Tuning Consistency With Apache Cassandra

Learn how to tune consistency levels with Apache Cassandra.

by Shivangi Gupta · Apr. 30, 19 · Database Zone · Tutorial

One of the challenges faced by distributed systems is how to keep the replicas consistent with each other. Maintaining consistency requires balancing availability and partitioning. Fortunately, Apache Cassandra lets us tune this balancing according to our needs. In this blog, we are going to see how we can tune consistency levels during reads and writes to achieve faster reads and writes.

Before digging more about consistency, let me first discuss CAP Theorem. CAP Theorem describes the tradeoffs in distributed systems; it states that any networked shared-data system can have at most two of three desirable properties:

- **Consistency** (C): All the nodes should have the same data at the same time
- **High availability**(A): Every request should be addressed
- **Tolerance to network partitions** (P): The system should continue to operate even in case of network partitions.

Consistency is very difficult to achieve in a distributed environment because we need all the replicas to be in sync with each within data centers and across data centers. As you can see in the diagram, Cassandra follows AP.

It optimizes Availability and Partition Tolerance itself, but for consistency, it gives flexibility by letting us tune it based on how much consistency we need in our data.

Let's get familiar with some terminologies that will be used:

- RF(Replication Factor) – Number of copies for data
- CL(Consistency Level) – Number of nodes required to acknowledge the read or write.

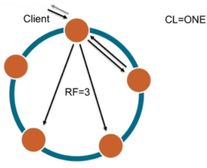
So, How Can We Tune Consistency?

Consistency levels are a part of the writes. While writing data, we need to mention the consistency level with which we want to write, and while reading data, we need to ask for a particular consistency level. This control has been given to the developer.

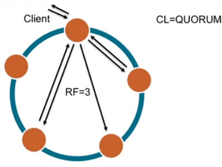
Let's take the example of writing data. For example, let's say that RF=3, meaning data is to be copied to 3 nodes.

How do we make sure the data is written completely to all the nodes? Yes, you are right; we need an acknowledgment that the work is complete. This is exactly what CL provides.

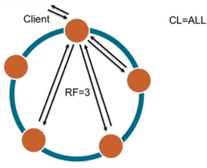
- **CL=ONE**
  Only one node is required to acknowledge the read or write. If acknowledgment from any one node is received, that write is marked as done, but the data is still being written to the other two nodes asynchronously. It's just that only one node's acknowledgment is included in the response. This is the fast consistency level.



- **CL=QUORUM**
  At least 51 percent of the nodes need to acknowledge write. Considering RF=3, we need 2 nodes out of 3 nodes to acknowledge the write; the third node will still get data asynchronously.
- **QUORUM = (Replication Factor / 2) + 1**



- **CL=ALL**
  This means that all the nodes need to acknowledge the write. This is not recommended until, and unless, you have a use case for it, as this will be equivalent to turning off the partition tolerance and availability in order to be consistent all the time.



So, as stated above, we can adjust the consistency according to the business needs. Let's see what different options we have:

Strong consistency: The data that we just wrote should be available when we read it stating there is no stale data.

But how can we achieve it?

- WRITE CL=ALL, READ CL=ONE: Not at all recommended for production environments as it will make writes slow.
- WRITE CL=QUORUM, READ CL=QUORUM: it will give high read and write speed without sacrificing availability.

Eventual consistency: The data we just wrote will be available on all nodes eventually (as copying to other nodes is done in background). This approach has low latency. Best useful for analytical data, time series data, and log data.

How Can We Maintain Consistency Across Multiple Data Centers?

LOCAL QUORUM: Only local replicas are considered in acknowledging the writes; data still gets written to the other data center. It provides strong consistency along with speed.

All the available consistency levels in Cassandra (weakest to strongest) are as follows:

- ANY
- ONE, TWO, THREE
- QUORUM
- LOCAL_ONE
- LOCAL_QUORUM
- EACH_QUORUM
- ALL: not in for availability, all in for consistency

For multiple data-centers, the best CL to be chosen are: ONE, QUORUM, LOCAL_ONE.

Consistency plays a very important role. Consistency and replication are glued together. Consistency is all about how up-to-date all the replicas at any given moment, and consistency level determines the number of replicas that need to acknowledge the success of read or write operation.

Hope you liked the blog! Please leave your questions and thoughts in the comments below.

References:

- Datastax
- Eventual Vs. Strong Consistency in Distributed Databases
- Academy Datastax
- Cassandra Documentation

Topics: APACHE CASSANDRA, APACHE, TUNING, DATABASE, CONSISTENCY, DATA CENTERS

Popular on DZone

- Best Practices for Data Pipeline Error Handling in Apache NiFi
- How I Stopped Coding Repetitive Service Components with Kong
- How to Submit a Post to DZone
- Top Practices for Containers, Kubernetes