Overview

- Background
- Data Model
- API
- Architecture
- Users
- Linearly scalability
- Replication and Consistency
- Tradeoff
Background

- Cassandra is a highly scalable, eventually consistent, distributed, structured key-value store.
- Cassandra was open sourced by Facebook in 2008, and it was designed to fulfill the storage needs of the Inbox Search problem. It is in production use at Facebook but is still under heavy development.
Background

• Cassandra is Dynamo and Bigtable’s lovechild.

• Like Dynamo, Cassandra is eventually consistent; Like BigTable, Cassandra provides a ColumnFamily-based data model.
Data Model

• Basic concepts:
  – Cluster: the machines(nodes) in a logical Cassandra instance. Cluster can contain multiple keyspaces.
  – Keyspace: a namespace for ColumnFamilies, typically one per application.
  – ColumnFamilies: contain multiple columns, each of which has a name, value, and a time stamp, and which are referenced by row keys.
  – SuperColumns: can be thought of as columns that themselves have sub columns.
Data Model

• Columns
  – The column is lowest/smallest increment of data. It is a tuple(triplet) that contains a name, a value and a timestamp.
  – Example in Java:
Data Model

• Super Column
  – A container for one or more columns

```java
public class SuperColumn {
    Byte[] name;
    // The key is equal to the name of the Column
    Map<Byte[] /* key */, Column> value = null;
}

SuperColumn sc = new SuperColumn();
sc.name = "person1";
sc.put("firstname", new Column("firstname", "Ronald");
sc.put("familyname", new Column("familyname", "Mathies");
```
Data Model

• Column Families (CF)
  – A container for columns, analogous to table in a relational database.
  – The columnFamily has a name, a map with a key and a value (which is a map containing columns).
Data Model

• Column Families (CF)

```java
public class ColumnFamily {
    Byte[] name;

    // The key is a user generated key
    Map<Byte[] /* key */, Column> value = null;
}
```
# Data Model

```java
class ColumnFamily {
    Byte[] name;

    // The key is a user generated key
    Map<Byte[] /* key */, Column> value = null;
}
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Eric Long&quot;</td>
<td>Columns</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>&quot;email&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;United Kingdom&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;registeredSince&quot;</td>
</tr>
<tr>
<td>&quot;John Steward&quot;</td>
<td>Columns</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>&quot;email&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;country&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;registeredSince&quot;</td>
</tr>
<tr>
<td>&quot;Ronald Mathies&quot;</td>
<td>Columns</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>&quot;email&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;country&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;registeredSince&quot;</td>
</tr>
</tbody>
</table>
Data Model

- **SuperColumnFamily**
  - The largest container, instead of having **Columns** in the inner most Map, we have **SuperColumns**. So it just adds an extra dimension.
Data Model

• Keyspaces
  – The container for column families. From an RDBMS point of view you can compare this to the schema, normally you have one per application.
The Cassandra API consists of the following three methods:

- `insert(table, key, rowMutation)`
- `get(table, key, columnName)`
- `delete(table, key, columnName)`

`columnName` can refer to a specific column within a column family, a column family, a super column family or a column within a super column.
API

• Thrift
  – Cassandra driver-level interface that the clients below build on. NOT recommend...

• High level clients:
  – Python(Telephus, Pycassa...)
  – Java(Hector, Pelops...)
  – .NET(FluentCassandra, Aquiles...)
  – PHP(phpcassa, SimpleCassie...)
  – Others...
## Architecture

### Architecture layers

<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</td>
<td>Memtable</td>
<td>Hinted handoff</td>
</tr>
<tr>
<td>Failure detection</td>
<td>SSTable</td>
<td>Read repair</td>
</tr>
<tr>
<td>Cluster state</td>
<td>Indexes</td>
<td>Bootstrap</td>
</tr>
<tr>
<td>Partitioner</td>
<td>Compaction</td>
<td>Monitoring</td>
</tr>
<tr>
<td>Replication</td>
<td></td>
<td>Admin tools</td>
</tr>
</tbody>
</table>
Architecture

• Write Path
  – First write to a disk commit log (sequential)
  – After write to log it is sent to appropriate nodes
  – Each node receiving write first records it in a local log, then makes update to memtables.
  – Memtables are flushed to disk when
    • Out of space
    • Too many keys (128 is default)
    • Time duration (Client provided)
Architecture

• When memtables written out two files go out:
  – DataFile(\textbf{SSTable})
  – Index File(\textbf{SSTable Index})

• When a commit log has had all its column families pushed to disk, it is deleted

• \textbf{Compaction}: Data files accumulate over time. Periodically data files are merged sorted into a new file(and creates new index).
Architecture

• Write properties:
  – No reads
  – No seeks
  – Fast
  – Atomic within ColumnFamily
  – Always writable

• Read properties:
  – Read multiple SSTables
  – Slower than writes (but still fast)
  – Seeks can be mitigated with more RAM
  – Scales to billions of rows
Users

- **Facebook**
  - Uses Cassandra to power Inbox Search, with over 200 nodes deployed. Abandoned in late 2010.
- **Twitter**
  - But not for tweets.
- **IBM**
  - Research in building a scalable email system based on Cassandra
- **Cisco’s WebEx**
  - Uses Cassandra to store user feed and activity in near real time.
Next Topics

1. Linearly scalability
2. Replication and Consistency
3. Tradeoff
Linearly Scalability

Key

N1

N2

N3

Nx
Consistent Hashing

Cause a problem...
Load Balance

N4 ➔ N3 ➔ N2 ➔ N1
Replication and Consistency

Replication
Tunable Eventually consistency
Replication (Simple Case)
## Tunable Consistency

<table>
<thead>
<tr>
<th>Write(W)</th>
<th>Read(R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>Description</td>
</tr>
<tr>
<td>ZERO</td>
<td>Cross fingers</td>
</tr>
<tr>
<td>ANY</td>
<td>1&lt;sup&gt;st&lt;/sup&gt; Response (Including HH)</td>
</tr>
<tr>
<td>One</td>
<td>1&lt;sup&gt;st&lt;/sup&gt; Response</td>
</tr>
<tr>
<td>QUORUM</td>
<td>N/2 + 1 Replicas</td>
</tr>
<tr>
<td>ALL</td>
<td>All Replicas</td>
</tr>
</tbody>
</table>
A Quorum Level Example(1)

N=3

Write Operation

N1

N2

N3
A Quorum Level Example

N=3

Read Operation

N1

N2

N3
A Quorum Level Example(3)

• But...

Tradeoff!
Final Question about Cassandra

Why write/read fast?
(1) No read/write locks
(2) Organize all the write operations into a sequential write which can maximize the disk’s throughput
(3) Flexible Data Model
Similiarity with Dynamo and Bigtable

Dynamo-like features
a. Symmetric, P2P architecture
   No Special nodes, No SPOF (Single Point Of Failure)
b. Gossip Based cluster management
c. Distributed hash table for data placement (DHT)
d. Tunable and Eventual Consistency

BigTable-like Features
a. Data Model
b. SSTable Disk Storage
   Append-only Commit Log
   MemTable (Buffer & Sort)
   Immutable SSTable Files
c. Hadoop Integration (Some ideas Based on GFS)
Thanks!