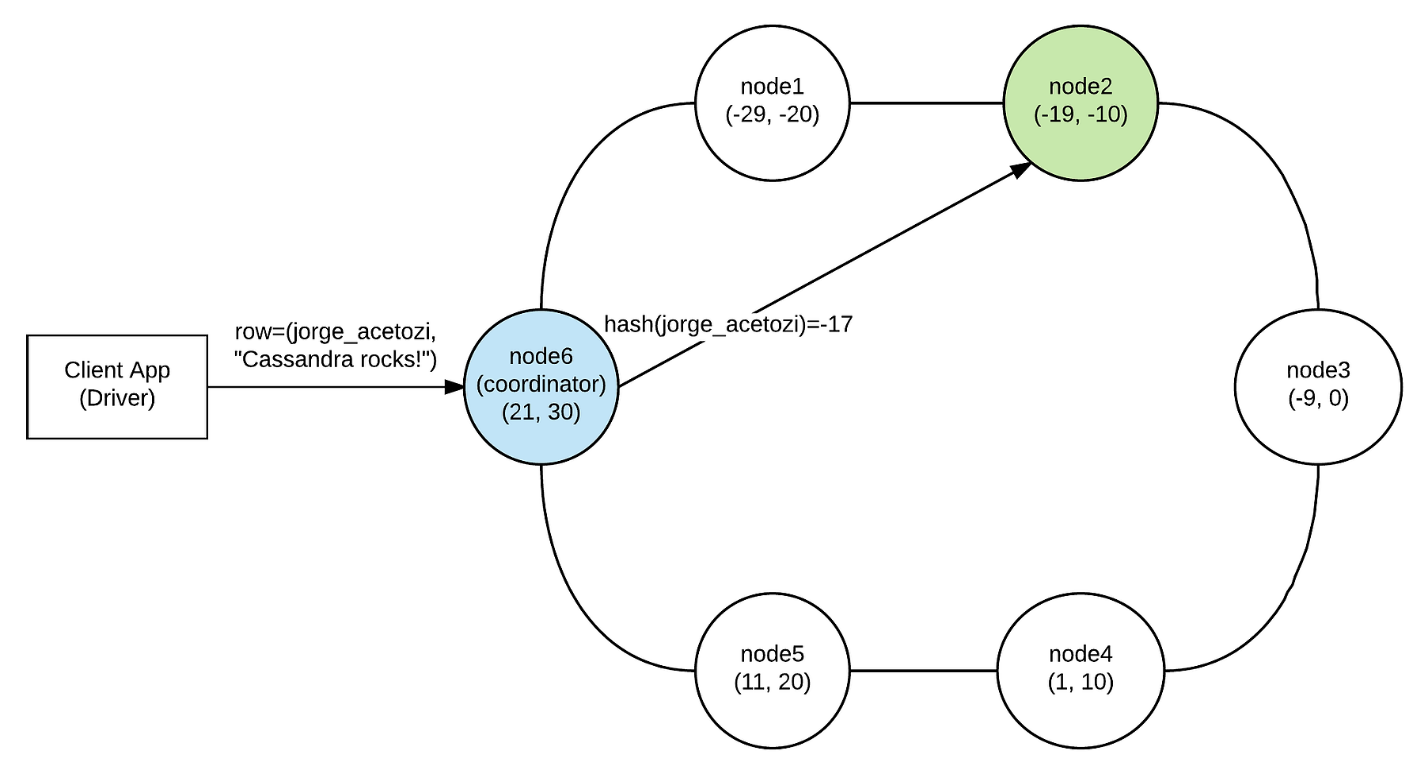
So, basically…

* Do you need fault tolerance and high availability? Use replication.
* Just bear in mind that using replication means you will pay with consistency (for most of the cases, this is not a problem. Availability is often more important than consistency).
* If consistency is not an issue for your domain, perfect. If it is, just increase the consistency level, but then you will pay with higher latency.(Higher Latency because the consistency Level is higher so the co-ordinator has to wait for all the writes to the respective replicas to succeed and come back.)
* If you want fault tolerance and high availability, strong consistency and low latency, then you should be the client, not the software engineer (Lol which means this is **NOT POSSIBLE**).

**Cassandra :** This NO SQL database is able to achieve high availability by distributing the data across a cluster of Nodes using consistent hashing. Each node also has a capability to store certain part of replica for one of the other nodes. Something similar to Coherence each node stores its own data and acts as replica for other nodes as well.

* + Reads will be super fast as consistent hashing redirects the read operation to a certain Node immediately.
  + Writes happen to the assigned consistent hash node as well as to some other nodes assigned as the replica nodes per the hashing/replica strategy/algorithm.
  + The above write strategy also means a lot of inter nodes communication which will increase with a high consistency level and eventually reduce latency in case of huge writes.
  + But this also means that the cluster can scale linearly with more need of read/write operations.
  + Cassandra is written in Java and it’s mainly used for time-series data such as metrics, IoT (Internet of Things), logs, chat messages, and so on.

How to handle node failures : As each Nodes data is backed up in some other node/s , the consistent hasher can re-assign the lost tokens to the respective replicas and the reads/writes will always be served while the Node comes back. This is in case of n/w failure of some node. Now because there is replication , it is paid with consistency level.

**Write Consistency Level**

Do you still remember that when the client sends a request to a Cassandra node, this node is called a coordinator and acts as a proxy between the client and the replica nodes?

Well, when you write to a table in Cassandra (inserting data, for example), you can specify the write consistency level. The write consistency level is the number of replica nodes that have to acknowledge the coordinator that its local insert was successful (success here means that the data was appended to the commit log and written to the memtable). As soon as the coordinator gets WRITE\_CONSISTENCY\_LEVEL success acknowledgments from the replica nodes, it returns success back to the client and doesn’t wait for the remaining replicas to acknowledge success.

For example, if an application issue an insert request with WRITE\_CONSISTENCY\_LEVEL=TWO to a table that is configured with REPLICATION\_FACTOR=3, the coordinator will only return success to the application when two of the three replicas acknowledge success. Of course, this doesn’t mean that the third replica will not write the data too; it will, but at this point, the coordinator would already have sent success back to the client.

There are many different types of write consistency levels you can specify in your write request. From the less consistent to full consistency: ANY, ONE, TWO, THREE, QUORUM, LOCAL\_QUORUM, EACH\_QUORUM, ALL.

