TIBCO BusinessEvents Loadbalancer

[Functional Spec]

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Date | Comments |
| 0.3 | Ashwin Jayaprakash | Aug 4, 2010 | Internal component |
| 0.2 | Ashwin Jayaprakash | July 30, 2010 | Grammar corrections |
| 0.1 | Ashwin Jayaprakash | July 30, 2010 | First draft |
|  |  |  |  |

# What is it?

This is a new module in BE.

# Briefly, why?

Most POCs are done in memory mode or Cache+memory mode. This is the simplest and cleanest way to demo the product.

However, when customers start implementing their real world project after the POC using Cache-only mode, they notice that the performance of the application drops significantly. The drop in performance is due to excessive cache synchronization and object transfers across the network. Scalability is achieved at the expense of latency. Most often throughput also suffers.

# In what version of BE will this be available?

Production ready in 4.1 \**Add disclaimer*\*

Working, alpha version is available in 4.0 code base.

# In detail, why?

When there is a cluster of more than 1 Inference Engine receiving events from EMS/RV, the sequence of events related to a transaction will be sent in round-robin fashion to all the Inference Engines. There is no session stickyness.

This means that the same concepts and events are downloaded repeatedly by each Inference Agent to its L1 cache instead of having to do it only once. This requires expensive version checks over the network to insure that the L1 cache always has the latest information.

Since related events can go to multiple Inference engines in parallel, a global Coherence lock has to be acquired. This also results in decreased throughput across the cluster because several threads are blocked across machines.

# What does the Loadbalancer do?

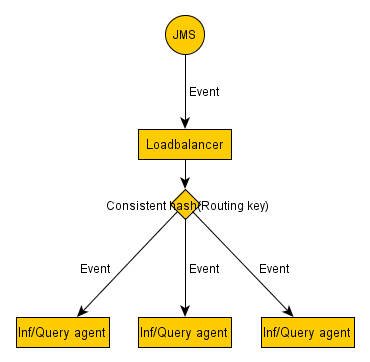
The Loadbalancer is meant to direct all related events from across many JMS queues to the same Inference Agent. This insures that the L1 cache is warm and does not have to be checked for version consistency. In a stable cluster topology/state, only global locking can be replaced with local locking.

In short, latency and throughput should increase drastically \**To be measured*\*.

# How does it work?

Any Inference or Query Agent can be turned into a Loadbalancer node. The Loadbalancer node sits in between the JMS server and the actual Inference/Query nodes and redirects events to a designated Inference/Query node based on the value of a routing key.

The routing requires no code change. It is merely a deployment procedure.



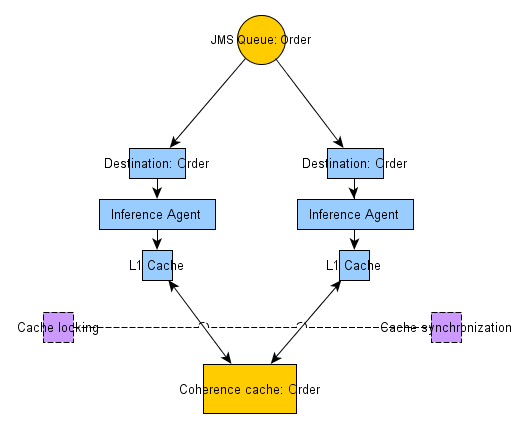
# How does it work, in detail?

The JMS destination of an Agent is set up to receive messages from the Loadbalancer instead of receiving messages directly from the JMS Server.

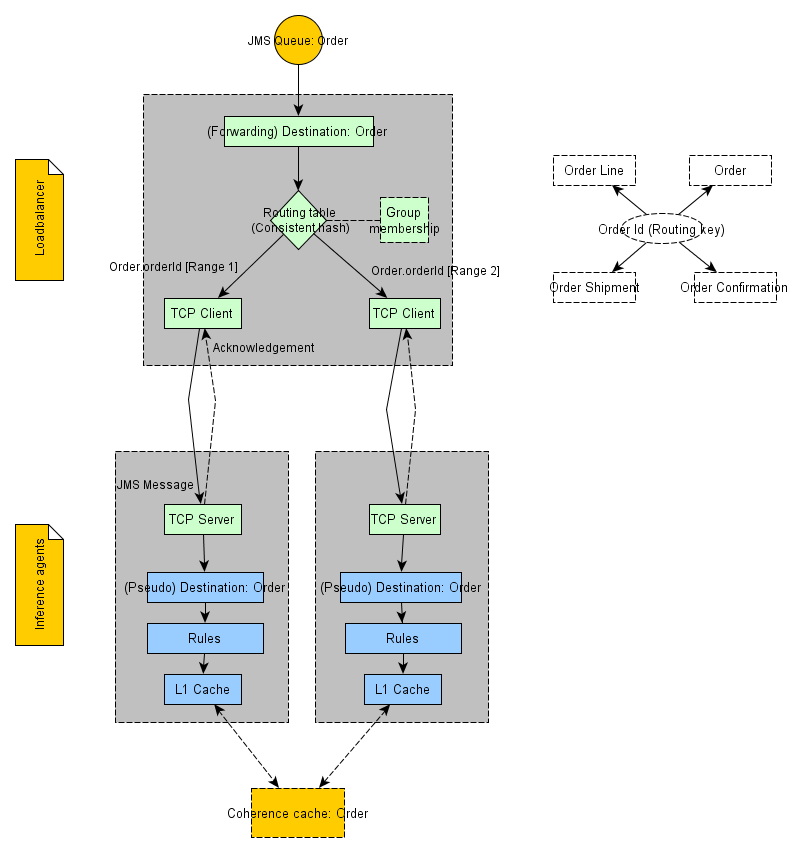
The Loadbalancer is setup to receive messages from the JMS server over the same destination that was meant for the Agents. When the Agents startup, the Loadbalancer starts relaying messages to the Agents based on a routing key.

The routing key can be any field in the Event that helps identify related events. Example: LoginEvent.userId or Order.orderId and OrderLine.orderId.

Before Loadbalancer



After Loadbalancer



# Architecture notes

Error handling and recovery

A single Loadbalancer would be a single point of failure. Therefore multiple instances of the Loadbalancer can be started.

The Loadbalancer has certain self-repair qualities that allow it to work correctly even with slow, unresponsive, error prone members:

* Connections are retried a few times before switching to alternate routes
* Members which are suspected of not functioning are checked occasionally allowing time for them to recover
* Group membership is not assumed to work perfectly. All changes are assumed to be sloppy
* When the cluster reaches a stable state, performance reaches its highest/optimal level
* Special headers are set in the events as hints to the client Agents. These hints can be used by the user code to perform Cache locking and synchronization if the cluster state/membership change is in progress. This allows fine grained control over global activities without incurring its cost during stable states
  + Example: LIKELY\_DUP\_DELIVERY, CONTENT\_ID, VERSION\_ID
* Acknowledgements, negative acknowledgements and timeouts are sent based on the situation to avoid message loss

All these features make the system more resilient in the face of errors.

Transport

TCP transport is the current mechanism to send messages and acknowledgements.

The transport is pluggable and is interface driven.

Group membership

A simple file based membership is currently implemented where all client agents write their contact details to a common directory. The Loadbalancers will poll, read these files and set up their hash tables and communicate with the client agents accordingly.

File based membership is also invaluable for testing since the errors are easy to simulate.

The group membership is pluggable and is interface driven.

Limitations (Current)

* Only TIBCO EMS Queues are supported
* EMS messages are expected to be Serializable
* Only TextMessageSerializer is known to work. Payloads and other formats have not been tested
* Messages and acknowledgements use Java serialization which might affect performance

Usage

Setup is property driven. Any BE project can be enabled to use the Loadbalancer with the correct system properties.

**Loadbalancer:**

be.agent.inference-class./Channels/JMS/NewDestination\_0.interceptor.classname =

com.tibco.cep.loadbalancer.impl.server.integ.ForwardingJmsDestination

be.agent.inference-class.loadbalancer./Channels/JMS/NewDestination\_0.routing.key = name

be.agent.inference-class.loadbalancer.membership.changeprovider.classname =

com.tibco.cep.loadbalancer.impl.server.membership.file.FileBasedMembershipChangeProvider

be.agent.inference-class.loadbalancer.membership.changeprovider.file.polldir = c:/temp/file\_poller

**Client:**

be.agent.inference-class./Channels/JMS/NewDestination\_0.interceptor.classname =

com.tibco.cep.loadbalancer.impl.client.integ.PseudoJmsDestination

be.agent.inference-class.loadbalancer.membership.publisher.classname =

com.tibco.cep.loadbalancer.impl.client.membership.file.FileBasedMembershipPublisher

be.agent.inference-class.loadbalancer.membership.publisher.file.publishdir = c:/temp/file\_poller

be.agent.inference-class.loadbalancer.member./Channels/JMS/NewDestination\_0.sink.transport = tcp

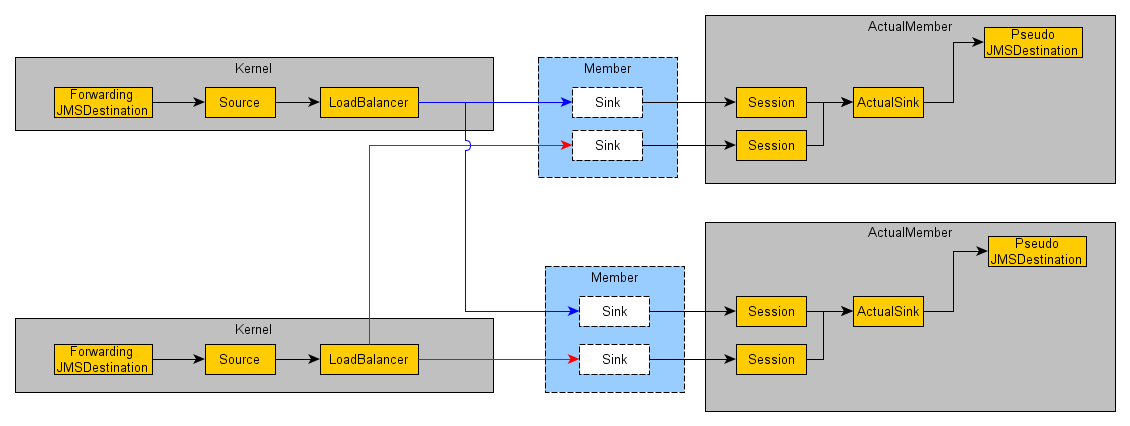
be.agent.inference-class.loadbalancer.member./Channels/JMS/NewDestination\_0.sink.tcp.hostname = localhost

be.agent.inference-class.loadbalancer.member./Channels/JMS/NewDestination\_0.sink.tcp.port = 33090

be.agent.inference-class.loadbalancer.member./Channels/JMS/NewDestination\_0.sink.tcp.so\_timeout = 3500

# Design notes

Components



**~~\*~~**