Apache YARN

Notes by Mannan Ul Haq (BDS-4A)

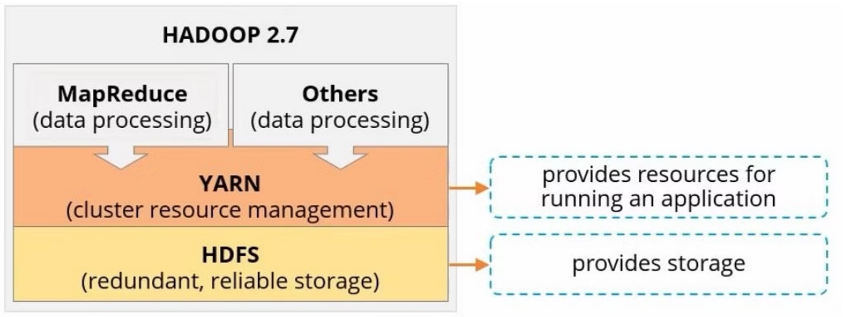
Apache YARN, which stands for Yet Another Resource Negotiator, is a key component of the Hadoop ecosystem that acts as a resource manager. It was created by splitting the processing engine and management functions of MapReduce to improve efficiency and scalability. Here's a simple breakdown of what YARN does:

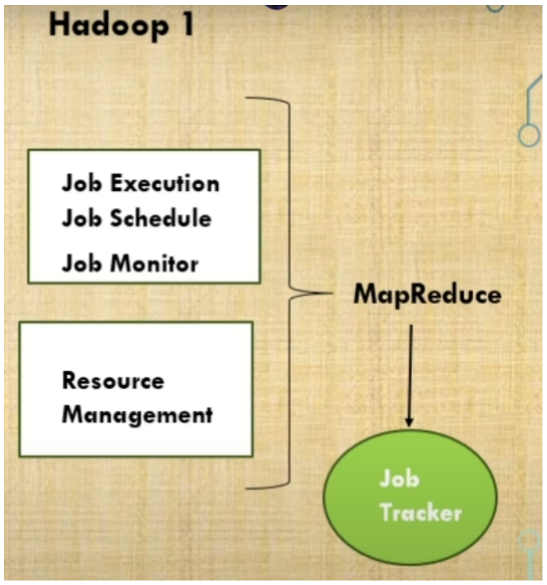
- **Monitor and manage workloads**: YARN keeps track of various tasks and jobs running on a Hadoop cluster, ensuring that resources like CPU and memory are allocated efficiently.
- **Maintain a multi-tenant environment**: YARN allows multiple users and applications to share the same cluster, ensuring fair resource distribution and preventing any single job from monopolizing the cluster's resources.
- **Manage high availability features**: YARN helps keep the Hadoop system reliable and available even if some components fail, ensuring continuous operation.
- **Implement security control**: YARN enforces security policies to protect data and control access, making sure that only authorized users and applications can perform certain actions.

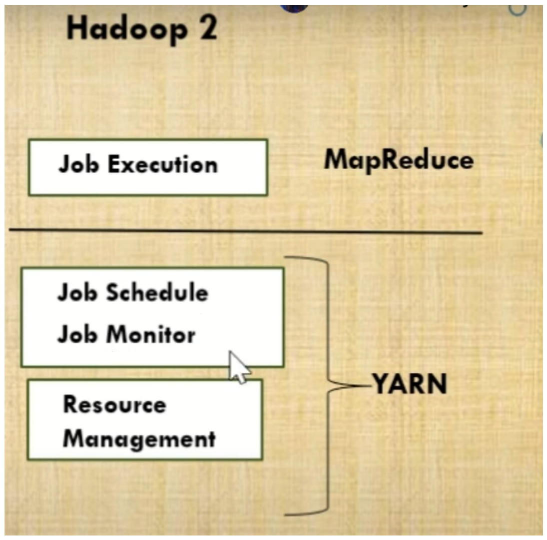
Need for YARN

Before 2012, users could write MapReduce programs using scripting languages such as Java, Python, and Ruby. They could also use Pig, a language used to transform data. No matter what language was used, its implementation depended on the MapReduce processing model.

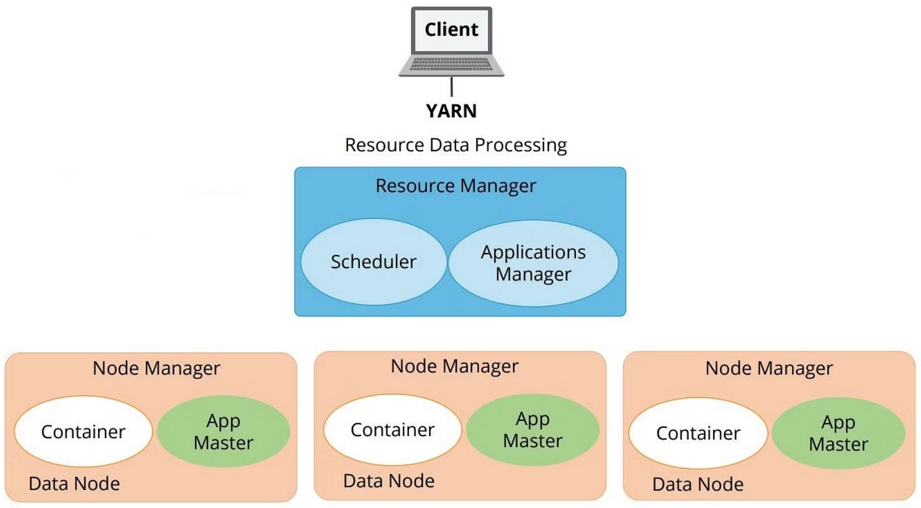
In May 2012, during the release of Hadoop version 2.0, YARN was introduced by Yahoo. You are no longer limited to working with the MapReduce framework anymore as YARN supports multiple processing models in addition to MapReduce, such as Spark. Other features of YARN include significant performance improvement and a flexible execution engine.


HADOOP 2.7
MapReduce
(data processing)
Others
(data processing)
YARN
(cluster resource management)
HDFS
(redundant, reliable storage)
provides resources for running an application
provides storage





YARN Architecture



The main components of YARN architecture include:

- **Client**: Submits MapReduce jobs to the system.
- **Resource Manager**: The master daemon of YARN, responsible for resource assignment and management among all applications. When it receives a processing request, it forwards it to the corresponding Node Manager and allocates the necessary resources. The Resource Manager has two major components:
    - **Scheduler**: Allocates resources to applications based on their needs and the availability of resources. It is a pure scheduler, meaning it doesn't handle tasks like monitoring or restarting tasks if they fail.

    - **Application Manager**: Manages application submissions, negotiates the first container from the Resource Manager, and restarts the Application Master container if a task fails.
- **Node Manager**: Manages individual nodes within the Hadoop cluster, handling applications and workflows specific to that node. It registers with the Resource Manager and sends heartbeats to report the node's health status. The Node Manager monitors resource usage, performs log management, and can kill containers based on directions from the Resource Manager. It also creates and starts container processes upon request from the Application Master.
- **Application Master**: Manages a single job submitted to a framework. It negotiates resources with the Resource Manager, tracks the status, and monitors the progress of its specific application. The Application Master requests containers from the Node Manager by sending a Container Launch Context (CLC), which includes all necessary information for the application to run. It also sends periodic health reports to the Resource Manager. Application Master communicates with the NameNode, how much resources (number of CPUs, number of nodes, memory required) will the job need. So, the NameNode will do its computation and figure out those things.
- **Container**: Represents a collection of physical resources (RAM, CPU cores, disk space) on a single node. Containers are launched using the Container Launch Context (CLC), which includes information like environment variables, security tokens, and dependencies required for the application to run.

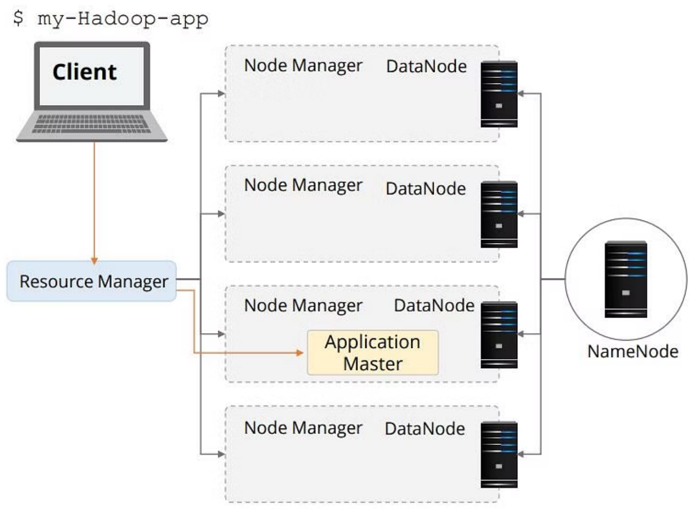
Advantages :

- **Flexibility:** YARN offers flexibility to run various types of distributed processing systems such as Apache Spark, Apache Flink, Apache Storm, and others. It allows multiple processing engines to run simultaneously on a single Hadoop cluster.
- **Resource Management:** YARN provides an efficient way of managing resources in the Hadoop cluster. It allows administrators to allocate and monitor the resources required by each application in a cluster, such as CPU, memory, and disk space.
- **Scalability:** YARN is designed to be highly scalable and can handle thousands of nodes in a cluster. It can scale up or down based on the requirements of the applications running on the cluster.
- **Improved Performance:** YARN offers better performance by providing a centralized resource management system. It ensures that the resources are optimally utilized, and applications are efficiently scheduled on the available resources.
- **Security:** YARN provides robust security features such as Kerberos authentication, Secure Shell (SSH) access, and secure data transmission. It ensures that the data stored and processed on the Hadoop cluster is secure.

Running an Application through YARN

Step 1 - Application submitted to the Resource Manager

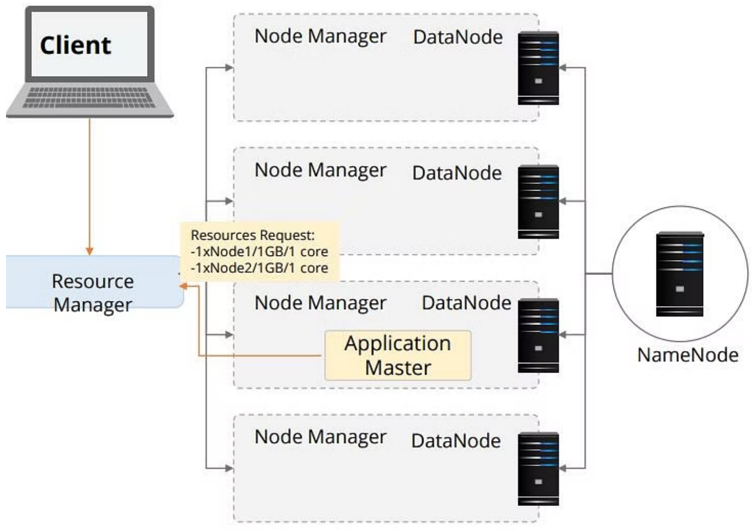
Users submit applications to the Resource Manager.



The Resource Manager maintains the list of applications on the cluster and available resources on the Node Manager. The Resource Manager determines the next application that receives a portion of the cluster resource.

Step 2 - Resource Manager allocates Container

When the Resource Manager accepts a new application submission, one of the first decisions the Scheduler makes is selecting a container. Then, the Application Master is started and is responsible for the entire life-cycle of that particular application.

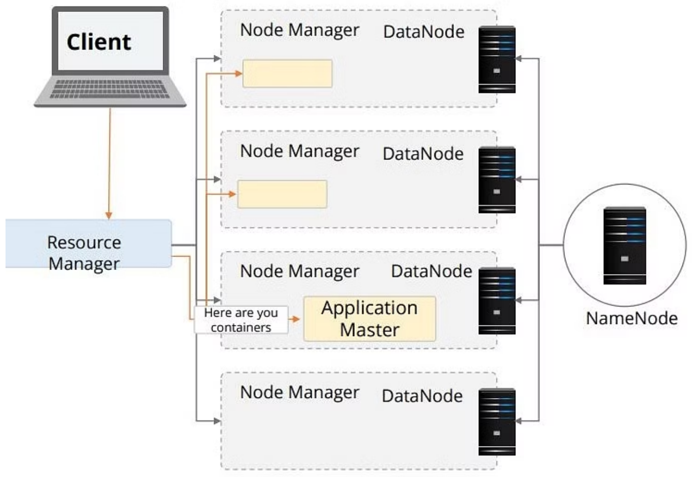


First, it sends resource requests to the ResourceManager to ask for containers to run the application's tasks.

A resource request is simply a request for a number of containers that satisfy resource requirements such as the following:

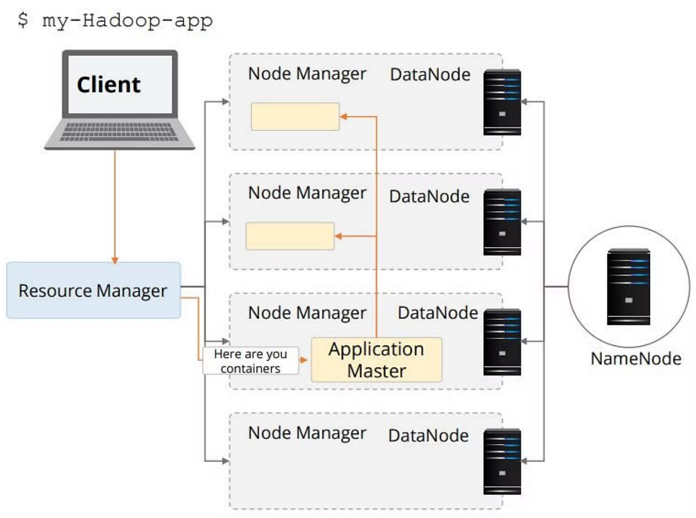
- Amount of resources expressed as megabytes of memory and CPU shares Preferred location, specified by hostname or rackname, Priority within this application and not across multiple applications.
- The Resource Manager allocates a container by providing a container ID and a hostname, which satisfies the requirements of the Application Master.

Step 3 - Application Master contacts Node Manager



After a container is allocated, the Application Master asks the Node Manager managing the host on which the container was allocated to use these resources to launch an application-specific task. This task can be any process written in any framework, such as a MapReduce task.

Step 4 -Resource Manager Launches Container

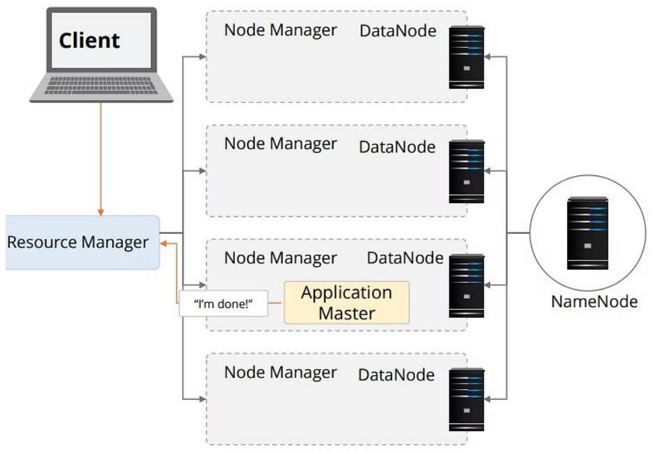


The NodeManager does not monitor tasks; it only monitors the resource usage in the containers.

For example, it kills a container if it consumes more memory than initially allocated.

Throughout its life, the Application Master negotiates containers to launch all of the tasks needed to complete its application. It also monitors the progress of an application and its tasks, restarts failed tasks in newly requested containers, and reports progress back to the client that submitted the application.

Step 5 - Container Executes the Application Master



After the application is complete, the Application Master shuts itself and releases its own container. Though the ResourceManager does not monitor the tasks within an application, it checks the health of the ApplicationMaster. If the ApplicationMaster fails, it can be restarted by the ResourceManager in a new container. Thus, the resource manager looks after the ApplicationMaster, while the ApplicationMaster looks after the tasks.

Execution Modes:

In Spark, there are two modes to submit a job:

- **Client Mode**
- **Cluster Mode**

Client Mode:

In client mode, Spark is installed on the local client machine, where the Driver program resides. The Driver program is the entry point to a Spark application, containing the SparkSession or SparkContext.

1. **Job Submission**: When a job is submitted using `spark-submit`, the request is sent to the Resource Manager.
2. **Application Master**: The Resource Manager launches the Application Master on one of the Worker nodes.
3. **Executors**: The Application Master then launches Executors (equivalent to Containers in Hadoop) where the job will be executed.

After launching, the Executors communicate directly with the Driver program on the client machine, and the output is returned to the client.

Drawback:

The client machine must remain available throughout the job's execution. If the client machine is turned off or disconnected, the connection between the Driver and Executors is broken, leading to job failure.

Cluster Mode:

In cluster mode, Spark is installed on the cluster rather than on the local machine.

1. **Job Submission**: Similar to client mode, the job is submitted using `spark-submit`, and the request goes to the Resource Manager.
2. **Application Master**: The Resource Manager launches the Application Master on one of the Worker nodes.
3. **Driver Program**: Unlike client mode, the Application Master launches the Driver program (with SparkSession/SparkContext) on the Worker node.

In cluster mode, the Spark driver runs inside the Application Master process managed by YARN on the cluster. This allows the client to disconnect after initiating the job, as the Driver program runs on the cluster.