Apache Cassandra

From POC to PROD: Twelve sprints of Cassandra

Christophe Schmitz
Director of Consulting, Europe
c christophe@instaclustr.com
Agenda

• Sprint 1 Data modeling
• Sprint 2 Cluster sizing
• Sprint 3 Stress test
• Sprint 4 Data modeling (revisit)
• Sprint 5 Analytics
• Sprint 6 Security
• **Sprint 7 Go Live**
• Sprint 8 Monitoring
• Sprint 9 Hello Compaction and Repair
• Sprint 10 Large partition and tombstone
• Sprint 11 Cluster expansion
Sprint 0: Choosing Cassandra
Why people choose Cassandra?

- High availability
- Linear scalability
- Low latency
When is Cassandra a good choice?

Schema relatively stable

Access pattern predictable

Update / delete infrequent

IOT / Timeseries / Transactions are good example
Sprint 1: Data Modeling
Quick introduction to Cassandra

Assumes:
Replication Factor=3 in both DCs / Consistency Level = Local Quorum

Sensor Id, Date, Timestamp, metrics1, ..

Partition Key 1
Partition Key 2
Partition Key 3

Synchronous Write
Asynchronous Write
Partitioning

<table>
<thead>
<tr>
<th>Sensor #</th>
<th>Date</th>
<th>Timestamp</th>
<th>Metric1</th>
<th>Metric2</th>
<th>Metric3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2015-01-01</td>
<td>20150101-000000</td>
<td>5.01</td>
<td>5.67</td>
<td>0.678</td>
</tr>
<tr>
<td>1</td>
<td>2015-01-01</td>
<td>20150101-000010</td>
<td>5.01</td>
<td>5.67</td>
<td>0.678</td>
</tr>
<tr>
<td>1</td>
<td>2015-01-01</td>
<td>20150101-000020</td>
<td>5.05</td>
<td>5.8</td>
<td>0.678</td>
</tr>
<tr>
<td>1</td>
<td>2015-01-02</td>
<td>20150101-000000</td>
<td>5.01</td>
<td>5.67</td>
<td>0.678</td>
</tr>
<tr>
<td>1</td>
<td>2015-01-02</td>
<td>20150101-000010</td>
<td>5.01</td>
<td>5.67</td>
<td>0.678</td>
</tr>
<tr>
<td>1</td>
<td>2015-01-02</td>
<td>20150101-000020</td>
<td>5.05</td>
<td>5.8</td>
<td>0.678</td>
</tr>
<tr>
<td>2</td>
<td>2015-01-02</td>
<td>20150101-000000</td>
<td>6.01</td>
<td>7.67</td>
<td>0.978</td>
</tr>
<tr>
<td>2</td>
<td>2015-01-02</td>
<td>20150101-000010</td>
<td>6.01</td>
<td>7.67</td>
<td>0.698</td>
</tr>
<tr>
<td>2</td>
<td>2015-01-02</td>
<td>20150101-000020</td>
<td>6.05</td>
<td>8.8</td>
<td>0.679</td>
</tr>
</tbody>
</table>

**Partition Key**: Sensor, Date

**Clustering Key**: Timestamp

**Primary Key**: ((Sensor, Date), Timestamp)
Design Approach

• Phase 1: Understand the data
  • Define the data domain: E-R logical model
  • Define the required access patterns: how will you select an update data?

• Phase 2: Denormalize based on access patterns
  • Identify primary access entities: driven by the access keys
  • Allocate secondary entities: denormalize by pushing up or down to the primary entities

• Phase 3: Review & tune
  • Review partition keys and clusters
    • Do partition keys have sufficient cardinality?
    • Is the number of records in each partition bounded?
    • Does the design consider delete and update impact?
  • Test & tune: check updates, review compaction strategy
Sprint 2: Cluster Sizing
Cluster sizing: consideration

- Disk usage (RF / MV)
- Read / Write throughput driven
- Availability / Consistency desired
Sprint 3: Stress test
Testing Cassandra applications

- Long running tests with background load are vital
  - Can run extremely high write loads for an hour or two but might take days to catch up on compactions
  - Don’t forget repairs
- Make sure your data volumes on disk are representative as well as read/write ops - cache hit rates can make a big difference to performance
- Mirror production data demographics as closely as possible (eg partition size)
- Don’t forget to include update/delete workload if applicable
- For core cassandra features, can test on reduce size and rely on scale-up but beware:
  - Secondary indexes
  - MVs
  - LWTs
Testing in practice

• Cassandra-test for synthetic testing
  • https://www.instaclustr.com/deep-diving-into-cassandra-stress-part-1

• JMeter for application level testing.
• JMeter / Cassandra plugin
Sprint 4: Data modeling - revisit
I want to query by a, or b, or c
   -> Materialized view

CREATE TABLE my_ks.my_tb (a int PRIMARY KEY, b int, c int, d int);

CREATE MATERIALIZED VIEW my_ks.my_tb_by_b AS
   SELECT a, b, c, d
   FROM my_ks.my_tb
   WHERE a IS NOT NULL AND b IS NOT NULL
   PRIMARY KEY (b, a);

I want to query by a, b, c, d, e, f, a+c, d+e ..... 
   -> Stratio Lucene
   -> DSE search (Solr)
   -> Elassandra (Cassandra + Elasticsearch)
Sprint 5: Analytics (and data migration)
Cassandra => fast => no join.
Spark on Cassandra

output.throughput_mb_per_sec
input.fetch.size_in_rows
input.reads_per_sec
Sprint 6: Security

“We’ve devised a new security encryption code. Each digit is printed upside down.”
Security

• At a minimum
  • Enable password auth
  • Enable client->server encryption (particularly if using public IPs to connect)
  • Enable internode encryption
  • Don’t use the default Cassandra user
  • Set up some read only user

• Best practice
  • Encrypt sensitive data at the client
    • Works well with typical C* access patterns where PK values are hashed anyway
    • Dates are the most common case of range selects and typically are not sensitive if other identifying data is encrypted
Sprint 7: Go Live!
Everything works!
Sprint 8: Monitoring and Alerting
Instaclustr console

JMX_exporter + prometheus + Grafana
## Monitoring Cassandra (Metrics + Alerting)

Items marked ** give an overall indication of cluster performance and availability.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Node Status</strong></td>
<td>Nodes DOWN should be investigated immediately.</td>
<td>Continuous, with alerting</td>
</tr>
<tr>
<td></td>
<td>org.apache.cassandra.net:type=FailureDetector</td>
<td></td>
</tr>
<tr>
<td><strong>Client read latency</strong></td>
<td>Latency per read query over your threshold</td>
<td>Continuous, with alerting</td>
</tr>
<tr>
<td></td>
<td>org.apache.cassandra.metrics:type=ClientRequest,scope=Read</td>
<td></td>
</tr>
<tr>
<td><strong>Client write latency</strong></td>
<td>Latency per write query over your threshold</td>
<td>Continuous, with alerting</td>
</tr>
<tr>
<td></td>
<td>org.apache.cassandra.metrics:type=ClientRequest,scope=Write</td>
<td></td>
</tr>
<tr>
<td>CF read latency</td>
<td>Local CF read latency per read, useful if some CF are particularly latency sensitive.</td>
<td>Continuous if required</td>
</tr>
<tr>
<td>Tombstones per read</td>
<td>A large number of tombstones per read indicates possible performance problems, and compactions not keeping up or may require tuning</td>
<td>Weekly checks</td>
</tr>
<tr>
<td>SSTables per read</td>
<td>High number (&gt;5) indicates data is spread across too many SSTables</td>
<td>Weekly checks</td>
</tr>
<tr>
<td><strong>Pending compactions</strong></td>
<td>Sustained pending compactions (&gt;20) indicates compactions are not keeping up. This will have a performance impact.</td>
<td>Continuous, with alerting</td>
</tr>
<tr>
<td></td>
<td>org.apache.cassandra.metrics:type=Compaction,name=PendingTasks</td>
<td></td>
</tr>
</tbody>
</table>
Sprint 9: Compaction and Repair
Compaction Intro

- Cassandra never updates sstable files once written to disk.
- Instead all inserts and updates are essentially (logically) written as transaction logs that are reconstituted when read.
- Compaction is the process of consolidating transaction logs to simplify reads.
- It’s an ongoing background process in Cassandra.
- Compaction ≠ Compression.
Repair Intro

- Reads every SSTable to be repaired
- Generates a merkle tree of data read.
- Send merkle tree to replicas, each replica compares each tree against every other.
- Any differences, Cassandra will stream the missing data
- Streamed data will be written as a new SSTable generating more compaction
Compaction and Repair

• Regular compactions are an integral part of any healthy Cassandra cluster.
• Repairs need to be run to ensure data consistency every gc_grace period.
• Can have a significant disk, memory (GC), cpu, IO overhead.
• Are often the cause of “unexplained” latency or IO issues in the cluster.
• Repair has a number of different strategies (sequential, parallel, incremental, sub range).
• Choose one that works best for you (likely to be either sub range or incremental).
Monitoring Compactions/Repair

- Monitor with nodetool compactionstats, tpstats & netstats

```bash
~ $ nodetool compactionstats -H
pending tasks: 518

<table>
<thead>
<tr>
<th>compaction type</th>
<th>keyspace</th>
<th>table</th>
<th>completed</th>
<th>total</th>
<th>unit</th>
<th>progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compaction</td>
<td>data</td>
<td>cf</td>
<td>18.71 MB</td>
<td>111.16 MB</td>
<td>bytes</td>
<td>16.83%</td>
</tr>
</tbody>
</table>

Active compaction remaining time: 0h00m05s
```

- A single node doing compactions can cause latency issues across the whole cluster, as it will become slow to respond to queries.

- Repair can often be the cause a large spike in compactions.
Sprint 10: Large partition and tombstone
Otherwise known as chasing 9’s
Partitioning: Diagnosing & Correcting

• Diagnosing
  • Overlarge partitions will also show up through long GC pauses and difficulty streaming data to new nodes
  • nodetool cfstats and nodetool cfhistograms provide partition size info. <10MB green, <100MB amber
  • Log file warnings - compacting large partition
  • Many issues can be identified from data model review

• Correcting
  • Correcting generally requires data model change although depending on the application, application level change may be possible
Examples of Large partitions

~ $ nodetool cfstats -H keyspace.columnfamily

... Compacted partition minimum bytes: 125 bytes
Compacted partition maximum bytes: 11.51 GB
Compacted partition mean bytes: 844 bytes

$ nodetool cfhistograms keyspace columnfamily

<table>
<thead>
<tr>
<th>Percentile</th>
<th>SSTables</th>
<th>Write Latency (micros)</th>
<th>Read Latency (micros)</th>
<th>Partition Size (bytes)</th>
<th>Cell Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>1.00</td>
<td>14.00</td>
<td>124.00</td>
<td>372</td>
<td>2</td>
</tr>
<tr>
<td>75%</td>
<td>1.00</td>
<td>14.00</td>
<td>1916.00</td>
<td>372</td>
<td>2</td>
</tr>
<tr>
<td>95%</td>
<td>3.00</td>
<td>24.00</td>
<td>17084.00</td>
<td>1597</td>
<td>12</td>
</tr>
<tr>
<td>98%</td>
<td>4.00</td>
<td>35.00</td>
<td>17084.00</td>
<td>3311</td>
<td>24</td>
</tr>
<tr>
<td>99%</td>
<td>5.00</td>
<td>50.00</td>
<td>20501.00</td>
<td>4768</td>
<td>42</td>
</tr>
<tr>
<td>Min</td>
<td>0.00</td>
<td>4.00</td>
<td>51.00</td>
<td>125</td>
<td>0</td>
</tr>
<tr>
<td>Max</td>
<td>5.00</td>
<td>446.00</td>
<td>20501.00</td>
<td>12359319162</td>
<td>129557750</td>
</tr>
</tbody>
</table>
Tombstones

- When a row is **deleted** in C* it is marked with a tombstone (virtual delete). Tombstones remain in the sstables for at least 10 days by default.
- A high ratio of tombstones to live data can have significant negative performance impacts (**latency**)
- Be wary of tombstones when: deleting data, updating with nulls or updating collection data types.

Diagnosing
- nodetool cfstats/cfhistograms and log file warnings
- slow read queries, sudden performance issues after a bulk delete

Correcting
- tune compaction strategy - LCS or TWCS can help in the right circumstances
- reduce GC grace period & force compaction for emergencies
- review data model/application design to reduce tombstones
Sprint 11: Cluster expansion
Cluster Changes

*Including*:

- Adding and removing nodes
- Replacing dead nodes
- Adding a Data Center
Ensure the cluster is 100% healthy and stable before making ANY changes.
**Adding Nodes**

- **How do you know when to add nodes?**
  - When disks are becoming >70% full.
  - When CPU/OS load is consistently high during peak times.

- **Tips for adding new nodes:**
  - If using logical racks, add one node to every rack (keep distribution even)
  - Add one node at a time.
  - During the joining process, the new node will stream data from the existing node.
  - A joining node will accept writes but not reads.
  - Unthrottle compactions on the JOINING node “nodetool setcompactionthroughput 0”
    - *But throttle again once node is joined.*
  - Monitor joining status with “nodetool netstats”
  - After the node has streamed and joined it will have a backlog of compactions to get through.
Replacing Nodes

• Replacing a dead node is similar to adding a new one, but add this line in the cassandra-env.sh before bootstrapping:

```bash
-Dcassandra.replace_address_first_boot=<dead_node_ip>
```

• This tells Cassandra to stream data from the other replicas.
  • Note this can take quite a long time depending on data size
  • Monitor with nodetool netstats

• If on >2.2.8 and replacing with a different IP address, the node will receive all the writes while joining.

• Otherwise, you should run repair.
  • If the replacement process takes longer than max_hint_window_in_ms you should run repair to make the replaced node consistent again, since it missed ongoing writes during bootstrapping (streaming).
Adding DC

Why?

- Distribute workload across data center or regions
- Major topology change
- Cluster migration
Adding DC: tips

• Ensure all keyspaces are using **NetworkTopologyStrategy**

• All queries using LOCAL_* consistency. This ensures queries will not check for replicas in the new DC that will be empty until this process is complete.

• All client connections are restricted to connecting only to nodes in the original DC. Use a data center aware load balancing policy such as DCAwareRoundRobinPolicy.

• Bring up the new DC as a stand alone cluster.
  • Provision nodes and configure Cassandra:
    • **cluster_name** in yaml must be the SAME as the original DC.
    • **DC name** in cassandra-rackdc.properties must be UNIQUE in the cluster.
    • Include **seed nodes** from the other DC.

• Join the new DC to the old one:
  • Start cassandra
  • Change replication on keyspaces
  • Execute nodetool rebuild <from existing dc> on 1-3 nodes at a time.

```sql
ALTER KEYSPACE my_ks WITH replication = { 'class': 'NetworkTopologyStrategy', 'DC1': 1, 'DC2': 1, 'DC3': 1};
```
- Sprint 1 Data modeling
- Sprint 2 Cluster sizing
- Sprint 3 Stress test
- Sprint 4 Data modeling (revisit)
- Sprint 5 Analytics
- Sprint 6 Security
- Sprint 7 Go Live
- Sprint 8 Monitoring
- Sprint 9 Hello Compaction and Repair
- Sprint 10 Large partition and tombstone
- Sprint 11 Cluster expansion
Christophe Schmitz
Director of Consulting, Europe
christophe@instaclustr.com

info@instaclustr.com
www.instaclustr.com
@instaclustr

• Managed service
• Consulting service
• Enterprise support contract