YARN: A Resource Manager for Analytic Platform

Tsuyoshi Ozawa
ozawa.tsuyoshi@lab.ntt.co.jp
ozawa@apache.org
About me

• Tsuyoshi Ozawa
• Research Engineer @ NTT
  • Twitter: @oza_x86_64
• Over 150 reviews in 2015
• Apache Hadoop Committer and PMC
  • Introduction to Hadoop 2nd Edition (Japanese)” Chapter 22 (YARN)
  • Online article: gihyo.jp “Why and How does Hadoop work?”

Copyright © 2016 NTT corp. All Rights Reserved.
Agenda

• 3 features of YARN
• Behaviors of DAG processing engine on YARN
  • Case study1: Tez on YARN
  • Case study2: Spark on YARN
    • How do they work and best practice
YARN

• A resource manager for Apache Hadoop
  • Being able to share resources not only across MapReduce job, but also across the other processing framework

• 3 kind of features
  1. Managing resources in cluster
  2. Managing history logs of application
  3. Mechanism for users to know locations running on YARN
YARN features
1. Resource Management
Why manages resources with YARN?

• Taking different data analytics platform based on workloads
  • Google uses Tenzing/MapReduce for batch/ETL processing, BigQuery for try and error, TensorFlow for machine learning

• There are 2 ways to do so:
  1. Separating each clusters
     • Pros: easy to separate workload and resource
     • Cons: difficult to copy big data between clusters
  2. Living together in the same cluster
     • Pros: no need to copy big data between clusters
     • Cons: difficult to separate resources
Design policy of YARN

• Hadoop attaches weight to scalability since Hadoop is for processing big data!

• Hence, YARN attaches weight to scalability!
  • Different frameworks living together in the same cluster without moving data
  • Launching masters of processing framework per job
    • At Hadoop MapReduce v1, master of MR get overload and reaches limit of scalability in large scale processing
Architecture of YARN

- **Master-Worker architecture**
  - Master manages resources in cluster (ResourceManager)
  - Worker manages containers per machine (NodeManager)

1. Submitting job
2. Allocating a container for master, launching master
3. Master requests containers to RM and launching worker on the container
Issue of resource management in YARN

• Issue
  • Allocated containers are not monitored: it’s allocated, but it has a possibility not to utilize them

• Suggestion by YARN community
YARN-1011(Karthik, Cloudera)
  • Monitoring all allocated containers and launching(over committing) tasks if it’s not used

• Effect
  • NodeManager can utilize more resources per node
YARN features

2. Managing history logs
What is history log?

• Application information which is running or completed
  • It’s useful to tune performance and debug!

• Types of history log
  1. Common information which all YARN applications have
     • Allocated containers, name of used queues
  2. Application specific information
     • E.g. MapReduce case: Mapper/Reducer’s counter/amount of shuffle etc.

• Server for preserving and displaying history logs → Timeline Server
What can Timeline Server do?

• YARN applications only show generic log with “YARN’s common information”

• Screenshot of Timeline Server
What can Timeline Server do?

- Rich performance analysis with YARN application specific information
  - examples: Tez-UI

Example of Tez-UI

• We can check job statistics in detail

From: http://hortonworks.com/wp-content/uploads/2015/06/Tez-UI-1024x685.png
Design philosophy of Timeline Server

- Scalability to number of applications
- Easy to access history logs
  - Users retrieve data via RESTful API
Current status of Timeline Server

• 3 versions: V1, V1.5, V2

• **V1 design**
  • Pluggable backend storage
    • LevelDB is default backend storage
  • Timeline Server includes Writer and Reader
    • Limited scalability

• **V2 design for scalability**
  • Distributed Writer and Reader
  • Scalable KVS as Backend storage
    • Focusing on HBase
  • Changing API drastically
    • Being enable to profiling across multiple jobs

• **(New!)V1.5 design**
  • HDFS backend storage with V2 API
  • Reader/Writer separation like V2
  • Please note the incompatibility between V1 and V1.5
Question from me

• Do you need other backends for Timeline Server?

• Alternatives
  • Time-series DB
    • Apache Kudu
  • RDBMS
    • PostgreSQL or MySQL
YARN feature
3. Service Registry
What is Service Registry?

• A service to know the location (IP, port) of YARN application
  • For running long-running jobs like HBase on YARN, Hive on Tez, MixServer in HiveMall

• Why do we need service registry?
  • To notify clients destination of write operation

1. Request to register port and address
2. Preserving
3. Get info
4. Connect
Behavior of applications on YARN
MR-inspired distributed and parallel processing framework on YARN

• New processing frameworks include a high-level DSL than MapReduce
  • Apache Tez: HiveQL/Pig/Cascading
  • Apache Spark: RDD/SQL/DataFrame/DataSets
  • Apache Flink: Table/DataSets/DataStream

• Features
  • More generic Directed Acyclic Graph = DAG, instead of MapReduce, describes jobs
    • One kind of compilation DSL to DAG with Spark, Hive on Tez, Flink
Why use DAG?

• DAG can express complex job, like Map – Map – Reduce, as single job
  • Why it is bad to increase number of jobs?
    • Before finishing Shuffle, Reduce cannot be started
    • Before finishing Reduce, next job cannot be started
→ Decreasing parallelism

Figure from: https://tez.apache.org/
Features of Modern processing frameworks

• The first feature is job description with DAG
  • The second one depends on processing framework

• Apache Tez
  • DAG + Disk-IO optimization + Low-latency query support
    • Hive/Pig/Spark/Flink can run on Tez

• Apache Spark
  • DAG + Low-latency in-memory processing calculation
    + Functional programming-style interface

• Apache Flink
  • DAG + Shuffle optimization
    + Streaming/Batch processing transparent interface
    • Like, Apache Beam
Workloads modern processing frameworks can handle

• Large-scale batch processing
  • Yes, MapReduce can handle it well
  • Sometimes MapReduce is still best choice for the stability

• Short query for data analytics
  • Trial and error to check the mood of data with changing queries
    • Google Dremel can handle it well
  • After evolving interface of MR(SQL), users need improvement of latency, not throughput
    • Currently, we take on different processing framework
    • However, it’s a just workaround

• YARN think the former is 1st citizen at first.
• How can we run the latter on YARN effectively?
Problem for running low-latency query on YARN

- **Overhead of job launching**
  - Because YARN needs 2 step to launch jobs: master and workers

- **Taking time to warm up**
  - Because YARN terminates containers after the job exits

- **Tez/Spark/Flink runs on server-side JVM, so these problems are remarkable**

→ **In this talk, I’ll talk about how Tez and Spark manages resources on YARN!**
Overview of Apache Tez

- **MapReduce-inspired DAG processing framework for large-scale data effectively under DSL (Pig/Hive)**
  - Passing key-value pair internally like MapReduce
  - Runtime DAG rewriting
    - DAG plan itself is dumped by Hive optimizer or Pig Optimizer
    - Being able to describe DAG without Sorting while shuffling

- **Example of Hive query execution flow**
  - Writing Hive query
  - Submitting Hive query
  - Dumping DAG of Tez by Hive optimizer
  - Executing Tez
Effective resource management in Tez

• Container reusing
  • Reusing containers instead of releasing containers in a job as possible as master can
  • Keeping allocated containers if session lives after completing jobs

Q. What is different between MapReduce’s Container Reuse and Tez’s one?

A.
  • Container reusing is removed by accident at the implementation of MapReduce on YARN
  • Session enable us to reuse containers across multiple jobs
Effective resource management in Tez (cont.)

• Long Lived and Process
  • Allocating containers without relationship with “Session”
    → Being able to reuse warm-up JVM and cache
    More low latency
  • LLAP uses Registry Service

• Q. Is it like daemon of database?
  A.
  • That’s right.
    The difference is that Tez can accelerate jobs by adding resources from YARN.
Tuning parameter

• **Rule**
  - Keeping container long can improve latency but can worsen throughput
  - Keeping container short can improve throughput (resource utilization) but can worsen latency

• **Container reuse**
  - tez.am.container.idle.release-timeout-min.millis
    - Keeping containers in this period
  - tez.am.container.idle.release-timeout-max.millis
    - Releasing containers after the period

• **LLAP**
  - Looking like being WIP

• **Note:** Hive configuration override Tez configuration, so please test it carefully
Tez on YARN summary

• Hive on Tez assumed batch processing at first
  • Releasing all containers after finishing jobs
    → Attaching more weight on throughput than latency

• As expanding use case of Hive, users need low latency for interactive queries running on REPL

• Improving latency based on high-throughput architecture
Overview of Apache Spark

• In-memory processing framework (originally!)

• Standalone mode is supported

• Spark on YARN is also supported

• Spark has 2 kinds of running mode on YARN
  • yarn-cluster mode
    • Submitting job (spark-submit)
  • yarn-client mode
    • REPL style (spark-shell)

• Both of them run on YARN
yarn-cluster mode

• Launching Spark driver on YARN container
  • Assuming spark-submit command

1. Submitting jobs via spark-submit

2. Launching master

3. Allocating resources for workers by Spark AppMaster
yarn-client mode

• Launching Spark driver at client-side
  • Assuming REPL like spark-shell

1. Launching spark-shell, Connecting to RM
2. Launching Master
3. Allocating resources for workers by Spark AppMaster
4. Enjoy interactive programming!!
Resource management of Spark on YARN

• Spark have a feature of container reuse of Tez natively
  • Spark can handle low-latency query very well

• However, Spark cannot “release containers”
  → State of tasks in memory will be lost
  → Cannot release containers still when allocated resources is not used
  → Decreasing resource utilization

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 2</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

![Diagram showing resource management of Spark on YARN](image-url)
Dynamic resource allocation (since Spark 1.2)

• Allocating/releasing containers based on workload
  • Intermediate data is persisted by using NodeManager’s Shuffle Service → Not in memory data
  • Triggers of allocating/releasing containers is timer

Doubling number of Executors if executors has tasks for a period

Releasing containers when executor is idle for a period

Exe 1  Exe 2  Exe 3  Exe 4 (idle)

NodeManager

Intermediate data

Spark AppMaster

Intermediate data
Tuning parameters

- **Rule**
  - Keeping container long can improve latency but can worsen throughput
  - Keeping container short can improve throughput (resource utilization) but can worsen latency
  → Please tune this points!

- **Max/Min/Initial number of YARN containers**
  → high numbers of executors increases throughput, but can decrease resource utilizations
  - spark.dynamicAllocation.maxExecutors
  - spark.dynamicAllocation.minExecutors
  - spark.dynamicAllocation.initialExecutors

- **A period to double number of containers for Executors when all Executors are active**
  → large value can improve latency and throughput but worsen stability and resource utilization
  - spark.dynamicAllocation.schedulerBacklogTimeout

- **A period to keep containers**
  → large value can improve latency but worsen resource utilization
  - spark.dynamicAllocation.sustainedSchedulerBacklogTimeout

- **A period to release containers which is empty**
  → large value can improve latency but worsen
  - spark.dynamicAllocation.executorIdleTimeout

- **Releasing executor with cache data if the executor is idle in the specified period**
  - spark.dynamicAllocation.cachedExecutorIdleTimeout
Spark on YARN summary

• Spark assumes interactive programming interface natively
  Container is not released by default
  → Attaching more weight on latency than throughput

• Adding a features to release containers dynamically when supporting YARN

• Increasing throughput based on low-latency architecture
Summary

• 3 kinds of features YARN provides
  1. Resource management in cluster
  2. Timeline Server
  3. Service registry

• Dive into resource management in Tez/Spark
  • Apache Tez, originated from batch processing
  • Apache Spark, originated from interactive processing

Both of Tez and Spark can process wider query by switching high throughput mode and low latency mode