**Problem Statement**

**Explain in brief the architecture of Apache Hadoop Yarn.**

YARN is yet another Resource Negotiator introduced in Hadoop 2.x to overcome the issue with Hadoop 1.x which is **Job Tracker overloading.** In Hadoop 1.x Job Tracker has three functions –Resource management, job monitoring and job scheduling.

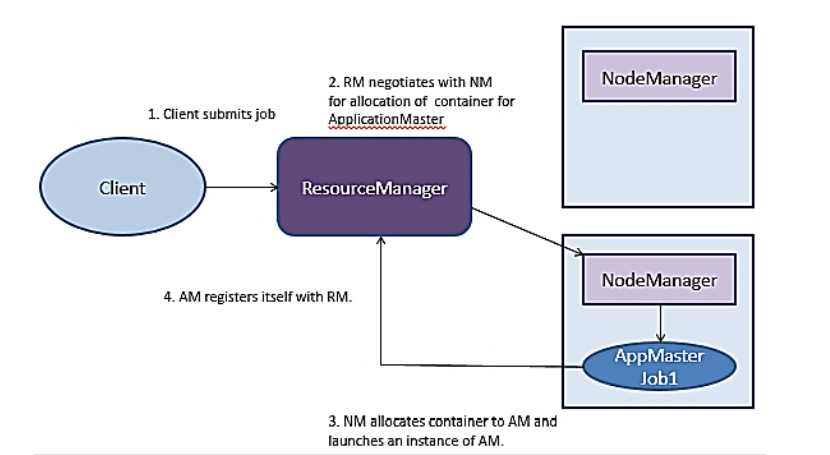
YARN separates all these function in separate daemon. YARN was introduced in Hadoop 2.x version which was brought I picture to overcome the limitation Hadoop 1.x which was Job Tracker overloading. YARN supports for other processing framework like Message Passing Interface, Graph Processing and many more such frameworks. It supports 50 processing frameworks. YARN to address the overloading issue of Job Tracker in Hadoop 1.x also divides the job tracker responsibilities of Hadoop 1.x into same separate components.

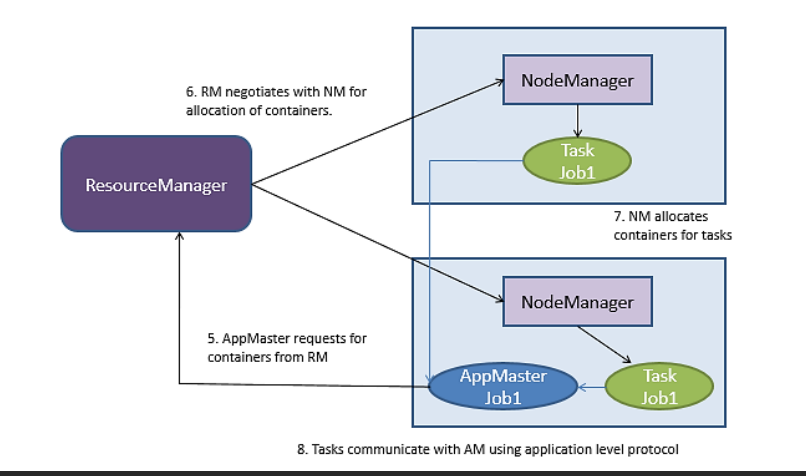
YARN is a latest management and processing framework of Hadoop. It was created by dividing the processing engine of Hadoop into smaller, more manageable parts. It also monitors and manages jobs submitted by user in a highly performance efficient manner.

YARN has five components-

1. **Resource manager-** Resource Manager is the Master of the processing part but it is different from master daemon because master daemon is a heavy duty object. It assign resources among applications for optional resource utilization. One cluster has one instance of resource manager.
2. **Node manager-** Node manager is the slave, when it starts, it announce itself to the Resource manager by means of Heart beat. It runs on each node and receives request from Resource Manager about resource allocation and maintain life cycle of containers.
3. **Application master-** Application Master is one per job, it orchestrates the job as soon as it becomes alive. Its temporary component and not a daemon. Actual instance is application master for resource and works with node manager to get those resources for task execution. Application master could be map reduce or any other processing framework.
4. **Scheduler-** Scheduler is a daemon, keeps the repository of all the containers on cluster. As a container comes into picture, it is register in scheduler. It is plugged with Resource Manager to help in resource allocation. Different schedulers allocates resources using different algorithms.
5. **Container-** Container is an abstraction sand runtime on commodity machine which allows the program to run a program unit. It is a set of allocated system resources. Containers are allocated and managed by node manager and are used by tasks. Containers depends on core of your machine like 4 containers in a quad core machine.
6. **Application Manager-** It is the master of all the application masters. It controls application master, if application master crashes, application master will go to scheduler and ask for a new container to run the process.

**YARN Architecture-**





**Steps of YARN-**

**Step 1:** Job/Application (which can be MapReduce, Java/Scala Application, DAG jobs like Apache Spark etc.) is submitted by the YARN client application to the Resource Manager daemon along with the command to start the Application Master on any container at Node Manager

**Step 2:** Application Manager Process on Master Node validates the job submission request and hands it over to Scheduler process for resource allocation

**Step 3:** Scheduler process assigns a container for Application Master on one slave node

**Step 4:** Node Manager Daemon starts the Application Master Service within one of its container using the command mentioned in Step 1, hence Application Master is considered to be the first container of any application

**Step 5:** Application Master negotiates the other containers from Resource Manager by providing the details like location of data on slave nodes, required cpu, memory, cores etc. 41 Overview (Contd.)

**Step 6:** Resource Manager allocates the best suitable resources on slave nodes and responds to Application Master with node details and other details

**Step 7:** Then, Application Master Send requests to Node Managers on suggested slave nodes to start the containers

**Step 8:** Application Master then manages the resources of requested containers while job execution and notifies the Resource Manager when execution is completed

**Step 9:** Node Managers periodically notify the Resource Manager with the current status of available resources on the node as to what information can be used by scheduler to schedule new application on the clusters

**Step 10:** In case of any failure of slave node, Resource Manager will try to allocate new container on other best suitable node so that Application Master can complete the process using new container