Introduction to Cassandra

CS185C: Introduction to NoSQL Databases
Suneuy Kim

References

[1] Cassandra: The Definitive Guide, 2nd Edition Distributed Data at Web Scale By Jeff Carpenter, Eben Hewitt, June 2016

[2] Mastering Apache Cassandra - Second Edition 2nd Edition

by Nishant Neeraj, Feburary 2015

Cassandra in 50 words

“Apache Cassandra is an open source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tuneably consistent, row-oriented database that bases its distribution design on Amazon’s Dynamo and its data model on Google’s Bigtable. Created at Facebook, it is now used at some of the most popular sites on the Web.”

In the rest of slides, C* stands for Cassandra

Distributed and Decentralized

- Distributed
  - It is capable of as well as optimized for running on multiple machines while appearing to users as a unified whole.
- Decentralized
  - Every node is identical.
  - Features Peer-to-peer (as compared to master-slave) protocol and uses gossip to maintain and keep in sync a list of nodes that are alive or dead
- In short, because C* is distributed and decentralized, there no single point failure, which supports high availability.

Elastic Scalability

- Scalability
  - An architectural feature of a system that can continue serving a greater number of request with little degradation in performance
  - Vertical scaling and horizontal scaling.
  - Horizontal scaling – software itself must have an internal mechanism for keeping its data in sync with the other nodes in the cluster
- Elastic scalability – a special property of horizontal scalability.
  - It means your cluster can seamlessly scale up and scale back down.

High availability and Fault Tolerance

- Availability of a system - Its ability to fulfill requests.
  - Multiple networked computers
  - Software they are running musts be capable of operating in a cluster.
  - Some facility for recognizing node failures and failing over request to another part of the system.
- C* is highly available
  - Failed nodes in the cluster can be replaced without downtime.
  - Data is replicated to multiple data centers.

Tunable Consistency

- Consistency
  - Consistency - A read always returns the most recently written value.
  - With scaling data stores, there are trade-offs between data consistency, node availability, and partition tolerance.
- Tunable consistency
  - C* allows you to easily decide the level of consistency you require in balance with the level of availability.

Tunable Consistency

- Degree of consistency
  - Strict consistency
    Any read will always return the most recently written value.
    A global clock that is capable of timestamping all operations
  - Causal consistency
    Causally related writes (one write occurs after another) must be read in sequence.
  - Weak (eventual) consistency
    All updates will propagate throughout all of the replicas but this may take some time.
    All replicas will be eventually consistent.

Tunable Consistency

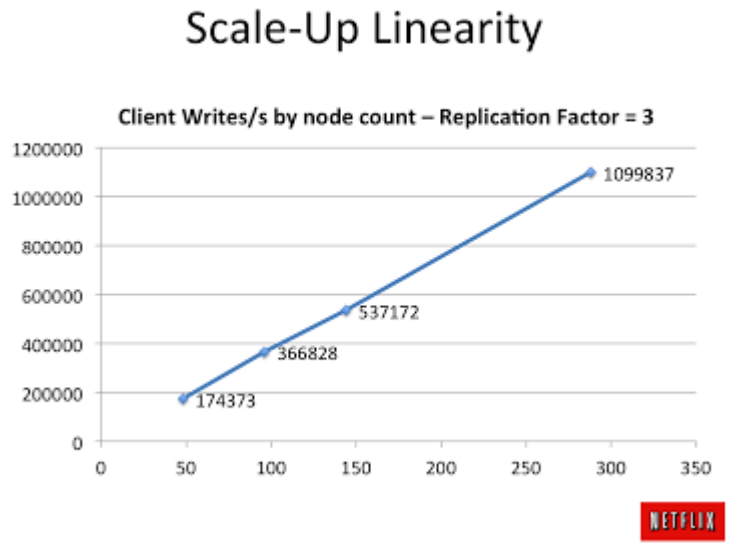
- Replication factor
  - How many times the data has to be copied
- Consistency level (ALL, QUARUM, etc)
  - How many replicas in the cluster must acknowledge a write or respond to a read to be considered successful
- C* has pushed the decision of consistency level out to the clients.
  - Example: ALL (the consistency level set to the replication factor) means the highest consistency but the lowest availability.

Row-Oriented

- Cassandra's data model is best described as a partitioned row store, in which data is stored in sparse multidimensional sorted hash tables.
  - Partitioned: Each row has a unique key and the keys are used to distribute the rows across multiple data stores.
  - Sparse multidimensional: Each row does not need to have all the same columns
  - Sorted: Column values are stored in a consistent sort order.
- A column-oriented (a.f.k. columnar) database is one in which the data is actually stored by columns. C* is NOT column-oriented.

High performance

- C* scales consistently and seamlessly to hundreds of terabytes.



http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html

- C* performs exceptionally well under heavy load.
- Very fast throughput for writes per seconds on basic commodity computers.

Flexible schema

- Schema free databases (e.g. Bigtable and MongoDB)
  - Very extensible and highly performant in accessing large amounts of data
  - Difficulty in determining the meaning and format of data
- C*'s flexible schema
  - CQL allows you to define a schema.
  - Thrift-based API that supports dynamic column creation has been deprecated starting with C* 3.0.
  - CQL collections to add content in a less structured form that can used to extend an existing schema.
  - Using CQL, the type of columns in certain instances can be changed.

  - Using ALTER TABLE, you can change the data type of a column.

  - Only newly inserted values will be created with the new type. However, the data type before must be compatible with the new data type specified.
  - CQL provides a support for the storage of JSON-formatted text.

What kind of projects C* is a good fit for?

- Large Deployments
  - If your application is expected to require
  - A single node - a RDBMS maybe a right fit.
  - Several nodes - C* might be a good fit
  - Dozens of nodes – C* might be a great fit.
- Application workloads that require high performance at significant write volumes with many concurrent client threads such as user activity updates, social network usage, recommendations/review, and application statistics.
- Geographical distribution of data
- Evolving applications – C* supports flexible schemas.