Big Data Frameworks CSE6001

Prof. Ramesh Ragala

September 26, 2020

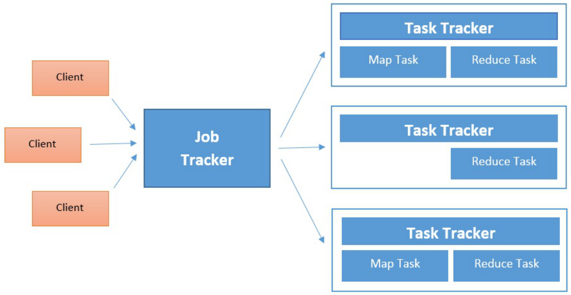
Limitations in MapReduce - 1

Map-Reduce-1 has two types of Daemons to control the job execution process:

- Jobtracker
  - It is a single master process, which coordinates all jobs running on the cluster and assigns map and reduce tasks to run on the Tasktrackers.
  - It takes care of job scheduling (matching tasks with tasktrackers)
  - It takes care of task progress monitoring
    - Keeping track of tasks
    - Restarting failed or slow tasks
    - Doing task book-keeping such as maintaining counter totals
  - It stores the job history for completed jobs
  - If a task fails, the jobtracker can reschedule it on a different tasktracker
- Tasktracker
  - It runs tasks and sends progress reports to the jobtrackers, which keeps a record of the overall progress of each job

Limitations in MapReduce - 1

Classical Version of Apache Hadoop (Map-Reduce-1)



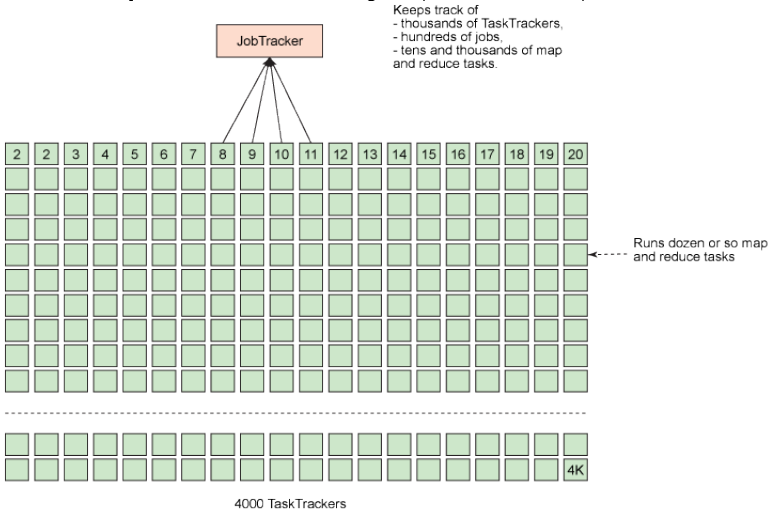
Limitations in MapReduce - 1

Large Hadoop cluster has limitation in scalability → single Job Tracker

- According Yahoo → a cluster of 5000 nodes and 40,000 task are running
- Solution - 1:
  - Smaller and less-powerful clusters has to be created and maintained
    - The smaller and larger Hadoop clusters had never used their computational resources with optimum efficiency
    - In Hadoop Mapreduce, the computational resources on each slave node are divided by cluster administrator into fixed-number of map and reduce slots
    - A node cannot run more map tasks than map slots at any given moment, even if no reduce tasks are running.
    - It harms the cluster utilization because when all map slots are taken (and we still want more), we cannot use any reduce slots, even if they are available, or vice versa
  - Solution - 2:
    - An alternative Programming model (such as graph processing provided by Apache Giraph) → designed to run mapreduce jobs only

Limitations in MapReduce - 1

Busy JobTracker on large Apache Hadoop Cluster



JobTracker
Keeps track of
- thousands of TaskTrackers,
- hundreds of jobs,
- tens and thousands of map and reduce tasks.
2
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Runs dozen or so map and reduce tasks
4K
4000 TaskTrackers

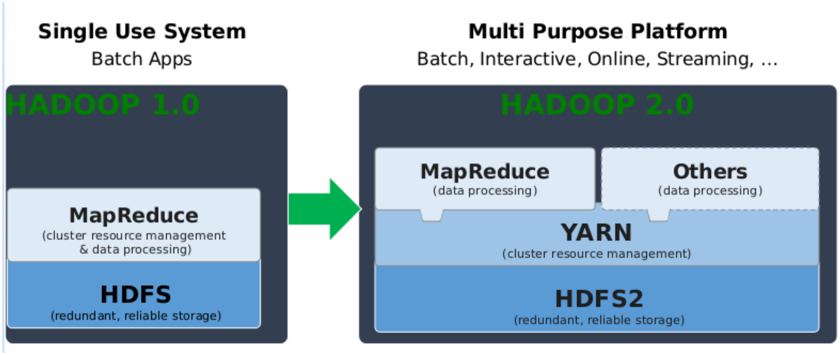
Limitations in MapReduce - 1

- High Availability (HA):
  - State needed for another daemon to take the work needed to provide the service, in the event of the service daemon fails.
  - Large amount of rapidly changing complex state in the jobtracker's memory makes difficult to retrofit HA into the jobtracker sevice.
  - Responsibilities of jobtracker are split into Resource Manager and Application Master in YARN.
- Utilization:
  - Tasktracker in Map Reduce -1 is configured with a static allocation of fixed-size slots.
  - Node Manager in YARN manages a pool of resources. Not Fixed number of designated slots
  - Application in can make request for what type of resources are needed.
- Multitenancy:
  - It opens up Hadoop to other types of distributed applications beyond mapreduce

Introduction to YARN

- YARN
  - **Y**et **A**nother **R**esource **N**egotiator
  - Arun Murthy created the original JIRA in 2008
  - YARN Released October 2013 with Apache Hadoop -2.0
  - YARN separates the resource management and processing components in Hadoop-2.0
  - Hadoop's Cluster resource management system
  - Introduced in Hadoop-2 to improve the mapreduce implementation
  - It also supports other distributed computing paradigms as well
  - YARN Provides APIs for requesting and working with cluster resources.$\rightarrow$ These APIs are not directly used by user code

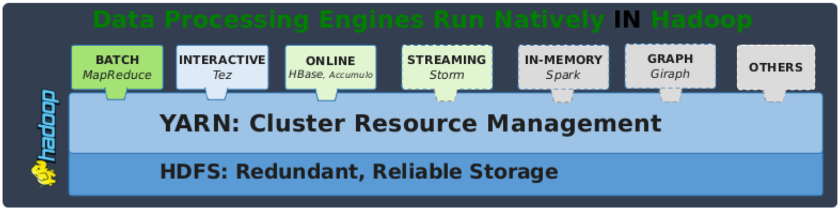
Hadoop 2.0: From Batch-only to Multi-Workload



Single Use System
Batch Apps
Multi Purpose Platform
Batch, Interactive, Online, Streaming, ...
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)

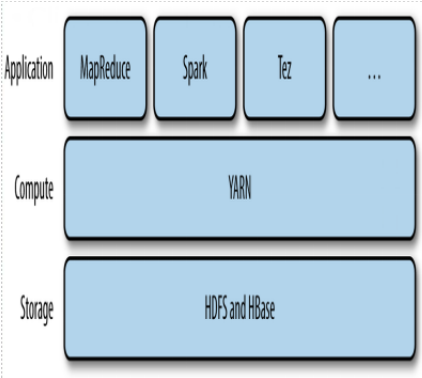
Key Driver Of Hadoop Adoption: Enterp Data Lake

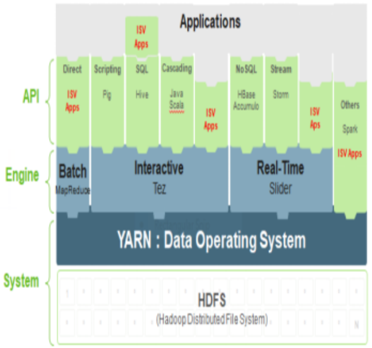
- Flexible:
  - Enables other purpose-built data processing models beyond MapReduce (batch), such as interactive and streaming
- Efficient:
  - Double processing **IN** Hadoop on the same hardware while providing predictable performance & quality of service
- Shared:
  - Provides a stable, reliable, secure foundation and shared operational services across multiple workloads



Data Processing Engines Run Natively IN Hadoop
BATCH MapReduce
INTERACTIVE Tez
ONLINE HBase, Accumulo
STREAMING Storm
IN-MEMORY Spark
GRAPH Giraph
OTHERS
YARN: Cluster Resource Management
HDFS: Redundant, Reliable Storage
hadoop

Introduction to YARN





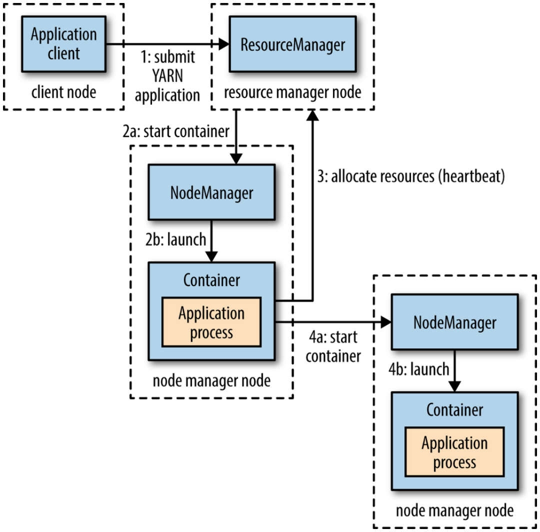
Introduction to YARN

- YARN provides it services with two daemons:
  - Resource Manager (One per Cluster) $\rightarrow$ To Manage the usage of resources in cluster
  - Node Manager (All the nodes in cluster) $\rightarrow$ To launch and monitor containers.
- A Container executes an application-specific process with a constrained set of resources (memory, cpu, etc..)

Anatomy of YARN Application Run

- **Step - 1:** A client contacts the resource manager and asks it to run an application master process
- **Step - 2:** The resource manager then finds a node manager that can launch the application master in a container
- **Step - 3 & 4:** It could simply run a computation in the container. it is running in and return the result to the client. Or it could request more containers from the resource managers and use them to run a distributed computation
- Hadoop's RPC is used to pass status updates and results back to the client, but these are specific to the application

Anatomy of YARN Application Run



Introduction to YARN

Resource Requests:

- It has flexible model for making resource requests
- A request for a set of containers
  - can express the amount of computer resources are required for each container (memory and CPU)
  - Locality constraints for the containers in the request → bandwidth efficiency
- Locality constraints can be used to request a container on a specific node or rack, or anywhere on the cluster (off-rack).
- What happens if locality constraint is not met????
  - Not met means → no allocation is made or the constraint can be loosend → How did the YARN solve this problem
  - Example → If a specific node was requested but it is not possible to start container on it
  - YARN will try to start a container on a node in the same rack or if that's not possible, on any node in the cluster

Introduction to YARN

Resource Requests:

- The common case of launching a container to process
  - The application will request a container on one of the nodes hosting the block's three replicas, or
  - On a node in one of the racks hosting the replicas, or
  - Failing that, on any node in the cluster
- YARN application can make resource requests at any time while it is running.
  - More dynamic approach whereby it requests more resources dynamically to meet the changing needs of the application
  - Example:
    - Map-Reduce : map and reduce
    - Map task containers are requested up-front, but the reducer task containers are not started until later
    - If any task fails, additional containers will be requested so the failed task can be re-run

Introduction to YARN

Application lifespan

- The lifespan of a YARN application can vary dramatically
  - Short-lived Applications
  - Long-lived Applications
- Categorize the applications in terms of how they map to the jobs that users run $\rightarrow$ Three models
  - One Application for one user job
    - MapReduce used this approach
  - One Application for workflow or user session jobs
    - Spark uses this approach
    - More efficient
    - Potential to cache intermediate data between jobs
  - Long – running application that is shared by different users
    - An application often acts in some kind of coordination role
    - Apache Slider use this approach
    - Impala uses this approach

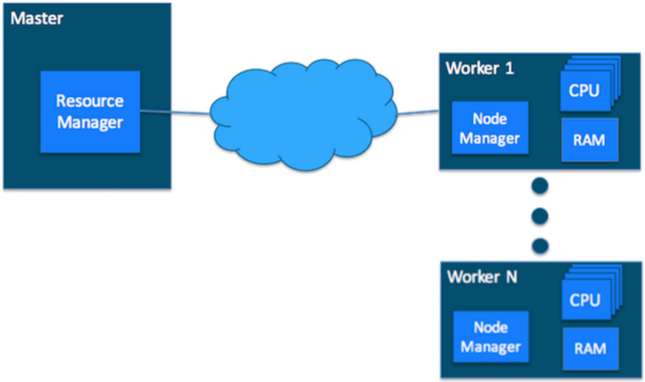
Introduction to YARN

Building YARN Applications:

- We can use existing application to write YARN application, instead of writing YARN application from scratch.
- Spark or Tez $\rightarrow$ execution of DAG of jobs on YARN
- Apache Slider
- Apache Twill $\rightarrow$ simple programming model for developing distributed application on YARN
- Distributed Shell application $\rightarrow$ part of the YARN to write distributed applications

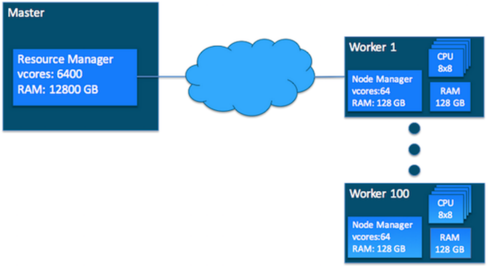
YARN cluster basics

- Master host with ResourceManager and Worker hosts with NodeManager



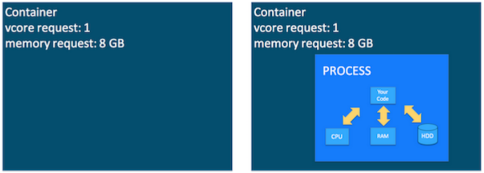
YARN CLUSTER BASICS

- YARN Requires a Global View:
  - two Resources: 1. vcores (usage share of CPU core) 2. memory
- Each NodeManager tracks its own local resources and communicates its resource configuration to the ResourceManager.
- Resource Manager keeps a running total of the cluster's available resources
- The ResourceManager knows how to allocate resources as they are requested.



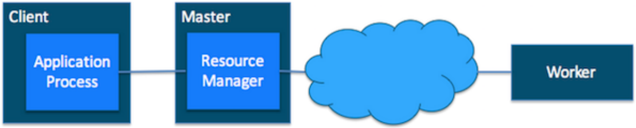
YARN CLUSTER BASICS

- Containers:
  - A container as a request to hold resources on the YARN cluster.
  - Once a hold has been granted on a host, the NodeManager launches a process called a task.
  - An application is a YARN client program that is made up of one or more tasks
  - For each running application, a special piece of code called an ApplicationMaster helps coordinate tasks on the YARN cluster. $\rightarrow$ The ApplicationMaster is the first process run after the application starts.



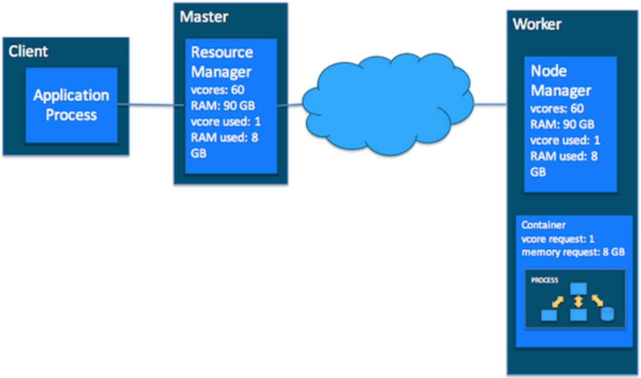
YARN cluster basics

The application starts and talks to the ResourceManager for the cluster:



YARN cluster basics

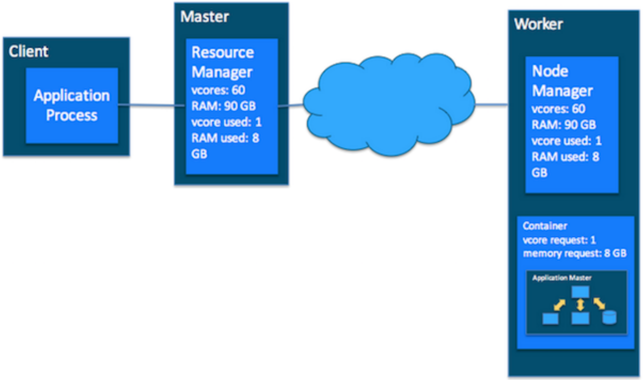
The ResourceManager makes a single container request on behalf of the



application

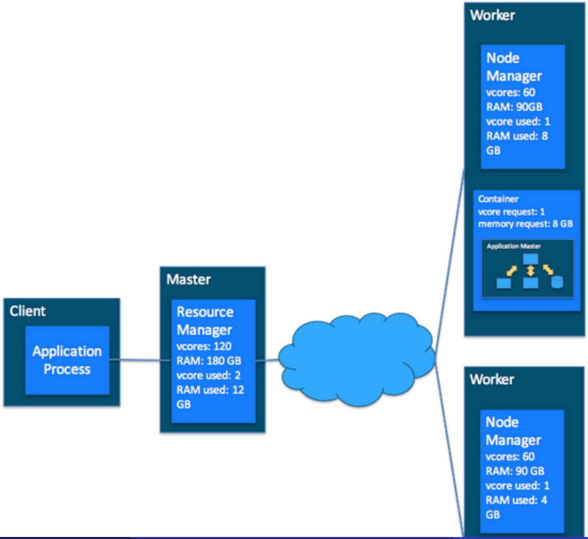
YARN CLUSTER BASICS

The ApplicationMaster starts running within that container



YARN cluster basics

The ApplicationMaster requests subsequent containers from the ResourceManager that are allocated to run tasks for the application.

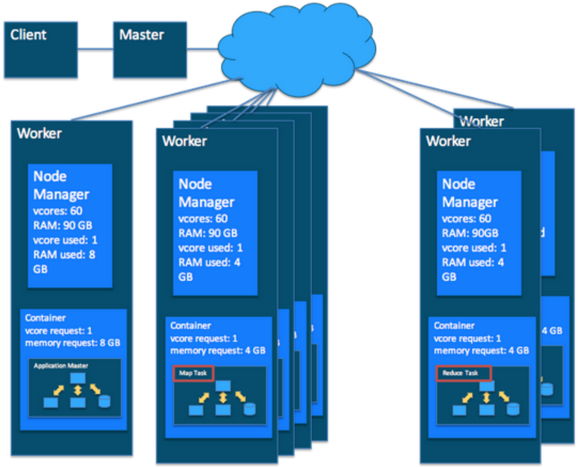


YARN cluster basics

- Once all tasks are finished, the ApplicationMaster exits. The last container is de-allocated from the cluster.
- The application client exits.

YARN Cluster basics

Putting it Together: MapReduce and YARN



Thank you