

Nirvana Clustering

Version	1.0
Issue date	14 January 2008
File name	clusteringWhitePaper.doc

Table of Contents

1. Introduction	3
1.1. About my-Channels	3
2. Clustering Concepts	4
3. Clustering – Server Concepts	5
3.1. Masters and Slaves	5
3.2. Quorum.....	5
3.3. Message Passing	6
3.4. Outages and Recovery.....	6
3.5. Creating clustered resources.....	7
4. Clustering – Client Components.....	8
4.1. Accessing clusters.....	8
4.2. Failover between cluster nodes.....	8
5. Conclusion.....	9

1. Introduction

This document describes the features and implementation of clustering found within Nirvana, my-Channels middleware messaging product.

At the highest level a cluster within the context of Nirvana represents a collection of realms (servers) that share a number of different types of information. In a cluster a messaging resource such as a topic or queue appears in every realm in said cluster. Every time the state of that resource changes the state change is updated on all realms in the cluster.

From a client perspective a cluster offers resilience and high availability. Nirvana clients automatically move from realm to realm in a cluster as required or when specific realms within the cluster terminate abnormally.

Clustering also offers a convenient way to replicate content between servers and ultimately offers a way to split large quantities of clients over different servers in different physical locations.

Finally, inline with most organisations business contingency and disaster recovery policies clustering supports remote distribution of data and continuous access to data.

1.1. About my-Channels

my-Channels provide technology that delivers and enables the next generation of connectivity, application extension and data interoperability within and beyond the enterprise.

my-Channels market-defined and driven solutions are specifically designed to meet and solve the growing challenges of integrating distributed and disparate enterprise infrastructures, extending the reach and use of applications, and ensuring seamless information exchange regardless of type, location or format of data.

2. Clustering Concepts

As described within the introduction of this document a Nirvana server process is called a realm. A realm is a container for a number of messaging resources, specifically:

1. Nirvana Channels
2. JMS Topics
3. Nirvana Queues
4. JMS Queues

These resources can be described as the common denominator between the publishers of data and the consumers or subscribers of data. Consequently these resources have a number of attributes that are also stored within the Nirvana realm, these are:

1. Access Control Lists
2. Type (Simple, Reliable, Persistent, Mixed, Transient)
3. Capacity (an optional parameter that determines the number of events a resource will hold)
4. TTL, Time To Live (an optional parameter that determines how long an event or message will remain accessible within a resource)

Within the context of a cluster a single instance of a channel, topic or queue (defined above) can exist on every node within the cluster. When this is the case all attributes associated with the resource are also propagated amongst every realm within the cluster. The resource in question can be written to or read from any realm within the cluster.

A Nirvana client accesses Nirvana realms and their resources through a custom URL called an RNAME. When accessing resources in a cluster clients use a comma separated array of RNAMEs. This comma separated array can be given to the client dynamically when the client connects to any member of a cluster. If a connection is terminated unexpectedly the Nirvana client automatically uses the next RNAME in its array to carry on.

Each event/message within Nirvana is uniquely identified by an event id regardless of whether it is stored on a channel, topic or queue. A clustered channel, topic or queue guarantees that every event published to it via any realm within the cluster will be propagated to every other realm in the cluster and will be identified with the same unique event id. This enables clients to seamlessly move from realm to realm after disconnection and ensure that they begin from the last event consumed based on this unique event id.

3. Clustering – Server Concepts

The basic premise for a nirvana cluster is that it provides a transparent entry point to a collection of realms that share the same resources and are in effect a mirror image of each other. A Nirvana cluster achieves this by the implementation of some basic concepts described below.

3.1. Masters and Slaves

A Cluster is a collection of realms as previously described. The relationship between these realms is that of a master and slave(s). Each cluster has 1 realm which is elected as master, and all other realms are deemed slaves. The master is the authoritative source of state for all resources within the cluster. For channels, topics and queues, each event published will be allocated the unique event id (described earlier) by the master, which is then propagated to each slave.

When the cluster is created, each member realm is provided with a flag that denotes whether the realm can be voted master within the cluster or not. This flag is accessible to system administrators.

3.2. Quorum

Quorum is the term used to describe the state of a fully formed cluster with a master that has been elected. In order to achieve quorum, certain conditions need to be met. Most importantly, >51% of the cluster nodes must be active/online in order for quorum to be achieved. For example, if we consider the situation where a 5 realm cluster exists, there must always be 3 realms active for the cluster to be online and operational.

Quorum, in conjunction with our deployment guidelines prevents ‘split brain’ or “multiple masters occurring”. My-Channels always recommend a cluster be created with an odd number of member realms, preferably in 3 separate locations. In a 5 realm cluster for example, realm1 and realm2 exist in location A, realm3 and realm4 exist in location B and realm5 exists in location C. All realms except realm5 are configured to be allowed to become master. Effectively realm5 is used purely to achieve quorum should location A or B become unavailable.

For the situation where we have a total cluster failure, i.e. all nodes are offline, the cluster will not be available, since the cluster is then deemed inaccessible.

When the master realm unexpectedly exits or goes offline for some reason, the remaining cluster nodes will elect a new master realm and continue to function. The process of the master election involves all remaining realms in the cluster. Each remaining realm submits a vote across the cluster that results in the new master once all votes are received and the number of votes is $\geq 51\%$ of the total cluster members.

3.3. Message Passing

Message passing between cluster realms enables state to be maintained across the member realms. The complexity of the message passing differs somewhat depending on the scenario.

For example, when using topics, where we publish to the master and subscribe from both master and slave nodes, the master will simply pass the event onto each slave for delivery with the correct event id, and each slave will maintain the same event id as set by the master.

When publishing to a topic on a slave node, the slave has to contact the master for the correct event id assignment before the event is then propagated to each slave.

When using queues, the message passing is much more complex, since each read is destructive, i.e. it is removed from the queue after it is delivered successfully. Consider the situation where we have 5 cluster realms, and each realm has a consumer connected to queue1. If we publish 5 events to the master realm's queue1 object, and the first event is consumed by consumer1 on the master (realm1), each slave realm must be notified of the consumption of the event from the queue and remove the event from its own local copy of queue1. The next event will be consumed by consumer2 on realm2. Realm2 will then notify the master of the event being consumed which will update its local queue1 and then propagate this to all other slave realms to update their own local stores.

The nirvana api, as well as the JMS specification defines transactional semantics for queue consumers which adds even more to the complexity of the message passing. For example, a consumer may effectively roll back any number of events it has consumed but not acknowledged. When an event is rolled back, it must then be re-added to the queue for delivery once again to the next available queue consumer (which may exist on any of the slave realms). Each event that is rolled back requires each slave realm to maintain a cache of the events delivered to transactional consumers in order for the event to be effectively restored should it be required. The state of this cache also must be maintained exactly the same across all cluster members. Once an event is acknowledged by the consumer, (or the session is committed) these events are no longer available to any consumer and no longer exist in any of the cluster member's queues.

From what we describe above, it is clear that certain scenarios require more message passing between cluster members than others. However, there are a huge number of benefits associated with using clusters in terms of scalability.

3.4. Outages and Recovery

Should any cluster member realm exit unexpectedly or become disconnected from the remaining cluster realms, when it is restarted or reconnected, it needs to fully recover the current state once it attempts to rejoin the cluster.

In order to achieve this, each clustered resource must recover the state from the current master. This involves complex evaluation of its own local stores against the master realm's stores to ensure that they contain the correct events and any events that no longer exist in any queues and topics are removed. With queues for example, events are physically stored in sequence but maybe consumed non-sequentially, (for example using message selectors that would consume and remove say every 5th event). This would result in a fairly sparse and fragmented store and adds to the complexity of recovering the correct state. However, Nirvana clusters successfully achieve this upon restart of any cluster member.

When a cluster member rejoins the cluster they automatically move into the recovery state until all local stores are recovered and its state is validated against the master.

3.5. Creating clustered resources

Channels, topics and queues can be created 'cluster wide' which ensures state is maintained across the cluster as described earlier. Once a channel, topic or queue is created as cluster wide, any operations upon that resource are also propagated to all cluster members.

There are a number of ways to create cluster resources once you have created your cluster. The Enterprise Manager application is a tool that provides access to all resources on any realm within a cluster or on stand alone realms. This graphical tool is written using the Nirvana Client and nAdmin APIs and allows resources to be created, managed and monitored from one central point.

Because the Enterprise Manager tool is written using our own APIs, all operations you can perform using the tool are also available programmatically using the APIs, allowing you to write customized applications for specific areas of interest within a realm or a cluster.

Once a realm is part of a cluster, you can centrally manage its resources and configuration. For example, realm access control lists (ACLs) can be updated on any member realm and the change will be propagated to all other member realms. Clustered channels and queue ACLs can also be changed on any cluster member and the change is then propagated to the other cluster members. Configuration changes, (in the enterprise manager tool) can also be made on one realm and propagated to all other realms in the cluster.

This provides a powerful way of administering your entire Nirvana environment and its resources.

4. Clustering – Client Components

The Nirvana API is a public API providing access to a realm server and its resources for the basis of performing pub/sub, message queue or Peer 2 Peer operations. Nirvana also provides a JMS API implementation which wraps a subset of the Nirvana Client API functionality within the JMS API specification.

The entry point for a Nirvana Client application is an nSession object. The nSession maps directly to a JMS Connection object. The attributes for the construction of a Nirvana nSession consist of one or more RNAME strings consisting of :

protocoll//host:port (nsp://www.my-channels.com:9000 for example)

4.1. Accessing clusters

To the client applications, whether Nirvana API or JMS, access to a realm is achieved through the use of an RNAME or array of RNAME(s), as described above. A Nirvana cluster is accessible by using the RNAME array associated with each of the cluster members.

For example, if we have a cluster consisting of 3 realms, your nirvana nSession can be constructed using the 3 RNAME URLs associated with each of the realms in the cluster.

Once connected to a realm in a cluster, you can then obtain references to nChannel and nQueue objects (or in JMS, create a Session followed by a topic or queue).

Once these object references are created, you can perform the desired operations on the resources.

4.2. Failover between cluster nodes

Using an array of RNAME urls allows client applications to seamlessly failover to different cluster nodes should they get disconnected for some reason.

For example, consider the following RNAME list that is used as the RNAMEs value for the nSession construction:

```
String[] rnames = {"nsp://host1:9000","nsp://host2:9000","nsp://host3:9000"};
```

When we first connect, the rnames[0] will be used by the session, and the client application will connect to this realm. However, should we disconnect from this realm, for example if host1 crashes, the client API will automatically reconnect the client application to the cluster member found at rnames[1].

5. Conclusion

Using Nirvana clusters provides many different benefits to those systems requiring a high level of scalability and continuous operation.

Client load can be spread among the cluster nodes with minimal impact on performance, thus providing an unprecedented level of scalability, with the added benefit of state being maintained between each node.