Using Hazelcast SQL with Payara Micro

The Payara® Platform - Production-Ready,
Cloud Native and Aggressively Compatible.

Contents

Hazelcast IDMG

“An in-memory data grid (IMDG) is a set of networked/clustered computers that pool together their random access memory (RAM) to let applications share data with other applications running in the cluster.”

Hazelcast IMDG allows you to store several different data structures. You can use Lists, Maps, Queue and other structures to organize the information stored in the Data Grid. They all implement the Java equivalent interfaces and process the information in a fashion we are familiar with in Java.

However, looping over the contents of the Map values, for example, is not the most efficient way to do this. When you loop over the data, you need to fetch them from the different cluster members as not all data is available in the local member. Data is usually stored throughout several members of the cluster. So when you loop over all entries, data will be fetched from different members, resulting in network overhead and latency.

Instead of looping over all entries of the local Map, you should create a Distributed Query within Hazelcast. The cluster will execute this query on each cluster member and send the result back to the member that requested the results. This way, a lot less network overhead does occur, and the search will be much faster. Each member of the cluster is involved and not only the instance embedded in your application.

Hazelcast SQL

Java developers are more familiar with standard Java interfaces like List and Map than with writing such a Distributed Query. The result is that data is searched locally in an inefficient way. Yet, many developers do know SQL or the variant used in JPA. With Hazelcast SQL, the goal is to search data within the In-Memory Data Grid more efficiently and easily. So you can write a SQL query that will search an IMap to return entries that match your query.

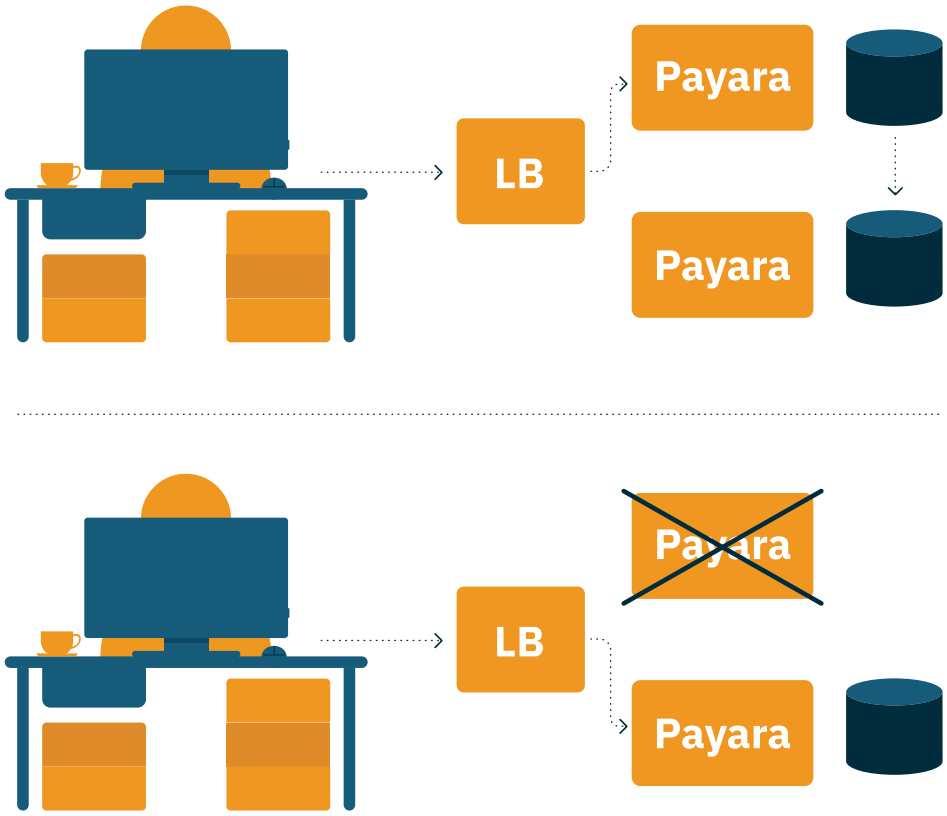
As an example, imagine an IMap with Person objects. Person has properties like name and age. You can write a query to return all people younger than 18. Under the hood, Hazelcast parses the query with Apache Calcite, queries each node using the Distributed Query, and aggregates the results: this means the search is performed as efficiently as possible.

Payara Domain Data Grid

The Payara Platform products can run your Java Enterprise application in many environments - including IoT and edge types like the RaspberryPi, over Virtual Machine, and Containerized and cloud environments. Payara Platform Enterprise is stable, supported software for enterprise designed for mission critical production systems and containerized Jakarta EE (Java EE) and MicroProfile applications. Choose Payara Server for reliable and secure deployments of Jakarta EE and MicroProfile applications in any environment, or Payara Micro for containerized microservices deployments with no installation, configuration, or code rewrites required.

The Payara Platform products also support clustering in an easy-to-set-up way within all environments, making use of the Hazelcast In-memory Data Grid for this purpose. The Data Grid makes it easy to share data between the different instances of your applications, provides support for the JCache specification, and you can use it to provide **session replication** for your environment. The HTTP session data is stored within The Hazelcast Data Grid and available across all Payara instances when activated.

If you store data in a session on a node and that node goes down, the user can be redirected to another node: the data is replicated and will be available.



However, it doesn't stop at session replication. You can share any data on any node and retrieve it on another node. You can think about it as a datastore with faster access.

A classic example of such usage is an e-commerce application. While storing a user's cart content in a database makes sense, accessing the latter on each request might be too slow. In that case, keep the cart in memory and let Hazelcast take care of the replication across the Payara nodes if the current node goes down.

When you start relying on Hazelcast to store data, the chances are that you'll need to query the cluster at one point or another. The good side about SQL databases is that they allow SQL queries - though you might point out it depends on the exact SQL version, that's a debate for another day.

On the other hand, most NoSQL datastores offer a query API, but it's, in general, a proprietary one that you need to learn for each store. Fortunately, the current trend of NoSQL vendors is to add a SQL API on top (in complement?) of their API. Hazelcast is no different.

It opens up a lot of new opportunities. Regarding the previous example of carts, you might keep the cart in the database because your analysts want to be able to get insight into their customers' habits: what's the average number of products in a cart, what's the average item price in a cart, etc.? But they don't want to learn a new API. By offering a SQL API, Hazelcast allows you to not compromise on performance while keeping your analysts happy.

Of course, there are plenty of other use-cases. If you use Hazelcast as a cache, you can now search your cache directly.

Using Hazelcast SQL

As Payara already provides the Hazelcast dependency, you only need to add the Hazelcast SQL JAR to start querying your cluster.

Start the Payara Micro node with the `--addjars` parameter and pass the location of the JAR, *e.g.*,

```
java -jar payara-micro-5.2021.3.jar --addjars hazelcast-sql-4.2.jar
```

At this point, you're ready to execute SQL queries on your cluster.

Hazelcast SQL queries require access to the Hazelcast Instance, and you need to know the name of the Hazelcast Map that contains the Java instances you want to search. The following example gives you an idea of how you can achieve it.

```
@ApplicationScoped
public class CountryService {

    @Inject
    private HazelcastInstance hzInstance;

    public List<String> getCountries(String continent) {
            List<String> result = new ArrayList<>();
            try (SqlResult queryResult = hzInstance
                  .getSql()
                  .execute(
                    "SELECT name FROM countries WHERE continentName = ?",
                    continent)
) {
      for (SqlRow row : queryResult) {
      String name = row.getObject(0);
}    }   result.add(name);
return result;

    }
}
```

In the above example, the CountryService class is defined as a CDI bean where we have injected the HazelcastInstance. The HazelcastInstance variable is the Hazelcast defined one so we have access to all functionality.

The query can be defined using the getSql() statement in a very similar fashion to regular SQL. In the example, we have written the following query:

```
SELECT name FROM countries WHERE continentName = ?
```

Hazelcast looks in the above case for a Map with the name *countries*. It returns the value of the property `name` from all entries in the map for those that match the `continent` value.

With Hazelcast SQL, you can turn Maps stored in the Domain Data Grid of Payara into a table and treat the properties of the POJO entries in that map as fields. As of now, SQL is in beta, and full ANSI-SQL is not fully supported. Expect additional capabilities in upcoming releases!

Conclusion

The Hazelcast In-Memory Data Grid is an efficient way of storing data in a distributed way within the memory of the different processes of the cluster. The distributed design, however, means that searching the data locally in your process is not efficient. All data needs to be 'moved' to your instance so that it can be accessed. Hazelcast allows distributed queries so that the search is performed where the data is, and only the results are transferred to your process.

With Hazelcast SQL, the Distributed Query capabilities are wrapped in another well-known concept by developers and easier to use. Since the Payara products already use Hazelcast IMDG, using the Hazelcast SQL capabilities is straightforward. You only need to add the additional JAR library and can start using it.

info@payara.fish

+44 207 754 0481

www.payara.fish