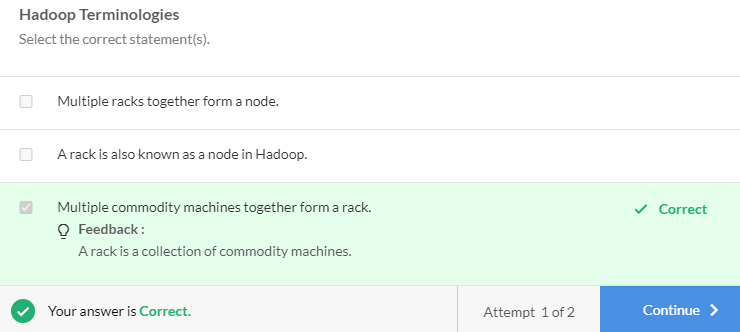
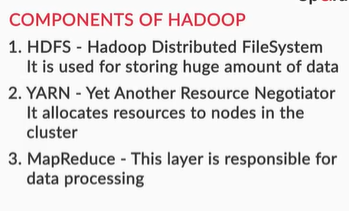
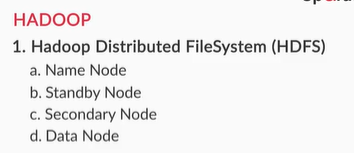


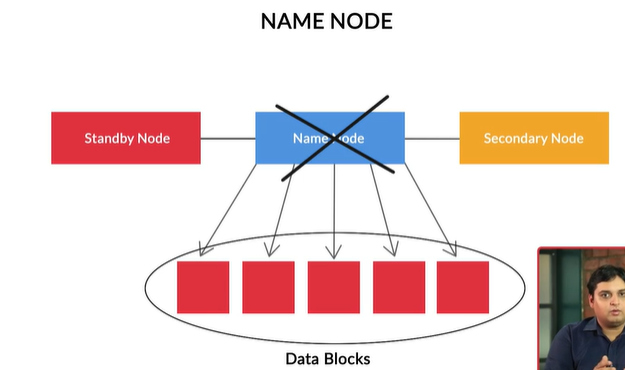
So these are the key terminologies in Hadoop:

* **Block:** The data is divided into smaller chunks called blocks.
* **Commodity machine** **or** **node:** This is a machine without any special hardware to process huge amounts of data, such as a personal computer. A commodity machine/node stores the data.
* **Rack:** This is a collection of commodity machines/nodes.
* **Data centres:** These are physical locations where multiple racks are stored together. A group of racks makes a data centre.
* **Cluster:**This is a collection of data centres. It can also be the case that only a certain rack in the data centre is a part of the cluster. So in other words, a cluster has many nodes which are part of a rack, which is a part of a data centre.
* **Client node:** This is used to read and write data in the Hadoop cluster. This special type of node is not a part of the cluster and does not store blocks.



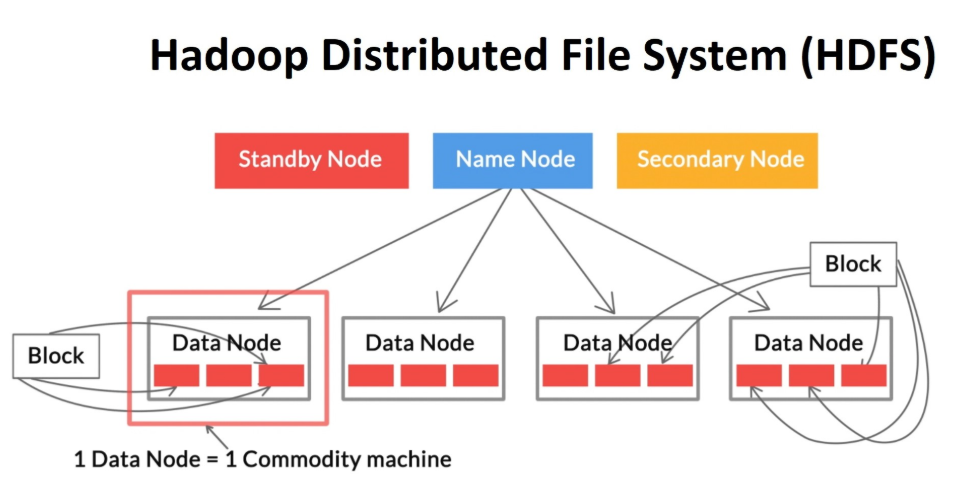


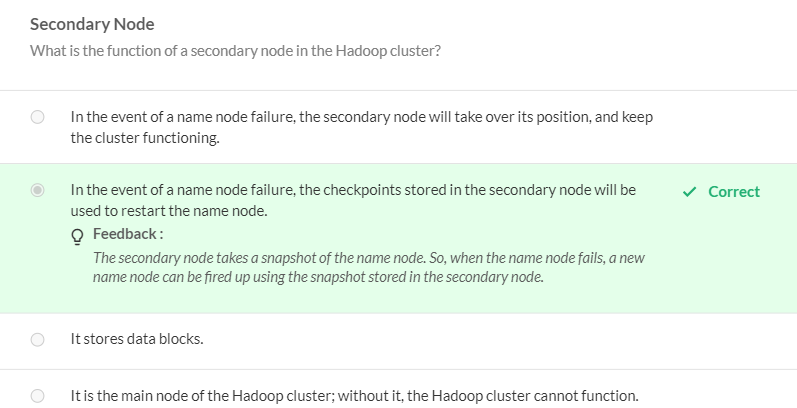


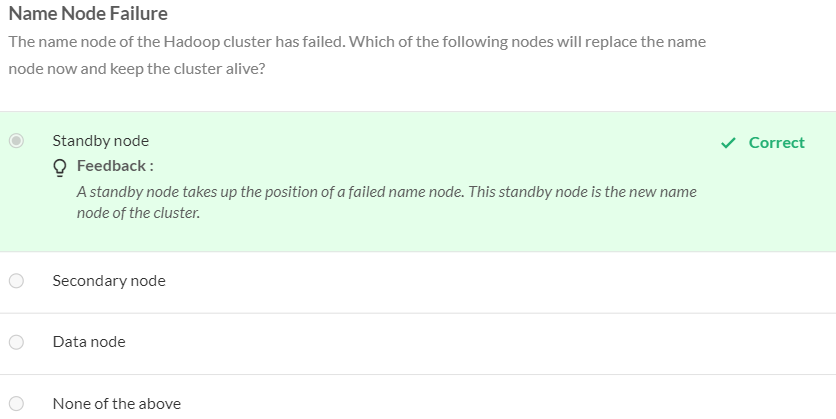


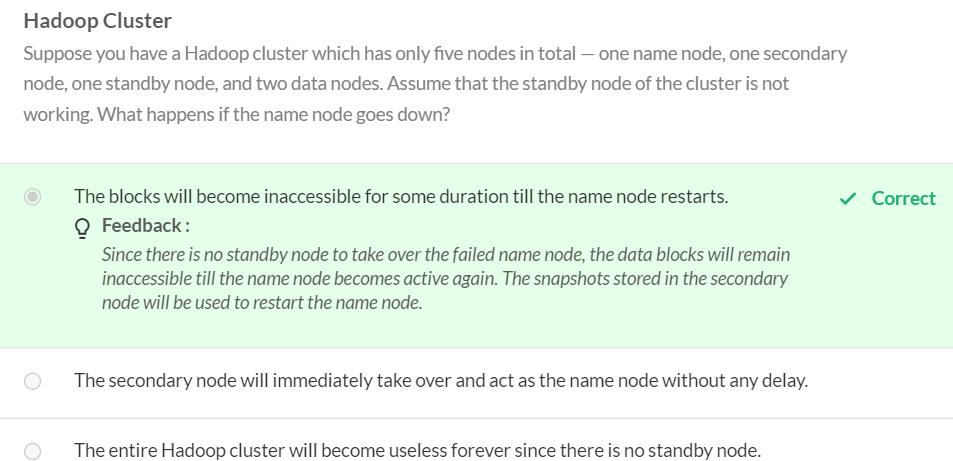
To summarise, apart from the name node, there are three other types of nodes:

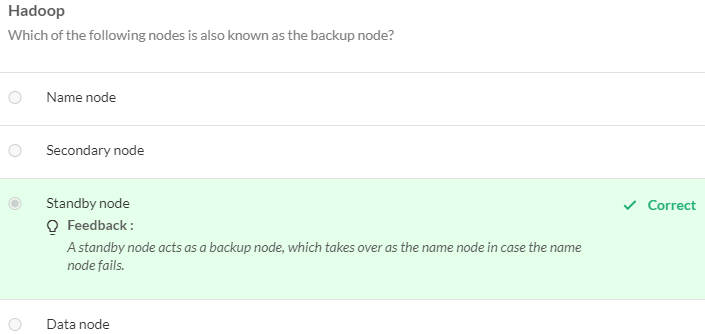
1. **Secondary node**: This node keeps an image (or a snapshot) of the metadata that is stored in the name node. It periodically takes this snapshot, which is then used later in the case of a name node failure, to restart the name node or to start a new one.
2. **Standby node:** This node replaces the name node if it fails and is also known as the backup node.
3. **Data node**: The data node stores data blocks. In a Hadoop cluster, many data nodes are available. They receive instructions from the name node and report back to it.

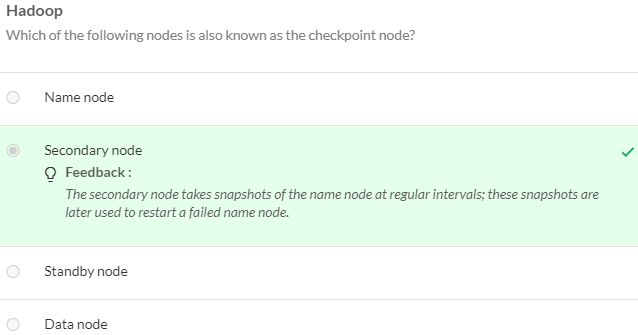












To avoid this problem of data loss, Hadoop has a concept of a **block replica.** When the data is being divided into blocks for storage on the HDFS, instead of storing a block of data on just one node, **each block is replicated a specific number of times**. This way, more than one block replica is then stored on the HDFS on multiple nodes.

But how many replicas should you create? This number is specified by what is known as the **replication factor.**

**The replication factor** is the number of replicas you want for each block in the cluster. If it is four, a total of four blocks — including the original block —will be stored in the Hadoop cluster. A file that would have ideally occupied 100 GB of space in the cluster will now require 400 GB for storage.

Consider a Hadoop cluster which has one name node, one standby node, one secondary node, and multiple data nodes. Now suppose a client node wants to write a file in the HDFS.

It will first contact the name node and request the location of the data nodes on which these blocks should be stored. For each block, the name node will provide the client node with a list containing the addresses of the data nodes where the block has to be copied. The client node will then copy the data blocks onto the data nodes assigned by the name node. Let’s assume that the replication factor is four. Each block will then be stored at four different locations in the Hadoop cluster.

So, what is the **advantage of storing multiple copies of each block**in the cluster? If a block gets corrupted or is inaccessible, you **can access copies of this block stored elsewhere**in the Hadoop cluster.

However, there's also a **tradeoff between safety and speed**: you want to access the replicated blocks as quickly as possible while keeping the time spent on transporting the blocks to a minimum.

Considering this tradeoff, you have to make a choice: should all the blocks be**placed on one data node**, or should you **distribute them among different data nodes**? What happens if you keep all the blocks on the same data node and that data node goes down?

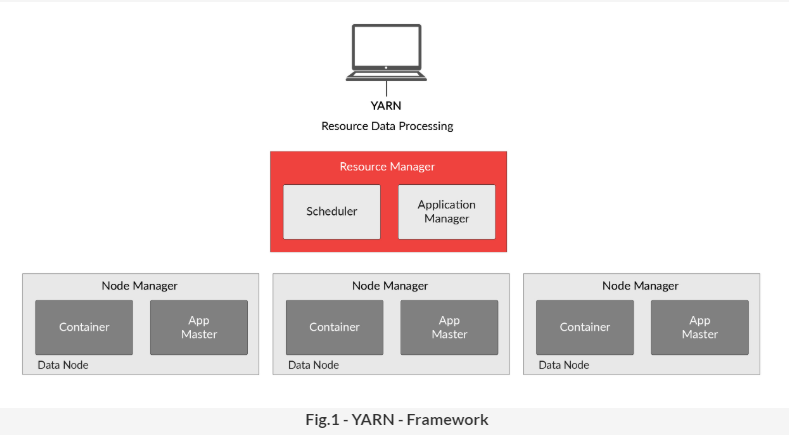
It is built on the concept of **distributed computing** according to which, it first divides the incoming big data (file) into multiple chunks called **blocks**. These blocks are stored on **data nodes**. The location of each block on the data nodes, or metadata, is stored in the **name node**.

If the name node fails or gets corrupted, the whole cluster becomes inaccessible. This is because only the name node knows where all the blocks are kept. You have two nodes in the cluster to handle **name node failure**. One is the **standby node**, which **replaces the failed name node**(almost immediately) and brings the cluster back up and running. The other node is the **secondary node**, which **stores a snapshot**of the information stored in the name node, which can be used to start a new name node in case of absence of any backup nodes.

You also looked at the concept of a **block replica**. Each block in the cluster is replicated multiple times so that in case a block gets corrupted, you have the same block stored somewhere else in the cluster as a backup.

YARN:

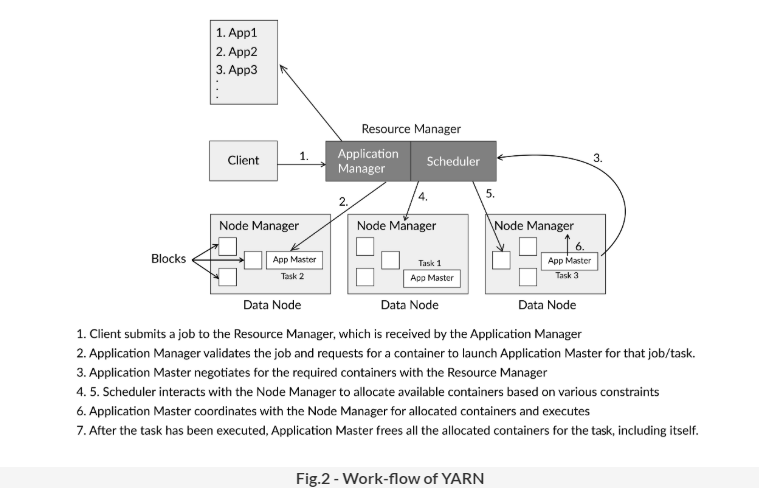
YARN’s function is to allocate resources, i.e. disk space, memory, network etc., for data processing in the cluster. After a job is divided into tasks (or sub-jobs), the resource manager in YARN allocates resources to execute the tasks. In earlier versions of Hadoop, MapReduce used to handle the resource management as well as the processing. YARN was initiated as a project to decouple the two functions.



Coming to the structure of YARN, there are 4 components in it:

1. **Resource manager**: It looks after the resources of the entire cluster. The slave machines run multiple tasks which require different resources for execution, such as memory (RAM), CPU, etc. These resources are allocated by the resource manager.
2. **Node manager**: The node manager keeps track of the resources running on the slave machine (one data node) it is residing on. It manages the resources allocated for each of the applications (also known as tasks) inside the node and communicates the status of the resources to the resource manager.
3. **Application Master**: A job is subdivided into multiple tasks. Each task is individually managed by the application master. It looks after the lifecycle of the task and sends timely requests to the resource manager for the required containers.
4. **Containers**: To perform each task, the application master will require different components like memory (RAM) or CPU or storage disks. These components are collectively termed as containers and are managed by the node manager. These containers are the actual place where the task is performed.

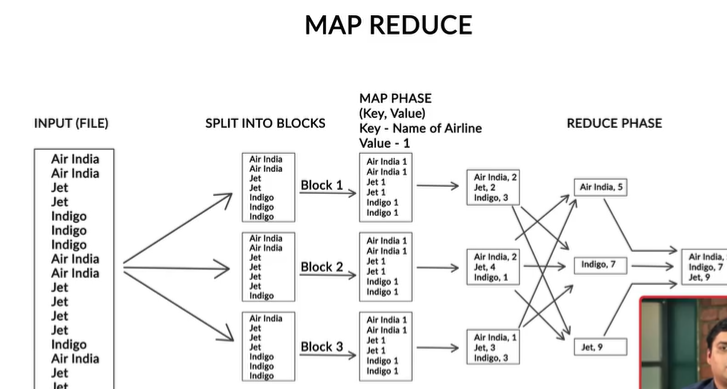
Before going ahead, please be aware that Resource Manager in YARN has 2 sub-components: **Scheduler** and **Application Manager**. The scheduler is required to allocate the required containers to run an application based on multiple constraints or capacity of the system. Applications Manager, on the other hand, receives the tasks (also called applications) that are submitted to the cluster by the client and keeps a track of all the applications running on the resource node.

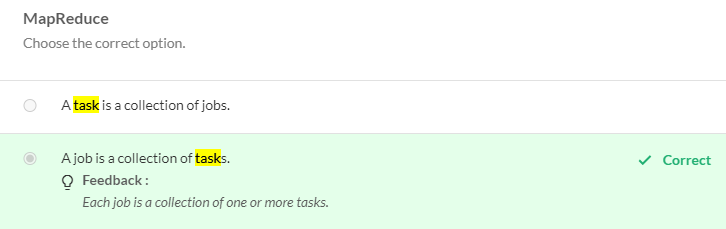


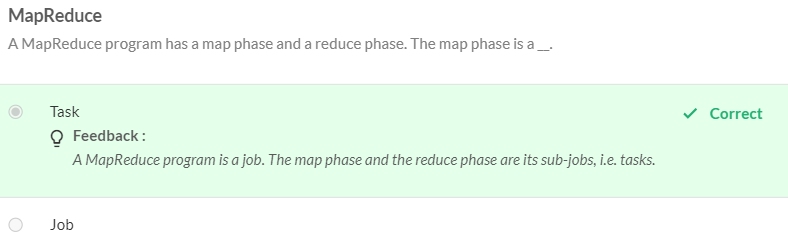
To process the data as required by the code, you want both the code and the data to be in the same location, to avoid any delay in processing. You have two options, either move the code to the location of the data or move the data to the location of the code. So which of the two do you move?

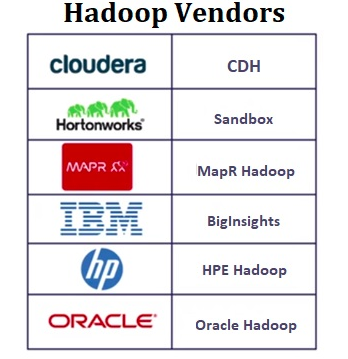
The data is generally in gigabytes, whereas the code is typically a few kilobytes. So, it is less expensive to transport the code to the location of the data, as opposed to shifting the data to the location of the code.

This is known as the **Principle of Data Locality**. Thus, in Hadoop, the code is transported to the location of the data, rather than vice versa.

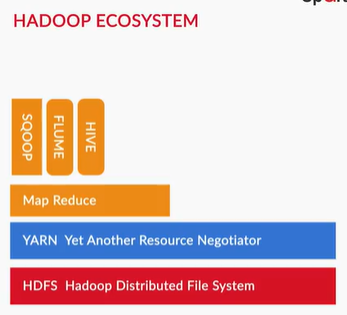








To run Hadoop, you may need a cloud service so that your local machine or commodity hardware can be connected to a cluster of nodes or machines. Some of the common cloud platform providers include AWS EMR — which is Amazon Web Services’ Elastic MapReduce — and Microsoft's Azure.



* **Sqoop and Flume**: These tools are designed to efficiently transfer bulk data between Hadoop, RDBMSs, and real-time streaming data sources. Sqoop is used to transfer data from an RDBMS to the HDFS. Likewise, if you are storing data in real time and want to store it in the HDFS, you would need Flume. So, Flume is used when you have a continuous flow of data and want to store it in the HDFS.
* **Hive**: It helps you access and analyse data in Hadoop with a query language such as SQL. HIVE-QL is the query language introduced for this purpose, which is similar to SQL.

YARN has **four**major components — Resource manager, Node manager, Application master and Containers.

The resource manager is located on a master machine, whereas the node manager, the application master and containers are located on the slave machines.

Consider that the data is stored in the form of blocks on the data nodes 1,2, and 5.

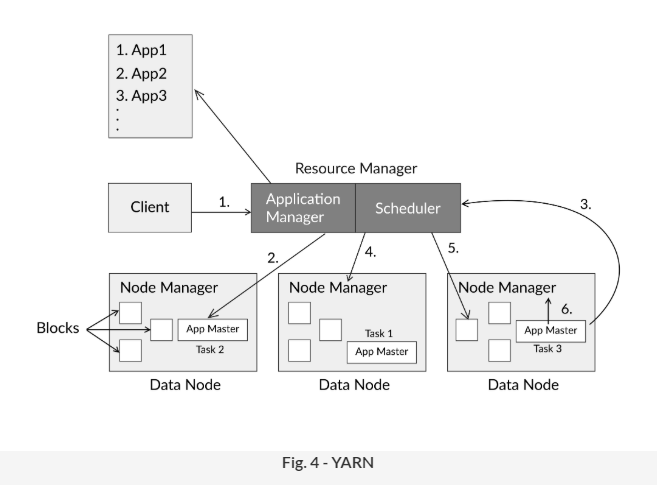
Let’s now look at how the YARN components interact. The **client** submits the job to the **resource manager**, which is accepted by the application manager present in it. The application manager checks if the requested task can be fulfilled by the resources of cluster or not. If not, then it rejects the task. In case the request is accepted, it moves ahead with the task of launching the **application master** for that task with the help of the scheduler.

It is always advised to keep the code file close to the data for faster processing. Hence, to get the location of the relevant data blocks, the scheduler contacts the name node. The name node replies with the location of the data nodes where the data required by the code is stored. In our case, the name node will reply that the data is stored on data nodes 1,2, and 5.

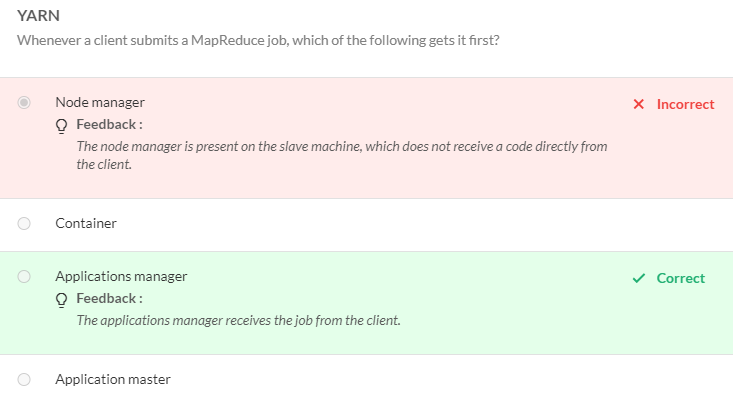
The scheduler will interact with the respective **node manager** for the available resources to launch an application master in the same node. If the resources are unavailable in that node, it will provide another node to the application manager for launching the application master.

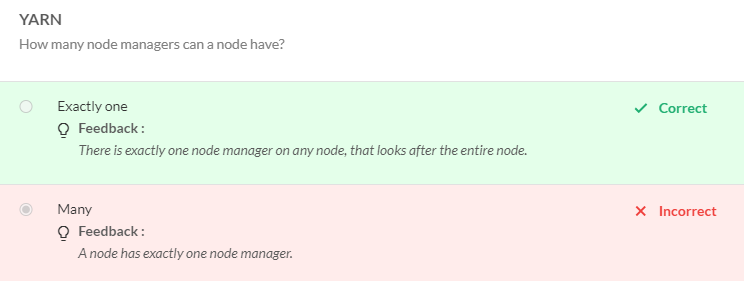
The status of the task is reported by the application master. The main task of the application master is to negotiate the resources (memory/CPU/network etc.) from the resource manager and then execute the task assigned to it using them.

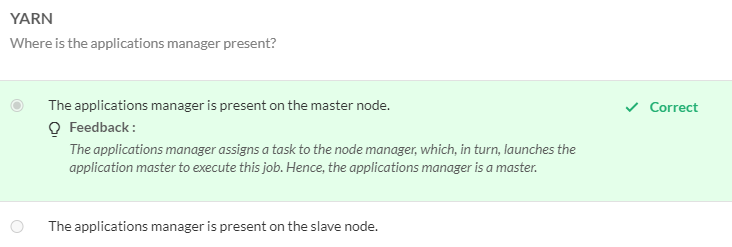
The resources provided to an application master to execute a task are known as **containers**. These containers are provided to the application master to run an application/task. You have exactly one node manager managing all the resources in a node. The node manager continuously reports the condition of the node to the resource manager, specifically the CPU and RAM usage of that node, the node’s health, etc.

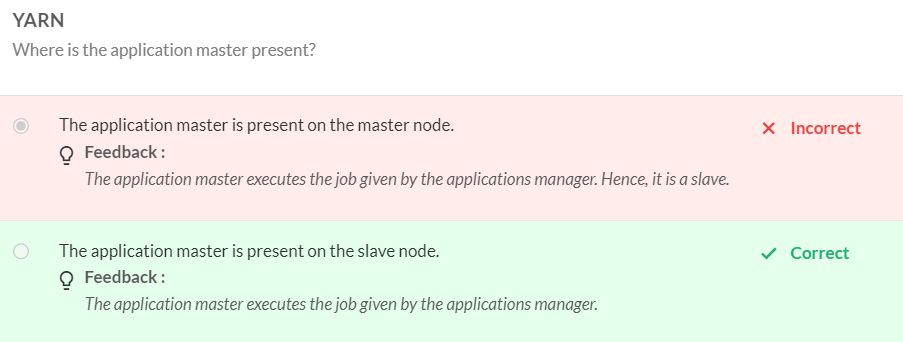


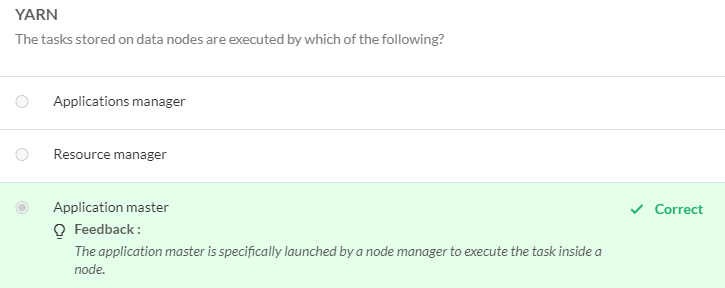
Remember that a client submits a MapReduce job to the application manager. The MapReduce job can have one or multiple mapper and reducer codes. Let’s assume that the MapReduce job has four sub-jobs, i.e. tasks. So, the node manager will launch exactly four application masters to process these codes, with each application master taking care of exactly one application. The application master then negotiates for containers and executes the task allotted to it. After completion, the requested containers and the application master free the occupied resources.











**YARN**, Yet Another Resource Negotiator, does the resource management for the Hadoop cluster.

It has four main components:

1. Resource Manager
2. Node Manager
3. Application Master
4. Containers

A **resource manager** manages the resource allocation in the entire Hadoop cluster and a**node manager** tracks the resource usage in a node. The node manager keeps track of all the resources available on a node. The resource manager has a component called the applications manager that receives the job from a client. Once the node receives the job, corresponding **application master**(s) are launched to execute that job. You have one application master per task inside a node. The application master(s) negotiate with the resource manager for the **containers**(resources like memory, processing units, etc.) to execute the task.

**MapReduce** is a programming framework which is used by Hadoop to process 'big data'.

MapReduce has two components: a map phase and a reduce phase. The function of the **map phase** is to convert the incoming data (stored in blocks) into **key-value** pairs. These pairs are then fed into the **reduce phase**. The function of the reduce phase is to aggregate these values on the basis of the keys across ALL the blocks in the cluster.