An Oracle White Paper
May 2011

Distributed Caching in the Oracle Enterprise Gateway

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Caching Overview

The Gateway provides general purpose caching facilities. The Gateway's caching facilities are frequently used as an optimization in a cluster of Gateways. It can be used to avoid re-doing computations or complex database access when the results remain valid for an extended period of time. In this case, subsequent requests for the same information can be served with the cached version rather than repeat the processing with the associated overheads.
Another use of distributed caching is in the transmission and sharing of data between multiple Gateways, for example the caching can be used in the throttling filter to limit access to a web service, in this scenario the usage count is shared between the Gateways in the cluster.
Cached requests/responses in the Gateway works on the principle that a unique key is set from a chosen message attribute from the message that needs to be cached. This key will be looked up in the chosen cache. If the key is not present in the cache, it will be stored in the chosen cache and the subsequent messages that have the same key will then be flagged as cached and follows a specific policy path.
Distributed caches are employed where several Gateways are deployed throughout a network in a clustered configuration. In this scenario, each Gateway has its own local copy of the cache but registers a cache event listener that replicates messages to the other caches so that add, put, remove, expiry, and delete events on a single cache are duplicated across all other caches.

It is important to note that caching might not be appropriate for all type of deployments. Careful considerations needs to be made in regards to expected volume and type of requests (the request/responses mainly static in nature or ever changing). Caching too much might also pose performance issues rather than solving it as this might overwhelm the available resources.

The Gateway uses Ehcache technology. For more information visit http://ehcache.org/.
Setup used for this Guide:

1. Two OEG Gateway's configured to use the same distributed cache
2. OEG Service Explorer as test client

Setting up the Distributed Caching and Caching Policy

Configuring a Distributed Cache

In a distributed cache, there is no master cache controlling all caches in the group. Instead each cache is a peer in the group and needs to know where all the other peers in the group are located. **Peer Discovery** and **Peer Listeners** are 2 essential parts of any distributed cache system. They can be configured by right-clicking on the **Caches** top-level node in the tree view of the Policy Studio and selecting the **Edit Cache Settings** menu option. Configure the following fields on the **Cache Settings** dialog:

Caching Settings:

1. In Policy Studio select the **External Connections** panel then right click on the **Caches** object and select the **Edit Cache Settings**
2. The following fields can be configured:
3. **Peer Provider Class:** By default, the built-in peer discovery class factory is used: net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory
4. **Properties Separator:** Specify the token that will be used as the separator for the list of properties in the next field.
5. **Properties:** The properties listed here dictate whether or not the peer discovery mechanism is automatic or manual.

   If the automatic mechanism is used, each peer uses TCP multicast to establish and maintain a multicast group. This is the default option since it requires minimal configuration and peers can be automatically added and removed from the group. Each peer pings the group every second. If a peer has not pinged any of the other peers after 5 seconds it is dropped from the group, while a new peer is admitted to the group if it starts pinging the other peers. To use automatic peer discovery ensure that the **peerDiscovery** setting is set to **automatic**. You can specify the multicast address and port using the **multicastGroupAddress** and **multicastGroupPort** settings. The time to live for multicast datagrams can be specified using the **timeToLive** setting.

   Alternatively, it is possible to configure a manual peer discovery mechanism, whereby each peer definitively lists the peers that it wants to communicate with. This should only really be used in networks where there are problems propagating multicast datagrams. To use a manual peer discovery mechanism, make sure the **peerDiscovery** setting is set to **manual.** The list of RMI URLs of the other peers in the group must also be specified, for example:

peerDiscovery=manual,rmiUrls=//server2:40001/sampleCache|//server3:40001/sampleCache|//server2:40001/sampleCache1|//server3:40001/sampleCache1

NOTE: Only the peers should be listed above and not the local host.

6. **Peer Listener Class:** The peer listener class specified here is responsible for listening for messages from peers in the group.
7. **Properties Separator:** Specify the token that will be used as the separator for the list of properties in the next field.
8. **Properties:** The properties entered here configure the way the listener behaves. Valid properties are as follows:

   • hostname (optional)

   This is the hostname of the machine on which the listener is listening. Note that, by default, this is set to "localhost", which maps to the local loopback address of 127.0.0.1, which is not addressable from another machine on the network. If you intend this cache to be used over the network, you should change this address to the IP address of the network interface on which the listener is listening.

   • port (mandatory)

   Specify the port on which the listener is listening, which by default is 40001. This setting is mandatory.

   • socketTimeoutMillis (optional)

   Enter the number of seconds that client sockets will wait when sending messages to this listener until they give up. The default setting is 2000ms.

9. **Notify replicators on removal of items during refresh:**

A server refresh will automatically purge all items from the cache. If this checkbox is enabled, the contents of each peer in the group will also be purged. This avoids a situation where a single peer is refreshed (and, therefore, has its contents purged) but the other peers in the group are not purged.

If this option is not checked, the refreshed peer will attempt to bootstrap itself to the other peers in the group resulting in the cache items becoming replicated in the refreshed cache.

1. Click on **Add** located in the lower right hand corner of Policy Studio
2. Select **Add Distributed Cache**
3. Configure the following fields:
4. **Cache Name:** When configuring a distributed cache across multiple Gateways the Cache Name has to be the same. For this guide **ResponseCache** is used.
5. **Maximum Element in Memory:** Default setting is 1000. Denotes the maximum number of objects that can be in memory at any one time.
6. **Maximum Elements on Disk:** Default is 1000. This field sets the maximum number of objects that can be stored in the disk store at any one time. A value of zero indicates an unlimited number of objects.
7. **Eternal:** Default is checked. If this option is checked, objects stored in the caches will never expire and timeouts will have no effect.
8. **Overflow to Disk:** Default is checked. Check this option if you want the cache to overflow to disk when the number of objects in memory has reached the amount set in the Maximum Elements in Memory field above.
9. The rest of the configuration options are optional:
10. **Time to Idle:** Default value is 0. This field determines the maximum amount of time (in seconds) between accesses that an object can remain idle before it expires. A value of zero indicates that objects can idle for infinity, which is the default value. Note that if the **Eternal** field above is checked, this setting will be ignored.
11. **Time to Live:** Sets the maximum time between when an object is created and when it expires. The default value is zero, which means that the object can live for infinity. Note that if the **Eternal** field above is checked, this setting will be ignored.
12. **Persist to Disk:** Not selected by default. If this field is checked, the disk store will be persisted between JVM restarts.
13. **Disk Expiry Interval:** Default value is 120 seconds. This setting configures the number of seconds between runs of the disk expiry thread.
14. **Disk Spool Buffer Size:** This setting indicates the size of memory (in MBs) to allocate the disk store for a spool buffer. Writes are made to this memory and then asynchronously written to disk. The default size is 30MB, however if you get **OutOfMemory** exceptions, you may consider lowering this value. On the other hand if you notice poor performance, you should increase the value.
15. **Eviction Policy:** Select the eviction policy that the cache will use to evict objects from the cache. The default policy is **Least Recently Used,** however it is also possible to use **First in First Out** and **Less Frequently Used.**
16. Under the **Event Listener** section the following options can be configured:
17. **Class Name:** The name of the listener factory class that enables this cache to register listeners for cache events, such as put, remove, delete, and expire.

18. **Properties Separator:** Specify the character to use to separate the list of properties in the field below.
19. **Properties:** This field can be used to list the properties to pass to the RMICacheReplicatorFactory. The following properties are available:

    • replicatePuts=true | false
    Determines whether new elements placed in a cache are replicated to other caches. Default is true.
    • replicateUpdates=true | false
    Determines whether new elements that override (i.e. update) existing elements with the same keyin a cache are replicated. Default is true.
    • replicateRemovals=true
    Determines whether element removals are replicated. Default is true.
    • replicateAsynchronously=true | false
    Determines whether replications are asynchronous (true) or synchronous (false). Default is true.
    • replicateUpdatesViaCopy=true | false
    Determines whether new elements are copied to other caches (true) or a remove message is sent (false). Default is true.
    • asynchronousReplicationIntervalMillis=[number of ms]
    The asynchronous replicator runs at a set interval of milliseconds. The default is 1000 and the minimum is 10. This property is only applicable if replicateAsynchronously=true.

20. Under the **Cache bootstrap** section the following options can be configured:
21. **Class Name:** Specifies a **BootstrapCacheLoader** factory that the cache can call on initialization to pre-populate itself. The **RMIBootstrapCacheLoader** bootstraps caches in clusters where **RMICacheReplicators** are used.
22. **Properties Separator:** The character entered here is used to separate the list of properties listed in the field below.
23. **Properties:** The properties listed here will be used to initialize the **RMIBootstrapCacheLoaderFactory**. The following properties are recognized:
    • bootstrapAsynchronously=true | false
    Determines whether the bootstrap happens in the background after the cache has started (true), or if bootstrapping must complete before the cache is made available (false). Default is true.
    • maximumChunkSizeBytes=[integer]
    Caches can potentially grow larger than the memory limits on the JVM. This property allows the bootstrapper to fetch elements in chunks. The default chunk size is 5000000 (i.e. 5MB).

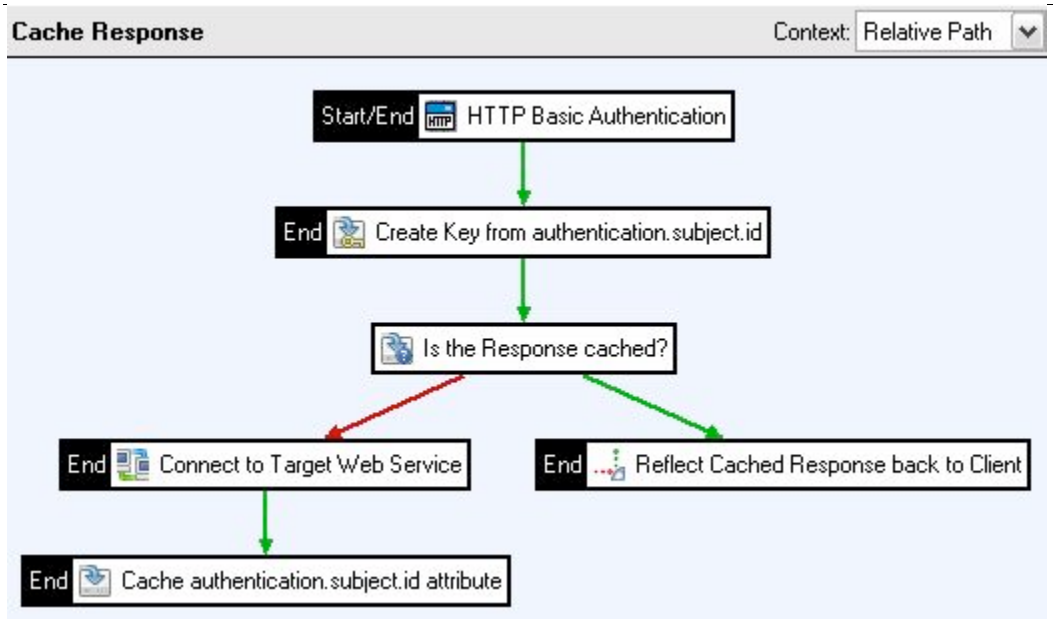
The Gateway can improve service response performance by caching responses. With caching enabled for a service, if the Gateway receives an identical request within a configurable time period, it serves the cached response. This avoids the message exchange with the backend server, improving performance and reducing load on backend servers.

For caching to work effectively, incoming requests must include a feature that can be used to determine the suitability of serving cached content. The feature may be, for example, a header or credential value. You must specify the header entries or parameters in an incoming request that are matched against the cache.

Caching however might not be appropriate for all types of services. Those that provide time-sensitive information is likely not appropriate for caching for example.

It will be demonstrated on how to set up a policy that will check whether a particular response message has been cached, and if not the message will be cached.
In this example, it will be demonstrated how to construct a circuit that will cache responses from the sample Calculator Web Service. The circuit is protected with basic http authentication, and the user credentials will be used to identify identical successive requests. In other words, if the Gateway receives 2 successive requests with an identical user credentials, it will return the corresponding response from the cache instead of routing the request to the Web Service.
This policy will use a distributed cache as setup in section 3.1 and is setup identical on two Gateway's on two different hosts.

The following diagram illustrates the complete circuit:



The logic of the circuit can be summarized as follows:

1. **Basic HTTP Authentication** will be enforced on this circuit against a user in the **local user store** in Policy Studio. The **Basic HTTP Authentication filter** will provide the attribute used that will be used to created the **unique key** that will be cached in the **distributed cache** namely: **authentication.subject.id**. This attribute will be used by the successive Cache filters in the policy.
2. The purpose of the **Create Key** filter is to configure what part of the request will be used to **identify unique requests**. In this example, the **authentication.subject.id** will be used as the unique key, which will then be used to lookup the appropriate response message from the cache.
3. The **Is Cached?** filter looks up the **authentication.subject.id** in the response cache to see if it contains the cached key. If it does, the response message that corresponds to this request is returned to the client.
4. If it doesn't, the request is routed to the Web Service, which processes the request and returns a response do the Gateway.
5. The Gateway then returns the response to the client and caches it in the response cache.
6. When the next identical request is received by the Gateway, the corresponding response will be located in the responses cache and returned immediately to the client, without being forwarded to the backend target service.

The following caching filters must be configured in order to create this circuit. A SimpleAxisServer Web Service is used as the target service for the purpose of this

document. For convenience, the routing filters will not be included since the configuration options here will depend on your target Web Service.
NOTE: This SimpleAxisServer service is shipped with the Gateway and can be started from the products /bin directory running axissimpleserver . This will run on port 7070.

Configure the policy:

1. Open **Policy Studio** and click on **Policies** in the navigation panel.
2. Right click on Policies root node and select **Add Policy** and
3. name the Policy **Cache Response.**

Configure a HTTP Basic Filter

1. Drag a **HTTP Basic** filter from the **Authentication** filter category.
2. Configure the **HTTP Basic** filter as:
3. Credential Format: **User Name.**
4. Repository Name: **Local User Store.**
5. Click **Finish** and right click on the **HTTP Basic** filter and select **Set as Start.**

Configure a Create a Key Filter

1. Drag a **Create Key** filter from the **Cache** filter category.
2. Configure the **Create Key** filter by choosing the Vordel message attribute to determine whether an incoming request is unique or not. Depending on the attribute chosen a valid predecessor filter that generates the attribute needs to be present. For this guide **authentication.subject.id** is used, which is generated by the **HTTP Basic** filter.
3. Click **Finish** and connect the **Create Key** and **HTTP Basic** filter via a success path.

Configure an Is Cached Filter

1. Drag an **Is Cached?** filter from the **Cache** filter category.
2. Configure the **Is Cached?** filter by selecting the cache that has been configured in section 3.1, **ResponseCache** in the list.
3. The **Attribute containing key** can be left default as: **message.key**
4. Change the **Overwrite attribute name if found** to: **authentication.subject.id**
5. Click **Finish** and connect the **Create Key** and **Is Cached?** filter via a success path.

Configure an Connect to URL Filter

1. Drag a **Connect to URL** filter from the **Routing** filter category
2. Configure the **Connect to URL** filter to point to the target web service.
3. For this guide the URL for this service is: http://192.168.0.49:7070/axis/services/urn:cominfo

4. Click on **Finish** and connect the **Connect to URL** filter to the **Is Cached?** filter via a failure path.

Configure a Cache Attributes filter

1. Drag a **Cache Attribute** from the **Cache** filter category.
2. Configure the **Cache attribute** filter by selecting the cache that has been configured in section 3.1, **ResponseCache** in the list.
3. The **Attribute containing key** can be left default as: **message.key**
4. Change the **Overwrite attribute name if found** to: **authentication.subject.id**
5. Click on **Finish** and connect the **Cache Attribute** filter to the **Connect to URL** filter via a success path.

Configure a Reflect Filter

1. Drag a **Reflect** filter from the **Utility** filter category and connect to the Is Cached? filter and connect via a success path. Settings can be kept default.
2. Select **Services** in the navigation window and right click on the **Default Services** service group under the Vordel Gateway process and **Add Relative Path**. Connect this path to **the Cache Response policy**. For this guide **/test** is used.

Refresh Gateway and Repeat Steps for Other Gateways

Refresh the Gateway by pressing the **F6** key.

Repeat the same steps on the second Gateway that is part of the distributed cache deployment.

Note: The policy above can be used as part of a larger policy and can be called via a Policy Shortcut filter before processing is continued. It can be part of a modular policy deployment. For more information on building policies please refer to the Vordel documentation that is included with the Gateway.

Testing the Distributed Cache

OEG Service Explorer tool will be used to send a request to the Gateway.

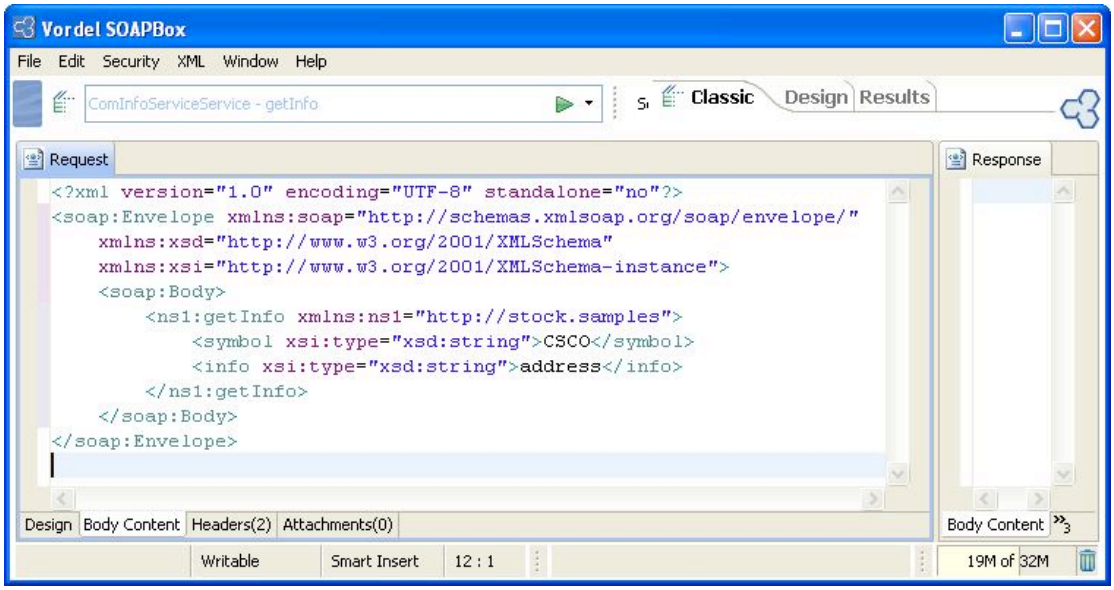
Testing the policy:

1. Open Service Explorer and click on the **Import WSDL** button.
2. Connect to the sample WSDL either via browsing to the WSDL file or connecting to the WSDL URL. For this guide http://192.168.0.49:7070/axis/services/urn:cominfo?wsdl is used.
3. The wizard will generate the required message based on the SOAP Operation chosen
4. Once the message is loaded it will be sent to the **Response Cache** policy created in section 3.2 of this guide.
5. Configure the URL that the message will be sent to: http://192.168.0.4936:8080/test. This is set via the **Request Settings…** window which is a drop down selection from the small black triangle ▼. Also set Security->HTTP Authentication to **HTTP Basic** with a valid username and password (e.g. admin/changeme).

In the **Request** tab on the left, paste in the following message::-

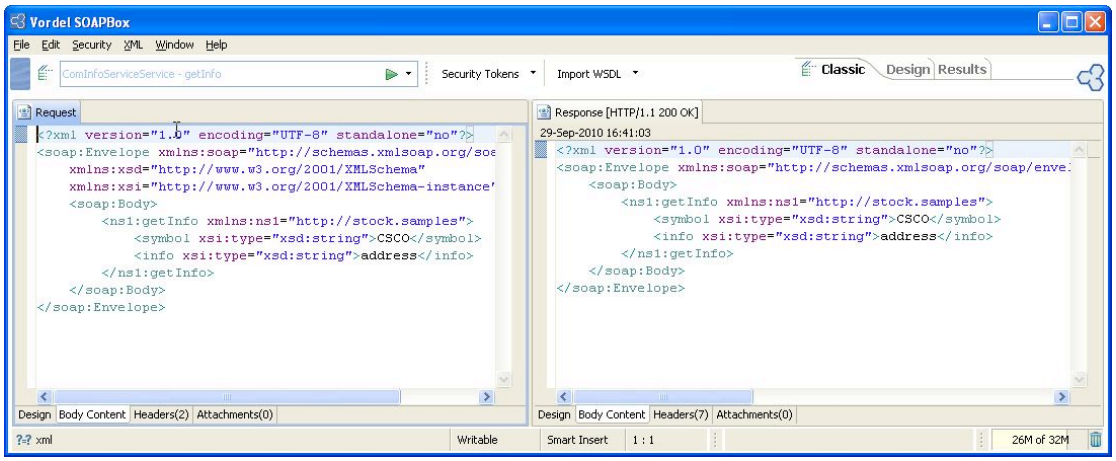
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
     <ns1:getInfo xmlns:ns1="http://stock.samples">
        <symbol xsi:type="xsd:string">CSCO</symbol>
        <info xsi:type="xsd:string">address</info>
     </ns1:getInfo>
   </soap:Body>
</soap:Envelope>
```

The message ready to be sent to the Gateway in OEG Service Explorer :



- Click on the **Send** ▶ button

The response received from the Gateway:



NOTE: Some data has been removed from the trace snippets for readability

Looking at the trace it can be seen the response has not been cached yet, and was cached when the service responded to the client.

Gateway 1 trace:

```
DEBUG   16:07:47:995 [5ff9b70] Incoming HTTP request:
method=POST, host=(unset), port=(unset), path=/test,
query=(unset), version=1.1
DEBUG   16:07:47:995 [5ff9b70] attaching current thread to
VM
DEBUG   16:07:47:995 [5ff9b70] done: env=0x8f6fcf4
DEBUG   16:07:47:999 [5ff9b70] handle type text/xml with
factory class com.vordel.mime.XMLBody$Factory
DEBUG   16:07:48:007 [5ff9b70] run circuit "Cache
Response"...
DEBUG   16:07:48:007 [5ff9b70] run filter [HTTP Basic
Authentication ] {
DEBUG   16:07:48:008 [5ff9b70]
VordelRepository.checkCredentials: username=admin
DEBUG   16:07:48:009 [5ff9b70] } = 1, in 2 milliseconds
DEBUG   16:07:48:009 [5ff9b70] run filter [Create Key from
authentication.subject.id] {
DEBUG   16:07:48:013 [5ff9b70] } = 1, in 4 milliseconds
DEBUG   16:07:48:013 [5ff9b70] run filter [Is Response
cached?] {
DEBUG   16:07:48:013 [5ff9b70] } = 0, in 0 milliseconds
ERROR   16:07:48:018 [5ff9b70] The message [Id-
0000012b632f559d-0000000001eeb372-1] logged Failure at
09.30.2010 16:07:48,015 with log description: Failure
while looking for key in the message attribute cache.
DEBUG   16:07:48:018 [5ff9b70] run filter [Connect to
Target Web Service] {
DEBUG   16:07:48:019 [5ff9b70]     get connection to host
192.168.0.49 port 7070 scheme http
DEBUG   16:07:48:020 [5ff9b70]     new endpoint
192.168.0.49:7070
DEBUG   16:07:48:020 [5ff9b70]     Resolved
192.168.0.49:7070 to:
DEBUG   16:07:48:020 [5ff9b70]         192.168.0.49:7070
DEBUG   16:07:48:020 [5ff9b70]     connected to
192.168.0.49:7070
DEBUG   16:07:48:027 [5ff9b70] } = 1, in 9 milliseconds
DEBUG   16:07:48:027 [5ff9b70] run filter [Cache
authentication.subject.id attribute] {
DEBUG   16:07:48:033 [5ff9b70] } = 1, in 6 milliseconds
```

```
DEBUG   16:07:48:033 [5ff9b70] ..."Cache Response"
complete.

DEBUG   15:11:53:153 [0554] Cache Response
```

Sending the same request the second time shows the following:

Gateway 1 trace:

```
DEBUG   16:08:09:394 [5cf6b70] Incoming HTTP request:
method=POST, host=(unset), port=(unset), path=/test,
query=(unset), version=1.1
DEBUG   16:08:09:394 [5cf6b70] done: env=0x837bfcf4
DEBUG   16:08:09:395 [5cf6b70] handle type text/xml with
factory class com.vordel.mime.XMLBody$Factory
DEBUG   16:08:09:397 [5cf6b70] run circuit "Cache
Response"...
DEBUG   16:08:09:397 [5cf6b70] run filter [HTTP Basic
Authentication ] {
DEBUG   16:08:09:397 [5cf6b70]
VordelRepository.checkCredentials: username=admin
DEBUG   16:08:09:397 [5cf6b70] } = 1, in 0 milliseconds
DEBUG   16:08:09:397 [5cf6b70] run filter [Create Key from
authentication.subject.id] {
DEBUG   16:08:09:397 [5cf6b70] } = 1, in 0 milliseconds
DEBUG   16:08:09:397 [5cf6b70] run filter [Is Response
cached?] {
DEBUG   16:08:09:397 [5cf6b70]     Key is in the cache
already
DEBUG   16:08:09:398 [5cf6b70] } = 1, in 1 milliseconds
DEBUG   16:08:09:398 [5cf6b70] run filter [Reflect Cache
Response back to Client] {
DEBUG   16:08:09:398 [5cf6b70] } = 1, in 0 milliseconds
DEBUG   16:08:09:398 [5cf6b70] ..."Cache Response"
complete.
```

The same request will now be sent to the second Gateway

Gateway 2 trace:

```
DEBUG   16:08:42:000 [1940] Incoming HTTP request:
method=POST, host=(unset), port=(unset), path=/test,
query=(unset), version=1.1
DEBUG   16:08:42:000 [1940] handle type text/xml with
factory class com.vordel.mime.XMLBody$Factory
DEBUG   16:08:42:000 [1940] run circuit "Cache
Response"...
DEBUG   16:08:42:000 [1940] run filter [HTTP Basic
Authentication ] {
DEBUG   16:08:42:000 [1940]
VordelRepository.checkCredentials: username=admin
DEBUG   16:08:42:000 [1940] } = 1, in 0 milliseconds
DEBUG   16:08:42:000 [1940] run filter [Create Key from
authentication.subject.id] {
DEBUG   16:08:42:000 [1940] } = 1, in 0 milliseconds
DEBUG   16:08:42:000 [1940] run filter [Is Response
cached?] {
DEBUG   16:08:42:000 [1940]     Key is in the cache
already
DEBUG   16:08:42:000 [1940] } = 1, in 0 milliseconds
DEBUG   16:08:42:000 [1940] run filter [Reflect Cache
Response back to Client] {
DEBUG   16:08:42:000 [1940] } = 1, in 0 milliseconds
DEBUG   16:08:42:000 [1940] ..."Cache Response" complete.
```

It can be seen that the request has already been cached when the request was sent to Gateway 1, and Gateway 2 sent the response that was cached by Gateway 1.

Appendix:

Using Log4j for additional Trace information

Additional Ehcache tracing can be enabled in the Gateway trace console/file by modifying the log4j.properties file that ships with the Gateway.
The Ehcache logger can be can be set to DEBUG mode, as shown below and this will output the communication between machines in using a distributed cache.
The log4j.properties file is located in the vordel_gateway_root/system lib directory. Add the following entries to the bottom of the existing entries.

```
# Add the Vordel Appender (note first line of file, how
Vordel include in rootLogger)
log4j.appender.Vordel=com.vordel.trace.VordelTraceAppender

log4j.logger.net.sf.ehcache=DEBUG
log4j.logger.net.sf.ehcache.bootstrap=DEBUG
log4j.logger.net.sf.ehcache.config=DEBUG
log4j.logger.net.sf.ehcache.constructs=DEBUG
log4j.logger.net.sf.ehcache.distribution=DEBUG
log4j.logger.net.sf.ehcache.event=DEBUG
log4j.logger.net.sf.ehcache.exceptionhandler=DEBUG
log4j.logger.net.sf.ehcache.extension=DEBUG
log4j.logger.net.sf.ehcache.hibernate=DEBUG
log4j.logger.net.sf.ehcache.jcache=DEBUG
log4j.logger.net.sf.ehcache.loader=DEBUG
log4j.logger.net.sf.ehcache.management=DEBUG
log4j.logger.net.sf.ehcache.store=DEBUG
log4j.logger.net.sf.ehcache.util=DEBUG
```

Refresh the Gateway.

The trace console and files will now contain verbose output from the caching operations:

```
2010-02-23 12:53:09,046 [Multicast Heartbeat Receiver
Thread]
```

```
DEBUG
net.sf.ehcache.distribution.MulticastKeepaliveHeartbeatRec
eiver - rmiUrls received //127.0.0.1:40001/GlobalTestCache
DEBUG   12:53:12:453 [06ac] rmiUrls received
//127.0.0.1:40001/GlobalTestCache
2010-02-23 12:53:12,453 [pool-1-thread-1]
DEBUG
net.sf.ehcache.distribution.MulticastRMICacheManagerPeerPr
ovider - Unable to lookup remote cache peer for
//127.0.0.1:40001/GlobalTestCache. Removing from peer
list. Cause was: GlobalTestCache
DEBUG   12:53:12:453 [02d8] Unable to lookup remote cache
peer for //127.0.0.1:40001/GlobalTestCache. Removing from
peer list. Cause was: GlobalTestCache2010-02-23
12:53:12,468 [pool-1-thread-1]
DEBUG
net.sf.ehcache.distribution.MulticastKeepaliveHeartbeatRec
eiver - Aborting processing of rmiUrls since failed to add
rmiUrl: //127.0.0.1:40001/GlobalTestCache
DEBUG   12:53:12:468 [02d8] Aborting processing of rmiUrls
since failed to add rmiUrl:
//127.0.0.1:40001/GlobalTestCache 2010-02-23 12:53:14,062
[Multicast Heartbeat Receiver Thread]
DEBUG
net.sf.ehcache.distribution.MulticastKeepaliveHeartbeatRec
eiver - rmiUrls received //127.0.0.1:40001/GlobalTestCache
```

Issue Resolution

Issue: Request and or response messages are not in cache when processed by a Gateway in a clustered install environment using distributed caching.

Resolution 1: Make sure that the Gateway's distributed cache name is set the same as the rest of the Gateway's in the clustered/distributed environment. See section 2.1.2 on how to set the Cache Name.

Resolution 2: The Gateway distributed cache listener is not contactable as the host IP is still on the default loop back address. See section 2.1.1 point 8 for changing the host IP address.

Issue: Any filter configured with a local/distributed cache aborts when a request is processed by the filter.

Resolution: Check the Gateway start-up trace and look for the following type of error:

```
ERROR   11:26:05:939 [77f7bbb0]     cannot reconfigure
module type CacheManager: unloading the entity:{name=Cache
Manager}: net.sf.ehcache.CacheException: No such device
net.sf.ehcache.CacheException: No such device
     at
net.sf.ehcache.distribution.MulticastRMICacheManagerPeerPr
ovider.init(MulticastRMICacheManagerPeerProvider.java:93)
     at
net.sf.ehcache.CacheManager.init(CacheManager.java:228)
     at
net.sf.ehcache.CacheManager.<init>(CacheManager.java:143)
     at
com.vordel.circuit.cache.CacheContainer.configure(CacheCon
tainer.java:83)
     at com.vordel.dwe.Service.refresh(Service.java:343)
     at
     ……..
```

This denotes that the cache manager did not load correctly, and the probable cause is that the discovery method is set to a setting that is not supported on the network that the Gateway clustered/distributed setup is installed in.
Resolution: Usually changing from multicast to manual peer discovery will solve this issue. Please refer to section 2.1.1 point.

Oracle Enterprise Gateway
May 2011
Author:

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Oracle is committed to developing practices and products that help protect the environment

SOFTWARE. HARDWARE. COMPLETE.