I wonder if I can do request-reply with this:

- **1 hazelcast instance/member (central point)**
- **1 application with hazelcast-client sending request through a queue**
- **1 application with hazelcast-client waiting for requests into the queue**

The 1st application also receives the response on another queue posted by the second application.

Is it a good way to proceed? Or do you think of a better solution?

Thanks!

java hazelcast

asked Jan 21 '13 at 17:53

unludo
2,953 ● 4 ● 30 ● 55

The last couple of days I also worked on a "soa like" solution using hazelcast queues to communicate between different processes on different machines.

My main goals were to have

1. "one to one-of-many" communication with garanteed reply of one-of-the-many's
2. "one to one" communication one way
3. "one to one" communication with answering in a certain time

To make a long story short, I dropped this approach today because of the follwoing reasons:

1. lots of complicated code with executor services, callables, runnables, InterruptedException's, shutdown-handling, hazelcast transactions, etc
2. dangling messages in case of the "one to one" communciation when the receiver has shorter lifetime than the sender
3. loosing messages if I kill certain cluster member(s) at the right time
4. all cluster members must be able to deserialize the message, because it could be stored anywhere. Therefore the messages can't be "specific" for certain clients and services.

I switched over to a much simpler approach:

1. all "services" register themselves in a MultiMap ("service registry") using the hazelcast cluster member UUID as key. Each entry contains some meta information like service identifier, load factor, starttime, host, pid, etc
2. clients pick a UUID of one of the entries in that MultiMap and use a DistributedTask (distributed executor service) for the choosen specific cluster member to invoke the service and optionally get a reply (in time)
3. only the service client and the service must have the specific DistributedTask implementation in their classpath, all other cluster members are not bothered
4. clients can easily figure out dead entries in the service registry themselves: if they can't see a cluster member with the specific UUID (hazelcastInstance.getCluster().getMembers()), the service died probably unexpected. Clients can then pick "alive" entries, entries which fewer load factor, do retries in case of idempotent services, etc

The programming gets very easy and powerful using the second approach (e.g. timeouts or cancellation of tasks), much less code to maintain.

Hope this helps!

answered Feb 14 '13 at 20:24

Peti
790 ● 9 ● 19

I think it's a good idea to make the bus manage the lifetime/ttl of a request. What I tried to avoid though is to have too many hazelcast nodes as it slows overall communication because of replication needed between the nodes. By the way are you client notified by the bus, or do you perform a king of polling? – unludo Feb 15 '13 at 16:05

In my first approach - the queue based approach - the clients used IQueue#take on a special-only-for-this-single-reply-IQueue instance. – Peti Feb 16 '13 at 19:32