CASSANDRA: A DECENTRALIZED STRUCTURED STORAGE SYSTEM

Avinash Lakshman, Prashant Malik - Facebook
Presented by Dhruv Patel
Overview

- Background
- Data Model
- Architecture
- Implementation
- Facebook Search Index
- Conclusion
- Discussion
History

- Developed by Facebook
- Designed to fulfill the storage needs of Facebook
- Index Search
  - Billions of writes/day
    - High write throughput
  - Scale with number of users
- Deployed as the backend storage system for multiple services within Facebook
Motivation

- High Scalability
  - Read & Write throughput increases linearly with number of nodes
- High Availability
  - Treats failures as norms rather than exceptions
- Fail Tolerance
Data Model

- Table: Distributed multi dimensional map, indexed by a key
- Value: Object which is highly structured
- Row Key: String – no size restriction
  - Normally 16-36 bytes long
- Operations are **atomic** on each row per replica
Data Model (contd.)

- Column Families (CF) – columns grouped together
  - Number of CFs not limited per table

- Types of Column Families:
  - Simple CF
  - Super CF – CF within CF

- Column Sort Order – application specific
  - Time
    - Facebook Inbox Search – results displayed in time sorted order
  - Name
Each Column has
- Name
- Value
- Timestamp

Column Access
- Simple column
  - column_family: column
- Super column
  - column_family:super_column:column
## Data Model (contd.)

<table>
<thead>
<tr>
<th>Key</th>
<th>User_Id</th>
<th>User_Name</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>6576e768-8r73-78df</td>
<td>786780</td>
<td>John</td>
<td>2010-04-17T18:10:11</td>
</tr>
<tr>
<td>2456e124-6y78-12ef</td>
<td>218745</td>
<td>Bob</td>
<td>2010-02-12T14:12:16</td>
</tr>
</tbody>
</table>
Figure adapted from [3]
API

Methods:
- insert (table, key, rowMutation)
- get (table, key, columnName)
- delete (table, key, columnName)

columnName:
- Column within column family
- Column family
- Super column family
- Column within a super column
System Architecture

- Partitioning – high scalability
- Replication – high availability and durability
- Cluster membership – how nodes are added/deleted
- Bootstrapping – how nodes start for the first time
- Scaling the Cluster
- Local Persistence
- Implementation Details
Partitioning

- Scale incrementally
- Nodes: logically structured in Ring Topology
  - Each node assigned a random value — position
  - Hashing on data-item’s key, assign to node
  - Walk the ring clockwise
  - This node — coordinator of the key
Partitioning (contd.)

A

B

C

D

E

h(key1)

h(key2)
Partitioning (contd.)

- Consistent Hashing

<table>
<thead>
<tr>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Departure or arrival of the node only affects its immediate neighbours</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Non-uniform data &amp; load distribution</td>
</tr>
<tr>
<td>• Unaware of node performance heterogeneity</td>
</tr>
</tbody>
</table>

Solution

- Assign nodes to multiple position on the ring
- Analyze load information
  - Load Distribution
    - Move lightly loaded nodes to alleviate heavily loaded nodes

Cassandra uses the 2nd approach
Data items are replicated at N nodes
Cassandra Replication Policies

- **Rack Unaware**
  - Replicate data on N-1 nodes after its coordinator node

- **Rack Aware**
  - Zookeeper chooses a node leader
  - Tells nodes the range they are replicas for

- **Datacenter Aware**
  - Leader is chosen at Datacenter level instead of Rack level

Metadata about ranges a node is responsible for

- Cached locally at each node and inside Zookeeper
  - Fault tolerant: Node crash and comes back – knows its range
Cluster Membership

- **Scuttlebutt**
  - Gossip based protocol
    - Inspired from real life rumor spreading
    - Periodic, Pairwise & Inter-node communication

- **Failure Detection**
  - Determine which node is up and down
  - Avoid attempts to communicate with unreachable nodes
Cluster Membership (contd.)

- Failure Detection
  - Ø Accrual Failure Detection
    - emits Ø, instead of binary value (up/down)
  - Ø = Suspicion level for each monitored node
    - Network and load condition
    - Node maintains sliding window of inter arrival time of gossip message from other nodes, Ø is calculated
  - If node is faulty: suspicion level increases with time
  - If node is correct: Ø will be constant set by the application, generally 0
Bootstrapping

- Node starts for the first time
- Reads configuration file – list of few contact points
  - Zookeeper
- Receives a random token for its position
- Persists mapping locally and in Zookeeper
- Token information is gossiped around the cluster
- Or manually by administrator via command line tool or Cassandra web interface
Scaling

- Joining node assigned a token
  - s.t. it can alleviate a heavily loaded node
  - Splitting the range

- Bootstrap algorithm initiated by CLT or web interface

- Overloaded node streams the data to new node using kernel-kernel copy

- 40MBPS
Local Persistence

- Relies on the local file system for data persistence

- Write Operation
  - Write to commit log in local disk of the node
    - Durability and recoverability
  - Update in-memory data structure
    - Only after successful write in commit log
  - When in-memory data structure crosses certain limit, it dumps itself to disk
  - Merge process runs in background to merge such files into one file
Local Persistence (contd.)

Write Implementation

- Key (CF1, CF2, CF3)
- Commit Log
  - Binary serialized
  - Key (CF1, CF2, CF3)
- Memtable (CF1)
- Memtable (CF2)
- Memtable (CF2)
- Dedicated Disk
  - K_{128} Offset
  - K_{256} Offset
  - K_{384} Offset
  - Bloom Filter
  - (Index in memory)
- Data file on disk
  - Data size
  - Number of Objects
  - Lifetime
  - <Key name>|<Size of key Data>|<Index of columns/supercolumns>|<Serialized column family>
    - ---
    - ---
  - BLOCK Index <Key Name> Offset, <Key Name> Offset
    - ---
    - ---
  - <Key name>|<Size of key Data>|<Index of columns/supercolumns>|<Serialized column family>

Figure adapted from [6]
Local Persistence (contd.)

- Compaction: files that are close to each other

**Figure adapted from [6]**
Local Persistence (contd.)

- **Read Operation**
  - Looks up in-memory data structure: newest to oldest
    - If not found, look into files on disk
  - **Bloom Filter**
    - Summarizing keys in one file
    - Avoid looking up files which do not contain the key
    - Stored in memory and on each data file
  - **Column Index**
    - Jump to right chunk on disk for column retrieval
    - At every 256k chunk boundary
Implementation Details

- Cassandra process on a single machine
  - Partitioning module
  - Cluster membership and failure detection module
  - Storage engine module

- Architecture is based on SEDA
  - Staged Event Driven Architecture
    - Operations transit from one stage to the next
    - Each stage can be handled by different thread pool
      - Gives high performance

Figure adapted from [8]
Commit Log Maintenance

- Commit log is rolled out after it reaches certain threshold (128MB)
- Each commit log contains header
  - Bit vector
  - Shows if column family has successfully persisted to disk
  - This header will be checked before purging the commit log
    - Make sure all the data is persisted to disk
Facebook Index Search

- Two kind of search
  - Term Search
    - Key: user_id
    - Super Column: words that make up the message
  - Interaction Search
    - Key: user_id
    - Super Column: recipients ids’

- For each of these super columns, individual message identifiers are the columns
- Term Search – not available currently!!!
Facebook Index Search (contd.)

- Faster search technique
  - User clicks on search bar
  - Asynchronous message is sent to Cassandra cluster
  - Prime the buffer cache with that user’s index
  - Search results likely to be in memory when query is executed

- 50+TB data on 150 node cluster
Conclusion

- High write throughput
- No single points of failure
- Linear scalability
- Durability
- Clever integration of Bigtable and Dynamo
Questions and Discussion

- Security
  - Need to use trusted environments
  - Development of Security Layer externally
- Comparison with other such system would have generated more persuasive results
- Dense reading
- No figures
References


[15] https://huu.la/blog/review-of-cassandra--a-decentralized-structured-storage-system


