HA Joram

1 Overview

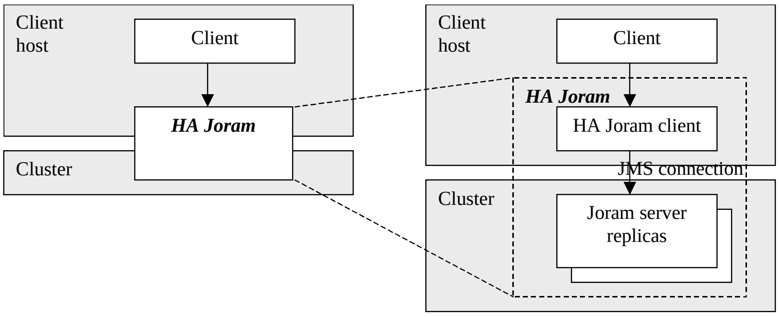
This specification describes how to build a Joram server providing High Availability (HA) by replicating it on a cluster. Two replication modes are specified: one for a remote client connected to the HA Joram server and one for a collocated client.

1.1 Remote client mode

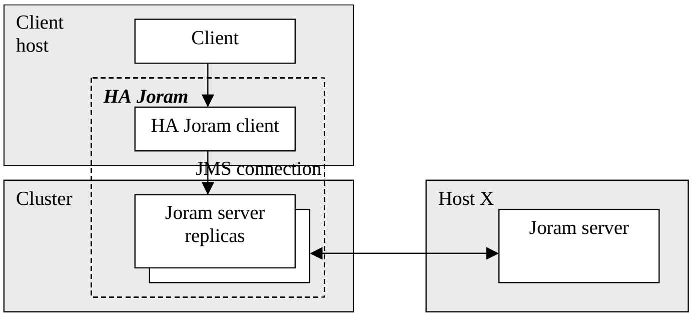
In this mode the Joram client is not replicated. Only the Joram server is replicated. There are two possible architectures depending on whether the remote client is directly connected to the replicated Joram server or indirectly through a standard Joram server.

1.1.1 Direct connection

The client is directly connected to the replicated Joram server. The HA Joram client layer manages the JMS connection with the Joram server replicas.

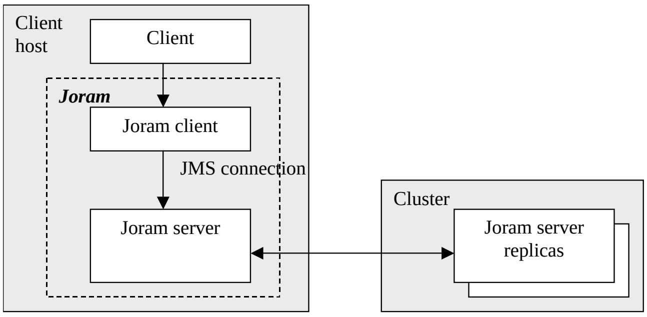


The JMS proxy of the client is deployed on the replicated server. The destinations used by the client may be deployed on the replicated server or on another server (replicated or not).



1.1.2 Indirect connection

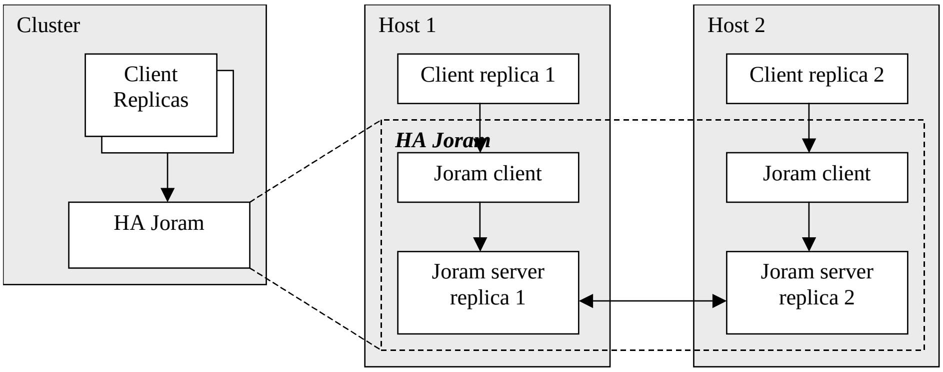
A client that needs *store and forward* facilities must be collocated with a local standard Joram server. The user proxy is deployed on the collocated Joram server. But the destinations can be deployed on a remote HA server. In this case the client is not directly connected with the HA server but indirectly through a standard Joram server.



1.2 Collocated client mode

This mode enables to replicate a client if its execution model is "event-reaction", i.e. it cyclically reacts to events (JMS messages) coming from some destinations (in a pushed or in a pulled way).

Only one replicated client is allowed to work at the same time: the one collocated with the active Joram replica. There is not any HA client layer because the clients are connected to their local server so the reconnection mechanism is useless.



Use constraints

The quality of service (QOS) provided by this mechanism is either "Duplicate-okay" or "At-most-once" depending on whether the client reacts before or after acknowledging the received messages. For an "Exactly-once" QOS the client must use XA transactions.

Moreover when subscribing to topics, durable subscriptions should be used in order not to lose any messages when the active replica fails.

2 Replication scheme

The replication strategy is master ownership and eager (synchronous) propagation.

2.1 Master ownership

The master-slave has been chosen for two reasons:

- when a client is remote (not replicated) it cannot use group communication (multicast) with the HA Joram but a point-to-point protocol like TCP. So the only way to ensure the consistency of the replicas is to designate one of them to be the front-end responsible for the connection with the client. The others must be updated by this replica.
- when the client is collocated (replicated), only one replica can be active at once: the master.

2.2 Eager propagation

The eager propagation is necessary to ensure the consistency of the replicated servers. In particular the messages sent by the client must not be lost. In order to prevent those message losses, the communication between the master and slaves must be synchronous.

Notice that some "asynchronous propagation" can be implemented in order to improve the performances. This propagation implies that the client buffers the messages it has sent in order to re-send them in case of failure. But it is nevertheless necessary to regularly synchronize the replica updates in order to enable the client to reset its buffer. So it is more a loose synchronous communication than a real asynchronous one.

Moreover if a message is delivered to any replica, it must also be delivered to every other replica (atomic delivery requirement: all-or-none).

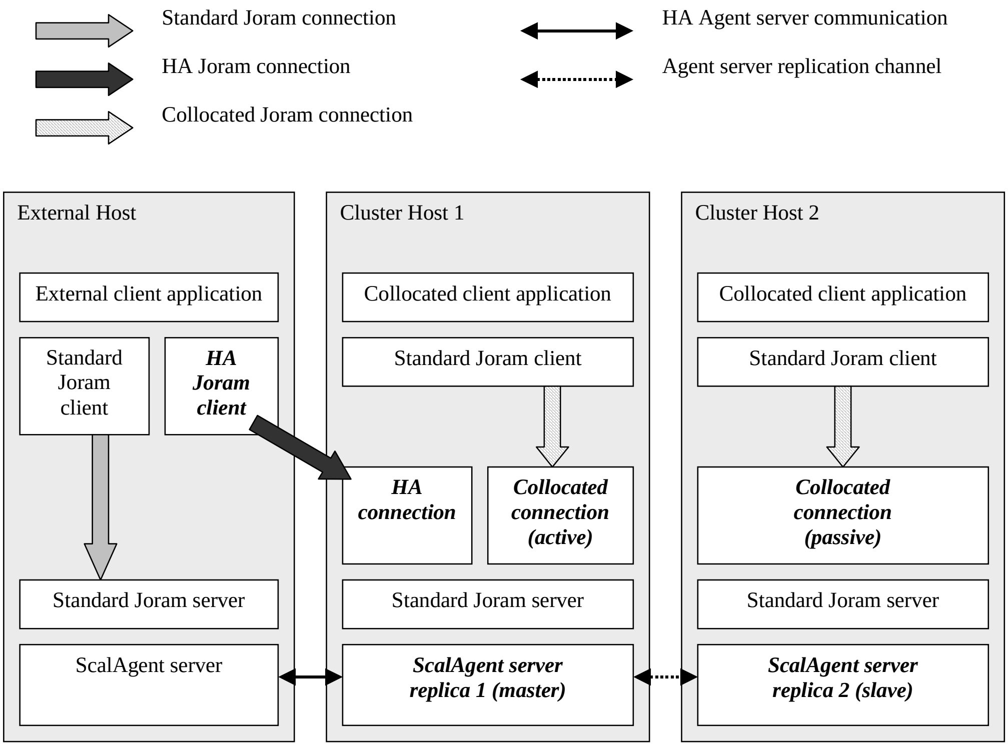
3 Architecture

The HA Joram is built on top of the HA ScalAgent platform that implements a master ownership with eager propagation strategy. A HA server is actually a group of servers, one of which is the master server that coordinates the other slave servers. An external server that communicates with the HA server is actually connected to the master server.

Each replicated Joram server executes the same code as a standard Joram server except for the communication with the clients. Two specific modules are defined: one for the HA connections (with remote clients) and the other for the collocated connections (with collocated clients).

On the client side only the remote clients need to be extended with a *HA Joram client* module. This module is responsible for seamlessly reconnecting to the HA server when the connection fails, i.e. to the same replica in case of a connection failure or to the new master replica if the current one failed.

The collocated clients use the standard *Joram client* module. If the server replica is the master then the connection is active enabling the client to use the HA Joram server. If the server replica is a slave then the connection opening is blocked until the replica becomes the master.


Standard Joram connection
HA Joram connection
Collocated Joram connection
HA Agent server communication
Agent server replication channel
External Host
External client application
Standard Joram client
HA Joram client
Standard Joram server
ScalAgent server
Cluster Host 1
Collocated client application
Standard Joram client
HA connection
Collocated connection (active)
Standard Joram server
ScalAgent server replica 1 (master)
Cluster Host 2
Collocated client application
Standard Joram client
Collocated connection (passive)
Standard Joram server
ScalAgent server replica 2 (slave)

4 HA ScalAgent platform

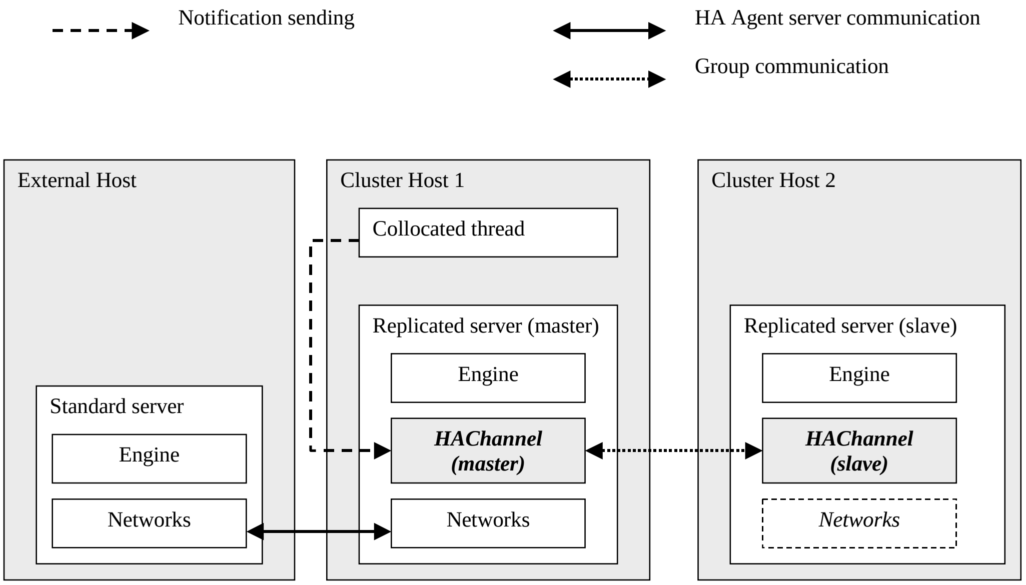
4.1 Architecture

A HA server is a group of servers replicated through an active duplication mechanism. All the servers are active, processing the same notifications in the same order. As a consequence their state are identical. The duplication coordination is ensured by a master replica. This coordination needs some specific communication properties such as reliable (synchronous) atomic FIFO multicast, failure detection and coordinator (master) election. *JGroups* (http://www.jgroups.org) is a reliable group communication toolkit written entirely in Java that provides these properties.

A HA server can communicate with other agent servers. Those servers are called “external servers”. An external server actually communicates with the master replica, the only one which networks (communication layer) are activated. The slave replicas are not directly accessible by an external server. Notice that an external server can also be an HA server.

A HA server can also receive notifications from a thread collocated with the master replica.

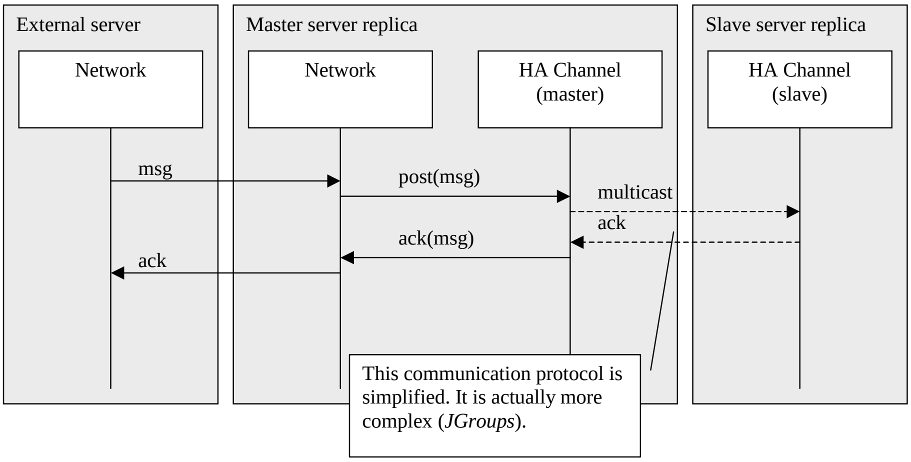
The following figure presents the architecture of the HA platform. Three servers are displayed. The first one is a standard server. Its two main components are the *Engine* that distributes the notifications to the local agents and the *Networks* in charge of the communication with other servers. The two other servers are two replicas from the same HA server. One is the master and the other a slave. They both own an additional component called *HAChannel* that ensures the active duplication. The *Networks* are only active on the master replica.



4.2 Communication with external servers

4.2.1 Receive

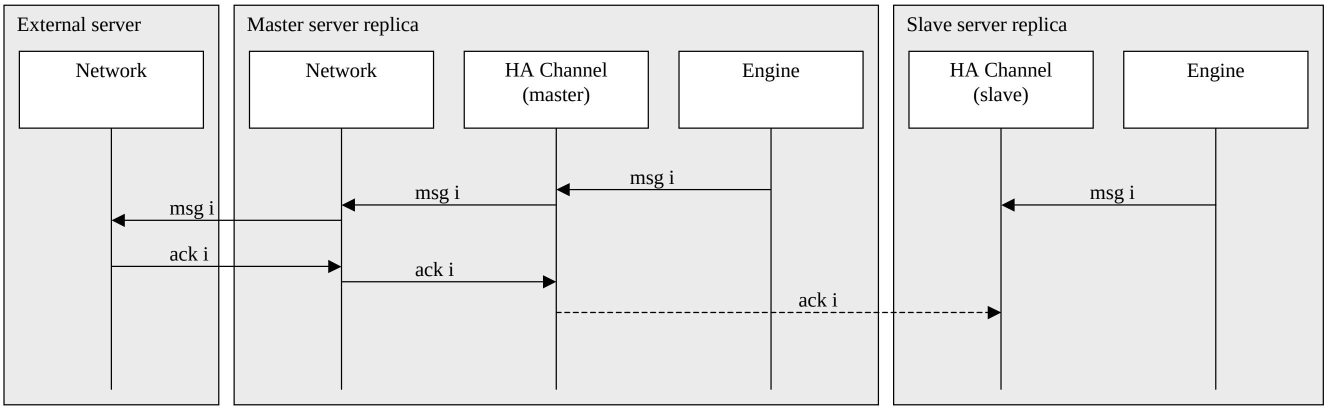
The network of the external server finds the address of the master server through its configuration file (*a3servers.xml*). An external server always communicates with the master server (slave servers are not directly accessible). So the master HA channel is the unique entry point for the external notifications which are hence serialized. The master channel multicasts the notifications to all the slave servers. As the multicast communication is FIFO, every slave channel consumes the notifications in the same order as the master. The consistency all the replicated servers also implies that the communication is reliable and atomic (a message will be received by all receivers, or none). Moreover the multicast is synchronous so the master server network can acknowledge the notification reception as soon as the notification has been broadcasted.



Notice that the master can decide to acknowledge the message before having received all the replies. *JGroups* allows to wait for the majority of all receivers to respond (mode *GET_MAJORITY* in *MessageDispatcher*).

4.2.2 Send

A HA server can send notifications to an external server. As each server replica reacts to the same notifications in the same order, each sends the same notifications in the same order. But only the master channel sends the notification to the external server (through its network). The slaves wait for the acknowledge in order to destroy the message. This acknowledge is received by the master channel that multicasts it to the slaves.

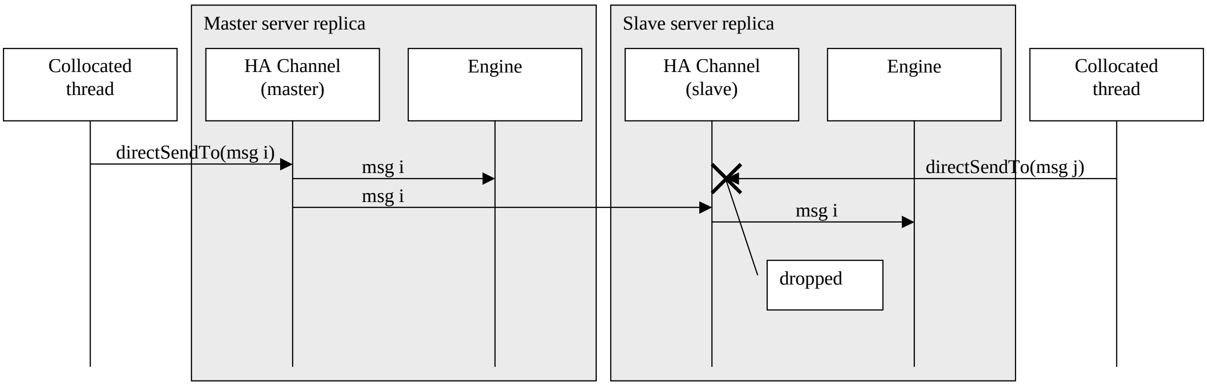


Notice that the acknowledge propagation could be loosened to an asynchronous broadcast (without reliability nor ordering). If a replica does not receive the acknowledge and if it becomes the master, then it will reemit the message to the external server. But as the message has already been received by the external server, it will be discarded.

4.3 Communication with collocated threads

4.3.1 Receive

A collocated thread can send a notification to an agent, by calling the method *directSendTo* of the *Channel*. The HA channel multicasts the message if it is the master or drops it if it is a slave.



4.3.2 Send

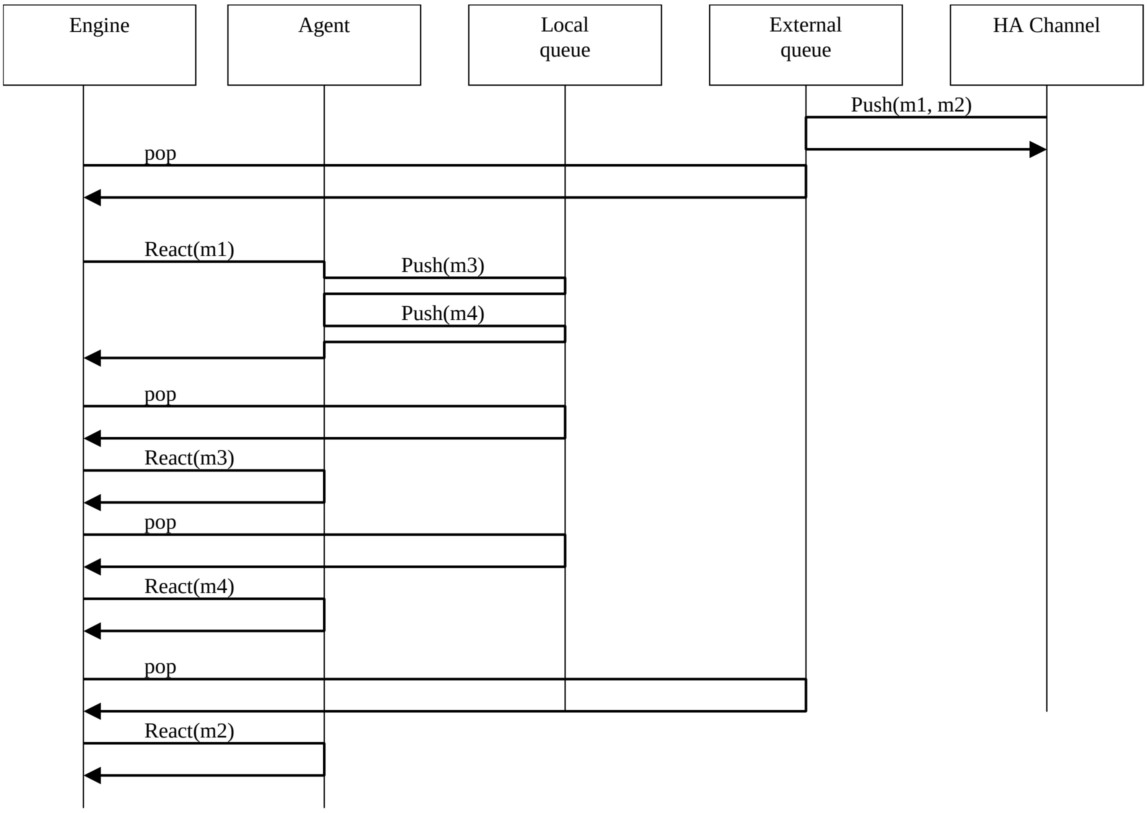
A replicated server can send messages to a collocated thread through a message queue (not a JMS queue, just a Java object that enables two threads to communicate).

Notice that if a collocated thread expects a reply from such a queue it will receive the reply whether the server is master or not as all the servers (master and slaves) handle the same notifications and execute the same reactions in the same order. So it is up to the collocated thread to change the treatment depending on the replica status master/slave.

4.4 Internal communication (Engine)

The HA channel pushes the messages in the local engine. All the replica engines must consume the messages in the same order. To ensure this, the engine owns two queues: the local and the external queues. The local queue is where the notifications from the local agents are posted. The external queue is where the external notifications (from the HA channel) are posted. The engine first gets the messages from the local queue. If it is empty, it consumes one message from the external queue and make the recipient local agent react. This agent may post notifications into the local queue. In this case the engine consumes those notifications, making the local agents react. When the local queue is empty, it consumes one notification from the external queue etc.

Notice that this mechanism implies that the local agents don't indefinitely send notifications to each other. The notifications from the external queue would never be consumed.



4.5 Failure and master election

The multicast communication layer (*JGroups*) is responsible for the failure detection (based on time-out) and the master election. All the replicas belong to a group. When a replica fails, a new *view* of the group is sent to each replica. In particular, this view indicates the replica that is the new master (group coordinator). The atomicity of the multicast ensures that the remaining replicas are consistent.

4.6 Replica configuration

4.6.1 Services

The services are only started on the master replica (not on the slaves). When a slave replica becomes the master, it starts its services.

A service is specified by a Java class that defines a method called *init* with one boolean parameter *firstTime* indicating if the service is started for the first time or not. When the first master replica starts the service, the boolean *firstTime* is true. When a slave replica becomes master and starts, the boolean *firstTime* is false.

Some services may deploy agents the first time they are started. If the first master replica fails during the initialization it is necessary to stop all the replicas and start again in order to ensure that all the agents have been deployed. Moreover the agents deployed by this service must be instantiated with a reserved identifier in order not to interfere with the identifiers generated by the replicas.

4.6.2 Networks

The declared networks must be the same for all the replicas. However, each network replica may have a different port value.

4.6.3 Configuration descriptor

One new element is added in the configuration DTD (*a3config.dtd)*: *cluster*. The element *cluster* defines a replicated server. It owns two attributes: the identifier and the name of the HA server (seen by the other servers).

Each replica is defined as a standard server except that the server identifier and name are defined at the *cluster* level. The identifier defined at the server level identifies the replica inside the cluster (two clusters can share the same replica identifiers).

```
<config>
  <domain name="D1"/>

  <cluster id="0" name="s0">
    <server id="0" hostname="localhost">
      <network domain="D1" port="16301"/>
      <service class="org.objectweb.joram.mom.proxies.ConnectionManager"
               args="root root"/>
      <service class="org.objectweb.joram.mom.proxies.tcp.TcpProxyService"
               args="2560"/>
    </server>
    <server id="1" hostname="localhost">
      <network domain="D1" port="16302"/>
      <service class="org.objectweb.joram.mom.proxies.ConnectionManager"
               args="root root"/>
      <service class="org.objectweb.joram.mom.proxies.tcp.TcpProxyService"
               args="2561"/>
    </server>
    <server id="2" hostname="localhost">
      <network domain="D1" port="16303"/>
      <service class="org.objectweb.joram.mom.proxies.ConnectionManager"
               args="root root"/>
      <service class="org.objectweb.joram.mom.proxies.tcp.TcpProxyService"
               args="2562"/>
    </server>
  </cluster>
</config>
```

4.7 Replica startup

4.7.1 Replicas startup synchronization

If the replicas are started all at once, all of them may think that they are masters. In order to avoid this situation (more than one master), each replica is told the number of replicas that must initially join the cluster. As long as this number is not reached, the replicas cannot start.

4.7.2 Starting parameters

The starting parameters are:

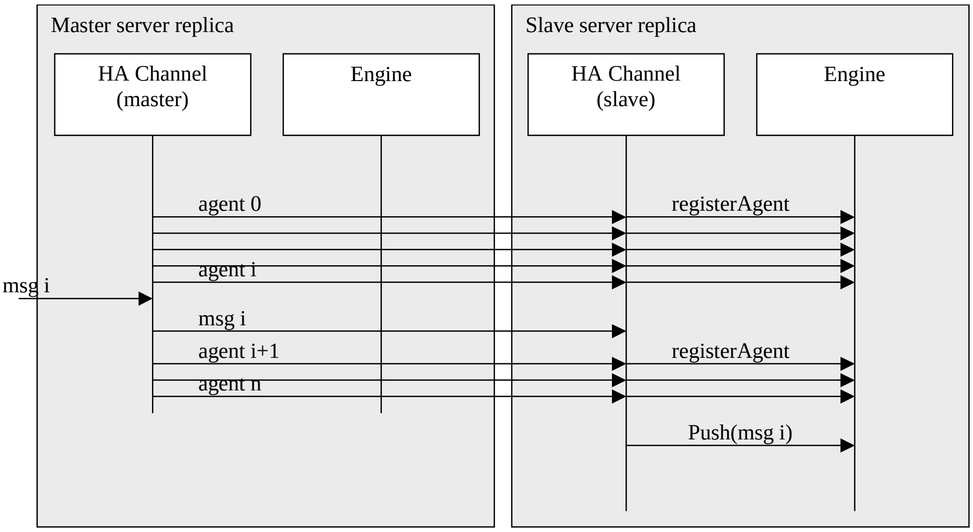
- identifier of the HA server
- storage path name
- identifier of the replica
- the environment property *nbClusterExpected* specifies the number of replicas initially expected in the cluster.

The command line to start the replica 1 of the HA agent server 0 is:

```
> java –DnbClusterExpected=3 AgentServer 0 s0 1
```

4.7.3 Dynamically adding a replica

When a view change shows that a new member joined the group, the master must send its state (agents) to the new member. The master’s state may be bulky, so it is asynchronously transferred to the new slave. In parallel the master may multicast notifications. These notifications cannot be delivered to the newly registered agents because those agents may have references to still unknown agents. So they are kept undelivered until the new replica state is wholly updated.



Notice that the undelivered notifications list may grow very fast during the state transfer. So an optimization is to do the state transfer in two steps:

- transfer all the agents and forget the arriving messages but mark the agents that have been modified by these messages
- re-transfer the modified agents and transfer the arriving messages

4.8 Master election

The master replica is defined to be the first server in the group list. When a replica receives a view change that shows that it is the first server, it decides to become the new master and starts its networks and services.

Network partitions

The handling of network partitions is not yet specified.

4.9 Connection from an external server to a replica

When a server decides to connect itself to a replicated server, it loads the configuration descriptor of the replicated server (*cluster*) and tries to connect each replica. Only the master replica accepts the connection demands. So the first replica that accepts the connection is the master.

An optimization is that a slave replica replies to the external server with the reference (host, port) of the master server.

5 HA Joram

5.1 Remote client mode

5.1.1 Reliable connection

The connection between a remote client and a HA Joram server must be reliable, i.e. it must not lose any messages. Messages are asynchronously sent and received in order to optimize the throughput between the client and the server.

The reliability of the communication is already ensured by the underneath protocol: TCP. But the TCP connection can be closed for two reasons:

- network failure
- Joram replica server failure

So the Joram HA connection layer must re-implement part of the TCP reliability mechanism by acknowledging messages and ensuring re-emission in case of failure.

5.1.2 Connection opening

The Joram client opens a TCP connection with the active Joram server replica and sends its user name and its user password. The server creates a connection context in order to handle the reliability. This context contains a connection identifier, the input and output messages counters and the output message queue filled by the agent that represents the Joram user (called proxy agent). The connection identifier is returned to the client in order to enable him to reconnect if the TCP connection fails.

5.1.3 Connection closure

The connection can be closed:

- by a closure order from the client
- after an idleness period detected by the server

Idle connection detection

A connection is declared idle if no message has been received for a specified time (timeout). The client is responsible for sending “keep-alive” messages in order to ensure that the connection stays open even if there is no Joram message to send.

5.1.4 Connection replication

The connection context is updated by the proxy agent. These updates occur when:

- The connection is opened
- An input message or an acknowledge is received from the client
- An output message is pushed (through the output message queue) by the proxy agent to the client
- The connection is closed

Each update is performed inside a reaction of the proxy agent. As the agent platform is replicated, the connection contexts are seamlessly replicated.

5.1.5 Master startup

The HA Joram master is automatically started by the HA ScalAgent platform. Each replica declares a service called “TcpProxyService” responsible for activating the TCP entry point. This service is started when a replica becomes master. Hence as soon as the HA platform has elected a new master server, the Joram master is started.

5.1.6 Recovering a failed connection

The HA Joram client owns the list of the replicas provided by the HA Joram server. When the client detects that the connection failed it tries to reconnect first to the same replica. If the reconnection fails for a specified time it switches to the next replica from the list and retries to connect. The client loops through the replicas list until it manages to connect to one of them.

Once the TCP connection is established with the replica, the client reopens the Joram connection by sending its user name, user password and the connection identifier. Then it reemits the unacknowledged messages. The server discards the messages already received and also reemits its unacknowledged messages. The client discards the messages already received.

The connection may have been closed by an idleness timeout. In this case the client gets a error: the high availability is broken.

5.2 Collocated client mode

A collocated client is connected to a Joram server through a collocated connection. This connection has two behaviors depending on whether the Joram server is master or not.

Master behavior

If the Joram server is the master then the connection works like a standard collocated connection with replication of the JMS requests.

The connection context is replicated in order to enable each Joram server replica to process the JMS requests. But it is not recovered after a failure.

Slave behavior

If the Joram server is a slave then the collocated connection opening must block until the replica becomes the master. When the connection is opened it is not possible to recover an already existing connection because the collocated client is not replicated. So a new connection must be opened.

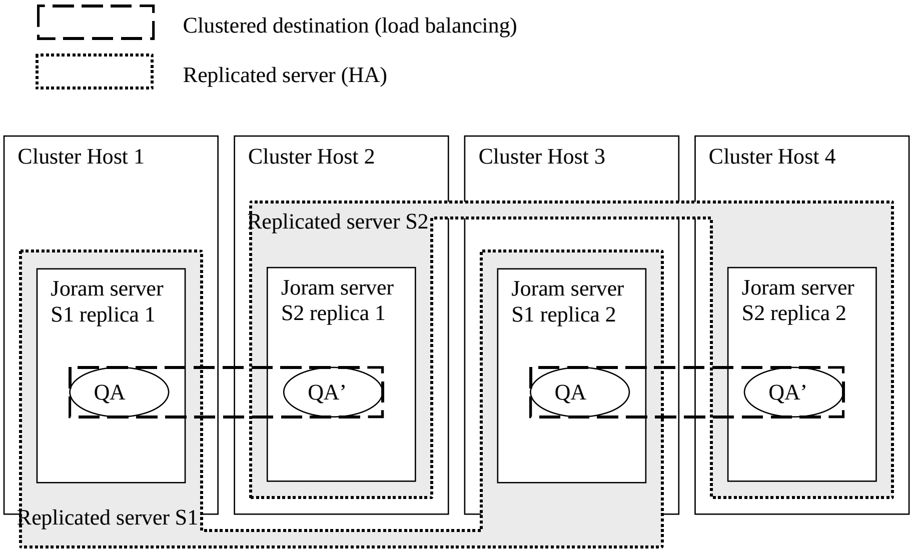
Contrary to the remote client mode that has almost no recovery stage (except the message reemission), this mode needs to clean the user proxy from the connection, subscription and transaction contexts that have been created by the replication of former collocated connections.

6 Mixing HA and load balancing

The HA mechanism can be easily mixed with the load balancing policy based on clustered destinations (already provided by Joram). The load is balanced between several HA servers. Each element of a clustered destination is deployed on a separate HA server.

Each of these clustered destinations (queues and topics) can be deployed on se HA servers.

The following figure shows an example of a destination called QA which is clustered for load balancing on two servers S1 and S2. Those two servers are replicated for HA on two hosts: S1 on (Host1, Host3) and S2 on (Host2, Host4).


Clustered destination (load balancing)
Replicated server (HA)
Cluster Host 1
Cluster Host 2
Cluster Host 3
Cluster Host 4
Replicated server S2
Joram server S1 replica 1
Joram server S2 replica 1
Joram server S1 replica 2
Joram server S2 replica 2
QA
QA'
QA
QA'
Replicated server S1