Running Cassandra on Apache Mesos across multiple datacenters at Uber

Abhishek Verma (verma@uber.com)

MesosCon, June 2016
About me

- MS (2010) and PhD (2012) in Computer Science from University of Illinois at Urbana-Champaign
- 2 years at Google, worked on Borg and Omega and first author of the [Borg paper](#)
- ~ 1 year at TCS Research, Mumbai
- Currently at Uber working on Cassandra Service
“Transportation as reliable as running water, everywhere. for everyone”

Uber’s mission
“Transportation as **reliable** as running water, everywhere. for everyone”

Uber’s mission
“Transportation as **reliable** as running water, everywhere. for **everyone**”

Uber’s mission
Cluster Management @ Uber

- Statically partitioned machines across different services
- Move from custom deployment system to everything running on Mesos
- Gain efficiency by increasing machine utilization
  - Co-locate services on the same machine
  - Can lead to 30% fewer machines
- Build stateful service frameworks to run on Mesos

“Large-scale cluster management at Google with Borg”, EuroSys 2015
Cassandra advantages

- Horizontal scalability
  - Scales reads and writes linearly as new nodes are added
- High availability
  - Fault tolerant with tunable consistency levels
- Low latency, solid performance
- Operational simplicity
  - Homogeneous cluster, no SPOF
- Rich data model
  - Columns, composite keys, counters, secondary indexes
- Integration with OSS: Hadoop, Spark, Hive
DCOS Cassandra Service
Mesosphere-Uber collaboration

https://github.com/mesosphere/dcos-cassandra-service

**Mesosphere**
- Keith Chambers
- Kenneth Owens
- Mohit Soni

**Uber**
- Abhishek Verma
- Karthik Gandhi
- Matthias Eichstaedt
- Teng Xu
- Zhiyan Shao
- Zhitao Li
Cassandra service architecture

- Web interface
  - Control plane API
- Deployment system
- Framework
  - dcos-cassandra-service
- Cluster 1
- Cluster 2
- Mesos master
  - Leader
- Mesos master
  - Standby
- Aurora (DC1)
- Aurora (DC2)
- ZooKeeper quorum
- DC2
- Client App uses CQL interface
- ...
Cassandra Service: Features

- Custom seed provider
- Increasing cluster size
- Replacing a dead node
- Backup/Restore
- Cleanup
- Repair
- Multi-datacenter support
Mesos primitives

- Persistent volumes
  - Data stored outside of the sandbox directory
  - Offered to the same task if it crashes and restarts
- Dynamic reservations
Plan, Phases and Blocks

- Plan
  - Phases
    - Reconciliation
    - Deployment
    - Backup
    - Restore
    - Cleanup
    - Repair

```json
{
  - phases: [
    - {
      id: "d68e15e1-1d5e-49a8-941f-243f8c6d55f",
      name: "Reconciliation",
      blocks: [
        - {
            id: "2820d1f1-0769-4085-a879-18e029a32f0c",
            status: "Complete",
            name: "Reconciliation",
            message: "Reconciliation complete",
            has_decision_point: false
          },
        status: "Complete"
      ],
    },
    - {
        id: "9d01561b-40cc-474a-92d3-c402a4f86f98",
        name: "Deploy",
        blocks: [
          - {
              id: "9268baf-6174-4477-a5e5-d610b166c0ab",
              status: "Complete",
              name: "node-0",
              message: "Deploying Cassandra node node-0",
              has_decision_point: false
            },
          - {
              id: "53bf64bb-1b1f-44bd-ac3e-5d9f0510c83",
              status: "Complete",
              name: "node-1",
              message: "Deploying Cassandra node node-1",
              has_decision_point: false
            }
        ],
        status: "Complete"
      },
      errors: [ ],
      status: "Complete"
    }
}
Spinning up a new Cassandra cluster

https://www.youtube.com/watch?v=gbYmjtDKSzs
Using the Cassandra cluster

https://www.youtube.com/watch?v=qgqO39DteHo
Automate Cassandra operations

● Repair
  ○ Synchronize all data across replicas
    ■ Last write wins
  ○ Anti-entropy mechanism
  ○ Repair primary key range node-by-node

● Cleanup
  ○ Remove data whose ownership has changed
    ■ Because of addition or removal of nodes
  ○ Cleanup node-by-node
Cleanup operation

https://www.youtube.com/watch?v=VxRLSi8MpYI
Failure scenarios

- Executor failure
  - Restarted automatically
- Cassandra daemon failure
  - Restarted automatically
- Node failure
  - Manual REST endpoint to replace node
- Scheduling framework failure
  - Existing nodes keep running, new nodes cannot be added
- Mesos master failure: new leader election
Experiments
Cluster startup

For each node in the cluster:

1. Receive and accept offer
2. Launch task
3. Fetch executor, JRE, Cassandra binaries from S3/HDFS
4. Launch executor
5. Launch Cassandra daemon
6. Wait for it’s mode to transition STARTING -> JOINING -> NORMAL
Cluster startup

For each node in the cluster:

1. **Receive** and accept offer
2. Launch task
3. Fetch executor, JRE, Cassandra binaries from S3/HDFS
4. Launch executor
5. Launch Cassandra daemon
6. Wait for it’s mode to transition STARTING -> JOINING -> NORMAL
Aurora hogs offers

- Aurora designed to be the only framework running on Mesos and controlling all the machines
- Holds on to all received offers
  - Does not accept or reject them
- Mesos waits for `--offer_timeout` time duration and rescinds offer
- `--offer_timeout` config
  - Duration of time before an offer is rescinded from a framework. This helps fairness when running frameworks that hold on to offers, or frameworks that accidentally drop offers. If not set, offers do not timeout.
- Set to 5mins in our setup, reduced to 10secs
Cluster startup time

Framework can start ~ one new node per minute
Long term solution: dynamic reservations

- Dynamically reserve all the machines resources to the “cassandra” role
- Resources are offered only to cassandra frameworks
- Improves node startup time: 30s/node
- Node failure replacement or updates are much faster
Tuning JVM Garbage collection

Changed from CMS to G1 garbage collector

Left: https://github.com/apache/cassandra/blob/cassandra-2.2/conf/cassandra-env.sh#L213
Right: https://docs.datastax.com/en/cassandra/2.1/cassandra/operations/ops_tune_jvm_c.html?scroll=concept_ds_sv5_k4w_dk__tuning-java-garbage-collection
Tuning JVM Garbage collection

Using cassandra-stress, 32 threads client

<table>
<thead>
<tr>
<th>Metric</th>
<th>CMS</th>
<th>G1</th>
<th>G1 : CMS Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>op rate</td>
<td>1951</td>
<td>13765</td>
<td>7.06</td>
</tr>
<tr>
<td>latency mean (ms)</td>
<td>3.6</td>
<td>0.4</td>
<td>9.00</td>
</tr>
<tr>
<td>latency median (ms)</td>
<td>0.3</td>
<td>0.3</td>
<td>1.00</td>
</tr>
<tr>
<td>latency 95th percentile (ms)</td>
<td>0.6</td>
<td>0.4</td>
<td>1.50</td>
</tr>
<tr>
<td>latency 99th percentile (ms)</td>
<td>1</td>
<td>0.5</td>
<td>2.00</td>
</tr>
<tr>
<td>latency 99.9th percentile (ms)</td>
<td>11.6</td>
<td>0.7</td>
<td>16.57</td>
</tr>
<tr>
<td>latency max (ms)</td>
<td>13496.9</td>
<td>4626.9</td>
<td>2.92</td>
</tr>
</tbody>
</table>

G1 garbage collector is much better without any tuning
Cluster Setup

- 3 nodes
- Local DC
- 24 cores, 128 GB RAM, 2TB SAS drives
- Cassandra running on bare metal
- Cassandra running in a Mesos container
Read Latency
Bare metal vs Mesos managed cluster

**Bare metal**

Mean: 0.38 ms  
P95: 0.74 ms  
P99: 0.91 ms

**Mesos**

Mean: 0.44 ms  
P95: 0.76 ms  
P99: 0.98 ms
Read Throughput
Bare metal vs Mesos managed cluster

Bare metal

Mesos
Write Latency
Bare metal vs Mesos managed cluster

**Bare metal**

![Write Latency (Bare Metal) Chart]

Mean: 0.43 ms  
P95: 0.94 ms  
P99: 1.05 ms

**Mesos**

![Write Latency (Mesos) Chart]

Mean: 0.48 ms  
P95: 0.93 ms  
P99: 1.26 ms
Write Throughput
Bare metal vs Mesos managed cluster

Bare metal

Mesos
Running across datacenters

- Four datacenters
  - Each running dcos-cassandra-service instance
  - Sync datacenter phase
    - Periodically exchange seeds with external dcs
- Cassandra nodes gossip topology
  - Discover nodes in other datacenters
Asynchronous cross-dc replication latency

- Write a row to dc1 using consistency level LOCAL_ONE
  - Write timestamp to a file when operation completed
- Spin in a loop to read the same row using consistency LOCAL_ONE in dc2
  - Write timestamp to a file when operation completed
- Difference between the two gives asynchronous replication latency
  - \textit{p50} : 44.69ms, \textit{p95} : 46.38ms, \textit{p99} : 47.44ms
- Round trip ping latency
  - 77.8ms
Questions?

verma@uber.com
Thank you

verma@uber.com