Benchmarking Graph Database in standalone and distributed for the Flight Dynamic System

BOULET-GILLY Edmond
Internship at Thales Services Labège during Summer 2018
Telecommunication and Network
France

Abstract

The performance of a cluster of Graph Database depends on the solution you choose software wise. We are going to see the impact of the solutions chosen by Neo4J and OrientDB when it comes to making the data availability and secure in distributed and standalone architecture. We will also see those Graph Databse to perform transactions and achieve stability when using the JAVA API.

1. Introduction

When designing cluster who will host terabyte of data one the key question is what software will manage the data. The software will dictate the availability, integrity and the security of the data base's content. Thales already choose OrientDB, years ago, to host the data gathered during satellite launch, each launch gathering terabyte of data. During summer 2018 they wanted to move their single instance database to a distributed one, to extend the availability of the data.

They had multiple options, Neo4J [1], the world leader Graph Database, OrientDB [2], the 4th, and JanusGraph [3], not well ranked but supported by Thales Group. They wanted the Database to be a Graph one, due to the structure of the data. But also an open sourced one.

2. The JAPI and Why not Tinkerpop

One of the aspect of the work was to test to latest release compatibility with Tinkerpop, the current API used to transfer data into the Graph Database. Sadly Neo4J did not support the Tinkerpop API, actually neo4J did not support anything than Cypher and the benchmarking was made in JAVA by executing Cypher request.

Regarding OrientDB, some test were made with the tinkerpop API but the results were underwhelming and after speaking with OrientDB dev team, they declared the API obsolete, stating that the Multi-Model API was a must-use.

In the end Janus Graph was the only who fully endorse Janus Graph.

3. Data specification

In order to have relevant test the data base used had the same structure as the one found in the Flight Dynamic System. The data has a similar structure as a folder-file one with multiple layer, link and type of objects. The majority of the weight being located in the "File" of 1MB.

The size of the graph can reach thousands on Nodes, which can be problematic in size considering that the goal is to generate graph from hundred of thousands of Nodes. Therefore in order to make the benchmarking less time and size intensive the weight of Node was considerably reduce.

4. Standalone instance

The first benchmarks were made in standalone mode in order to test the base functionality of each engine. During the benchmark the engine were running on a local machine with little to no interaction by the user.

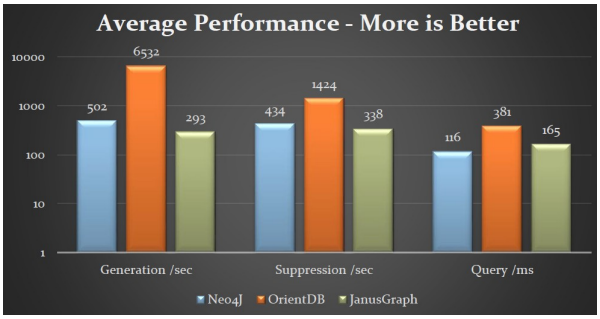

Figure1 : standalone performance

We can observe that OrientDB perform better than Janus Graph and Neo4J by a long shot in Generation as the scale it he previous figure is logarithmic.

5. Distributed Architecture Janus Graph

Sadly we could not perform more test Janus Graph as the Distributed architecture's development status did not hint satisfying performance.

6. Distributed Architecture OrientDB

The Architecture implemented in OrientDB (and the one advised in the documentation) is called High-availability. Each member of the cluster has a copy of the entire database and can answer and compute transaction.

In order to protect the database, OrientDB implements the principle of Core Majority. Which means that order that a transaction can permanently be validated, it has to reach more that half the total cluster. This guarantee that transaction doesn't collide with each other and that a transaction can be validated even if one member of the cluster leave the cluster.

OrientDB were benchmarked in Single-Thread, Multi-Thread, Batched and single transaction mode.

Sadly combining Multi-Thread and Batch mode had unexpected consequences on OrientDB as transaction were colliding with one another and would lock the database permanently.

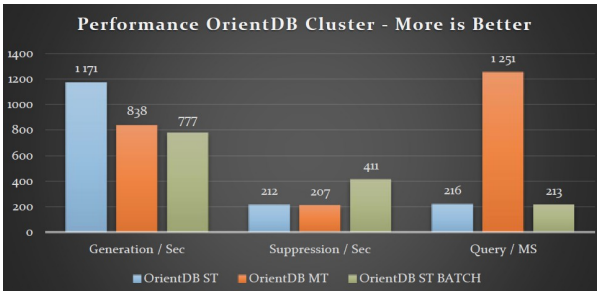

Figure 2 : OrientDB Cluster performance

7. Distributed Architecture Neo4J

Neo4J implements causal clustering : some nodes of the cluster are going to act as replica, copy the data and answer read operation, while other act as core, preform all kind of transaction.

With that said, core Node doesn't work independently, all the transaction are routed to the master Node who is in charge of validating it and transferring it to the other core Node.

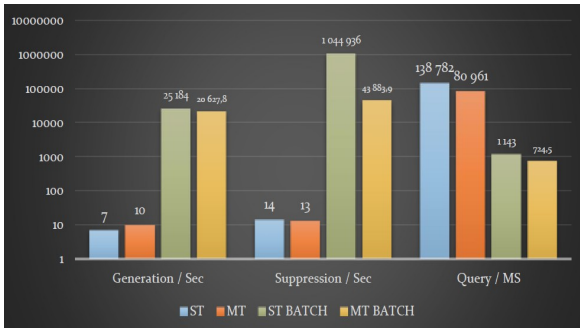

Figure 3 : Neo4 Cluster performance

8. Footnotes

In the end we can compute all the results in one figure:

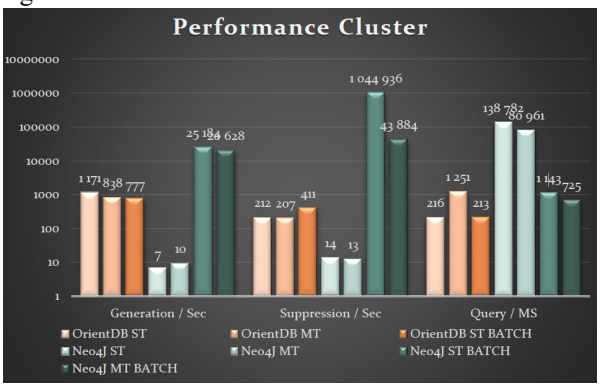

Figure 4 : OrientDB-Neo4J performance

On the side note, the cluster was composed of three instance, the engines were deployed with Ansible [3] with OrientDB 3.0.X, Neo4J 3.4.0 and JanuGraph 0.2.0.

References

[1] Neo4J : https://neo4j.com/

[2] OrientDB : https://orientdb.com/

[3]JanusGraph : http://janusgraph.org/

[4] Ansible: https://www.ansible.com/