Cassandra: Beyond Bigtable

Jonathan Ellis  CTO, DataStax
Five years of Cassandra

0.1
Jul-08

0.3
Jul-09

0.6
May-10

0.7
Feb-11

1.0
Dec-11

1.2
Oct-12

2.0
Jul-13

DSE
Bigtable + Dynamo
Bigtable + Dynamo

• LSMT / SSTables
Bigtable + Dynamo

• LSMT / SSTables
  • Runtime “column” (cell) definition
Bigtable + Dynamo

• LSMT / SSTables
  • Runtime “column” (cell) definition
  • Schema-agnostic
Bigtable + Dynamo

• LSMT / SSTables
  • Runtime “column” (cell) definition
  • Schema-agnostic
  • Size-tiered compaction
Bigtable + Dynamo

• LSMT / SSTables
  • Runtime “column” (cell) definition
  • Schema-agnostic
  • Size-tiered compaction
• Gossip-based cluster status + failure detection
Bigtable + Dynamo

- LSMT / SSTables
  - Runtime “column” (cell) definition
  - Schema-agnostic
  - Size-tiered compaction
- Gossip-based cluster status + failure detection
- Hinted handoff
Bigtable + Dynamo

• LSMT / SSTables
  • Runtime “column” (cell) definition
  • Schema-agnostic
  • Size-tiered compaction
• Gossip-based cluster status + failure detection
• Hinted handoff
• Read repair
Bigtable + Dynamo

• LSMT / SSTables
  • Runtime “column” (cell) definition
  • Schema-agnostic
  • Size-tiered compaction
• Gossip-based cluster status + failure detection
• Hinted handoff
• Read repair
• Anti-entropy repair
Bigtable + Dynamo

• LSMT / SSTables
  • Runtime “column” (cell) definition
  • Schema-agnostic
  • Size-tiered compaction
• Gossip-based cluster status + failure detection
• Hinted handoff
• Read repair
• Anti-entropy repair
• Eventually consistent
... with some differences
... with some differences

• SuperColumns
... with some differences

- SuperColumns
- Indexes
... with some differences

• SuperColumns
• Indexes
• Timestamp-based conflict resolution
  [link](http://www.datastax.com/dev/blog/why-cassandra-doesnt-need-vector-clocks)
Bigtable-inspired API

```cpp
list<ColumnOrSuperColumn> get_slice(
    1:required binary key,
    2:required ColumnParent column_parent,
    3:required SlicePredicate predicate,
    4:required ConsistencyLevel consistency_level)
```
Two years ago

- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use
Two years ago

- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use
Two years ago

- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use
Two years ago

- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use
Two years ago

- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use
Two years ago

- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use
Two years ago

- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use
User defined types

CREATE TYPE address (  
    street text,  
    city text,  
    zip_code int,  
    phones set<text>
);

CREATE TABLE users (  
    id uuid PRIMARY KEY,  
    name text,  
    addresses map<text, address>
);

SELECT id, name, addresses.city, addresses.phones FROM users;

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>addresses.city</th>
<th>addresses.phones</th>
</tr>
</thead>
<tbody>
<tr>
<td>63bf691f</td>
<td>jbellis</td>
<td>Austin</td>
<td>{'512-4567', '512-9999'}</td>
</tr>
</tbody>
</table>
CREATE TABLE songs (  id uuid PRIMARY KEY,  artist text,  album text,  title text,  data blob,  tags set<text>  );

CREATE INDEX song_tags_idx ON songs(tags);

SELECT * FROM songs WHERE 'blues' IN tags;

<table>
<thead>
<tr>
<th>id</th>
<th>album</th>
<th>artist</th>
<th>tags</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>5027b27e</td>
<td>Country Blues</td>
<td>Lightnin' Hopkins</td>
<td>{'acoustic', 'blues'}</td>
<td>Worrying My Mind</td>
</tr>
</tbody>
</table>
Cassandra is a...

• Partitioned row store with extensions
• Typed document database
• Object database
Paxos / CAS

Session 1

SELECT * FROM users
WHERE username = 'jbellis'

[empty resultset]

INSERT INTO users (...)
VALUES ('jbellis', ...)

Session 2

SELECT * FROM users
WHERE username = 'jbellis'

[empty resultset]

INSERT INTO users (...)
VALUES ('jbellis', ...)
Prepare / promise
Propose / accept
Read / results

- Leader
- Replica
- Leader
- Replica
- Leader
- Replica
- Leader
- Replica
Commit / acknowledge
CREATE TABLE paxos (  
  row_key blob,  
  cf_id UUID,  
  in_progress_ballot timeuuid,  
  proposal_ballot timeuuid,  
  proposal blob,  
  most_recent_commit_at timeuuid,  
  most_recent_commit blob,  
  PRIMARY KEY (row_key, cf_id)
)
Implications

• 4 round trips vs 1 for normal updates
• Paxos state is durable
• Linearizable consistency with no leader election or failover
• `ConsistencyLevel.SERIAL`
INSERT INTO USERS (username, email, ...) VALUES (‘jbellis’, ‘jbellis@datastax.com’, ... ) IF NOT EXISTS;

UPDATE USERS
SET email = 'jonathan@datastax.com', ...
WHERE username = 'jbellis'
IF email = 'jbellis@datastax.com';
Triggers

CREATE TRIGGER <name> ON <table>
USING <classname>;
Trigger implementation

class MyTrigger implements ITrigger
{
    public Collection<RowMutation> augment (ByteBuffer key, ColumnFamily update)
    {
        ...
    }
}
Atomicity?
Batches

Coordinator Node

Batchlog Node

Red Replica

Yellow Replica

Blue Replica
Batches

Coordinator Node

Batchlog Node

Red Replica

Yellow Replica

Blue Replica
Batches

Coordinator Node

Batchlog Node

Red Replica

Yellow Replica

Blue Replica
Batches

- Coordinator Node
- Batchlog Node
- Red Replica
- Yellow Replica
- Blue Replica
Batches

Coordinator Node

Batchlog Node

Red Replica

Yellow Replica

Blue Replica
Batches

Coordinator Node

Batchlog Node

Red Replica

Yellow Replica

Blue Replica
Batches

Coordinator Node

Batchlog Node

Red Replica

Yellow Replica

Blue Replica
Batches

Coordinator

Batchlog Node

Red Replica

Yellow Replica

Blue Replica
Batches

Coordinator

Batchlog Node

Red Replica

Yellow Replica

Blue Replica
Batches

Coordinator

Batchlog Node

Red Replica

Yellow Replica

Blue Replica
On-Heap/Off-Heap

Java Process

On-Heap Managed by GC

Off-Heap Not managed by GC
Read path (per sstable)

Bloom filter

Memory

Disk
Read path (per sstable)

Memory

Bloom filter

Disk

Partition key cache
Read path (per sstable)

Bloom filter

Partition summary

Partition key cache

Memory

Disk
Read path (per sstable)

Memory

Disk

Partition index

Partition summary

Bloom filter

Partition key cache

Partition
Read path (per sstable)

0x...

Bloom filter

Partition summary

Partition key cache

Partition index

Compression offsets

Memory

Disk
Read path (per sstable)

Memory → Compression offsets → Partition index → Data

Partition summary → Bloom filter

Disk → Partition key cache
Off heap in 2.0

Partition key bloom filter
1-2GB per billion partitions

Compression offsets
Partition summary
Partition key cache

Memory
Disk
Data
Partition index

Bloom filter
Off heap in 2.0

Compression metadata
~1-3GB per TB compressed
Off heap in 2.0

Partition index summary
(depending on rows per partition)
Compaction

• Size-tiered
• Leveled
• Others?
Size-tiered compaction
Leveled compaction
Sad leveled compaction
STCS in L0
HLL and compaction
HLL and compaction
HLL and compaction
Data-aware compaction?

• Append-only workloads
  • No compaction necessary in trivial case; still needed for clustered scans

• Append-mostly workloads?
  • Bounded window for out-of-order updates
Rapid Read Protection

C* eager retry, 1 node killed after 450s - stress-read

Operations/Second vs. Elapsed time in seconds
Typical reads

- 90% busy
- 30% busy
- 40% busy

Client

Coordinator
Typical reads

Client -> Coordinator

90% busy
30% busy
40% busy
Typical reads

- Client: 40% busy
- Coordinator: 90% busy
- 30% busy
- 40% busy
Typical reads

Client

Coordinator

90% busy

30% busy

40% busy
A failure

Client

Coordinator

90% busy

30% busy

40% busy
A failure

- Client
- Coordinator
- 90% busy
- 30% busy
- 40% busy
A failure

Client

Coordinator

90% busy

30% busy

40% busy
A failure

Client

Coordinator

90% busy

40% busy

X
A failure

Client

Coordinator

90% busy

40% busy

timeout
Failure with read protection

Client

Coordinator

90% busy

30% busy

40% busy
Failure with read protection

Client

Coordinator

90% busy

30% busy

40% busy
Failure with read protection

Client

Coordinator

90% busy

30% busy

40% busy
Failure with read protection
Failure with read protection

- Client
- Coordinator
- 90% busy
- 40% busy

Red X indicates failure.
Failure with read protection

Client

Coordinator

90% busy

40% busy

X
Failure with read protection

- Client
- Coordinator
- 90% busy
- 40% busy

Success
# Latency (mid-compaction)

| Speculative Retry | Averages from the middle 80% of values: | Total operation time | cmd: |-n 60000000 -o read -i 5 -K 20 |
|-------------------|----------------------------------------|----------------------|-------------------|
| NONE              | interval_op_rate: 39156               | 00:26:10             |                   |
|                   | interval_key_rate: 39156              |                      |                   |
|                   | latency median: 0.8                   |                      |                   |
|                   | latency 95th percentile: 2.6          |                      |                   |
|                   | latency 99.9th percentile: 48.8       |                      |                   |
| ALWAYS            | interval_op_rate: 34823               | 00:29:11             |                   |
|                   | interval_key_rate: 34823              |                      |                   |
|                   | latency median: 0.8                   |                      |                   |
|                   | latency 95th percentile: 2.6          |                      |                   |
|                   | latency 99.9th percentile: 36.5       |                      |                   |
| 75percentile      | interval_op_rate: 36764               | 00:27:41             |                   |
|                   | interval_key_rate: 36764              |                      |                   |
|                   | latency median: 0.8                   |                      |                   |
|                   | latency 95th percentile: 3.0          |                      |                   |
|                   | latency 99.9th percentile: 16.8       |                      |                   |
| 99percentile      | interval_op_rate: 40466               | 00:25:28             |                   |
|                   | interval_key_rate: 40466              |                      |                   |
|                   | latency median: 0.8                   |                      |                   |
|                   | latency 95th percentile: 2.7          |                      |                   |
|                   | latency 99.9th percentile: 19.6       |                      |                   |
More-efficient repair
More-efficient repair
More-efficient repair
More-efficient repair
More-efficient repair
More-efficient repair
More-efficient repair
More-efficient repair
More-efficient repair
DELETE FROM users
WHERE username = 'jbellis'
When can we purge?

• gc_grace_seconds
Pain points
Pain points

• Easy to write a query that is $O(N)$ in the number of tombstones
  • Tombstones must be read-repaired
Pain points

• Easy to write a query that is O(N) in the number of tombstones
  • Tombstones must be read-repaired

• Clumsy hammer
  • tombstone_warn_threshold: 1000
  • tombstone_failure_threshold: 100000
Pain points

• Easy to write a query that is $O(N)$ in the number of tombstones
  • Tombstones must be read-repaired

• Clumsy hammer
  • tombstone_warn_threshold: 1000
  • tombstone_failure_threshold: 100000

• http://www.datastax.com/dev/blog/cassandra-anti-patterns-queues-and-queue-like-datasets