ADVANCED DATABASES CIS 6930
Dr. Markus Schneider

Group 2
Archana Nagarajan, Krishna Ramesh, Raghav Ravishankar, Satish Parasaram
Drawbacks of RDBMS

- Vertical Scaling.
- ACID doesn’t always hold good for Big Data.
- Sharding becomes very difficult.
- High Availability is complicated.

Master 🔄 Replication Lag 🔄 Slave

MySQL
Why No SQL?

- Non-Relational.
- Mostly Open Source.
- Easy Scalability and High Efficiency.
- No need to develop a detailed database model.
- Stores large volumes of data.
- Capable of performing agile operations.
- Object oriented programming.
Definition of Cassandra

Cassandra is a

- Distributed
- High Performance
- Extremely scalable
- Fault tolerant
- Post relational database solution
● Facebook released Cassandra as an open-source project on Google Code in July 2008.
● In March 2009 it became an Apache Incubator project.
● On February 17, 2010 it graduated to a top-level project.
How Big is Cassandra Today?
Key Features of Cassandra

- Distributed and Decentralized.
- Linear Scalability.
- Fault Tolerance.
- Handles huge amounts of data.
- Horizontal Scaling.
- Replication.
- Extremely Fast.
Distributed Architecture: Node

- Reads and Writes Data.
- Data Storage.
- Has an array of commands.
Distributed Architecture: Ring

- Clustering System.
- Token Ranges: $-2^{63}$ to $2^{63}-1$.
- Partitioner.
Distributed Architecture: Ring

- Co-ordinator.
- Four States: Joining, Leaving, Up and Down.
- Peer to Peer.
- V nodes.
Distributed Architecture: Gossip

Information is thus, sent to all the nodes in the cluster.
Distributed Architecture: Gossip

Node 1

**Endpoint state**
- Generation = 5
- Version = 22

**Application State:**
- Status = Normal
- Dc = dc - west
- Rack = rack1
- Schema = c2a2b
- Load = 100.0
- Severity = 0.75
Distributed Architecture: Gossip

Node 1

EP: 127.0.0.1
HB: 100:20
LOAD: 86

EP: 127.0.0.2
HB: 175:40
LOAD: 68

Node 2

EP: 127.0.0.1
HB: 100:15
LOAD: 86

EP: 127.0.0.2
HB: 175:50
LOAD: 68

Digest

SYN
End:Gen:Ver
127.0.0.1 : 100 : 20
127.0.0.2 : 175 : 40

ACK
End : Gen : Ver
127.0.0.1 : 100:15
127.0.0.2 : 175:50
Distributed Architecture: Snitch

- Determines each node's rack and data center.
- The topology of the cluster.
- Configured in Cassandra.yaml.

There are mainly 2 groups of snitches. They are as follows:

1) Cloud Based Snitches.
2) Regular Snitches.
Characteristic Features of Cassandra
Replication Factor

- Similar to MySQL Sharding
- Point of Failure, Data loss
- Node reboot, network failure, Power loss, natural calamities
- Update patches
- What about RF=2?
- Simple Strategy

Create KEYSPACE socialdata
With REPLICATION = {
‘class’ : ‘SimpleStrategy’,
‘replication_factor’ : ‘1’
}
Replication Factor

- Data replicated to 3 nearby nodes
- Node can be down
- Is RF=3 better than RF=2?
- Odd values of RF better
- RF = 3 used in production
Client

Node 3, Node 5 are down
Coordinator is down
All nodes equally likely to be coordinator
New coordinator randomly selected
Data fetched from Node 4
Network Topology Strategy
Create KEYSPACE socialdata
With REPLICATION = {

- Can lose a node
- Can lose an entire DC and be online
- Remote Coordinator
- High Availability
- cassandra.yaml
Consistency Levels

- **CL = 1**
  Coordinator chooses the closest node to respond, acknowledge

- **CL = QUORUM**
  51% of replicas to respond back
  RF = 2, RF = 3
  49% of nodes can be down

- **CL = ALL, Strong consistency**
  If any node down, no data

- **Digest, Checksum**

Consistency Levels in Writes and Reads

- **CL = ALL WRITE**, **CL = ONE READ**

- **RF=3, CL = WRITE QUORUM, READ QUORUM, ATLEAST ONE OVERLAP**

- **RF=3, CL=QUORUM ACROSS DCs leads to latency**
Consistency Levels across data centers

Create KEYSPACE socialdata
With REPLICATION = {
    ‘DC-West’ : ‘2’ }

CL = LOCAL_QUORUM, QUORUM = latency of 100ms for response
During Inserts, Updates, Deletes, RF=3
- Node 2 is down
- Inconsistency handled
- Hints are stored on the Coordinator, Node 5
- Node 2 is back up and Resyncs its data
- cassandra.yaml hinted_handoff default set to 3 hours
Read Repair

RF=3, CL = ALL

- Select operation
- Digest and checksum dint match
- Request for timestamp
- Update nodes with latest value based on timestamp
Cassandra Write Path

Incoming Insert data

Memory

Disk

Memtable - Partition Key 58

UID5121  Charles  TX  1473323770
UID1348  Robert  TX  1473323772
UID3451  Ryan    WI  1473323778
UID9632  Michael FL  1473323775

SSTable – Sorted String Table

ID1    UID5121  Charles  TX  1473323770
ID2    UID1348  Robert  TX  1473323772
ID3    UID3451  Ryan    WI  1473323778
ID4    UID9632  Michael FL  1473323775

Commit Log

Writes to disk
### Cassandra Write Path

- **Memtable – Partition Key 58**

<table>
<thead>
<tr>
<th>UID</th>
<th>Name</th>
<th>State</th>
<th>State ID</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>UID2101</td>
<td>Kevin</td>
<td>FL</td>
<td></td>
<td>1473323780</td>
</tr>
<tr>
<td>UID2109</td>
<td>Richard</td>
<td>FL</td>
<td></td>
<td>1473323782</td>
</tr>
<tr>
<td>UID3451</td>
<td>Ryan</td>
<td>NY</td>
<td></td>
<td>1473323788</td>
</tr>
<tr>
<td>UID2191</td>
<td>Steven</td>
<td>CA</td>
<td></td>
<td>1473323785</td>
</tr>
</tbody>
</table>

### Memory

### Disk

**SSTable 1**

<table>
<thead>
<tr>
<th>ID</th>
<th>UID</th>
<th>Name</th>
<th>State</th>
<th>State ID</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID1</td>
<td>UID5121</td>
<td>Charles</td>
<td>TX</td>
<td></td>
<td>1473323770</td>
</tr>
<tr>
<td>ID2</td>
<td>UID1348</td>
<td>Robert</td>
<td>TX</td>
<td></td>
<td>1473323772</td>
</tr>
<tr>
<td>ID3</td>
<td>UID3451</td>
<td>Ryan</td>
<td>WI</td>
<td></td>
<td>1473323778</td>
</tr>
<tr>
<td>ID4</td>
<td>UID9632</td>
<td>Michael</td>
<td>FL</td>
<td></td>
<td>1473323775</td>
</tr>
</tbody>
</table>

**SSTable 2**

<table>
<thead>
<tr>
<th>ID</th>
<th>UID</th>
<th>Name</th>
<th>State</th>
<th>State ID</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID5</td>
<td>UID2101</td>
<td>Kevin</td>
<td>FL</td>
<td></td>
<td>1473323780</td>
</tr>
<tr>
<td>ID6</td>
<td>UID2109</td>
<td>Richard</td>
<td>FL</td>
<td></td>
<td>1473323782</td>
</tr>
<tr>
<td>ID7</td>
<td>UID3451</td>
<td>Ryan</td>
<td>NY</td>
<td></td>
<td>1473323788</td>
</tr>
<tr>
<td>ID8</td>
<td>UID2191</td>
<td>Steven</td>
<td>CA</td>
<td></td>
<td>1473323785</td>
</tr>
</tbody>
</table>

flushed
Cassandra Read Path - SSTable

Reference: http://www.datastax.com
Cassandra Read Path - SSTable

Key Cache

<table>
<thead>
<tr>
<th>Token</th>
<th>Byte Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>6,224</td>
</tr>
</tbody>
</table>

RAM

<table>
<thead>
<tr>
<th>Tokens</th>
<th>Byte Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 20</td>
<td>0</td>
</tr>
<tr>
<td>21 - 55</td>
<td>32</td>
</tr>
<tr>
<td>56 - 100</td>
<td>48</td>
</tr>
</tbody>
</table>

HDD

<table>
<thead>
<tr>
<th>Token</th>
<th>Byte Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>1,120</td>
</tr>
<tr>
<td>18</td>
<td>3,528</td>
</tr>
<tr>
<td>21</td>
<td>4,392</td>
</tr>
<tr>
<td>36</td>
<td>6,224</td>
</tr>
<tr>
<td>58</td>
<td>7,192</td>
</tr>
<tr>
<td>92</td>
<td>9,120</td>
</tr>
</tbody>
</table>

Reference: http://www.datastax.com
Cassandra Read Path - SSTable

- Determines probabilistically if a value is not in a SSTable
- Gives false positives but zero false negatives
- Eliminates need to search across multiple SSTables
Compaction

SSTable 1

1  Johnny (92)
2  Betsy (49)
3  Nicholi (85)
4  Sue (41)
5  Sam (96)

SSTable 2

1  Johnny (181)
2  (X) (176)
3  Norman (148)
5  (X) (184)
6  Henrie (134)

SSTable 3

Reference: http://www.datastax.com
Compaction

SSTable 1
1  Johnny (92)
2  Betsy (49)
3  Nicholi (85)
4  Sue (41)
5  Sam (96)

SSTable 2
1  Johnny (181)
2  (X) (176)
3  Norman (148)
5  (X) (184)
6  Henrie (134)

SSTable 3
1  Johnny (181)
3  Norman (148)
4  Sue (41)
5  (X) (184)
6  Henrie (134)

Reference: http://www.datastax.com
Compaction Strategy

- min_sstable_size 50Mb
- min_threshold 4 – Minimum number of SSTables required for compaction
- max_threshold 32 – Maximum number of SSTables allowed for compaction
- tombstone_compaction_interval – 86400secs
CAP Theorem

- Availability
- Consistency
- Partition Tolerance

MySQL
Cassandra
Data Model
Column Oriented DBs

- The storage of data is column value wise
- Column values are mapped back to the row-keys
Column Families

- Resembles a table in RDBMS
- Each column family can have more than one column
- Number of columns can vary for different rows
Column Families

- Multi-versioned
- One column family can have variable no of columns
- Cells within a Column family are sorted physically
- Very Sparse, most cell has NULL value

Column Family: User

<table>
<thead>
<tr>
<th>rowid</th>
<th>Col_name</th>
<th>ts</th>
<th>Col_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>name</td>
<td>v1</td>
<td>Ricky</td>
</tr>
<tr>
<td>u1</td>
<td>email</td>
<td>v1</td>
<td><a href="mailto:ricky@gmail.com">ricky@gmail.com</a></td>
</tr>
<tr>
<td>u1</td>
<td>email</td>
<td>v2</td>
<td><a href="mailto:ricky@ya.com">ricky@ya.com</a></td>
</tr>
<tr>
<td>u2</td>
<td>name</td>
<td>v1</td>
<td>Sam</td>
</tr>
<tr>
<td>u2</td>
<td>phone</td>
<td>v1</td>
<td>650-3456</td>
</tr>
</tbody>
</table>

Column Family: Social

- One File per Column Family
- Data inside file is physically sorted
- Sparse: NULL cell does not materialize

<table>
<thead>
<tr>
<th>rowid</th>
<th>Col_name</th>
<th>ts</th>
<th>Col_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>friend</td>
<td>v1</td>
<td>u10</td>
</tr>
<tr>
<td>u1</td>
<td>friend</td>
<td>v1</td>
<td>u13</td>
</tr>
<tr>
<td>u2</td>
<td>friend</td>
<td>v1</td>
<td>u10</td>
</tr>
<tr>
<td>u2</td>
<td>classmate</td>
<td>v1</td>
<td>u15</td>
</tr>
</tbody>
</table>
Data Model in Cassandra

- Hybrid between key-value store and column oriented databases
- Column family - analog of a RDBMS table
- Row - identified uniquely by a key, has values as columns, all rows need not have same number of columns
Data Model in Cassandra (..contd)

• Keyspace - analog of a RDBMS schema, outermost container of data.
• Number of column families in Keyspace is fixed.
• Most basic attributes of a Keyspace are - Replication factor, Replica placement strategy
Data Distribution

• Rows are partitioned through a partition key which is the first component of a primary key

• Two ways to partition:
  • Random
  • Ordered
CQL (Cassandra Query Language)

- Way to interact with Cassandra
- Syntax is very similar to that of SQL, but far less capable
- No joins, no subqueries
Queries

• Create Keyspace - CREATE KEYSPACE "users" WITH replication = {
  'class': 'Strategy name',
  'replication_factor': 'No. Of replicas'};

• Consistency - CONSISTENCY QUORUM

• Capture - CAPTURE 'dest_file.txt'

• Source - SOURCE 'myfile.txt'

• Copy - COPY airplanes (name, mach, year, manufacturer) TO 'temp.csv'
Monitoring Cassandra Cluster

- Java Management Extension (JMX) can be used to monitor Cassandra cluster
- Several JMX compliant tools are available
Applications
When to use Cassandra

- Highly scalable.
- Reliable cross-datacenter replication
- Excellent choice for real-time Analytics workload. Faster write operations
- Higher insertion rates
- Can be integrated with Hadoop, Hive and Apache Spark for Batch Processing
- Tunable Consistency and CAP parameters.
Throughput comparison

Throughput for workload Read/Write

Throughput for workload Read/Scan/Write
Tradeoffs

Read latency for workload Read/Write

Write latency for workload Read/Write
Drawbacks of Cassandra

- Transactions are not supported (ACID or otherwise)
- Eventual consistency isn’t sufficient always. Eg: Trading stocks.
- No support for ad-hoc queries
- Cannot perform complex queries
Connecting applications to Cassandra : Drivers

- Drivers help connect the applications to Cassandra database

- Driver languages :
  - Python
  - Java
  - C
  - C++
  - Ruby
  - And many more.


- API’s are similar for all the languages
Setup

- Create a cluster object
- Use the cluster to obtain a session
- Session manages all the connections to the cluster

```python
#connect to the cluster and the keyspace “sample”
from cassandra.cluster import Cluster
cluster = Cluster()
session = cluster.connect('sample')
```

Session object listens to the changes in the cluster and the driver reacts to the same
NetFlix - Challenges faced:

- Single datacenter meant single point of failure
- Users grew exponentially
- And every 2 weeks, the site was down for maintenance
What was required?

- More reliable and fault tolerant data storage
- High availability of member information, streaming quality video data in a more robust fashion
- Flexibility of streaming the video data from multiple devices
What Cassandra offered?

- Created better business agility for Netflix.
- No downtime as there are no schemas.
- No fear of data loss because replication means no single point of failure.
- Open-source model provided Netflix the flexibility to implement their own backup, recovery system and replication strategy.
Instagram : Shift from Redis to Cassandra

- Memory limitations!!
- Cut the costs to the point where they were paying around a quarter of what they were paying before.
- Primarily used for fraud detection, newsfeed and inbox
Facebook: Why Cassandra

- Cross datacenter replication
- Designed to address the storage needs of inbox search problem.
- Provided high write throughput
- Exploited the timestamp property provided by Cassandra
Facebook : Shift from Cassandra to HBase

- Eventual consistency model not suitable for the new messenger.
- Hbase – simpler consistency model
- High scalability and performance, auto load balancing.
- Hadoop – widely used by Facebook and HDFS being the distributed file system for both Hadoop and Hbase.