Cassandra
By Avinash Lakshman and Prashant Malik
Presented by Joel Sharin
Motivation

- Facebook needed a solution to Inbox Search
  - High write throughput
  - Many failures
    - Failure is the norm
- Continuous Growth
- Eventual consistency
  - “Knobs” to tune tradeoffs between consistency, durability and latency
BACKGROUND

• Like Google’s Big Table
  • Column Families
  • Memtables
  • SSTables
• Aspects of Dynamo
  • Consistent hashing
  • Partitioning
  • Replication
• Criticized by some as having little novel contribution
DATA MODEL

- NoSQL
  - SQL focuses on complicated queries and analysis
    - Large datasets often don’t need
    - Not usually the best for write heavy workloads
  - Multidimensional Map, indexed by key
DATA MODEL

keyspace

column family

settings (eg, partitioner)

settings (eg, comparator, type [Std])

column

name
value
clock
Data Model

Column

Name  colA  value1  Value
DATA MODEL

Row

key

a
colA
value1
colFoo
aval
milk
white

columns
DATA MODEL
API

- `insert(table, key, rowMutation)`
  - No locks in the critical path
  - Sequential disk access
  - Atomicity guarantee for a key
  - “Always Writable” – accept writes during failure scenarios
  - Synchronous or async
- `get(table, key, columnName)`
  - Routed to closest node, or to all which can wait for quorum
- `delete(table, key, columnName)`
  - Can be routed to any node, which will reroute as needed
SYSTEM ARCHITECTURE

- All nodes are identical
- No master or single point of failure
- Scalable and robust
  - Adding nodes is simple

- replica synchronization
- state transfer
- concurrency and job scheduling
- request marshalling/routing
- system monitoring and alarming
- configuration management
Partitioning

- Consistent hashing using a ring (like Dynamo)
- Each node has a position on the ring
- Data is hashed to a position on the ring
  - The node responsible is the first node after it in the ring
- New nodes or crashed nodes only affect immediate neighbors
PARTITIONING

• Handling overloaded nodes
  • Nodes can be assigned multiple places on the ring
    ● This is how Dynamo operates
    ● Cassandra switched to this method
  • New nodes can be added to help overloaded nodes
    ● Cassandra started with this method
PARTITIONING

Diagram showing a circular partitioning scheme with nodes labeled A, B, C, D, E, and F. Arrows indicate the flow based on the hash functions h(key1) and h(key2). The number of partitions is indicated as N=3.
**Replication**

- Each data item is replicated at $N$ nodes (configurable)
- Each coordinator manages the replication
  - “Rack Unaware”
    - $N - 1$ Successors
  - “Rack Aware” (within a datacenter) and “Datacenter Aware”
    - Zookeeper to select leader
    - All nodes on joining the cluster contact the leader who tells them for what ranges they are replicas for
MEMBERSHIP

- Scuttlebutt gossip protocol used to send membership information out
  - $O(\log(n))$ rounds to send messages out
- Failure detector uses the gossip channel to detect when machines may have gone down
Gossip
Gossip
Gossip
ACCRUAL FAILURE DETECTOR

• Valuable for system management, replication, load balancing etc.
• Defined as a failure detector that outputs a value, \( \Phi \), associated with each process.
• The value output, \( \Phi \), represents a suspicion level.
  • If \( \Phi = 1 \), then the likelihood that we will make a mistake is about 10%. The likelihood is about 1% with \( \Phi = 2 \), 0.1% with \( \Phi = 3 \)
• Applications set an appropriate threshold, trigger suspicions and perform appropriate actions.
• In Cassandra the average time taken to detect a failure is 10-15 seconds with the \( \Phi \) threshold set at 5
Bootstrap:

- Nodes choose random positions on the ring at first (can also read from Zookeeper)
  - Saved to Zookeeper and gossiped
  - Other nodes can route requests
- Node outages are usually considered transient and don’t result in rebalancing
  - Nodes must be explicitly added
Scaling the Cluster

• New nodes are given tokens to alleviate heavily loaded nodes
• Old node streams data to the new node
  • 40MB/sec at the time of paper release
  • Optimize with a bittorrent like procedure
Local Persistence

- Local filesystems at nodes are used to store data
- Write to commit log and update in memory structures
- Commit log
  - Sequential writes to maximize disk throughput
- In memory data structures
  - Dumped to disk when size is too large
- Over time, many files exist on disk
  - Compaction process merges them
- Reads access memory, then disk chronologically
  - Filter used before checking an entire file
LOCAL PERSISTENCE

![Diagram showing local persistence with components like write data, memtable, commit log, SSTable, and flush.]
Implementation Details

- Event driven
- Java
- Module based
- System control messages use UDP
- Application messages use TCP
- Asynchronous replication for high throughput
- Commit log purged after size > 128MB, keeping track of what’s been written to disk
PRACTICALITY

• No transactions, which can be difficult
• Though cassandra is decentralized, coordination via Zookeeper was felt necessary
• Facebook ended up switching off of it for Inbox Search, though uses it for Instagram
• Separate tools for monitoring
EVALUATION

• Facebook later switched off of Cassandra in favor of HBase
  ○ This was apparently due to consistency models
• There is no comparison of Cassandra vs. other storage systems in the paper
EVALUATION

- Used to handle the original Inbox Search problem

<table>
<thead>
<tr>
<th>Latency Stat</th>
<th>Search Interactions</th>
<th>Term Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>7.69ms</td>
<td>7.78ms</td>
</tr>
<tr>
<td>Median</td>
<td>15.69ms</td>
<td>18.27ms</td>
</tr>
<tr>
<td>Max</td>
<td>26.13ms</td>
<td>44.41ms</td>
</tr>
</tbody>
</table>
EVALUATION

- MySQL > 50 GB
  - Data Writes Average : ~300 ms
  - Reads Average : ~350 ms
- Cassandra > 50 GB
  - Data Writes Average : 0.12 ms
  - Reads Average : 15 ms
Questions?