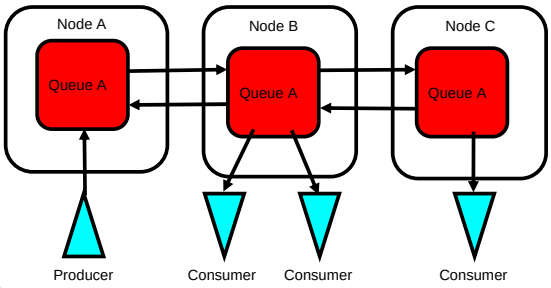
37

Distributed destinations

- State of the art distributed destinations (unlike some competing products)
- Multiple consumers for the same queue distributed across the cluster
- Multiple consumers for the same durable subscription distributed across the cluster
- Combine with multiple persistent stores then we have a highly scaleable distributed messaging system
- Fully recoverable ACID transactions guaranteed across the cluster

38

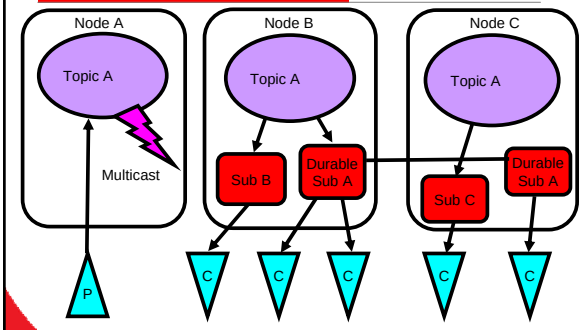
Distributed Queue



Producers and Consumers can connect to any node
Only one consumer will receive the message sent by the producer

39

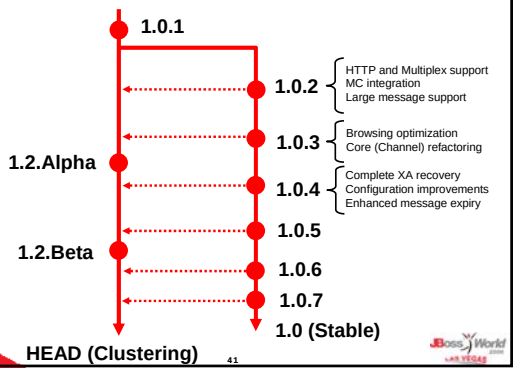
Distributed Topic



Subscriptions on the topic can be spread across the cluster
There can be multiple consumers of the durable subscription on different nodes

40

Roadmap



41

Q&A

Slide 38

T1 Mention
Tim, 6/9/2006