Apache Hadoop Yarn: Yet Another Resource Negotiator

Fan Jun Bo

Problems for Hadoop 1.X

Tight coupling of specific programming model with resource management infrastructure => abuse the use of MapReduce.

centralised handling of job' s control flow => endless scalability concerns of the scheduler.

Target for YARN

Decoupling programming model from resource management infrastructure

Delegate scheduling functions to per-application components

History works

100 billion nodes with 1 trillion edges

Dreadnaught(max 800 nodes) to Hadoop (Scalability)

Multiple Business flow => Multi-tenancy (Hadoop on Demand)

HOD => Allocate Hadoop clusters on shared hardware

HOD's each cluster per job => Serviceability

HOD's shared HDFS

HOD Shortcomings

Torque's unawareness of locality **Locality Awareness**

Long job latency led by cluster allocation **High Cluster Utilisation**

JobTracker's Failure leads to lost of all running jobs and manually recovery is needed **Reliability/Availability**

Hadoop would manage more tenants, diverse of cases and raw data with low authorisation model currently **Secure**

Map Reduce is not suitable for every tasks **Support for Programming Model Diversity**

# of map and reduce slots are fixed by cluster operator so fallow map resources would not help for reducer **Flexible Resource Model**

Fit for old version of Hadoop to avoid "second system syndrome" **Backward Compatibility**

Architecture of YARN

YARN: platform layer responsible for resource management

Resource Manager

Application Master

Node Manager

RM for YARN

Centralised view of cluster resource (global scheduling ability by various schedulers)=> fairness capacity locality across tenants

Dynamically allocate leases(containers) to applications

container is logical bundle of resource(eg <2GB, 1Core>) to a specific Node(eg Node3)

persistent storage for accepted jobs

2 external interfaces:
Clients to submit jobs with secure control
AM dynamically access to resource

1 internal interface: cluster monitoring and resource access management towards NM

overall resource demand of an application ignore local optimisation and internal flow

Request resources back from an application(AM decides how to compress resource usage by migration tasks, yielding containers)

RM is NOT for:
providing status or metrics for running app(AM)
serving framework specific reports of completed jobs(pre-framework daemon)

AM for YARN

Head of an application to coordinate with the app process

Build request model => encode them to heartbeat message => send to RM => receive container lease => update execution plan(late-binding)

Optimise of locality

AM decides the success or failure of container status by the information from NM through RM

NM for YARN

NM is a worker daemon in YARN

NM is responsible for container lease, managing container dependencies, monitoring their execution

Container Launch Context(CLC) describes all containers.{environment variables, dependencies stored remotely, security tokens etc }. NM is for configuring CLC for a container

NM kill containers directed by RM/AM

Periodically monitoring the health of physical node
NM provides local service to container(eg log aggregation to HDFS once done)

Admin could configure NM set of pluggable services(eg preserve some output until the app done)

Frameworks for YARN

0. Submit CLC for AM to RM

1. RM starts AM and register the AM

2. AM send ResourceRequest to RM and receive allocation

3. AM construct CLC to NM for launching containers and mounting/managing them

4. AM is done, it unregister from RM and exit cleanly.

YARN at Yahoo! and Experiments

Applications and Frameworks

Hadoop MapReduce: MapReduce model

Apache Tez: DAG execution framework

Spark: machine learning

Dryad: DAG

Giraph: vertex-centric graph computation framework

Storm: real-time processing engine