

Building reliable distributed systems

Lecture 5

Giorgia Lodi

lodi@dis.uniroma1.it

Outline

- An introduction to JBoss application server
 - JBoss 4.x architecture
 - JBoss 5.x architecture
 - How to install and run it
- The JBoss Clustering Service
 - The clustering framework
 - Automatic discovery of clustered nodes
 - How JBoss uses JGroups for group communication
 - Distributed cluster-wide hot deployment
 - HA-JNDI
 - Fail-over and load balancing
 - At RMI
 - At HTTP
 - Clustering EJBs

What is JBoss?

- An open source application server that implements the J2EE specifications
 - J2EE specifications provide a component-based approach to the design, development, assembly and deployment of enterprise applications
 - Robust suite of middleware services that rely on Java 2 Standard Edition (J2SE)
 - Enterprise Java Beans
 - Java Servlet and Java Server Pages
 - Java Database Connectivity (JDBC)
 - Java Transaction API
 - Java Message Service (JMS)
 - ...

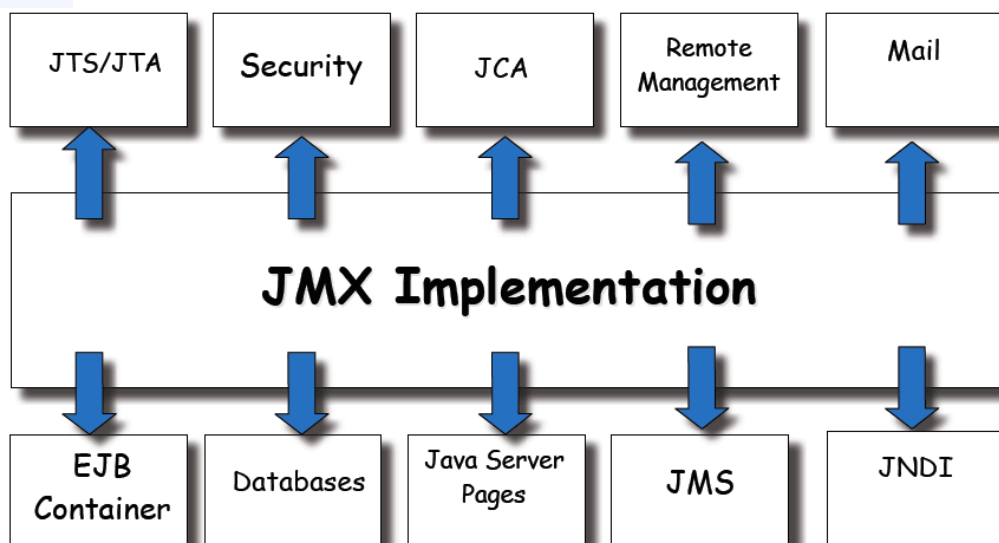
Only JBoss?

- No ☺
 - J2EE application servers
 - Open source
 - JOnAS (ObjectWeb consortium)
 - Geronimo (Apache Software Foundation)
 - ...
 - Commercial
 - BEA Web Logic
 - IBM WebSphere
 - ...
 - Other component-based technologies
 - Corba Component Model (CCM)
 - Microsoft .Net
- The most popular open-source application server
 - Many versions available
 - 3.x
 - 4.x
 - Latest one 5.x

JBoss 4.x Application Server

- Collection of middleware services for communication, persistence, transactions, security, and clustering
- Services interoperate by means of a microkernel, i.e., Java Management eXtension (JMX)
 - JMX provides Java developers with a common software bus
 - Allows them to integrate components (e.g., modules, containers)
 - Components are declared as Managed Beans (MBeans) services
 - Implementation of manageable resources in JBoss
 - Represented by Java objects that expose interfaces
 - Consisting of methods to be used for invoking the MBeans

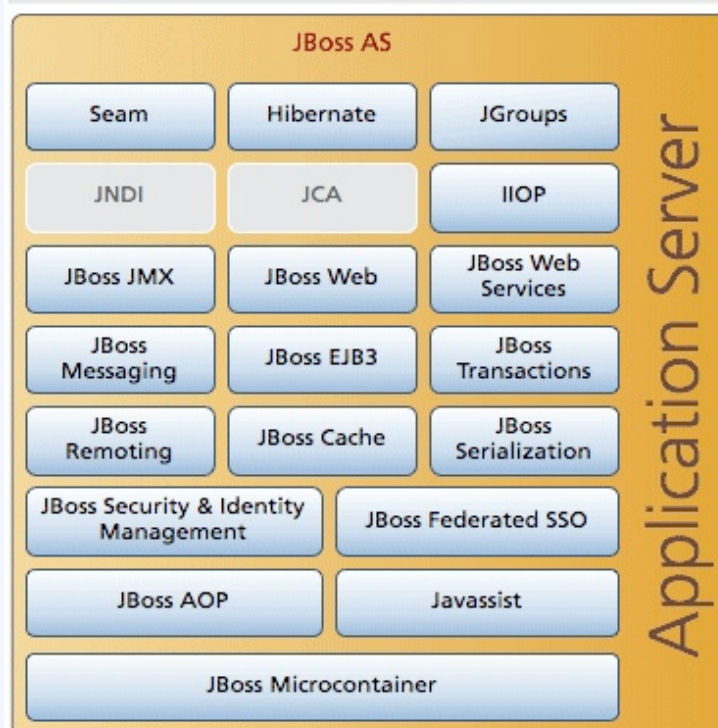
JBoss 4.x Application Server



JBoss 5.x Application Server

- It uses the microcontainer
 - It is a refactoring of JBoss's JMX Microkernel to support direct Plain Old Java Object (POJO) deployment and standalone use outside the JBoss application server
 - It allows the application server to integrate enterprise services with a Servlet/JSP and EJB containers, deployers and management utilities in order to provide a standard Java EE environment
 - If you need additional services
 - simply deploy these on top of Java EE to provide the functionality you need
 - Since it is very lightweight and deals with POJOs it can be also used to deploy services into a Java ME runtime environment
 - New possibilities for mobile applications to take advantage of enterprise services without requiring a full J2EE application server

JBoss 5.x Application Server



JBoss Installation

- You should have already installed java J2DK 1.5 or higher
- Set the JAVA_HOME to point where java is installed
- Download the JBoss Application server
 - You can download the src code and build it
 - jboss-XXX-src.tar.gz
 - You can download the software already built (unzip the file)
 - jboss-XXX.zip

How to run and shutdown JBoss

Run

- cd jboss-XXX/bin
 - ./run.sh (under Linux) or run.bat (under Windows)
 - You are running the default server
- There exist three server confs:
 - Minimal
 - starts the core server container without any of the enterprise services
 - Default
 - all the basic services of the J2EE specifications
 - All
 - default conf + IIOP + clustering and load balancing services

Shutdown

- You can terminate the server by using ctrl-c

JBoss Clustering Service

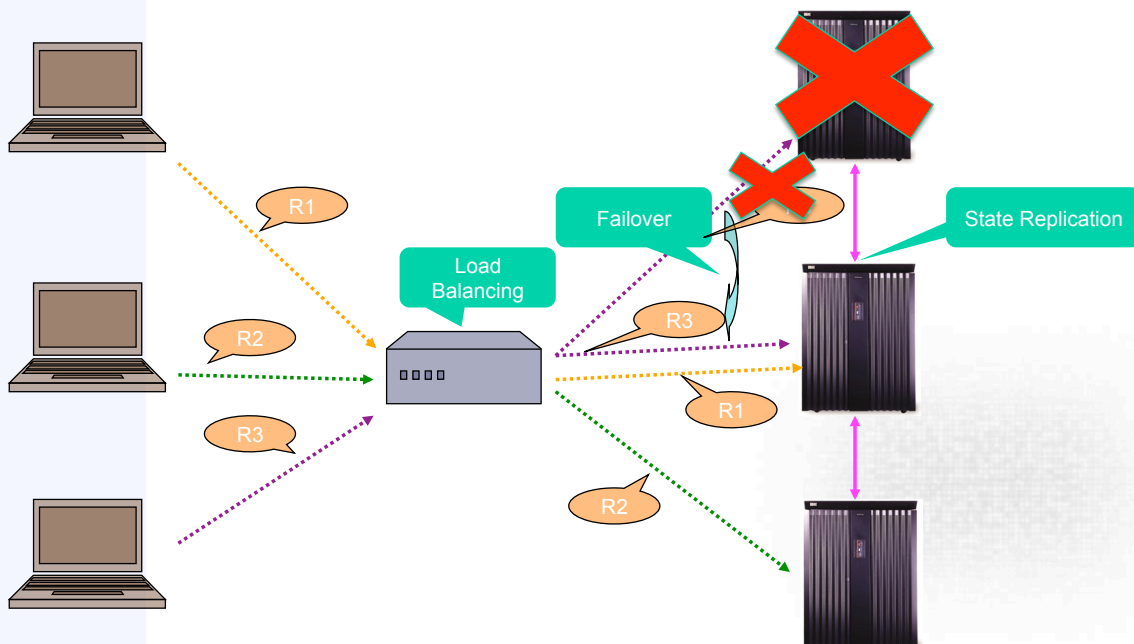
Why Clustering?

- It allows developers to run an application on several parallel servers while providing a single view to application clients
- It is crucial for
 - Fault tolerance
 - If one or more servers fails, the application is still accessible via the surviving servers
 - Load Balancing
 - Load is distributed across different servers
 - Scalability
 - Performance can be improved by simply adding more nodes to the cluster and performing load balancing
 - Highly availability
 - The clustering infrastructure supports the redundancy needed for high availability
 - It is obtained through load balancing and fault tolerance as well

Why JBoss Clustering?

- Transparent
 - No stub re-compiling!
 - Client does not know about clustering
 - Cluster is maintained automatically
 - Modular/stack approach
- Open source
 - No extra cost to activate clustering

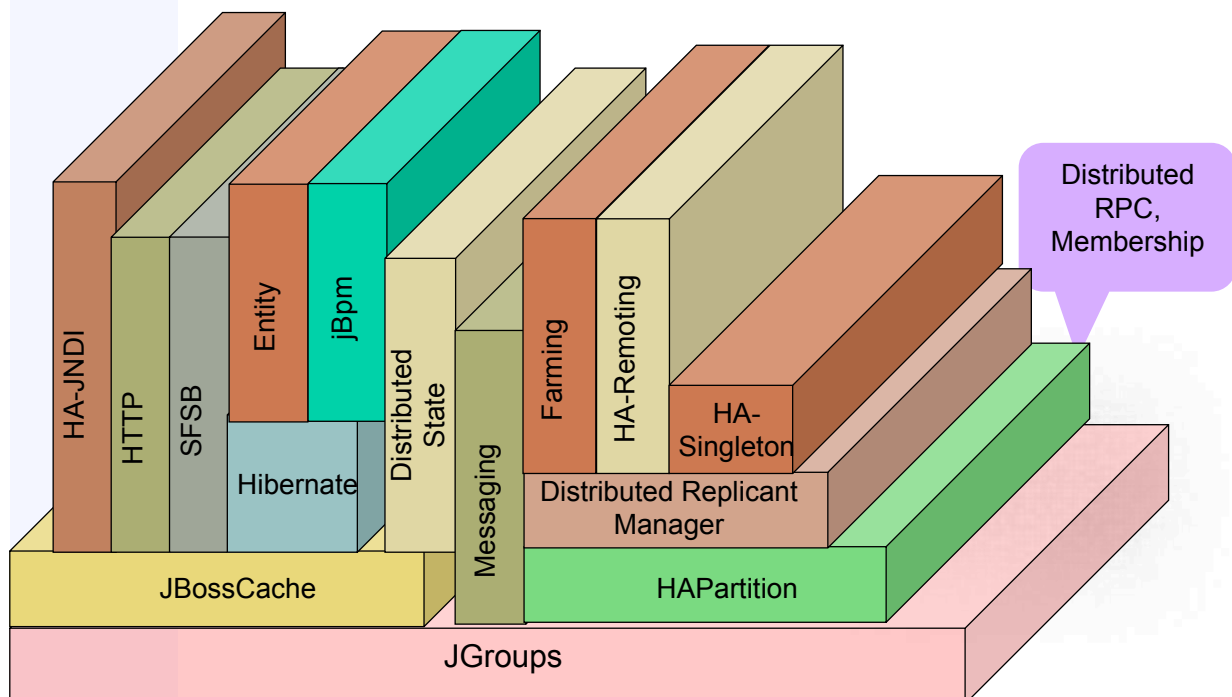
Overview of Clustering functionalities



Running JBoss Clustering

- Use JBoss' "all" configuration
 - run.bat -c all
 - ./run.sh -c all
- The "all" configuration contains everything needed for clustering
 - It has all the libraries for clustering
 - E.g., JGroups.jar, jboss-cache.jar, etc.

Modular/stack approach



Clustering divided in 3 parts

- Cluster communication (inside cluster)
 - Based on JGroups
 - Method calls across all cluster nodes
 - HA-JNDI, Farming, etc...
- Client-cluster communication
 - Formerly HA-RMI
 - Load balancing and failover logic for fat clients
- State replication (inside cluster)
 - JBoss Cache
 - HTTP session, SFSBs, DistributedState

Clustering divided in 3 parts

- Cluster communication (inside cluster)
 - Based on JGroups
 - Method calls across all cluster nodes
 - HA-JNDI, Farming, etc...
- Client-cluster communication
 - Formerly HA-RMI
 - Load balancing and failover logic for fat clients
- State replication (inside cluster)
 - JBoss Cache
 - HTTP session, SFSBs, DistributedState

Cluster definition

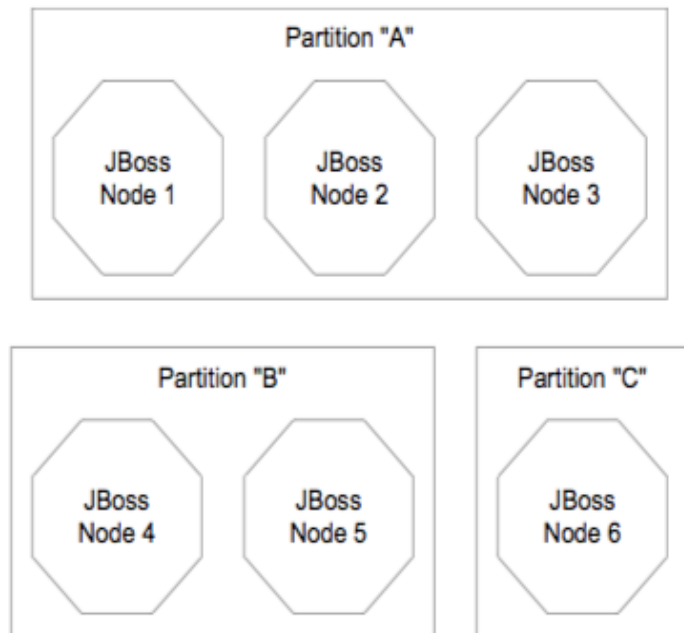
- A JBoss cluster (or partition) consists of a set of nodes
 - Nodes communicate with each other and work toward a common goal
 - A node in JBoss is a JBoss server instance
 - More than one JBoss server instance can be deployed in one single machine
 - Pay attention to the service ports: they must be changed!

JGroups Channels

- Communication between nodes is handled by the JGroups group communication library
- A JGroups Channel
 - Tracks who is in the cluster
 - Enables reliable exchanging of messages between the cluster members
- JGroups channels with the same configuration and name dynamically discover each other and form a group
- Nodes can be dynamically added to or removed from clusters at any time
 - by starting or stopping a Channel with a configuration and name that matches the other cluster members
- In essence, a JBoss cluster is a set of AS server instances each of which is running an identically configured and named JGroups Channel

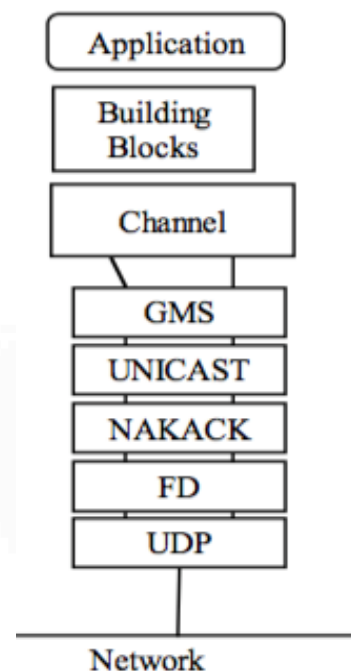
JGroups Channels

- On a same network, even for the same service, we may have different clusters
 - In order to differentiate them, each cluster must have an individual name



JGroups Configuration

- It is built on top of a stack of network communication protocols
 - Protocols provide transport, discovery, reliability and failure detection, and cluster membership management services



JGroups Configuration

- `cluster-service.xml` file in `/deploy` folder
 - Describes configuration for the default cluster partition
- JGroups configurations appear as a nested attribute in cluster related MBean services
- `PartitionConfig` attribute in the `ClusterPartition` MBean is an XML string
 - Describes and configures the JGroups stack of protocols
 - All the JGroups configuration data is contained in the `<Config>` element under the `PartitionConfig` attribute
 - Default configuration uses UDP with IP multicast
 - multicast identified by address and port number

JGroups Configuration

```
<mbean code="org.jboss.ha.framework.server.ClusterPartition"
name="jboss:service={jboss.partition.name:DefaultPartition}">
... ..
  <attribute name="PartitionConfig">
    <Config>
      <UDP mcast_addr="{jboss.partition.udpGroup:228.1.2.3}"
        mcast_port="{jboss.hapartition.mcast_port:45566}"
        tos="8"
        ... ..
        <PING timeout="2000"
          down_thread="false" up_thread="false"
          num_initial_members="3"/>
        <MERGE2 max_interval="100000"
          down_thread="false" up_thread="false"
          min_interval="20000"/>
        .....
        <FD timeout="10000" max_tries="5"
          down_thread="false" up_thread="false" shun="true"/>
        <VERIFY_SUSPECT timeout="1500" down_thread="false"
up_thread="false"/>
        .....
        <pbcast.STATE_TRANSFER down_thread="false" up_thread="false"/>
    ... ..
  </attribute>
</mbean>
```

JGroups Channels

- By default JBoss uses four separate JGroups Channels
 - Two broad categories
 - The Channel used by the general purpose HAPartition service
 - Three Channels created by JBoss Cache for special purpose caching and cluster wide state replication

HA Partition

- HAPartition is a general purpose service used for a variety of tasks in AS clustering
- It is an abstraction built on top of a JGroups Channel
- It provides support for making/receiving RPC invocations on/from one or more cluster nodes
- It supports a distributed registry of which clustering services are running on which cluster members
- It provides notifications to interested listeners when the cluster membership changes or the clustered service registry changes
- It is the core of many clustering services
 - smart client-side clustered proxies
 - EJB 2 SFSB replication and entity cache management
 - farming,
 - HA-JNDI
 - ...

HA Partition: example

```
<mbean code="org.jboss.ha.framework.server.ClusterPartition"
  name="jboss:service=DefaultPartition">
  <! -- Name of the partition being built -->
  <attribute name="PartitionName">
    ${jboss.partition.name:DefaultPartition}</attribute>
  <! -- The address used to determine the node name -->
  <attribute name="NodeAddress">${jboss.bind.address}</
  attribute>
  <! -- Determine if deadlock detection is enabled -->
  <attribute name="DeadlockDetection">False</attribute>
  <! -- Max time (in ms) to wait for state transfer to
  complete. Increase for large states -->
  <attribute name="StateTransferTimeout">30000</attribute>
  <! -- The JGroups protocol configuration -->
  <attribute name="PartitionConfig">... .. </attribute>
</mbean>
```

HA Partition

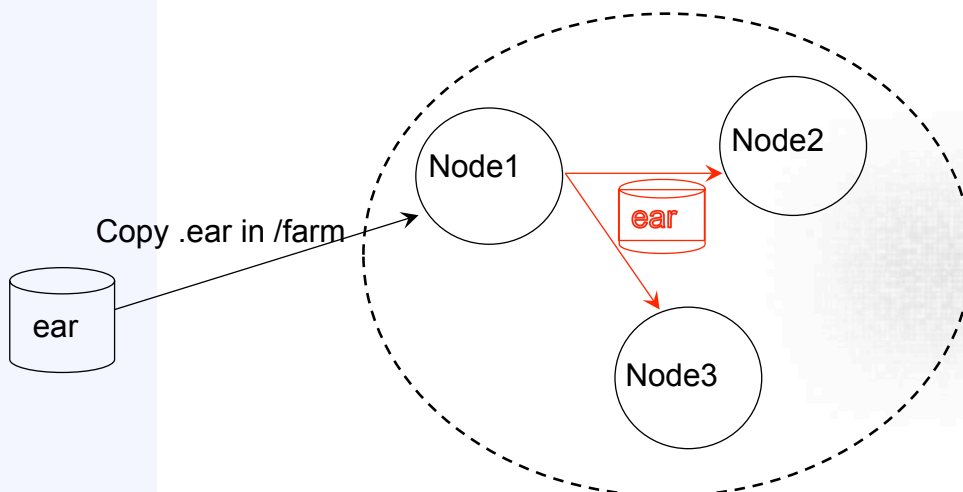
- In order for nodes to form a cluster, they must have
 - the exact same `PartitionName` and the `PartitionConfig` elements
- Changes in either element on some but not all nodes would cause the cluster to split

Types of deployments

- JBoss' staff recommends use of homogeneous deployment
 - Each application component replicated in all clustered nodes
 - Requests do not need to span on different nodes
- Potentially, heterogeneous deployment is also allowed
 - Application components can span on different nodes
 - Not recommended for lack of distributed transaction manager for example

Homogeneous deployment

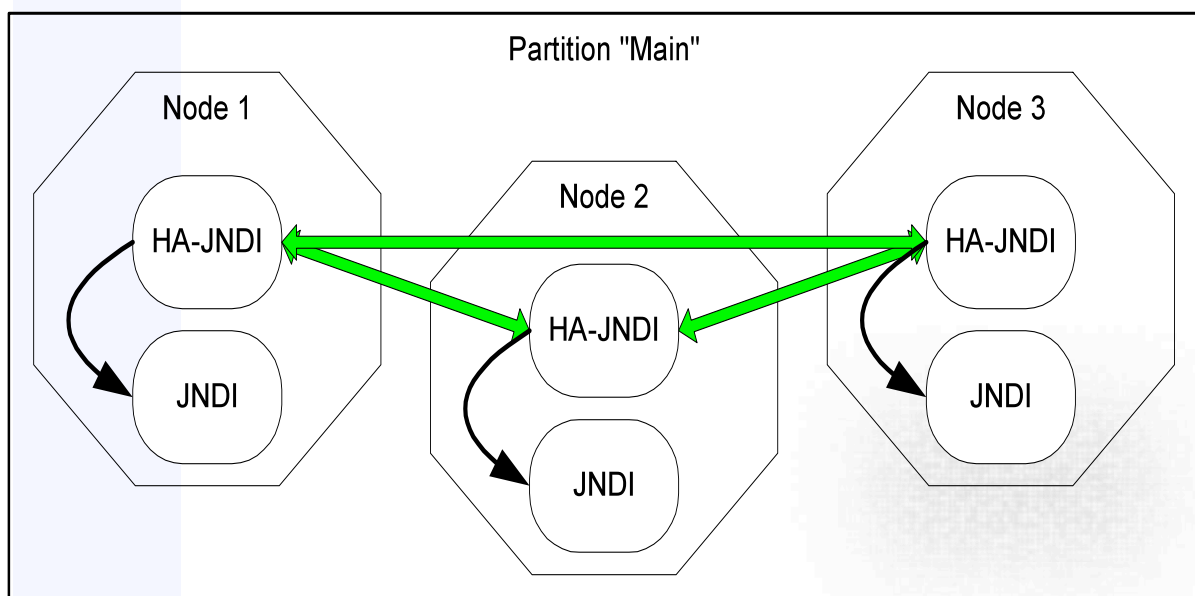
- Realized using JBoss farming service
 - application copied into JBoss /farm directory



Clustering JNDI

- The client obtains an HA-JNDI proxy object and invokes JNDI lookup services on the remote server through the proxy
- On the server side, the HA-JNDI service maintains a cluster-wide context tree
 - The tree is always available as long as there is one node left in the cluster
 - Each node in the cluster also maintains its own local JNDI context tree
 - The HA-JNDI service on that node is able to find objects bound into the local JNDI context tree

Clustering JNDI



Clustering JNDI

- A remote client does a lookup through HA-JNDI
 - If the binding is available in the cluster-wide JNDI tree, return it
 - If the binding is not in the cluster-wide tree, delegate the lookup query to the local JNDI service and return the received answer if available
 - If not available, the HA-JNDI services asks all other nodes in the cluster if their local JNDI service owns such a binding
 - returns the answer from the set it receives
 - If no local JNDI service owns such a binding, a `NameNotFoundException` is finally raised

Clustering divided in 3 parts

- Cluster communication (inside cluster)
 - Based on JGroups
 - Method calls across all cluster nodes
 - HA-JNDI, Farming, etc...
- Client-cluster communication
 - Formerly HA-RMI
 - Load balancing and failover logic for fat clients
- State replication (inside cluster)
 - JBoss Cache
 - HTTP session, SFSBs, DistributedState

JBoss Clustering Configurations

- Two types of clustering configurations
 - JBoss fat client clustering
 - JBoss thin client clustering

JBoss Fat Client Clustering

- Client uses smart proxies
- HA smart proxy has logic inside
 - Load Balancing policies, e.g., RoundRobin
 - Failover capabilities
- HA smart proxy logic can be pluggable
 - User can define his/her own logic
 - Failover policy
 - Load balancing based on weight or load

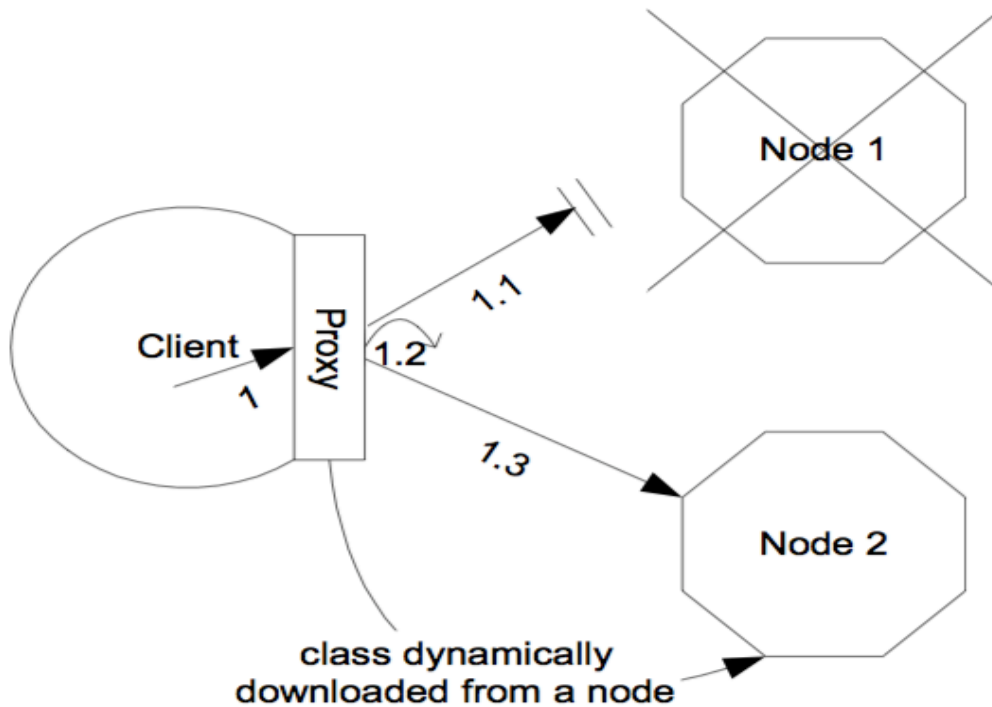
JBoss Thin Client Clustering

- Front end load balancing mechanism
 - Hardware
 - JBoss group suggests F5 BIG-IP
 - Software
 - Apache + JBossWeb
 - Apache mod_jk/mod_proxy
 - Tomcat

Fail-over and load balancing: RMI

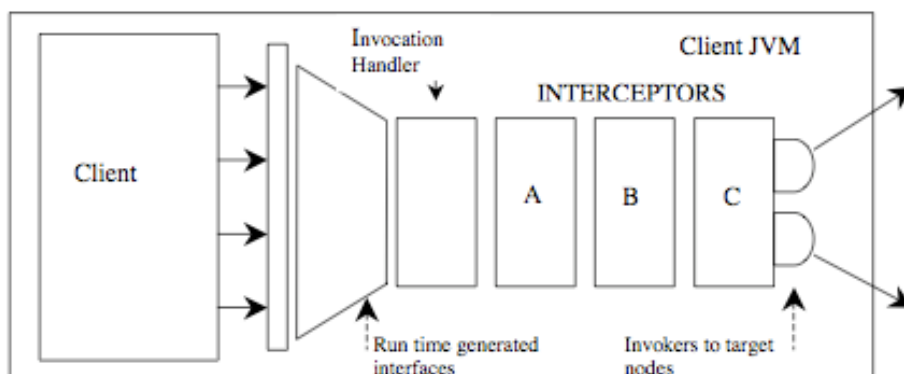
- RMI fail-over and load balancing
 - Clustering logic included into the client stubs (HA-RMI)
 - Stub contains the list of available clustered nodes + load balancing policy
 - Random
 - Round Robin
 - First Available
 - If cluster topology changes, next client invocation server piggybacks new list of target nodes
 - Works only on remote invocations
 - The list of target nodes is maintained automatically, using JGroups
 - The client stub
 - receives a reply from the invoked server
 - unpacks the list of target nodes from that reply
 - updates the current list of target nodes with the received one
 - terminates the client RMI

Fail-over and Load Balancing: RMI

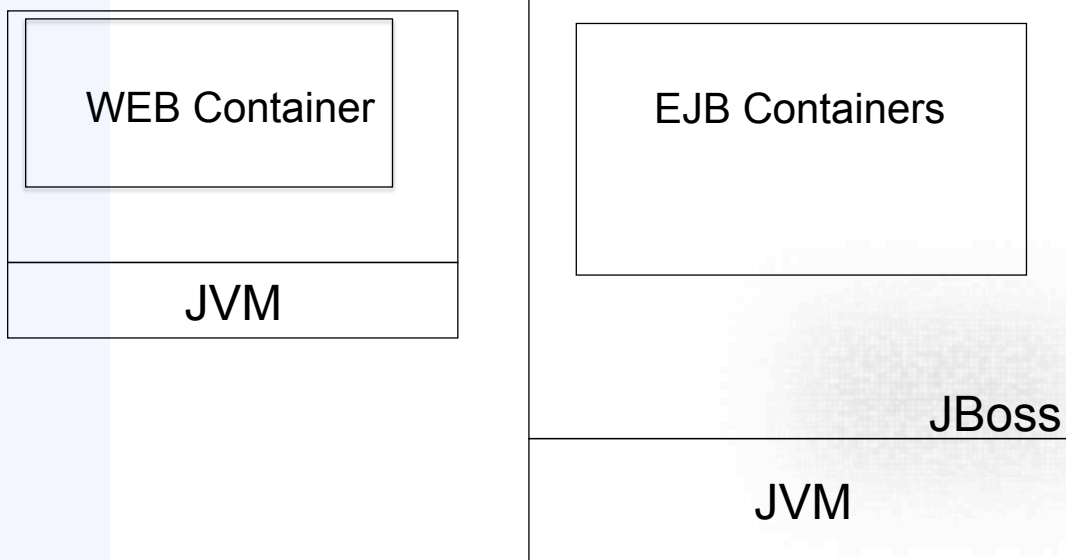


HA RMI: Interceptors

- When the stub's interface is invoked, the invocation is translated from a typed call to a de-typed call
- The de-typed invocation is passed through a set of client-side interceptors
- The load balancing and failover mechanisms are located in the last interceptor of the chain

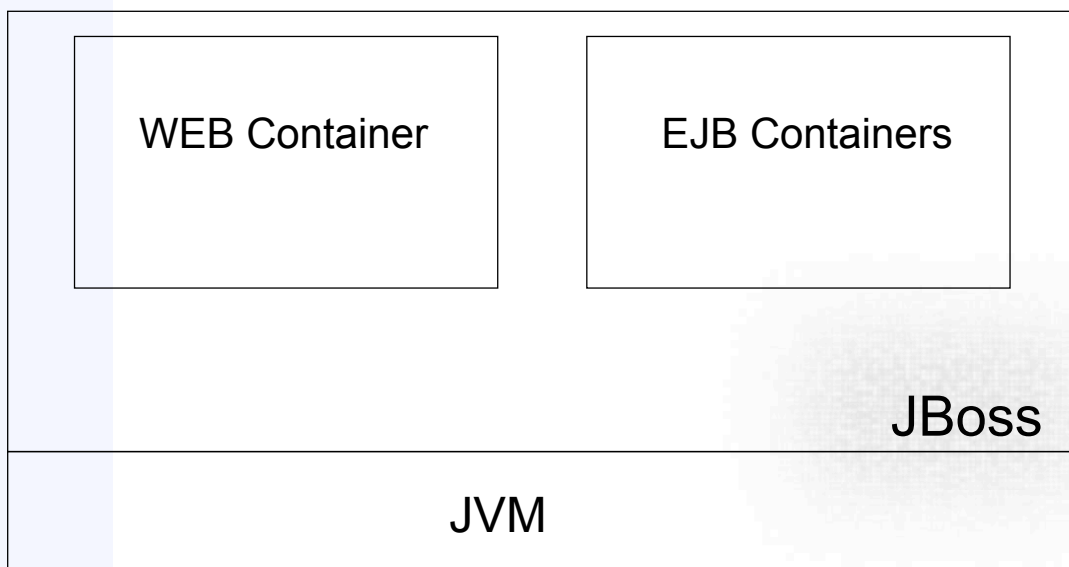


Fail-over and load balancing: RMI



RMI fail-over and load balancing are possible!!

Fail-over and load balancing: RMI



RMI Fail-over and load balancing do not apply!!!

Clustering divided in 3 parts

- Cluster communication (inside cluster)
 - Based on JGroups
 - Method calls across all cluster nodes
 - HA-JNDI, Farming, etc...
- Client-cluster communication
 - Formerly HA-RMI
 - Load balancing and failover logic for fat clients
- State replication (inside cluster)
 - JBoss Cache
 - HTTP session, SF SBs, DistributedState

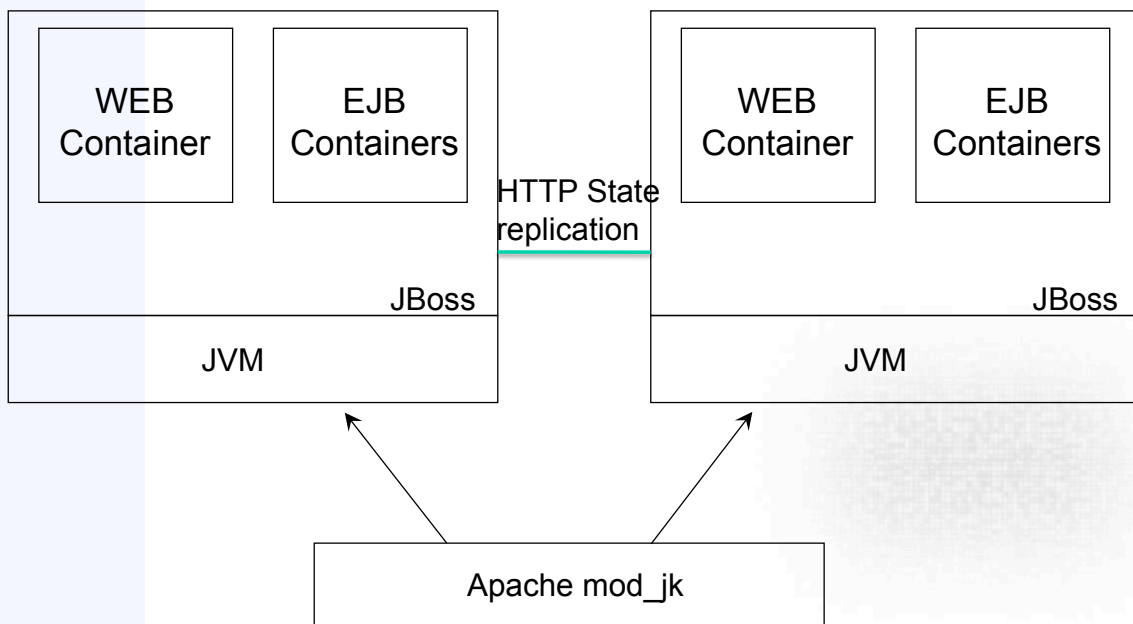
JBoss Cache

- JBoss Cache is a fully featured distributed cache framework that can be used standalone or in any application server environment
 - It caches frequently accessed Java objects in order to dramatically improve the performance of applications
 - it is easy to remove data access bottlenecks - such as connecting to a database
- In JBoss, it provides cache services for
 - HTTP sessions
 - EJB 3.0 session beans
 - EJB 3.0 entity beans
 - Each cache service is defined in a separate Mbean, and creates its own JGroups Channel
- JBoss Cache is a cluster-aware cache; that is, state is always kept in sync with other servers in the cluster when there are concurrent updates
 - Any state stored in JBoss Cache is resilient to server crashes or restarts
 - It can either invalidate or update its state

Fail-over and Load Balancing: HTTP

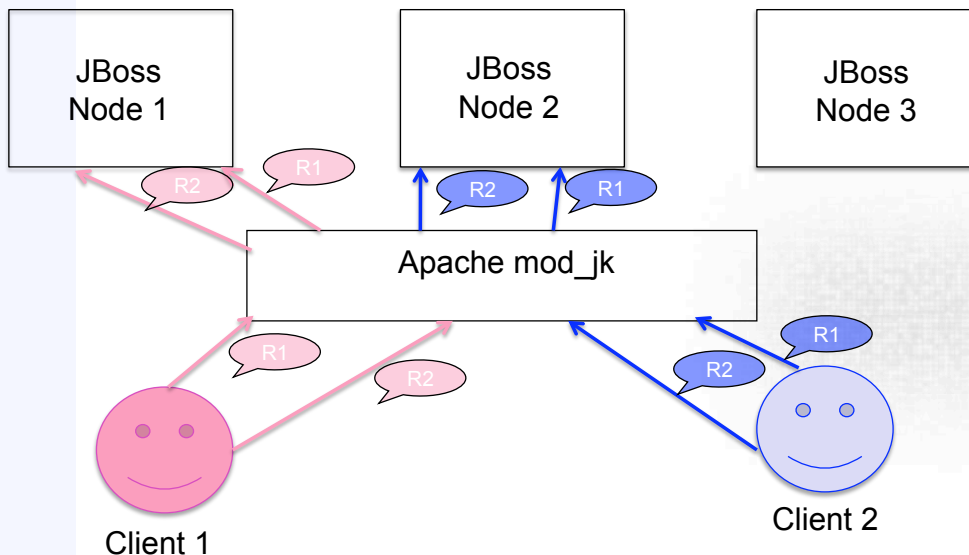
- Session state replication handled by JBoss
 - in "all" JBoss conf included files for session state replication
 - configure HTTP session replication with JBoss Cache
- Load balancing handled by external software or hardware components
 - typically use software components such as Apache+mod_jk (recommended by JBoss)

Fail-over and Load Balancing: HTTP



Alternately...

- If you use sticky sessions you do not need HTTP session replication
- All requests coming from the same client are sent to the same node
- In the example round robin policy is applied per-client



Sticky Session: Pros and Cons

- Pros
 - You may disable HTTP session replication that is an expensive operation
 - All requests from the same clients are processed by the same server
- Cons
 - If server that serves the requests fails, you lose your session state
- It is a tradeoff and depends on the type of application you are deploying

Hybrid solution: Sticky session and HTTP state replication

- You may enable both sticky session and HTTP session replication
 - In case of server crash the client session is replicated in the remaining servers
- Asynchronous vs synchronous HTTP replication
 - In case you use sticky session, it may be better enabling an asynchronous HTTP replication that is more scalable
- You can also specify the granularity of the replication for optimization purposes
 - Entire session
 - Modified attributes
 - More efficient than session granularity
 - Modified field
 - If size of the attribute is huge, replication of modified field is efficient

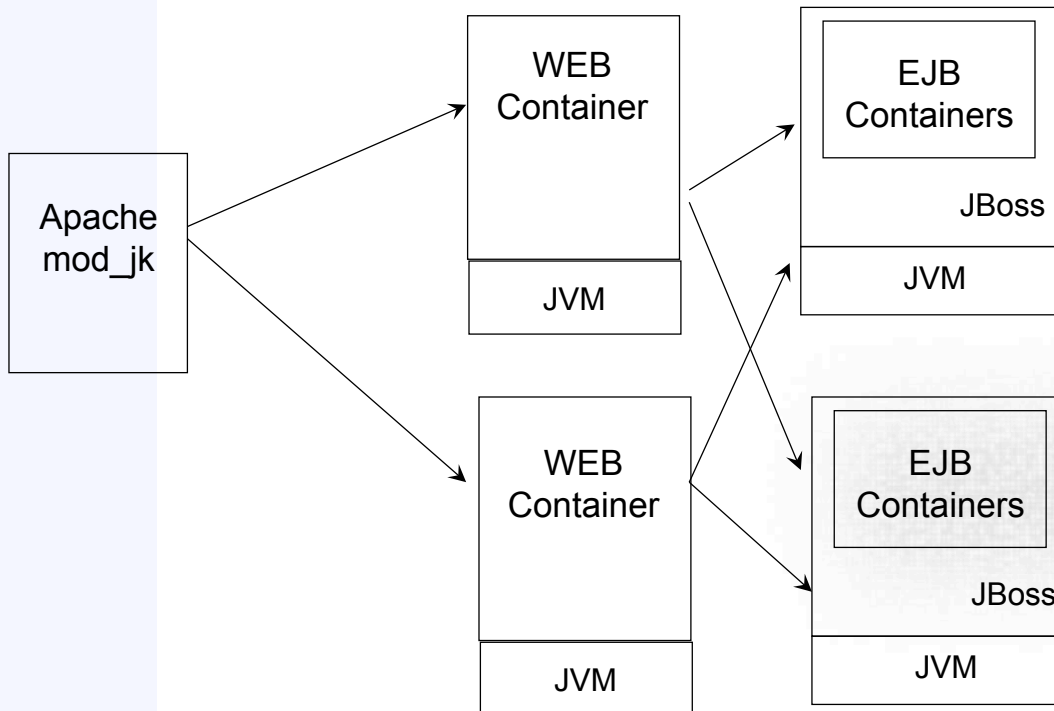
HTTP session replication: configuration

- Session replication is enabled per web application in `WEB-INF/web.xml`
 - Add `<distributable/>` to your webapp
- JBoss specific stuff (asynchronous vs synchronous replication) goes into `WEB-INF/jboss-web.xml`
- Session replication is configured in Tomcat

```
deploy/tc5-cluster-service.xml
```

```
...  
<!-- Valid modes are LOCAL, REPL_ASYNC and REPL_SYNC -->  
<attribute name="CacheMode">REPL_SYNC</attribute>  
...
```

Hybrid: Thin and Fat Client Clustering



Clustering EJBs

- **Stateless Session Beans**
 - As no state involved, calls can be load balanced on any node
- **Stateful Session Beans**
 - It is necessary to manage the state
 - The states of all stateful session beans are replicated and synchronized across the cluster
 - Each time the state of a bean changes
 - JBoss Cache provides the session state replication for EJB 3.0 stateful session beans

Clustering EJBs

- Entity Beans
 - 2.x: do not use clustering
 - Introduces data synchronization problems
 - they do not have a distributed locking mechanism and a distributed cache
 - 3.0
 - Entity bean clustering service deals with distributed caching and replication
 - Through JBoss Cache
- In general, you have to annotate your application in order to instruct the EJB container that the bean is clustered

Troubles

- If you are not able to successfully run JBoss clustering work, it does not mean that it does not work 😊
 - Network issues (firewall, multicast disabled, etc.)
 - JGroups issues
 - JGroups comes with a handy set of Trouble Shooting tools

Exercise JBoss clustering

- Try to write a "hello world" J2EE application
 - You find some solutions via web
 - Remember to annotate it in order to instruct JBoss that the application is clustered
- Deploy it homogeneously in a JBoss cluster
- Have fun!

References

- Bela Ban, "JBoss Clustering Overview", 2006
- JBoss Application Server 4.x, Chapter 16. Clustering, "High Availability Enterprise Services via JBoss Clusters", available online at
 - <http://docs.jboss.org/jbossas/jboss4guide/r4/html/cluster.chapt.html>
- Brian Stansberry, Galder Zamarreno JBoss Application Server 5, "Clustering Guide", November 2008
- Any other problem?

www.google.com