Al-Mustaqbal University
College of Sciences
Intelligent Medical System Department

جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الانظمة الطبية الذكية

Lecture (6): YARN (Yet Another Resource Negotiator)

Subject: Big Data Analysis in Healthcare
Level: Fourth
Lecturer: Asst. Lecturer Qusai AL-Durrah
Duration: Two hours

1. Introduction

YARN, short for **Yet Another Resource Negotiator**, represents the *second-generation* Hadoop architecture that separates computation management from resource management.
It was introduced in **Hadoop 2.0** to overcome the scalability, fault-tolerance, and flexibility limitations of the original MapReduce v1 system.

In MapReduce v1, a single component — the **JobTracker** — was responsible for both resource allocation and job coordination. This centralized control caused system bottlenecks and made large-scale cluster management inefficient. YARN solves this problem by introducing a **modular and distributed design** where each application manages its own execution, while a global **ResourceManager** coordinates resources across the cluster.

In the context of healthcare big data, YARN enables multiple analytic workloads — such as machine learning, streaming analysis, and patient data mining — to share the same cluster resources efficiently and safely.

2. Learning Objectives

By the end of this lecture, students should be able to:

1. **Understand**: the motivation behind the development of YARN.
2. **Identify:** YARN's main architectural components and their roles.
3. **Describe:** the YARN execution workflow in detail.
4. **Compare:** YARN with the original MapReduce model.
5. **Explain:** how YARN supports multi-framework environments (MapReduce, Spark, Tez, etc.).
6. **Apply:** YARN concepts to healthcare data-processing scenarios.

3. Evolution from Classic MapReduce to YARN

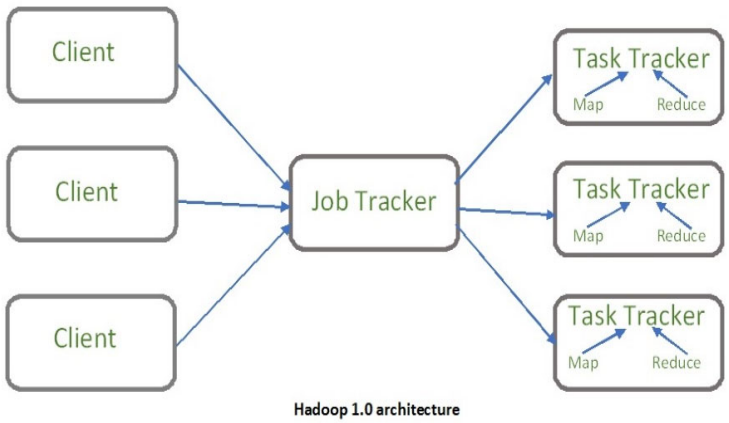
3.1 Limitations of MapReduce v1

The traditional Hadoop MapReduce architecture (also called MRv1) relied on two daemons:

- **JobTracker** (Master)
- **TaskTracker** (Worker)

While effective for small clusters, this model struggled in large-scale environments due to:

- **Scalability bottlenecks:** The single JobTracker could not manage thousands of concurrent tasks.
- **Lack of flexibility:** Only MapReduce jobs could run — other processing frameworks (e.g., graph processing or streaming) were not supported.
- **Resource fragmentation:** Static "map" and "reduce" slots caused inefficient utilization.
- **Single point of failure:** If the JobTracker failed, all jobs in progress were lost.



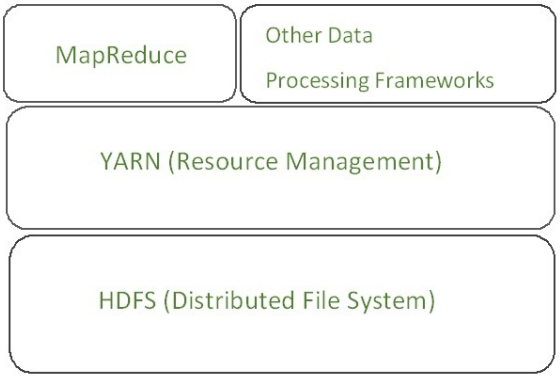
Hadoop 1.0 architecture

3.2 YARN's Design Philosophy

YARN rearchitected Hadoop into a **general-purpose data-processing platform** capable of running *multiple application types* on the same cluster. The responsibility of Job tracker is split between the ResourceManager and ApplicationMaster.

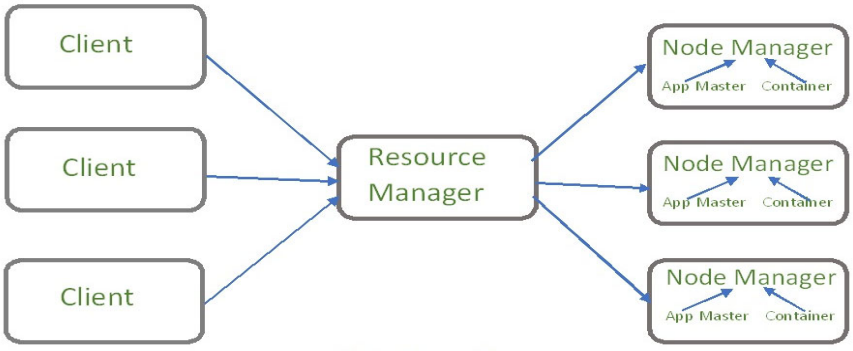
- **Resource Management (by ResourceManager)**
- **Job Execution (by ApplicationMaster)**

This design makes Hadoop flexible enough to support new engines such as **Spark**, **Hive on Tez**, **Flink**, and **Storm**, all operating under YARN's unified resource control



Hadoop 2.0

4. YARN Core Components



Hadoop Yarn architecture

4.1 Client

The Client is the entity that initiates and submits the application (such as a MapReduce job) to the YARN framework. It communicates with the Resource Manager to request execution, monitors the job status, and can also interact with the Application Master for progress updates. It essentially acts as the user's interface to launch and manage applications on the Hadoop cluster.

4.2 ResourceManager (RM)

It is the master daemon of YARN and is responsible for resource assignment and management among all the applications. Whenever it receives a processing request, it forwards it to the corresponding node manager and allocates resources for the completion of the request accordingly. It has two major components:

- **Scheduler:** It performs scheduling based on the allocated application and available resources. It is a pure scheduler; means it does not perform other tasks such as monitoring or tracking and does not guarantee a restart if a task fails. The YARN scheduler supports plugins such as Capacity Scheduler and Fair Scheduler to partition the cluster resources.
- **Application manager:** It is responsible for accepting the application and negotiating the first container from the resource manager. It also restarts the Application Master container if a task fails.

4.2 NodeManager (NM)

It takes care of individual node on Hadoop cluster and manages application and workflow and that particular node. Its primary job is to keep-up with the Resource Manager. It registers with the Resource Manager and sends heartbeats with the health status of the node. It monitors resource usage, performs log management and also kills a container based on directions from the resource manager. It is also responsible for creating the container process and start it on the request of Application master.

In other words; Each data node runs a **NodeManager** responsible for:

- Managing containers (the execution units).
- Monitoring resource usage (CPU, memory, disk, network).
- Sending **heartbeats** to the ResourceManager.
- Launching and cleaning up containers as instructed by ApplicationMasters.

If the RM is the "administrator," the NMs are the "department managers" ensuring that each ward or machine operates within resource limits.

4.3 ApplicationMaster (AM)

An application is a single job submitted to a framework. For each job, a dedicated **ApplicationMaster** is launched within a container. It:

- Negotiates resources from the RM.
- Schedules and monitors tasks across multiple NMs.
- Handles task failures and restarts if necessary.

The application master requests the container from the node manager by sending a Container Launch Context (CLC) which includes everything an application needs to run. Once the application is started, it sends the health report to the resource manager from time-to-time.

After the job finishes, the AM shuts down, freeing its resources. This decentralized control eliminates the single point of failure that existed in MapReduce v1.

4.4 Containers

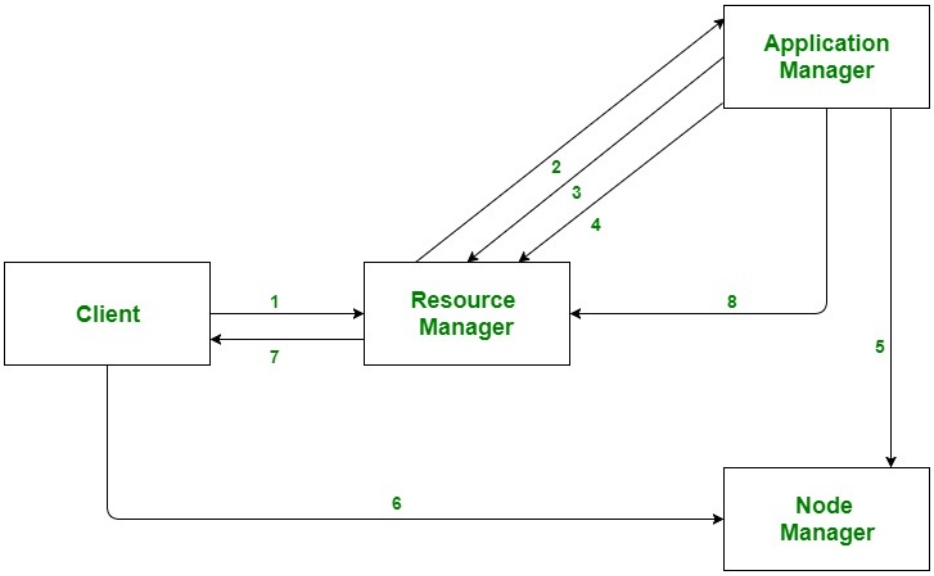
Containers are the fundamental **resource units** in YARN, representing allocated bundles of CPU cores, memory, and storage on a NodeManager. They are lightweight, isolated environments (similar to mini virtual machines) where

individual tasks execute.
The AM decides how many containers are required and requests them dynamically from the RM. The containers are invoked by Container Launch Context(CLC) which is a record that contains information such as environment variables, security tokens, dependencies etc.

5. YARN Execution Workflow

Step-by-Step Flow:

1. **Job Submission:**
   The client submits an application (e.g., MapReduce) to the ResourceManager.
2. **ApplicationMaster Launch:**
   RM allocates the first container on an available NodeManager and launches the AM within it.
3. the Application Manager registers itself with the Resource Manager
4. **Resource Negotiation:**
   The AM requests additional containers from the RM according to job requirements.
5. **Node Manager notification:**
   The Application Manager notifies the Node Manager to launch containers.
6. **Task Execution:**
   NodeManagers start containers and execute assigned tasks.
7. **Progress Monitoring:**
   Client contacts Resource Manager/Application Manager to monitor application's status. The AM monitors each task, manages retries, and reports progress back to the client via RM.
8. **Job Completion:**
   After all tasks finish, the AM notifies the RM and terminates. RM then releases all containers.



6. Scheduling in YARN

YARN supports several scheduling policies to manage how resources are distributed among users or departments:

Scheduler Type	Description	Use Case
FIFO Scheduler	First-come, first-served; simple but unfair.	Small test clusters.
Capacity Scheduler	Divides resources into queues with minimum guaranteed capacities.	Multi-tenant institutions like hospitals or universities.
Fair Scheduler	Allocates unused resources evenly among active jobs.	Dynamic environments with mixed workloads.

Schedulers ensure fairness and priority control — e.g., assigning higher capacity to time-critical medical analytics while keeping research jobs running at lower priority.

7. YARN Cluster Example

Component	Example in Healthcare Cluster
ResourceManager	Allocates containers for different research departments (e.g., genomics, imaging)
NodeManager	Runs on each compute node in the hospital data center
ApplicationMaster	Manages a specific job (e.g., MRI image classification)
Containers	Execute task units (e.g., individual patient image analysis)

8. YARN vs. Classic MapReduce

Feature	MapReduce v1	YARN (MRv2)
Job Management	Single JobTracker	Per-job ApplicationMaster
Resource Control	Static slots	Dynamic containers
Fault Tolerance	Limited	Improved, distributed
Framework Support	MapReduce only	Multiple (Spark, Tez, Flink, etc.)
Scalability	Limited to ~4000 nodes	Scales to 10,000+ nodes

9. YARN Advantages & Disadvantages

9.1 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.

9.2 Disadvantages

- **Complexity:** YARN adds complexity to the Hadoop ecosystem. It requires additional configurations and settings, which can be difficult for users who are not familiar with YARN.
- **Overhead:** YARN introduces additional overhead, which can slow down the performance of the Hadoop cluster. This overhead is required for managing resources and scheduling applications.
- **Latency:** YARN introduces additional latency in the Hadoop ecosystem. This latency can be caused by resource allocation, application scheduling, and communication between components.

- **Single Point of Failure:** YARN can be a single point of failure in the Hadoop cluster. If YARN fails, it can cause the entire cluster to go down. To avoid this, administrators need to set up a backup YARN instance for high availability.
- **Limited Support:** YARN has limited support for non-Java programming languages. Although it supports multiple processing engines, some engines have limited language support, which can limit the usability of YARN in certain environments.

10. YARN Advantages & Disadvantages

YARN redefines Hadoop as a **multi-application resource manager** rather than just a MapReduce system. By separating resource negotiation and job management, it enables **efficient resource sharing, flexible workload management, and high scalability** — all of which are essential for healthcare big data systems that integrate batch, streaming, and AI-driven analytics.