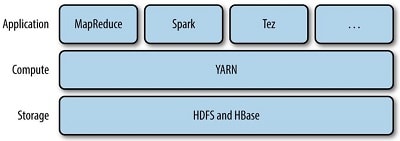
**BWhat is YARN?**

Apache YARN (**Yet Another Resource Negotiator**) is Hadoop’s cluster resource management system. YARN was introduced in Hadoop 2 to improve the MapReduce implementation, but it is general enough to support other distributed computing paradigms as well.

**YARN** provides APIs for requesting and working with cluster resources, but these APIs are not typically used directly by user code. Instead, users write to higher-level APIs provided by distributed computing frameworks, which themselves are built on YARN and hide the resource management details from the user.

Below figure shows some distributed computing frameworks (MapReduce, Spark,and so on) running as YARN applications on YARN.



**IIs YARN a replacement of Hadoop MapReduce?**

**YARN** is not a replacement of Hadoop but it is a more powerful and efficient technology that supports MapReduce and other distributed computing frameworks and is also referred to as MapReduce version 2.

**AWhy YARN?**

With older versions of Hadoop, you were limited to executing **MapReduce jobs** only. This was great if the type of work you were performing fit well into the MapReduce processing model, but it was restrictive for those wanting to perform graph processing, iterative computing, or any other type of work.

In Hadoop 2 the scheduling pieces of MapReduce were separated and reworked into a new component called YARN. The YARN doesn’t know or care about the type of applications that are running, nor does it care about keeping any historical information about what has executed on the cluster. Because of this design YARN can scale beyond the levels of MapReduce.

**IWhat are the YARN responsibilities?**

YARN is responsible for two activities:

* Responding to a client’s request to create a container(A container is in essence a process, with a contract governing the physical resources that it’s permitted to use).
* Monitoring containers that are running, and terminating them if needed(Containers can be terminated if a YARN scheduler wants to free up resources so that containers from other applications can run, or if a container is using more than its allocated resources).

**IWhat are the benefits YARN brings in to Hadoop?**

* Yarn does efficient utilization of the resource. There are no more fixed map-reduce slots. YARN provides central resource manager. With YARN, you can now run multiple applications in Hadoop, all sharing a common resource.
* YARN is backward compatible. This means that existing MapReduce job can run on Hadoop 2.0 without any change.
* YARN fixed the old mapreduce scalability issue and can now run on larger clusters than MapReduce 1.
* The biggest benefit of YARN is that it opens up Hadoop to other types of distributed application beyond MapReduce. MapReduce is just one YARN application among many.

**AExplain how scalability issue is fixed in YARN?**

In YARN, (Contrast to the jobtracker in MapReduce 1), instance of an application say a MapReduce job has a dedicated application master, which runs for the duration of the application. This model is actually closer to the original Google MapReduce paper, which describes how a master process is started to coordinate map and reduce tasks running on a set of workers.

MapReduce 1 hits scalability bottlenecks in the region of 4,000 nodes and 40,000 tasks, because of the reason that the jobtracker has to manage both jobs and tasks. YARN overcomes these limitations by virtue of its split resource manager/application master architecture which is designed to scale up to 10,000 nodes and 100,000 tasks.

**BWhat are the key components of YARN?**

The basic idea of YARN is to split the functionality of resource management and job scheduling/monitoring into separate daemons. YARN consists of the following different components

* **ResourceManager** - The ResourceManager is the YARN master process, and its sole function is to arbitrate resources on a Hadoop cluster. It responds to client requests to create containers, and a scheduler determines when and where a container can be created
* **NodeManager** - The NodeManager is the slave process that runs on every node in a cluster. Its job is to create, monitor, and kill containers. It services requests from the ResourceManager and ApplicationMaster to create containers, and it reports on the status of the containers to the ResourceManager.
* **ApplicationMaster** - ApplicationMaster is a per-application component which doesn’t perform any application-specific work, as these functions are delegated to the containers. Instead, it is responsible for negotiating resource requirements for the resource manager and working with NodeManagers to execute and monitor the tasks.

The ApplicationMaster is also responsible for the specific fault-tolerance behavior of the application. It receives status messages from the ResourceManager when its containers fail, and it can decide to take action based on these events (by asking the ResourceManager to create a new container), or to ignore these events.

* **Container** - A container is an application-specific process that’s created by a NodeManager on behalf of an ApplicationMaster with a constrained set of resources (Memory, CPU, etc.)
* **YARN child** - After submitting the application, application master dynamically launch YARN child to do the MapReduce tasks.

**IWhat is ResourceManager in YARN?**

The ResourceManager is the YARN master process, and its only function is to arbitrate resources on a Hadoop cluster. It responds to client requests to create containers, and a scheduler determines when and where a container can be created.

The ResourceManager has two main components - Scheduler and AppicationsManager

* **Scheduler** - The scheduler is responsible for allocating resources.
* **ApplicationManager** - The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure.

**IWhat is ApplicationMaster in YARN?**

ApplicationMaster is a per-application component which doesn’t perform any application-specific work, as these functions are delegated to the containers. Instead, it is responsible for negotiating resource requirements for the resource manager and working with NodeManagers to execute and monitor the tasks.

The ApplicationMaster is also responsible for the specific fault-tolerance behavior of the application. It receives status messages from the ResourceManager when its containers fail, and it can decide to take action based on these events (by asking the ResourceManager to create a new container), or to ignore these events.

**BWhat are the scheduling policies available in YARN?**

YARN scheduler is responsible for scheduling resources to user applications based on a defined scheduling policy. YARN provides three scheduling options-

* **FIFO Scheduler** - FIFO scheduler puts application requests in queue and runs them in the order of submission (first in, first out). Requests for the first application in the queue are allocated first; once its requests have been satisfied, the next application in the queue is served, and so on.
* **Capacity Scheduler** - Capacity scheduler has a separate dedicated queue for smaller jobs and starts them as soon as they are submitted.
* **Fair Scheduler** - Fair scheduler dynamically balances and allocates resources between all the running jobs. Just after the first (large) job starts, it is the only job running, so it gets all the resources in the cluster. When the second (small) job starts, it is allocated half of the cluster resources so that each job is using its fair share of resources.

**AHow do you setup ResourceManager to use CapacityScheduler?**

You can configure the ResourceManager to use **CapacityScheduler** by setting the value of property yarn.resourcemanager.scheduler.class to org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler in the file **conf/yarn-site.xml**.

**AHow do you setup ResourceManager to use FairScheduler?**

You can configure the ResourceManager to use **FairScheduler** by setting the value of property yarn.resourcemanager.scheduler.class to org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler in the file **conf/yarn-site.xml**.