OpenDaylight Clustering – What’s new in Boron

Robert Varga, Jan Medved

With help from Peter Gubka, Vratko Polak, Tomas Cere, Quentin Chen
What Makes OpenDaylight Different?

- Netconf Server
- Northbound RESTCONF
- Application
- Protocol Plugins/Adapters
- Controller Core
- Controller Apps/Services
- Protocol Plugin
- Protocol Plugin
- Protocol Plugin
- Application
- OSS/BSS, External Apps
- Network Devices
ODL Software Architecture – Designed for Clustering

OSS, BSS, External Apps

Network Devices

Netconf Server
RESTCONF
REST
Application
Application
Protocol Plugin
Netconf Client

Apps/Services

“Kernel”

3, 5, 7, … 2n+1 nodes

RPCs
Notifications

Data Change Notifications

Model-Driven (MD-SAL)

Data Store

Conceptual Data Tree - Config

Conceptual Data Tree - Operational

Yang Model

Network Devices

Data Change Notifications

Model-Driven (MD-SAL)

Data Store

Conceptual Data Tree - Config

Conceptual Data Tree - Operational

Yang Model

Netconf Server
RESTCONF
REST
Application
Application
Protocol Plugin
Netconf Client

Apps/Services

“Kernel”

3, 5, 7, … 2n+1 nodes

RPCs
Notifications

Data Change Notifications

Model-Driven (MD-SAL)

Data Store

Conceptual Data Tree - Config

Conceptual Data Tree - Operational

Yang Model

Netconf Server
RESTCONF
REST
Application
Application
Protocol Plugin
Netconf Client

Apps/Services

“Kernel”

3, 5, 7, … 2n+1 nodes

RPCs
Notifications

Data Change Notifications

Model-Driven (MD-SAL)

Data Store

Conceptual Data Tree - Config

Conceptual Data Tree - Operational

Yang Model
What’s Wrong with Clustering in Beryllium?

1. Bugs
   - Oversights:
     • Global RPC is not really global
     • DataTreeChangeNotifications not global
   - Corner cases do not work:
     • Unfixable w/o major rewrite of Frontend – Backend protocol
     • Raft algorithm implementation
     • Incorrect 3PC implementation

2. Performance
   - Hey, why not go to an external database and an external message bus?

3. Difficult/impossible to use in applications
   - Virtually no Beryllium app (OF/FRM, Netconf, BGP/PCEP, ….) cluster-ready
   - No Beryllium app uses clustering properly
   - No common HA/Clustering app model
So what was done with Clustering for Boron

1. Bug fixing & code reorganization:
   - Centralized state transition queueing
   - Leadership handover race fixes
   - Predictable persistence upgrades
   - Mechanism for frontend to retry transactions (BUG-5280, slated for SR1)

2. Performance improvements:
   - Dynamic shards

3. Started on Application Clustering/HA model:
   - Singleton service
Refresher: Conceptual Data Tree

- Node 1
- Node 2
- Node 3
- In-Memory Data Store (IMDS)
- Clustered Data Store (CDS)
- Shard1 DS
- Shard2 DS
- Shard3 DS
- Shard4 DS
- Default Shard DS

Raft decides Shard Leader placement

Data Replication

“Shard Anchor / Shard Root”
Existing Clustered Data Store - Details

Node 1

App → Concurrent Data Broker

Distributed Data Store

CONFIG

OPERATIONAL

Shard Manager

Shard 1

Shard 2

Akka Clustering

Supervise

Node 2

Node 3

FrontEnd → BackEnd
Data Store Evolution in Boron and Beyond

y.d.a. = Yang Data API
DAB/PC - Directed Acyclic Graph Producer/Consumer
Shard Performance Testing: Test Setup

Single Producer Test

- List/Array of x string elements divided into 1 … n shards
- Single Producer (Test) writes data to shards in Round-Robin or Random
- Producer handles tx responses asynchronously (Futures)

Multi-Producer Test

- List/Array of x string elements divided into 1 … n shards
- (Test) writes data to Shard Workers (Producers) in Round-Robin or Random fashion
- Worker handles tx responses asynchronously (Futures)
Performance Results: Single Producer

![Graph showing performance results for different numbers of shards and listeners. The x-axis represents the number of shards (1, 2, 4, 8, 16) and the y-axis represents performance in k data/sec. The graph shows that performance decreases as the number of shards increases, with variations depending on the number of listeners (0, 1, 10, 100).]
Performance Results: Multiple Producers

The graph shows the performance of a system with multiple producers under different conditions. The x-axis represents the number of shards, ranging from 1 to 16, and the y-axis shows the performance in thousands of data/sec. The graph includes lines for different listener counts: 0, 1, 10, and 100 listeners.

- The blue line represents 0 listeners, showing a steady increase in performance as the number of shards increases.
- The orange line represents 1 listener, showing a similar trend but slightly below the 0 listener line.
- The brown line represents 10 listeners, with a performance curve that is higher than the 1 listener line.
- The red line represents 100 listeners, showing the highest performance among the lines, particularly notable with a significant increase at 16 shards.

The graph indicates that increasing the number of shards generally improves performance, with the 100 listeners condition showing the most significant improvement.
Performance Results: Impact of Clustering on CDS

![CDS Performance Graph]

- Single Node
- Leader
- Follower

- BA PUT
- DOM PUT
- BA SINGLETX
- DOM SINGLETX

OPEN DAYLIGHT
Performance Results: Lessons Learned

• Small transactions are dominated by initialization cost, charged to producer
  – Affects a single thread’s ability to saturate backend

• Batching effectiveness goes up with backend latency
  – Many listeners, complex transaction processing, messaging latency

• Listening across shards results in heavy backend contention
  – Increases latency in notification delivery
  – Results in more events being batched hence spikes are observable

• Dynamic sharding improves performance with multiple applications
  – Per-application shards result in better isolation and improved parallelism
  – Single-threaded applications are unable to fully saturate IMDS
Getting & Running the Performance Test Code

• Clone coretutorials:
  git clone https://git.opendaylight.org/gerrit/coretutorials.git

• Build a distribution with sharding performance tests:
  cd coretutorials/clustering/shardingsimple
  mvn clean install

• Run the distribution:
  ./karaf/target/assembly/bin/karaf –Xmx4096m

• Run the test script:
  cd coretutorials/clustering/scripts/sharding
  ./shardingbenchmark.py –help (will print all the parameters in the script)
  More info in coretutorials/clustering/scripts/sharding/site/asciidoc/scripts-user-manual.adoc
OpenDaylight “Internal” Application Model

Service/Application:
- Code + Data (Shards, subtrees of the Conceptual Data Tree)
- Service/Application instances SHOULD be co-located with shard replicas
- Active Service/Application instance SHOULD be co-located with all Shard Leaders it “owns”
- Apps can be grouped for "fate sharing"
Singleton Service

• Guarantees a cluster-wide single active instance of an app or a group of apps
• Used by OF Plugin, FRM, GBP, LISP Mapping Server
Singleton Demo Application

- 3 simple RPC services: “Global” RPC, Routed RPC, “Local” RPC
  - Accessible via RESTCONF
  - Returns the location (node) where the service is running
  - Started when becoming active (instantiateServiceInstance() called)

Node1
- Global RPC
- Local RPC
- Routed RPC

Node2
- Global RPC
- Local RPC
- Routed RPC

Node3
- Global RPC
- Local RPC
- Routed RPC

RESTCONF

Singleton Service

Browser

RAFT

Browser

Singleton Service

Node1

Node2

Node3
Demo
Getting & Running the Example Code

• Clone coretutorials:
  git clone https://git.opendaylight.org/gerrit/coretutorials.git

• Build a distribution with the example app:
  cd coretutorials/clustering/singletonsimple
  mvn clean install

• Run it in a clustered environment:
  • A VM/host with Docker installed
  • See scripts in coretutorials/clustering/scripts how to build a clusterized ODL Docker image
  • See ansible playbook for cluster install in coretutorials/clustering/scripts

• Detailed information:
  coretutorials/clustering/singletonsimple/src/site/asciidoc/overview.adoc
What do we want to do for Carbon

1. Bug fixing & code reorganization:
   – API duplicity cleanup (org.opendaylight.{yangtools, mdsal, controller})
   – CDS async persistence and replication
   – Fix CDS three-phase commit (3PC) protocol backend

2. Performance improvements:
   – Dynamic shards for CDS
   – Data Broker cleanups
   – Dormant CDS replicas

3. Started on Application Clustering/HA model:
   – API to move Shard leader towards a Producer
Thank you
Data Store Sharding

Node 1:
Conceptual Data Tree

Clustered Data Store (CDS)

Shard1 DS

Default Shard DS

Shard2 DS

Shard3 DS

Shard4 DS

Raft decides Shard Leader placement

Node 1:

Node 2:

Node 3:

OpenDaylight