Cassandra

Wa’el Belkasim
Arash Akhlaghi
Badrinath Jayakumar
Sparse, multi-dimensional data model, storage architecture

Cluster management, fault tolerant, replication

Started by Facebook

Born 2008
Column or Row Oriented DB?

<table>
<thead>
<tr>
<th>id</th>
<th>Name</th>
<th>Age</th>
<th>Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jerry</td>
<td>15</td>
<td>M</td>
</tr>
<tr>
<td>2</td>
<td>Tom</td>
<td>14</td>
<td>M</td>
</tr>
<tr>
<td>3</td>
<td>Tweety</td>
<td>16</td>
<td>F</td>
</tr>
</tbody>
</table>

Logical Schema

Row Oriented Database

Column Oriented Database

A little of both...
Cassandra

• Cassandra is a highly scalable, **eventually consistent**, distributed, fault tolerant, **structured key-value** store.
  • So it is row-oriented **and** column-structured.

• Cassandra can be used as both a real-time data store and a read-intensive database for Business Information Systems. It is classified as a NoSQL data store.

• What lead to Cassandra?
  • Facebook’s inbox search function

• **Eventually Consistent?**
  • Building reliable distributed systems on a global scale demands trade-offs between consistency and availability
  • Consistency: when something is written, it is expected all reads after the read will have access to that written data
  • In Cassandra, due to the distributed nature, there is no such hard guarantee
  • However, we can say, it **eventually** reaches a consistent state...
Cassandra Consistency

**CAP Theorem** – Pick 2/3: Consistency, Availability, Partition tolerance

What is Cassandra’s choice?

- Availability
- Partitioning
- But it is ‘eventually’ consistent because...
- Cassandra has several levels of consistency: i.e. it will block subsequent reads/writes for a time depending on a user setting & replica count
- And so **eventually** all the nodes will have the data replicated, thus eventually consistent
Current Users

Netflix
Adobe
Digg
Twitter
HP
IBM
Rackspace
Cisco
Reddit
...

[Images of Netflix, Twitter, and IBM logos]
Interfacing with Cassandra

- Cassandra Command Line Interface
  - Original Cassandra
  - Latest version (1.2.4) (Release April 11 2013)
- CQL (Cassandra Query Language)
  - Supports subset of SQL features
  - Standard DDL, DML commands can be used
  - Group by and Order by is not supported
  - Latest version 3.0
# Architecture Overview

- Peer to peer distributed system
- Read/Write anywhere design
- Gossip Protocol for communication
- Use of commit log
- In memory structure: memtable and SSTable

<table>
<thead>
<tr>
<th>Core Layer</th>
<th>Middle Layer</th>
<th>Top Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messaging service</td>
<td>Commit log</td>
<td>Tombstones</td>
</tr>
<tr>
<td>Gossip Failure detection</td>
<td>Memtable</td>
<td>Hinted handoff</td>
</tr>
<tr>
<td>Cluster state</td>
<td>SSTable</td>
<td>Read repair</td>
</tr>
<tr>
<td>Partitioner</td>
<td>Indexes</td>
<td>Bootstrap</td>
</tr>
<tr>
<td>Replication</td>
<td>Compaction</td>
<td>Monitoring</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Admin tools</td>
</tr>
</tbody>
</table>
Architecture Overview

1. Client Writes to any Cassandra Node
2. Coordinator Node replicates to nodes and Zones
3. Nodes return ack to coordinator
4. Coordinator returns ack to client
5. Data written to internal commit log disk

If a node goes offline, hinted handoff completes the write when the node comes back up.

Requests can choose to wait for one node, a quorum, or all nodes to ack the write

SSTable disk writes and compactions occur asynchronously

Why Cassandra?

- Gigabyte to Petabyte scalability
- Flexible Schema Design
- Linear Performance of gains through node addition
- Cassandra Query Language (like SQL)
- Data Compression
- No need for separate caching layer
- Tunable data consistency
Data Model - Bottom up Approach

- Column
- Row key
- Column Family
- Super Column Family
- Key Space
- Cluster
Mapping to Relational World

<table>
<thead>
<tr>
<th>Relational Model</th>
<th>Cassandra Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Keyspace</td>
</tr>
<tr>
<td>Table</td>
<td>Column Family (CF)</td>
</tr>
<tr>
<td>Primary key</td>
<td>Row key</td>
</tr>
<tr>
<td>Column name</td>
<td>Column name/key</td>
</tr>
<tr>
<td>Column value</td>
<td>Column value</td>
</tr>
</tbody>
</table>
Column

Basic Data structure

- Since its written in java, this byte[] is java byte[].
- Choosing no of columns is a client option.
- A single column value may not be larger than 2GB.
- Not possible to query Clock information.
Column Continued

- Columns names are sorted by the “comparator” type defined on their enclosing column family.

Sample Column Family Definition

```sql
create column family SAMPLECOLUMNFAMILY
with comparator = UTF8Type
and default_validation_class = 'UTF8Type'
and key_validation_class = 'UTF8Type';
```

- Column values can’t be sorted
### Column Continued

<table>
<thead>
<tr>
<th>Internal Type</th>
<th>CQL Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BytesType</td>
<td>blob</td>
<td>Arbitrary hexadecimal bytes (no validation)</td>
</tr>
<tr>
<td>AsciiType</td>
<td>ascii</td>
<td>US-ASCII character string</td>
</tr>
<tr>
<td>UTF8Type</td>
<td>text, varchar</td>
<td>UTF-8 encoded string</td>
</tr>
<tr>
<td>IntegerType</td>
<td>varint</td>
<td>Arbitrary-precision integer</td>
</tr>
<tr>
<td>Int32Type</td>
<td>int</td>
<td>4-byte integer</td>
</tr>
<tr>
<td>InetAddressType</td>
<td>inet</td>
<td>IP address string in xxx.xxx.xxx.xxx form</td>
</tr>
<tr>
<td>LongType</td>
<td>bigint</td>
<td>8-byte long</td>
</tr>
<tr>
<td>UUIDType</td>
<td>uuid</td>
<td>Type 1 or type 4 UUID</td>
</tr>
<tr>
<td>TimeUUIDType</td>
<td>timeuuid</td>
<td>Type 1 UUID only (CQL3)</td>
</tr>
<tr>
<td>DateType</td>
<td>timestamp</td>
<td>Date plus time, encoded as 8 bytes since epoch</td>
</tr>
<tr>
<td>BooleanType</td>
<td>boolean</td>
<td>true or false</td>
</tr>
<tr>
<td>FloatType</td>
<td>float</td>
<td>4-byte floating point</td>
</tr>
<tr>
<td>DoubleType</td>
<td>double</td>
<td>8-byte floating point</td>
</tr>
<tr>
<td>DecimalType</td>
<td>decimal</td>
<td>Variable-precision decimal</td>
</tr>
<tr>
<td>CounterColumnType</td>
<td>counter</td>
<td>Distributed counter value (8-byte long)</td>
</tr>
</tbody>
</table>

Source: Datastax
Rows

- A logical grouping of columns
- Encoding type defined in CF(Column Family).
- Wide Rows, Skinny Rows

create column family SAMPLECOLUMNFAMILY with comparator = UTF8Type and default_validation_class = 'UTF8Type' and key_validation_class = 'UTF8Type';
Column Family

A *column family* is a container for an ordered collection of rows, each of which is itself an ordered collection of columns.
Column Family Syntax

CREATE COLUMN FAMILY users WITH comparator = UTF8Type AND key_validation_class=UTF8Type AND column_metadata = [
{column_name: full_name, validation_class: UTF8Type}
{column_name: email, validation_class: UTF8Type}
{column_name: state, validation_class: UTF8Type}
{column_name: gender, validation_class: UTF8Type}
{column_name: birth_year, validation_class: LongType} ];
Column Family Options

• **keys_cached**
  
The number of locations to keep cached per SSTable. This doesn’t refer to column name/values at all, but to the number of keys, as locations of rows per column family, to keep in memory in least-recently-used order.

• **rows_cached**
  
The number of rows whose entire contents (the complete list of name/value pairs for that unique row key) will be cached in memory.

• **comment**
  
This is just a standard comment that helps you remember important things about your column family definitions.

• **read_repair_chance**
  
This is a value between 0 and 1 that represents the probability that read repair operations will be performed when a query is performed without a specified quorum, and it returns the same row from two or more replicas and at least one of the replicas appears to be out of date. You may want to lower this value if you are performing a much larger number of reads than writes.

• **preload_row_cache**
  
Specifies whether you want to prepopulate the row cache on server startup.
Super Column Family

• A container for super columns sorted by their names. Like Column Families, Super Column Families are referenced and sorted by row keys.
• To use a super column, you define your column family as type Super
• Composite Keys
Important facts

• Super columns were one of the updates that Facebook added to Google’s Bigtable data model.

• **Do not** use super columns. They are a legacy design from a pre-open source release. This design was structured for a specific use case and does not fit most use cases. Additionally, super columns are not supported in CQL 3.
Key Space

A cluster is a container for keyspaces—typically a single keyspace. A keyspace is the outermost container for data in Cassandra.

create keyspace SampleKS
    with strategy_options = {replication_factor:1}
    and placement_strategy =
'org.apache.cassandra.locator.SimpleStrategy'
Cluster

It can be single node, multiple node, or multiple data center cluster.
Configure through cassandra.yaml
Ring is a common word to refer it.

A little demo..
Read and write in Cassandra
Write Mechanism
Read/Write speed with different NO SQL system.

![Bar chart showing read/write speeds for different NO SQL systems: HBase, Cassandra, and MongoDB.](chart.png)
Tunable Data Consistency

• Choose between strong and eventual consistency depending on the need. (All to any node responding)
• Can be done on a per operation basis and for both reads and writes.
• Handles multi data center operations.
Selecting Strategy for writes

• ZERO: The write operation will return immediately to the client before the write is recorded; the write will happen asynchronously in a background thread, and there are no guarantees of success.

• ANY: Ensure that the value is written to a minimum of one node, allowing hints to count as a write.

• ONE: Ensure that the value is written to the commit log and memtable of at least one node before returning to the client.

• QUORUM: Ensure that the write was received by at least a majority of replicas \((\text{replication factor} / 2) + 1\).

• ALL: Ensure that the number of nodes specified by replication factor received the write before returning to the client. If even one replica is unresponsive to the write operation, fail the operation.
Hinted Handoffs

• Cassandra attempts to write a row to all replicas for that row.
• If all replica nodes are not available, a hint stored on one node to update any downed with the row once they are available again.
• If no replica nodes are available for a row, the use of the ANY consistency level will instruct the co-ordinator node to store a hint and the raw data which it passes to the replica nodes when they are available.
Selecting Strategy for reads

- ZERO Unsupported. You cannot specify CL.ZERO for read operations because it doesn’t make sense. This would amount to saying “give me the data from no nodes.”
- ANY Unsupported. Use CL.ONE instead.
- ONE Immediately return the record held by the first node that responds to the query. A background thread is created to check that record against the same record on other replicas. If any are out of date, a read repair is then performed to sync them all to the most recent value.
- QUORUM Query all nodes. Once a majority of replicas ((replication factor / 2) + 1) respond, return to the client the value with the most recent timestamp. Then, if necessary, perform a read repair in the background on all remaining replicas.
- ALL Query all nodes. Wait for all nodes to respond, and return to the client the record with the most recent timestamp. Then, if necessary, perform a read repair in the background. If any nodes fail to respond, fail the read operation.
Read Repair

• Frequently read data remains consistent
• When a read is done, the co-ordinator node compares the data from all remaining replicas to confirm the row in the background and if they are inconsistent, issues writes to the out-of-date replicas to update the row to reflect the most recently written values.
• Read repair can be configured per column family and is enabled by default.
Client Layer

Low level Client
  Thrift API
High level Client for Java
  • Firebrand
  • Hector
  • Pelops
  • PlayOrm
  • Kundera
Why Hector API?

- High level Object Orient API
- Fail Over support
- Connection Pooling
- JMX support
Database Design - Cricket Application

<table>
<thead>
<tr>
<th>ODI_PLAYER</th>
<th>PLAYER BY COUNTRY</th>
<th>MATCHES PLAYED</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CF&gt;</td>
<td>&lt;ROWKEY&gt;</td>
<td>&lt;CF&gt;</td>
</tr>
<tr>
<td></td>
<td>#PlayerID</td>
<td>#PlayerID</td>
</tr>
<tr>
<td></td>
<td>+name</td>
<td>+Trophy</td>
</tr>
<tr>
<td></td>
<td>+country</td>
<td>+venue</td>
</tr>
<tr>
<td></td>
<td>+Image</td>
<td>+runs scored</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+wickets taken</td>
</tr>
</tbody>
</table>

- <CF> ODI_PLAYER
- <CF> PLAYER BY COUNTRY
- <CF> MATCHES PLAYED
Lets see the application !!
References


ii) Datastax documentation

iii) Cassandra - A Decentralized Structured Storage System
     Avinash Lakshman, Prashant Malik
Thanks
&
Questions!!