Building Distributed Systems with Apache Helix!

Aditya Auradkar – Data Infrastructure @ LinkedIn
Lifecycle of a system

- Single Node
- Multi Node
- Fault Tolerant
- Fully Elastic
Ok… so lots of common problems!
Solve Problems Once

- What if you need to add a common feature to all systems?
  - For example: Zone Awareness

- Implementing in multiple systems is tricky and error prone

- Develop feature in common system

- Solve problem only once
Takeaway

- Distributed systems have very similar challenges
- Shared challenges take the most time to solve
- Unrelated to the actual problem domain
  - Kafka is a messaging system
  - Cassandra is a storage system
  - Pinot is OLAP
- Apache Helix abstracts these concepts in a reusable manner
A brief history

- Built at LinkedIn in 2011
  - Lots of systems being built in parallel

- Built to support Espresso and Databus
  - Espresso is LinkedIn’s distributed document store
  - Databus is LinkedIn’s change capture pipeline
  - Adopted in several other systems since then

- Open Source since 2012

- Apache TLP since 2014
Helix Usage

LinkedIn

Uber

Box

Yahoo!

Pinterest

Instagram

etc..
Helix Features – High Level

- Generic cluster management service

- Automated resource assignment
  - Map resources to a pool of nodes
  - Automated load balancing of resources

- Dynamic addition and removal of resources

- Node failure detection and recovery

- Dynamic node addition
Helix Features – High Level

- Automated Pluggable state model
  - Every system has a different definition of state

- Pluggable rebalancing

- Scheduled workflows
  - Index rebuilds, backups

- Throttled State Transitions

- Dynamic configuration changes
## Core Concepts

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>A group of nodes</td>
</tr>
<tr>
<td>Resource</td>
<td>A logical entity (database, message topic etc..)</td>
</tr>
<tr>
<td>Partition</td>
<td>Some subset of a resource</td>
</tr>
<tr>
<td>Replica</td>
<td>Standby copy of a partition</td>
</tr>
<tr>
<td>Current State</td>
<td>State of a replica (Online/Offline, Master/Slave etc..)</td>
</tr>
<tr>
<td>State Transition</td>
<td>Change in current state to any of the allowed states</td>
</tr>
</tbody>
</table>
Core Concepts

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant</td>
<td>A process registered with the cluster eligible to host replicas. Example: storage node in a distributed database</td>
</tr>
<tr>
<td>Spectator</td>
<td>A process interested in reading the state of the cluster</td>
</tr>
<tr>
<td>Controller</td>
<td>The component of Helix that generates state transitions</td>
</tr>
</tbody>
</table>
Ideal State

- Helix generated assignment is IdealState
- Best possible assignment
- Mapping from location to state for each resource
  - Replicas are never placed on the same node
- Persisted under /root/IdealState/<resource-name>
Ideal State
External View

- External View represents the current state of the cluster

- Can be different from IdealState

- If External View == Ideal State
  - Cluster is in steady state

- Helix issues transitions needed to reach IdealState

- Persisted under /root/ExternalView/<resource-name>
State Models

- Helix Controller tries to ensure that IdealState == ExternalView

- How does Helix compute the transitions?
  - Finite State Machine needed to govern behavior
  - State Machine has constraints

- State Machine is pluggable
  - Every application can have a different state model
State Models - Rebalancer

- Helix controller runs rebalancing algorithms

- Rebalancer is triggered on cluster events
  - Node start or stop
  - Node failure
  - Addition or removal of nodes
  - IdealState changes
State Models – Online Offline

- **Two States**
  - Online, Offline

- **Constraints**
  - All replicas can be online
  - Start in Offline
  - Transitions throttled per resource

- **Suited for**
  - Read only data
  - If all replicas are equal
State Models – Master Slave

- Three States
  - Master, Slave, Offline

- Constraints
  - One Master
  - Start in Offline
  - Offline cannot transition to Master
  - Transitions throttled per resource

- Suited for a database
State Models – Leader Standby

- Two States
  - Leader, Standby

- Constraints
  - Only 1 Leader

- Suited for single master applications
Rebalancing Strategies

<table>
<thead>
<tr>
<th></th>
<th>Full Auto</th>
<th>Semi Auto</th>
<th>Custom</th>
<th>User Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replica Location</td>
<td>Helix</td>
<td>App</td>
<td>App</td>
<td>App</td>
</tr>
<tr>
<td>Replica State</td>
<td>Helix</td>
<td>Helix</td>
<td>App</td>
<td>App</td>
</tr>
</tbody>
</table>
Rebalancing Strategies – Full Auto

```json
{
  "id" : "MyResource",
  "simpleFields" : {
    "REBALANCE_MODE" : "FULL_AUTO",
    "NUM_PARTITIONS" : "3",
    "REPLICAS" : "2",
    "STATE_MODEL_DEF_REF" : "MasterSlave"
  },
  "listFields" : {
    "MyResource_0" : [],
    "MyResource_1" : [],
    "MyResource_2" : []
  },
  "mapFields" : {
  }
}
```
Rebalancing Strategies – Full Auto

Node1
- P1: Master
- P3: Slave

Node2
- P2: Master
- P1: Slave

Node3
- P3: Master
- P2: Slave
Rebalancing Strategies – Full Auto

Node 1
- P1: Master
- P3: Slave

Node 2
- P2: Master
- P1: Slave

Node 3
- P3: Master
- P2: Slave
Rebalancing Strategies – Full Auto

Node1
- P1: Master
- P2: Slave
- P3: Slave

Node2
- P2: Master
- P3: Master
- P1: Slave

Node3
- P3: Master
- P2: Slave
Rebalancing Strategies – Full Auto

Node1
- P1: Master
- P2: Slave
- P3: Slave

Node2
- P2: Master
- P3: Master
- P1: Slave

Node3
- P3: Master
- P2: Slave
Rebalancing Strategies – Semi Auto

```json
{
  "id" : "MyResource",
  "simpleFields" : {
    "REBALANCE_MODE" : "SEMI_AUTO",
    "NUM_PARTITIONS" : "3",
    "REPLICAS" : "2",
    "STATE_MODEL_DEF_REF" : "MasterSlave",
  },
  "listFields" : {
    "MyResource_0" : ["Node1", "Node2"],
    "MyResource_1" : ["Node2", "Node3"],
    "MyResource_2" : ["Node3", "Node1"]
  },
  "mapFields" : {
  }
}
```
Rebalancing Strategies – Semi Auto

Node 1
- P1: Master
- P3: Slave

Node 2
- P2: Master
- P1: Slave

Node 3
- P3: Master
- P2: Slave
Rebalancing Strategies – Semi Auto

Node1
- P1: Master
- P3: Slave

Node2
- P2: Master
- P1: Slave

Node3
- P3: Master
- P2: Slave
Rebalancing Strategies – Semi Auto

Node1
- P1:Master
- P3:Master

Node2
- P2:Master
- P1:Slave

Node3
- P3:Master
- P2:Slave
Rebalancing Strategies – Semi Auto

Node1
- P1:Master
- P3:Master

Node2
- P2:Master
- P1:Slave

Node3
- P3:Master
- P2:Slave
Rebalancing Strategies – Custom

- Replica placement and state controlled by application
- Application implements and registers callback
- Callback invoked on cluster state change
- Recompute IdealState in callback
- Helix issues transitions based on new IdealState
  - Constraints must not be violated
Architecture
Architecture diagram

ZK
- Ideal State
- External View
- Configs
- Live Instances

Helix Controller

Spectator
- Agent

Node1
- Agent
  - P1:Master
  - P3:Slave

Node2
- Agent
  - P2:Master
  - P1:Slave

Node3
- Agent
  - P3:Master
  - P2:Slave

Cluster Participants

©2015 LinkedIn Corporation. All Rights Reserved.
Zookeeper Structure

- **CONFIGS**
- **CLUSTER**
  - HELIX QUICKSTART
- **PARTICIPANT**
  - localhost_12000
  - localhost_12001
  - localhost_12002
- **RESOURCE**
- **INSTANCES**
  - localhost_12000
  - localhost_12001
  - localhost_12002
- **LIVEINSTANCES**
  - localhost_12000
  - localhost_12001
Controller as a service

- Controller must not be a single point of failure

- Deploy multiple controllers
  - Controllers elect a leader

- Controllers can manage multiple clusters

- Each controller is leader for subset of clusters
Controller as a Service

- ZK
  - Ideal State
  - External View
  - Configs
  - Live Instances

Controllers

Cluster 1
Cluster 2
Cluster 3

Participants
Controller as a Service

ZK
- Ideal State
- External View
- Configs
- Live Instances

Controllers
- Cluster 1
- Cluster 2
- Cluster 3

Participants
Controller as a Service

Participants

Cluster 1

Cluster 2

Cluster 3

Controllers

ZK

Ideal State

External View

Configs

Live Instances

Participants
Real Life Examples (Espresso)
Espresso – Distributed Data Store

- Espresso is a distributed database

- Source of truth database for critical data at LinkedIn
  - Profile, Inbox, Advertising etc.

- Kafka for internal replication, MySQL storage engine

- Uses Helix for cluster management
  - Master Slave State model
  - Semi Auto Rebalance
Espresso Architecture

Helix Controller

Client

Router

Spectator

Participants

ZK

Node1
- Agent
- P1:Master
- P3:Slave

Node2
- Agent
- P2:Master
- P1:Slave

Node3
- Agent
- P3:Master
- P2:Slave

Storage Nodes
Espresso Architecture

- **Helix Controller**
- **Client**
- **Router**
- **Storage Nodes**

1. **Disconnect**
2. **External View Change**
3. **Routing Failure**
4. **P3: S-> M, Node 1**
5. **P3: S-> M, Node 1**

**Node1**
- Agent
- P1:Master
- P3:Slave

**Node2**
- Agent
- P2:Master
- P1:Slave

**Node3**
- Agent
- P3:Master
- P2:Slave

**ZK**

©2015 LinkedIn Corporation. All Rights Reserved.
Espresso Architecture

1) Disconnect

2) External View Change

3) P3 Available

4) P3: S-> M, Node 1

5) P3: S-> M, Node 1

6) P3 Available

Storage Nodes

Node1
- Agent
- P1:Master
- P3:Master

Node2
- Agent
- P2:Master
- P1:Slave

Node3
- Agent
- P3:Master
- P2:Slave

Helix Controller

ZK

Client

Router
Define State Model

```java
StateMachineDefinition.Builder builder = new StateMachineDefinition.Builder(STATE_MODEL_NAME);

// Add states and their rank to indicate priority. A lower rank corresponds to a higher priority
builder.addState(MASTER, 1);
builder.addState(SLAVE, 2);
builder.addState(OFFLINE);

// Set the initial state when the node starts
builder.initialState(OFFLINE);

// Add transitions between the states.
builder.addTransition(OFFLINE, SLAVE);
builder.addTransition(SLAVE, OFFLINE);
builder.addTransition(SLAVE, MASTER);
builder.addTransition(MASTER, SLAVE);
```
Define Constraints

```java
// static constraint: upper bound of 1 MASTER
builder.upperBound(MASTER, 1);

// dynamic constraint: R means it should be derived based on the replication factor for the cluster
// this allows a different replication factor for each resource without
// having to define a new state model
builder.dynamicUpperBound(SLAVE, "R");
```
public static class MasterSlaveStateManager extends TransitionHandler {

    public void onBecomeSlaveFromOffline(Message message, NotificationContext context) {
        // Bootstrap data and setup replication from master
    }

    public void onBecomeSlaveFromMaster(Message message, NotificationContext context) {
        // Setup replication from new master
    }

    public void onBecomeMasterFromSlave(Message message, NotificationContext context) {
        // Catch up from previous master. Stop replica fetchers. Ready to accept writes
    }

    public void onBecomeOfflineFromSlave(Message message, NotificationContext context) {
        // Stop replication
    }
}
Spectator Code

```java
HelixManager manager = HelixManagerFactory.getZKHelixManager(clusterName,
    instanceName,
    InstanceType.SPECTATOR,
    zkConnectString);

manager.connect();

RoutingTableProvider routingTableProvider = new RoutingTableProvider();
manager.addViewChangeListener(routingTableProvider);
instances = routingTableProvider.getInstances("myDB", "myDB_1", "ONLINE");

// Application-specific code to send a request to one of the instances
result = instances.get(random).sendRequest(request);
```
**Define Cluster and Resources**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup Cluster</td>
<td><code>.helix-admin.sh --zkSvr localhost:10000 --addCluster MYCLUSTER</code></td>
</tr>
<tr>
<td>Add Participants</td>
<td><code>.helix-admin.sh --zkSvr localhost:10000 --addNode MYCLUSTER localhost:1999</code></td>
</tr>
<tr>
<td>Define Resource</td>
<td><code>.helix-admin.sh --zkSvr localhost:10000 --addResource MYCLUSTER MyDB num-partitions MasterSlave</code></td>
</tr>
<tr>
<td>Rebalance</td>
<td><code>.helix-admin.sh --zkSvr localhost:2199 --rebalance MYCLUSTER MyDB replication-factor</code></td>
</tr>
</tbody>
</table>
Real Life Examples (Databus)
Databus – Distributed Change Capture

- Consistent change capture for databases at LinkedIn
  - Connectors for Oracle and Espresso

- Consumed by downstream systems
  - Search, Graph, HDFS Ingestion

- Size based retention for sources

- Uses Helix for cluster management
  - Online Offline State model
  - Customized Rebalance
Databus Architecture

Helix Controller

Participants

ZK

Espresso

Participants

Node1
Agent
Rebalancer
P1:Online
P2:Online
P3:Online

Node2
Agent
P1:Online
P2:Online
P3:Online

Node3
Agent
P1:Online
P2:Online
P3:Online

Storage Nodes

©2015 LinkedIn Corporation. All Rights Reserved.
More

- **Cluster Configuration**
  - Can set configuration at the instance and cluster level
  - Configuration can change dynamically

- **Admin APIs for Cluster Management**
  - Java API, CLI and REST Interface

- **Local agent for non-jvm clients**
Acknowledgements

- Kishore Gopalakrishna – Helix VP and Sr Staff Engineer @ LinkedIn
- Lei Xia – Sr Engineer @ LinkedIn

Resources

- helix.apache.org
- Twitter - @ApacheHelix
Even our workspace is Horizontally Scalable!