CASSANDRA: A DECENTRALIZED STRUCTURED STORAGE SYSTEM

Avinash Lakshman, Prashant Malik
Facebook

Presented by: Besat Kassaie
Agenda

• Background
• Data Model
• Architecture
• Implementation
• Facebook Inbox Search
• Conclusion
• Discussion
History

- Cassandra was created by Facebook to fulfill the storage needs of the Facebook Inbox Search
- Facebook open-sourced Cassandra in 2008 to Apache
- The latest version released by Apache is 2.1.0
Motivations

**High Scalability:** Read/write throughput increases linearly when number of nodes increases

**High Availability:** Cassandra treats failures as the norm rather than exception

**High write throughput:** By efficient disk access policy and flexible consistency level
Cassandra & CAP

• CAP theorem: In any given system, we can strongly support only two out of consistency, availability and partition tolerance.
# Related Systems

<table>
<thead>
<tr>
<th></th>
<th>Cassandra</th>
<th>Bigtable</th>
<th>Dynamo</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Model</strong></td>
<td>Column-Oriented</td>
<td>Column-Oriented</td>
<td>Key-Value</td>
</tr>
<tr>
<td><strong>CAP Theorem</strong></td>
<td>AP</td>
<td>CP</td>
<td>AP</td>
</tr>
<tr>
<td><strong>Distributed</strong></td>
<td>Decentralized P2P</td>
<td>Master-Slave</td>
<td>Decentralized P2P</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Related Systems

Google Bigtable
- Column Families
- Memtables
- SSTables

Amazon Dynamo
- Consistent hashing
- Partitioning
- Replication

Cassandra
Agenda

• Background
• Data Model
• Architecture
• Implementation
• Facebook Inbox Search
• Conclusion
• Discussion
Data Model

"A table is a distributed multi-dimensional map indexed by a key"

- Operations are **atomic** on each row per replica.

- Spreads data over nodes

- Multiple columns per key

- String with no size restriction
Data Model

- Columns are grouped into Column Families (CF):
  - CFs have to be defined in advance → structured storage system
  - The number of CFs is not limited per table

- Types of Column Families:
  - Simple
  - Super (nested Column Families)

- Column
  - Has (Name, Value, Timestamp) and Can be ordered by timestamps or name

- Row
  - Can have a different number of columns
Data Model

Simple Column Family

Super Column Family

Figure taken from [3]
API

- Insert(table, key, rowMutation)
- Get(table, key, ColumnName)
- Delete(table, key, ColumnName)
Agenda

• Background
• Data Model
• Architecture
• Implementation
• Facebook Inbox Search
• Conclusion
• Discussion
Architecture

- **Partitioning**
  - How data is partitioned across nodes to achieve high scalability

- **Replication**
  - How data is duplicated across nodes to achieve high availability and durability

- **Cluster Membership**
  - How nodes are added/deleted to the cluster

- **Bootstrapping**
  - How nodes start for the first time
Partitioning

- Partitions data through consistent hashing
  - Order preserving hash function
- Nodes are structured in a ring
  - Each node receives a value representing its position
  - Hashing rounds off after certain value to support ring structure
- Hashed value of data key determines data position in the ring
  - Walking the ring clockwise first node will be the coordinator node
Figure adapted from [4]
## Partitioning

**Consistent Hashing:**

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Nodes departure/arrival only affects the immediate neighbors</td>
<td>• Non-uniform load distribution</td>
</tr>
<tr>
<td></td>
<td>• Unaware of the node performance heterogeneity</td>
</tr>
</tbody>
</table>

**Solutions**

1. Assigning nodes to multiple positions in the ring (Virtual Nodes)
2. Analyze nodes’ load information and change the nodes’ location

**Cassandra uses the second approach**
Replication

- Data is replicated at N (replication factor) hosts

- Cassandra Replication Policies:

<table>
<thead>
<tr>
<th>Rack Unaware</th>
<th>Replicate data at N-1 successive nodes after its coordinator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rack Aware</td>
<td>‘Zookeeper’ chooses a leader which tells nodes the range they are replicas for</td>
</tr>
<tr>
<td>Datacenter Aware</td>
<td>Similar to Rack Aware but leader is chosen at Datacenter level instead of Rack level</td>
</tr>
</tbody>
</table>
Rack Unaware Replication

Figure adapted from [4]
Membership

- Cluster membership: based on an anti-entropy Gossip based mechanism (called Scuttlebutt)

- Gossip:
  - Network Communication protocols based on real life rumor spreading

- Anti Entropy Gossip:
  - Repairs replicated data by comparing and reconciling differences
Failure Detection

- Distribution is changed to Exponential
- Cassandra is the first implementation of Accrual Failure Detection in a gossip based configuration
Bootstrapping and Scaling

• Bootstrapping
  • A node starts for the First time
  • Receives a random token for its position in the ring
  • Persists mapping locally and in Zookeeper
  • Gossips the token information to others

• Scaling:
  • Joining node receives a token to help an overloaded node
  • Overloaded node copy a related range of data to the new node
Agenda

• Background
• Data Model
• Architecture
• Implementation
• Facebook Inbox Search
• Conclusion
• Discussion
Persistence Components

Commit Log File
- Is an append only file
- Has a dedicated local disk

MemTable
- In-memory data structure
- One memtable for each column family

SSTable (Sorted Strings Table)
- On-disk data structure
- Unchangeable once written
Write Implementation

No Lock is required
+ Append only Commit Log file = High write throughput
Compaction Process

- Keys are merged
- Columns are combined
- Records with deleted flag are discarded
- A new index is created
Read Repair

replication_factor = 3

CONSISTENCY_LEVEL = ONE

Figure adapted from [7]
Read Path

- Search the in-memory data structures
- Disk lookup is required when data is not in memory
- Each SSTable has a Bloom filter and index file
- Bloom filter is consulted to reduce the number of files for search
- Index is used to access the right chunk of disk
Staged Event-Driven Architecture (SEDA)

- SEDA is a concurrency model consisting of some stages.
- Stage is a basic unit of work
  - a queue,
  - an event handler
  - a thread pool
- Operations transit from one stage to the next.
- Each stage can be handled by a different thread pool.

High performance
Commit Log Maintenance

- The commit log is rolled out after its size reaches a threshold

- Each commit log contains a header to show whether each updated memtable persisted

- The header will be checked to make sure that all data is persisted before purging the commit log
Implementation modules

- Cassandra Modules in each node:
  - Partitioning
  - Cluster membership and failure detection
  - Storage engine

- Modules are implemented in Java

- The architecture is based on SEDA
Agenda

• Background
• Data Model
• Architecture
• Implementation
• Facebook Inbox Search
• Conclusion
• Discussion
Facebook Inbox Search

• Two types of search:
  • Term search
    • All messages containing a specific term
  • Interaction search
    • All messages between two specific users

Currently term search does not work in Facebook!
Inbox Search Schema

Super Column 1: Term1
msgID_i msgID_j msgID_k ...... ...

Super Column N: TermN
msgID_f msgID_h msgID_s ...... ...

Row Key <user id>

<table>
<thead>
<tr>
<th>Column Family1</th>
<th>Column Family 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super Column1</td>
<td>Super Column N</td>
</tr>
<tr>
<td></td>
<td>Super Column K</td>
</tr>
</tbody>
</table>

Super Column 1: UserID 1
msgID_i msgID_j msgID_k ...... ...

Super Column K: UserID K
msgID_f msgID_h msgID_s ...... ...

Super Column1
......

Super Column N
......

Super Column K
......
Agenda

• Background
• Data Model
• Architecture
• Implementation
• Facebook Inbox Search
• Conclusion
• Discussion
## First release vs 2.0

<table>
<thead>
<tr>
<th>Feature</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Model</td>
<td>• No super column</td>
</tr>
<tr>
<td></td>
<td>• Terminology is changed</td>
</tr>
<tr>
<td>API</td>
<td>• CQL offered</td>
</tr>
<tr>
<td>Partitioning</td>
<td>• Virtual nodes added</td>
</tr>
<tr>
<td>Replication</td>
<td>• All replicas are not equal for reads</td>
</tr>
<tr>
<td></td>
<td>• No Zookeeper</td>
</tr>
<tr>
<td>Persistence</td>
<td>• Automatic memtable sizes</td>
</tr>
<tr>
<td></td>
<td>• Automatic flush policy</td>
</tr>
</tbody>
</table>
Point to be considered ..... 

• **Cassandra power**: 
  • High write throughput 
  • No single points of failure 
  • Linear scalability 

• **Cassandra weakness**: 
  • No Join 
  • Atomic only per row 
  • Thinking in Reverse for data modeling
References


Thank you!

Q&A
Discussion

• Security
  • Many NoSQL databases like Cassandra, do not have security features similar to what we see in GRANT/REVOKE operations in relational databases.
  • The products themselves say that they are only designed to be accessed from “trusted environments”.
  • Isn’t this a restriction?

• Calculus based
  • The relational model has a strong Relational Algebra base.
  • There is not such a foundation for NoSQL databases.
  • Can NoSQL databases be a lasting solution despite this fact?