Distributed Systems
236351
Tutorial 9 - Cassandra

Yehontan Buchnik
(Based on Dolev Adas slides)

Technion

January 10, 2019
Motivation

- Facebook is the largest social network in the world
  - Hundreds of millions of users
  - Tens of thousands of servers spread over the globe
  - It is reasonable to assume that some servers will fail
  - Facebook is here, and it aims to be even bigger than it is today

- Thus, Facebook Distributed Database has to provide:
  - High performance
  - Reliability
  - Efficiency
  - Continuous growth
  - Scalability
  - Treat failures as a norm
Cassandra
Overview

- Designed in Facebook by Avinash Lakshman and Prashant Malik
- Synthesis of well known techniques such as Amazon’s Dynamo and Google’s Big Tables
- Decentralized
  - Every node in the cluster has the same role
  - No single point of failure
- Scalable
  - Read and write throughput both increase linearly as new machines are added
- Fault-tolerant
  - Data is automatically replicated to multiple nodes for fault-tolerance
  - Replication across multiple data centers is supported
- Tunable consistency
  - from "writes never fail" to "block for all replicas to be readable"
- Query language
  - CQL (Cassandra Query Language) is an SQL-like interface alternative to the traditional RPC interface
Cassandra
Data Model

- A table in Cassandra is a distributed multi dimensional map indexed by a key
- Every operation under a single row key is atomic per replica
- Columns are grouped together into sets called column families very much similar to what happens in the Bigtable
Cassandra API

- insert(table, key, rowMutation)
- get(table, key, columnName)
- delete(table, key, columnName)
Cassandra Architecture

Partition

1. Client → Node → Replica
2. Client → Node → Replica
3. Client → Node → Replica
4. Client → Node → Replica
5. Client → Node → Replica
6. Client → Node → Replica
Cassandra Architecture

Partition

- Cassandra partitions data across the cluster using consistent hashing
- The nodes arranged in a logical "ring"
  - Each node in the system is assigned a random value which represents its position on the ring
  - Each data item identified by a key which determines its position on the ring. To locate an item is done by walking the ring clockwise to find the first node with a position larger than the item's position. This node is called the **coordinator** of the item
  - Thus, each node becomes responsible for the region in the ring between it and its predecessor node on the ring
Cassandra Architecture

Partition

In order to deal with an unfair data load on the ring Cassandra offers two methods.

- Virtual nodes (from version 1.2) - like in Dynamo
- Analysing load information on the ring and have lightly loaded nodes move on the ring to alleviate heavily loaded nodes
  - Useful for heterogeneous clusters
Cassandra Architecture

Partition

Ring without VNodes

Node 1
A
F
E

Node 2
B
A
F

Node 3
C
B
A

Node 4
D
C
B

Node 5
E
D
C

Node 6
F
E
D

Ring with VNodes

Node 1
B
E
G
K

Node 2
A
P
M
O

Node 3
K
G
C
N

Node 4
M
O
I
H

Node 5
E
P
I
A

Node 6
H
C
B
O

J
L

Cassandra Architecture
Partition

Ring with VNodes

Node 1
K E M F
D J B N
P C A G
I H L O

Node 2
F I K C
B N A G

Node 3
H O B M
J M E I
D L H D
P E L N

Node 4
K F K F
C A O J
Cassandra Architecture

Replication

- Cassandra relax the quorum notion in the sense that an item is replicated at $N$ servers
  - $N$ is the replication factor configured "per-instance"
  - $N$ refers to the replicas' number in the same datacenter
- The coordinator is in charge of the replication of the data items that fall within its range
- Multiple replication policies. For example:
  - **Rack Unaware**: replication strategy then the non-coordinator replicas are chosen by picking $N - 1$ successors of the coordinator on the ring
  - **Rack Aware**: using Zookeeper to locate the replicas
  - **Datacenter Aware**: similar Rack Aware but in the datacenter level
- A client can indicate whether to store its transaction over multiple datacenters
Cassandra Architecture

Membership

- Cluster membership in Cassandra is based on very efficient anti-entropy Gossip mechanism
- Within the Cassandra system Gossip is not only used for membership but also to disseminate other system related control state
- Cassandra uses a modified version of the **Accrual Failure Detector**
  - Does not emit a Boolean value stating a node is up or down. Instead the failure detection module emits a value which represents a suspicion level for each of monitored nodes
- In Cassandra failure detection is also used to avoid attempts to communicate with unreachable nodes during various operations
Cassandra Architecture

Bootsrapping

- When a node starts for the first time, it chooses a random token for its position in the ring.
- For fault tolerance, the mapping is persisted to disk locally and also in Zookeeper.
- The token information is then gossiped around the cluster - causing all nodes to be familiar with the new node.
- To join a cluster the node has to know a few nodes within the cluster
  - configuration file
  - Zookeeper
- A node outage rarely signifies a permanent departure and therefore should not result in re-balancing of the partition assignment or repair of the unreachable replicas.
Cassandra Architecture

Local Persistence - Write Path
Cassandra Architecture

Local Persistence - Write Path

- Typical write operation involves a write into a commit log for durability and recoverability and an update into an in-memory data structure
- The write into the in-memory data structure is performed only after a successful write into the commit log
- When the in-memory data structure crosses a certain threshold, it dumps itself to disk
- Over time many such files could exist on disk and a merge process runs in the background to collate the different files into one file
- All writes are sequential to disk and also generate an index for efficient lookup
Cassandra Architecture

Local Persistence - Read Path
Cassandra Architecture
Local Persistence - Read Path

- A typical read operation first queries the in-memory data structure before looking into the files on disk.
- The files are looked at in the order of newest to oldest.
- In order to prevent lookups into files that do not contain the key, a bloom filter, summarizing the keys in the file, is also stored in each data file and also kept in memory.
Cassandra Consistency

R/W Consistency

The Cassandra consistency level is defined as the minimum number of Cassandra nodes that must acknowledge a read or write operation before the operation can be considered successful.

The client can specify desired R/W consistency:

- **Any**: always succeeds
- **One**: R/W from/to at least one replica node
- **Quorum**: a majority \((N/2 + 1)\) of the replicas must respond
- **Local_one**: at least one in the local datacenter
- **Local_Quorum**: a majority of the replicas in the local datacenter
- **Each_Quorum**: a majority of the replicas in each datacenter must respond
- **All**: all of the replicas must respond
Cassandra Requests

Write Operation

- A client contact to some node in Cassandra cluster (not necessarily the coordinator of the item)
  - if a write does not belong to the node, proxied to where the write belongs

- Hinted handoff
  - When a write is performed and a replica is down, the coordinator node stores the request for some time
  - After a node discovers from gossip that a node for which it holds hints has recovered, the node sends the data row corresponding to each hint to the target
  - If insufficient replica targets are alive to satisfy a requested consistency level, an exception is thrown with or without hinted handoff
  - Unlike Dynamo’s replication model - Cassandra does not default to sloppy quorum
Cassandra Requests
Read Operation

- when a query is made against a given key, we perform a digest query against all the replicas of the key and push the most recent version to any out-of-date replicas

- If a lower consistency level than ALL was specified, this is done in the background after returning the data from the closest replica to the client, otherwise, it is done before returning the data
Cassandra Requests

Request Illustration
Lightweight Transactions

Cassandra is able to provide **linearizable consistency** (isolation of ACID)

- For example, two users attempting to create a unique user account in the same cluster could overwrite each other’s work with neither user knowing about it
  - What about Paxos?
  - This is **compare-and-set** operation. Thus an ordinary Paxos won’t be enough (why?)
Lightweight Transactions

Modified Paxos
Lightweight Transactions

Modified Paxos

Cassandra has its own version of 4 phases Paxos that enables the above.

- (like in Paxos) - The leader picks a ballot and sends it to the participating replicas. If the ballot is the highest a replica has seen, it promises to not accept any proposals associated with any earlier ballot. Along with that promise, it includes the most recent proposal it has already received.

- (different than Paxos) - Read the current value of the row to see if it matches the expected one.

- (like Paxos) - If a majority of the nodes promise to accept the leader’s proposal, it may proceed to the actual proposal.

- (different than Paxos) - Reset the Paxos state for subsequent proposals (why?)