Infrastructure - Story #8525

timeout exceptions thrown from Hazelcast disable synchronization

2018-03-27 22:36 - Rob Nahf

Status:	In Progress	**Start date:**	2018-11-16
Priority:	High	**Due date:**	
Assignee:	Rob Nahf	**% Done:**	30%
Category:	d1_synchronization	**Estimated time:**	0.00 hour
Target version:	CCI-2.4.0		
Story Points:			

Description

Very occasionally, synchronization disables itself when RuntimeExceptions bubble up. The most common of these is when the Hazelcast client seemingly disconnects, or can't complete an operation, and a java.util.concurrent.TimeoutException is thrown.

These are usually due to network problems, as evidenced by timeout exceptions appearing in both the Metacat hazelcast-storage.log files as well as d1-processing logs.

Temporary problems like this should be recoverable, and so a retry or bypass for those timeouts should be implemented. It's not clear whether or not a new HazelcastClient should be instantiated, or whether the same client is still usable. (Is the client tightly bound to a session, or does it recover?) If a new client is needed, preliminary searching through the code indicates that refactoring the HazelcastClientFactory.getProcessingClient() method is only used in a few places, and the singleton behavior it uses can be sidestepped by removing the method and replacing it with a getLock() wrapper method (that seems to be the dominant use case for it). See the newer SyncQueueFacade in d1_synchronization for guidance on that. If the client is never exposed, it can be refreshed as needed.

```
root@cn-unm-1:/var/metacat/logs# grep FATAL hazelcast-storage.log.1
[FATAL] 2018-03-27 03:15:19,380 (BaseManager$2:run:1402) [64.106.40.6]:5701 [DataONE] Caught error
 while calling event listener; cause: [CONCURRENT_MAP_CONTAINS_KEY] Operation Timeout (with no res
ponse!): 0


[ERROR] 2018-03-27 03:15:19,781 [ProcessDaemonTask1]  (SyncObjectTaskManager:run:84) java.util.con
current.ExecutionException: java.lang.RuntimeException: java.util.concurrent
.TimeoutException: [CONCURRENT_MAP_REMOVE] Operation Timeout (with no response!): 0
java.util.concurrent.ExecutionException: java.lang.RuntimeException: java.util.concurrent.TimeoutE
xception: [CONCURRENT_MAP_REMOVE] Operation Timeout (with no response!): 0
    at java.util.concurrent.FutureTask.report(FutureTask.java:122)
    at java.util.concurrent.FutureTask.get(FutureTask.java:192)
    at org.dataone.cn.batch.synchronization.SyncObjectTaskManager.run(SyncObjectTaskManager.java:7
6)
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.RuntimeException: java.util.concurrent.TimeoutException: [CONCURRENT_MAP_REMO
VE] Operation Timeout (with no response!): 0
    at com.hazelcast.impl.ClientServiceException.readData(ClientServiceException.java:63)
    at com.hazelcast.nio.Serializer$DataSerializer.read(Serializer.java:104)
    at com.hazelcast.nio.Serializer$DataSerializer.read(Serializer.java:79)
    at com.hazelcast.nio.AbstractSerializer.toObject(AbstractSerializer.java:121)
    at com.hazelcast.nio.AbstractSerializer.toObject(AbstractSerializer.java:156)
    at com.hazelcast.client.ClientThreadContext.toObject(ClientThreadContext.java:72)
    at com.hazelcast.client.IOUtil.toObject(IOUtil.java:34)
    at com.hazelcast.client.ProxyHelper.getValue(ProxyHelper.java:186)
    at com.hazelcast.client.ProxyHelper.doOp(ProxyHelper.java:146)
    at com.hazelcast.client.ProxyHelper.doOp(ProxyHelper.java:140)
    at com.hazelcast.client.QueueClientProxy.innerPoll(QueueClientProxy.java:115)
    at com.hazelcast.client.QueueClientProxy.poll(QueueClientProxy.java:111)
    at org.dataone.cn.batch.synchronization.type.SyncQueueFacade.poll(SyncQueueFacade.java:231)
    at org.dataone.cn.batch.synchronization.tasks.SyncObjectTask.call(SyncObjectTask.java:131)
    at org.dataone.cn.batch.synchronization.tasks.SyncObjectTask.call(SyncObjectTask.java:73)
```

Subtasks:			
Task # 8747: Adjust package configuration to persist changes between updates			New
Related issues:			
Related to CN REST - Bug #7706: Hazelcast Runtime exception halts synchroniza...	Closed	2016-04-04	

History

#1 - 2018-03-27 22:46 - Rob Nahf

- Related to Bug #7706: Hazelcast Runtime exception halts synchronization added

#2 - 2018-03-28 17:10 - Rob Nahf

going to need to know what type of exception is the cause: It seems to be a generic RuntimeException. see the source code:
https://github.com/hazelcast/hazelcast/blob/maintenance-2.x/hazelcast/src/main/java/com/hazelcast/impl/ClientServiceException.java

#3 - 2018-11-13 00:09 - Dave Vieglais

- Priority changed from Normal to Urgent

We are seeing this issue with increased frequency, possibly related to the volume of content held in the map.

In the short term, this service needs to be configured to retry on timeout to prevent synchronization from shutting down on a single timeout.

Longer term solution involves deprecation of hazelcast for inter-CN synchronization of content. Typical error is:

```
[ERROR] 2018-11-12 21:17:34,239 [ProcessDaemonTask1]  (SyncObjectTaskManager:run:84) java.util.concurrent.Exec
utionException: java.lang.RuntimeException: [CONCURRENT_MAP_CONTAINS_KEY] Operation Timeout (with no response!
): 0
java.util.concurrent.ExecutionException: java.lang.RuntimeException: [CONCURRENT_MAP_CONTAINS_KEY] Operation T
imeout (with no response!): 0
    at java.util.concurrent.FutureTask.report(FutureTask.java:122)
    at java.util.concurrent.FutureTask.get(FutureTask.java:192)
    at org.dataone.cn.batch.synchronization.SyncObjectTaskManager.run(SyncObjectTaskManager.java:76)
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
    at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.RuntimeException: [CONCURRENT_MAP_CONTAINS_KEY] Operation Timeout (with no response!): 0
    at com.hazelcast.impl.ClientServiceException.readData(ClientServiceException.java:63)
    at com.hazelcast.nio.Serializer$DataSerializer.read(Serializer.java:104)
    at com.hazelcast.nio.Serializer$DataSerializer.read(Serializer.java:79)
    at com.hazelcast.nio.AbstractSerializer.toObject(AbstractSerializer.java:121)
    at com.hazelcast.nio.AbstractSerializer.toObject(AbstractSerializer.java:156)
    at com.hazelcast.client.ClientThreadContext.toObject(ClientThreadContext.java:72)
    at com.hazelcast.client.IOUtil.toObject(IOUtil.java:34)
    at com.hazelcast.client.ProxyHelper.getValue(ProxyHelper.java:186)
    at com.hazelcast.client.ProxyHelper.doOp(ProxyHelper.java:146)
    at com.hazelcast.client.ProxyHelper.doOp(ProxyHelper.java:140)
    at com.hazelcast.client.MapClientProxy.containsKey(MapClientProxy.java:219)
    at org.dataone.cn.batch.synchronization.type.AbstractListenableMapAdapter.containsKey(AbstractListenableMa
pAdapter.java:49)
    at org.dataone.cn.batch.synchronization.type.SyncQueueFacade.poll(SyncQueueFacade.java:220)
    at org.dataone.cn.batch.synchronization.tasks.SyncObjectTask.call(SyncObjectTask.java:131)
    at org.dataone.cn.batch.synchronization.tasks.SyncObjectTask.call(SyncObjectTask.java:73)
    ... 2 more
```

#4 - 2018-11-13 17:10 - Dave Vieglais

Just noticed that in /etc/dataone/storage/hazelcast.xml the number of backups is set to 3, which means HZ is trying to make 4 copies of everything (self + 3 copies) which of course doesn't work since there's only 3 CNs.

backup-count should be set to "2" for both hzSystemMetadata and hzObjectPath (actually all structures).

Additional suggestions:

Allow read from the backup copies. Backups are created synchronously, so a backup copy will be consistent with the owner's copy. Hence it should be safe to read a backup copy instead of always requesting the owner's copy which is the default.

```
<read-backup-data>true</read-backup-data>
```

The eviction policy is currently set to dump 25% of the entries when the maximum of 3,000,000 is reached. This seems completely arbitrary and is likely using a huge amount of memory. Instead, we could set the limit much lower and rely on efficient population of the map on demand from postgres. Suggest keeping 5000 entries in memory:

```
<max-size policy="cluster_wide_map_size">5000</max-size>
```

On startup, MapLoader.loadAllKeys is used to pre-populate the hazelcast map. However, entries are also loaded on demand when a key (i.e. PID) is requested. Is there any benefit to pre-populating the map when there is little chance that the populated entries will be relevant to the current operations? Suggest changing MapLoader.loadAllKeys to return NULL. Then entries will be loaded on demand. This would be a change in Metacat.

See: https://docs.hazelcast.org/docs/2.4.1/manual/single_html/index.html#Persistence

#5 - 2018-11-15 18:27 - Dave Vieglais

WAN replication is available in the version of hazelcast we are using (2.4.1) though becomes an enterprise "feature" on later versions. If we are planning to drop hazelcast for inter-CN sync, then enabling WAN replication may buy us some time for stability before needing to change technologies.

https://docs.hazelcast.org/docs/2.4.1/manual/single_html/index.html#WanReplication

#6 - 2018-11-15 18:34 - Dave Vieglais

wrt to MapLoader.loadAllKeys, it appears this has already been done:

https://github.com/NCEAS/metacat/blob/master/src/edu/ucsb/nceas/metacat/dataone/hazelcast/SystemMetadataMap.java#L114

#7 - 2018-11-15 19:08 - Rob Nahf

Reducing the number of backups seems like a no-brainer.

Regarding pre-population and how many to hold in memory, I think we need to consider the impact on lookup performance and also the hit distribution. Reducing from 750k in memory to 5k in memory I think really ups the odds that a db lookup + serialization will be needed.

If 95% of the time an HZ map lookup is going to take the same amount of time as a CN.getSystemMetadata call, it may not be worth using hazelcast at all.

Ideally we could pre-populate the map (upon startup) with items likely to be needed, based on archive status and modification date.

I also think because of how we index OREs, we need a larger cache, or else a large ORE will cycle the cache for clients. We didn't seem to have pre-loading or stability issues that long ago, so I would suggest keeping 100k-200k in memory. that's still at least 35% of what we are currently using, and would scale going from 3 CNs to 2 CNs.

Given the current analysis of xerces processing time for the view service, maybe there's a similar performance drag affecting systemMetadata serialization as well. JAXB serialization may also be slower than JiBX, as well.

#8 - 2018-11-16 17:16 - Dave Vieglais

The following changes have been deployed on production:

1. <backup-count>2</backup-count>
2. <max-size policy="cluster_wide_map_size">50000</max-size>
3. <read-backup-data>true</read-backup-data>

After tomcat restart sync appears stable. (previously sync was failing with timeouts after an hour or so).

Performance of getSystemMetadata, getObject, and the view service appear unaffected.

wrt. Rob's comments - the reduction was from 3,000,000 entries held in memory. I dropped it down to 50k which may be on the low side, but the primary goal is to get synchronization stable first.

The system metadata map is used to mirror sysmeta between the CNs. An alternative is to use postgres replication but that would involve major refactoring.

We need to do more profiling to determine what the bottlenecks are for basic operations like getSystemMetadata. I suspect authorization as the main blocker, but really need to measure.

wrt. JAXB / JiBX and xerces - this may well be a bottleneck as well. moving in and out of xml is pretty slow.

#9 - 2018-11-16 17:25 - Dave Vieglais

- % Done changed from 0 to 30

- Priority changed from Urgent to High

- Status changed from New to In Progress

Adding tasks to ensure config changes are persisted across updates