Deploying & Managing distributed apps on YARN

Steve Loughran, Devaraj Das & Ted Yu

{stevel, ddas, tyu} at hortonworks.com
About myself

- HBase committer / PMC member
- Slider committer
- YARN / HDFS contributor
Hadoop as Next-Gen Platform

**Single Use System**  
*Batch Apps*

**HADOOP 1.0**

- **MapReduce**  
  (cluster resource management & data processing)

- **HDFS**  
  (redundant, reliable storage)

**HADOOP 2.0**

- **MapReduce**  
  (data processing)

- **Others**  
  (data processing)

- **YARN**  
  (cluster resource management)

- **HDFS2**  
  (redundant, reliable storage)

**Multi Purpose Platform**  
*Batch, Interactive, Online, Streaming, …*
Availability (always-on)

Flexibility (dynamic scaling)

Resource Management (optimization)

Applications Run Natively IN Hadoop

YARN (Cluster Resource Management)

HDFS2 (Redundant, Reliable Storage)
HDFS2 (Redundant, Reliable Storage)

YARN (Cluster Resource Management)

HBase

YARN (Cluster Resource Management)

HDFS2 (Redundant, Reliable Storage)
Step 1: Hoya: On-demand HBase

JSON config in HDFS drives cluster setup and config

1. Small HBase cluster in large YARN cluster
2. Dynamic, self-healing
3. Freeze / thaw
4. Custom versions & configurations
5. More efficient utilization/sharing of cluster
YARN manages the cluster

- Servers run YARN Node Managers
- NM's heartbeat to Resource Manager
- RM schedules work over cluster
- RM allocates containers to apps
- NMs start containers
- NMs report container health
Client creates App Master
AM deploys HBase with YARN
HBase & clients bind via Zookeeper
YARN notifies AM of failures
JSON Specification

{
"schema": "http://example.org/specification/v2.0.0",
"metadata": {
},
"global": {
"yarn.vcores": "1",
"yarn.memory": "256",
},
"components": {
"rs": {
"yarn.memory": "512",
"yarn.priority": "2",
"yarn.instances": "4"
},
"slider-appmaster": {
"yarn.instances": "1"
},
"master": {
"yarn.priority": "1",
"yarn.instances": "1"
}
}
}
boolean flexCluster(ConfTree updated) {
    appState.updateResourceDefinitions(updated);
    return reviewRequestAndReleaseNodes();
}

void onContainersCompleted(List<ContainerStatus> completed) {
    for (ContainerStatus status : completed) {
        appState.onCompletedNode(status);
    }
    reviewRequestAndReleaseNodes();
}
Limitations

• Needs app with built in discovery/binding protocol
• Static configuration – no dynamic information
• Kill-without-warning is the sole shutdown mechanism
• Custom Java in client & App Master for each service
• Client code assumed CLI – embedded/PaaS use as common.
“Imagine starting a farm of tomcat servers hooked up to an HBase cluster you just deployed – servers processing requests forwarded by a load-balancing service”
Slider: evolution of & successor to Hoya

1. Unified packaging format for deployable applications
2. Service registration and discovery
3. Propagation of dynamic config information back to clients
4. Client API for embedding – CLI only one use case.

Goal: no code changes to deploy applications in a YARN cluster
Packaging "RPM for a datacenter"

- Packaging format for deployable applications
- metadata, template configurations
- template logic to go from config to scripts
- simple .py scripts for starting different components in different YARN containers
- future: snapshot, graceful stop
Service Registration and Discovery

- Applications to publish port bindings (URLs…)
- Applications to publish other config data
- Client APIs to find targeted instance, retrieve bindings and other data

- See also YARN-913
Slider – the tool

• **Slider**
  - Java tool
  - Completely CLI driven

• **Input: cluster description as JSON**
  - Specification of cluster: node options, ZK params
  - Configuration generated
  - Entire state persisted

• **Actions: create, freeze/thaw, flex, exists <cluster>**

• **Can change cluster state later**
  - Add/remove nodes, started / stopped states
**Model**

### Persisted
- **Specification**
  - resources.json
  - appconf.json &c

### Rebuilt
- **NodeMap**
  - model of YARN cluster

### Transient
- **Event History**
  - application history

- **Component History**
  - persistent history of component placements

- **Component Map**
  - container ID -> component instance

- **Container Queues**
  - requested, starting, releasing
Slider as a Service

1. builds
2. queues
3. launches
4. reads
5. deploys components
6. controls via REST API

Application

YARN RM

Slider AM

YARN NM

HDFS

resources

appconf

Components

Components

Components
YARN-896: long-lived services

1. Container reconnect on AM restart ✓
2. YARN Token renewal on long-lived apps
3. Containers: signalling (YARN-445), >1 process sequence
4. AM/RM managed gang scheduling
5. Anti-affinity hint in container requests
6. Service Registry (YARN-913)
7. Logging
SLAs & co-existence with MapReduce

1. Make IO bandwidth/IOPs a resource used in scheduling & limits

2. Need to monitor what's going on w.r.t IO & net load from containers \(\rightarrow\) apps \(\rightarrow\) queues

3. Dynamic adaptation of cgroup HDD, Net, RAM limits

4. Could we throttle MR job File & HDFS IO bandwidth?
Status as of April 2014

• Initial agent, REST API, container launch
• Package-based deployments of HBase, Ambari, Storm
• Hierarchical configuration JSON evolving
• Service registry - work in progress
• Incubator: proposal submitted
Questions?

hortonworks.com