Navneet Prabhakar
Instructor's Name
29 February 2020

Hazelcast as In Memory Data-grid and Lock Mechanism

What is hazelcast ?

Hazelcast is an in Memory distributed datagrid, that is used to store data in memory so we don't have to hit database all the time. Its very similar to that of gemfire/geode in a way but offers a bit more flexibility in terms of partitioning data, managing cluster and with the latest introduction of CP subsystem, provides synchronised computing across the distributed environment.

Lets clear few terms before diving into the getting started guide and how we used to solve our problems.

First, What is CP Subsystem ?

CP Subsystem in Hazelcast is a new implementation of Hazelcast's concurrency APIs on top of Raft consensus algorithm. These implementations are in coordination with the CAP (Consistency, Availability, Partition resistance) principle.

Hazelcast provides us a dynamic and elastic clustering capabilities. We can separately configure backups for our data structures, scale out/in our cluster, or replace failed nodes quite easily.

Not since this clarification is out of our way lets get started.

How to install hazelcast ?

Hazelcast is a java based project, which means starting a hazelcast can be as easy as running a jar from terminal. Oh! Wait, it indeed is all we need.(yay!). Just inter the following command at your terminal and your hazelcast cluster is ready.

java -jar -Dhazelcast.config=<your config>.xml hazelcast.jar

The XML file contains the configuration for hazelcast.

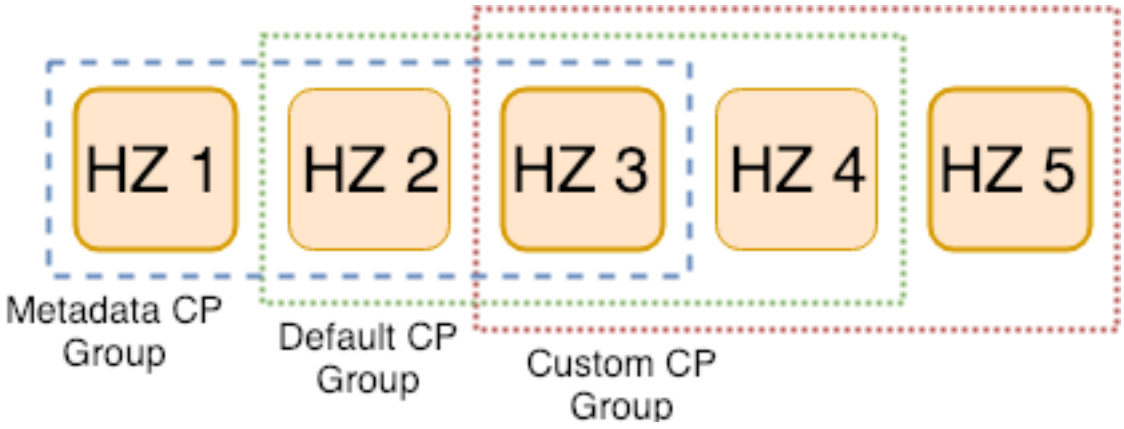
For the article perspective we will stick to the java based configuration from now onwards.

Hazelcast **master** is decided based on the longevity of a node in cluster i.e the more time is cluster spend in a node the higher is chances for it to be master. Rights to restarting a CP subsystem is strictly provided only to master of hazelcast cluster.

Lets start with a basic hazelcast cp subsystem

```
Config config = new Config();
config.getCPSubsystemConfig().setCPMemberCount(5);
config.getCPSubsystemConfig().setGroupSize(3);
HazelcastInstance hz1 = Hazelcast.newHazelcastInstance(config);
HazelcastInstance hz2 = Hazelcast.newHazelcastInstance(config);
HazelcastInstance hz3 = Hazelcast.newHazelcastInstance(config);
HazelcastInstance hz4 = Hazelcast.newHazelcastInstance(config);
HazelcastInstance hz5 = Hazelcast.newHazelcastInstance(config);

for (HazelcastInstance hz : Arrays.asList(hz1, hz2, hz3, hz4, hz5)) {

hz.getCPSubsystem().getCPSubsystemManagementService().awaitUntilDiscoveryC
ompleted(1, TimeUnit.MINUTES);
    System.out.println(hz.getCluster().getLocalMember() + " initialized
the CP subsystem with identity: "
            + hz.getCPSubsystem().getLocalCPMember());
}

CPSubsystemManagementService cpSubsystemManagementService =
hz1.getCPSubsystem().getCPSubsystemManagementService();
CPGroup metadataGroup =
cpSubsystemManagementService.getCPGroup(CPGroup.METADATA_CP_GROUP_NAME).ge
t();
assert metadataGroup.members().size() == 3;
System.out.println("Metadata CP group has the following CP members: " +
metadataGroup.members());

// Let's initiate the Default CP group

hz1.getCPSubsystem().getAtomicLong("counter1").incrementAndGet();

CPGroup defaultGroup =
cpSubsystemManagementService.getCPGroup(CPGroup.DEFAULT_GROUP_NAME).get();
assert defaultGroup.members().size() == 3;
System.out.println("Default CP group has the following CP members: " +
defaultGroup.members());

// Let's create another CP group

String customCPGroupName = "custom";
hz1.getCPSubsystem().getAtomicLong("counter2@" +
customCPGroupName).incrementAndGet();

CPGroup customGroup =
cpSubsystemManagementService.getCPGroup(customCPGroupName).get();
assert customGroup.members().size() == 3;
```

In the code above we configured CP subsystem to have 5 CP members, and CP Groups will be formed of 3 CP members. Members of CP group are randomly chosen among all CP members available in CP Subsystem each time a new CP group is created.In our example we have a ***Metadata*** and ***Default*** CP groups, and we create another CP Group named, "custom". These 3 CP groups run independently. Each CP group will have its own Raft leader to commit incoming operations to its majority. By this way we can distribute our workload among multiple CP groups and hazelcast members.



This is a descriptive diagram of CP subsystem initialised with 5 CP members and 3 CP groups.

So what happens in case of failure. Lets consider a case when there's a CP subsystem with 5 CP members (CP group size is also 5) and 2 of them crashes. Business as usual but unfortunately this scenario is risky. What happens if another CP member crashes, The majority of CP groups will not be satisfied anymore and the CP Subsystem will lose its availability. There is more than one way to deal this situation. First we can reduce the size of CP Subsystem by removing the failed nodes. When we remove those 2 members, the CP subsystem will have 3 CP members remaining and the majority will be reduced to 2. Now we can handle crash of 3rd CP member and majority will be reduced to 2 members. To recover the CP Subsystem back to 5 CP members, we can promote the regular AP members to CP role, then our majority will be restored to 3.

Lucky we. We can make hazelcast do this automatically for us.

In practical scenarios when a hazelcast member is removed from member list, it isn't automatically removed from CP member list. Reason being, failure detection in asynchronous

distributed system is not perfect. What hazelcast actually detects is the responsiveness of a CP member. Maybe that a member is just slow under high load, or it is having a long GC pause. There maybe a network problem or the member actually crashed. We can set the auto removal of CP member from CP Subsystem in our configuration.

```
Config config = new Config();
CPSubsystemConfig cpSubsystemConfig = config.getCPSubsystemConfig();
cpSubsystemConfig.setCPMemberCount(5);
cpSubsystemConfig.setSessionHeartbeatIntervalSeconds(1);
cpSubsystemConfig.setSessionTimeToLiveSeconds(5);
cpSubsystemConfig.setMissingCPMemberAutoRemovalSeconds(10);
```

Even after doing all this if a CP Subsystem has to lose majority we can alway restart out CP Subsystem. The restart rights being reserved to master of hazelcast.

Data in hazelcast is stored in a very peculiar manner. Lets say we have 3 nodes and three servers In this case all the data stored by each node consists of a partial data that is stored by a server itself and a replica of another server. Like

Node 1(Server A) may have data for Server A, and server C. Node 2(Server B) may have data for Server A and Server C. And Node 3 (Server C) may have data for Server C and Server B. Once a member crashes all its data and replicas are redistributed in cluster.

We all are already familiar with distributed lock mechanism. But what really troubles us is What if Lock holder dies or plays dead. Lets discus the scenario and its possible solution in much detail.

Lets imagine a situation in which a system shuts down while holding a lock. Well hazelcast has a solution for it. In order to deal with client failure, hazelcast has a mechanism to track liveness of a hazelcast servers and client in a unified manner: **CP Session.** This mechanism is being used in two CP data structures that manages the resource ownership; namely **FencedLock**, and **ISemaphore**. Hazelcast creates a new CP session whenever a hazelcast server or client makes its very first lock or Semaphore acquire request. After that all requests of the caller are associated with this session. Hazelcast server or client maintains single CP Session for a given CP Group. Even if its interacting with dozens of Locks or semaphores.

Hazelcast keeps the CP session alive for sometime after we committed the last operation associated with it. If a client doesn't perform any operation on its own but otherwise is live, its

proxy implementation will automatically send periodic heartbeat operations. If client dies or has a very long hiccup, the session times out and hazelcast closes it and releases all the locks and semaphores permits, and cancels all pending requests.

```
Config config = new Config();
CPSubsystemConfig cpSubsystemConfig = config.getCPSubsystemConfig();
cpSubsystemConfig.setCPMemberCount(3);
cpSubsystemConfig.setSessionHeartbeatIntervalSeconds(1);
cpSubsystemConfig.setSessionTimeToLiveSeconds(10);
HazelcastInstance hz1 = Hazelcast.newHazelcastInstance(config);
HazelcastInstance hz2 = Hazelcast.newHazelcastInstance(config);
HazelcastInstance hz3 = Hazelcast.newHazelcastInstance(config);

// Hazelcast member One acquires the lock
hz1.getCPSubsystem().getLock("my-lock").lock();

// Member One crashes. After some time, the lock
// will be auto-released due to missing CP session heartbeats:
hz1.getLifecycleService().terminate();
FencedLock lock = hz2.getCPSubsystem().getLock("my-lock");
while (lock.isLocked()) {
    Thread.sleep(TimeUnit.SECONDS.toMillis(1));
    System.out.println("Waiting for auto-release of the lock...");
}

System.out.println("The lock was automatically released");
```

In code above code sample a hazelcast member acquires a lock and crashes. The rest of CP Group realises the configured timeout has passed without any operation from that member and closes its session. This causes the lock to be released.

To be precise, the CP SubSystem resets the CP session timeout in 3 circumstances:

- When session owner performs a FencedLock or ISemaphore operation
- When the session owner sends a periodic heartbeat
- When the current Raft leader of CP group crashes and a new leader is elected. This must be done in order to give session owner time to discover new Raft leader and start sending operation/heartbeats to it.

What if a Lock Owner Plays dead, But Isn't ?

Lets consider a scenario when a hazelcast client or server acquires a lock, then hits a long GC pause or becomes partitioned from the cluster. Since it will not be able to commit session heartbeats in the meantime, its CP Session will be eventually closed and another hazelcast client can acquire this lock. If first client wakes up again or reconnects to cluster, it mayn't immediately notice that it has lost ownership of lock. In this case, multiple clients thinks that they hold lock. If they attempt to operate on shared resource, they can break the system.

To prevent such situation we can choose to have a potentially infinite CP session time to live, but then we will have liveness issues.

Hazelcast offers a fencing mechanism to deal with this scenario. **Each FencedLock instance contains a fencing token which is incremented each time the lock instance moves from the available to locked state.** Before attempting any side-effectful actions, the lock holder must pass the fencing token to all external services it will use thereby fencing off previous lock holders. It must also include the token in every request. The service must persist the largest observed fencing token and reject any incoming requests whose fencing token is less than the current one.

Below is he way in which we can get the fencing token.

```
Config config = new Config();
CPSubsystemConfig cpSubsystemConfig = config.getCPSubsystemConfig();
cpSubsystemConfig.setCPMemberCount(3);
HazelcastInstance hz1 = Hazelcast.newHazelcastInstance(config);
HazelcastInstance hz2 = Hazelcast.newHazelcastInstance(config);
HazelcastInstance hz3 = Hazelcast.newHazelcastInstance(config);

// The lock switches from the available state to the held state.
FencedLock hz1Lock = hz1.getCPSubsystem().getLock("my-lock");
hz1Lock.lock();
long fence1 = hz1Lock.getFence();
hz1Lock.unlock();

// The lock switches from the available state to the held state.
FencedLock hz2Lock = hz2.getCPSubsystem().getLock("my-lock");
hz2Lock.lock();
long fence2 = hz2Lock.getFence();

assert fence2 > fence1;

// The lock is already held by the second instance.
```

```
// Making a reentrant lock acquire.
hz2Lock.lock();
long fence3 = hz2Lock.getFence();

assert fence3 == fence2;

hz2Lock.unlock();
hz2Lock.unlock();

// The lock switches from the available state to the held state.
hz2Lock.lock();
long fence4 = hz2Lock.getFence();
hz2Lock.unlock();
assert fence4 > fence3;
```

Also please refer to the example in repository to see how talking can be done with external services