**HADOOP ARCHITECTURE**

Hadoop follows distributed architecture.

Types of distributed architectures:

1. Shared architecture

All data is accessible from all cluster nodes. Any machine can read or write any portion of data it wishes.

1. Shared nothing architecture

Each node will be processing its local disk data. No need to fetch from other sources.

Each machine has sole access, and hence sole responsibility, for the data it holds. It does not share responsibility with other machines.

**Hadoop architecture of Hadoop 1/MR1**

It is controlled and maintained by five daemons (components). Daemon is a sub-system which keep on running. Five daemons are-

**Master daemons (All communicate with each other)**

NAMENODE

JOBTRACKER

SECONDARY NAMENODE

**Slave daemons (All communicate with each other)**

DATANODE

TASKTRACKER

Here Namenode, Datanode and Secondary Namenode are called as HDFS/Storage daemons.

Jobtracker and Tasktracker are called MapReduce/Processing daemons.

**PROCESS TO STORE DATA IN HDFS:**

Namenode- Responsible for storage and controls entire cluster.

* When HDFS storage request is submitted by client,

eg $ Hadoop fs –copyFromLocal file1.txt myHDFSdir

* This request is sent to NameNode (NN). It will fetch file size from client. Eg 100MB
* The NN will contact configuration file for block size configured and number of replicas configured. Eg Block size = 64MB and Replicas = 3
* NN will pass this info to client and datanodes (DN) and provide client, datanode info where the blocks shall be stored.

**Architecture while storing data in HDFS:**



* First DN will divide the file into blocks including replicas after receiving the file from client.

Eg. Block1 (B1)🡪 3 times

Block2 (B2)🡪 3 times

* NN will choose the right machines for saving these file blocks. Decision making criteria used while DN selection is-

1. High configuration
2. Nearest location
3. More free space

Eg. By this algorithm DNs selected are

1,3,5 --🡪 for block B1

2,4,1 --🡪 for block B2

* First NN will take decision for storing original copies, later for the duplicate copies.(Always first copy is original).
* This selective info is sent to first DN and storage order is sent to proper data nodes.
* Now first DN will distribute the data blocks to the selected DNs. In this selected list, first DN may also be the member.
* Once storage is done, NN will register metadata(info about data, ie file block metadata)
* While storing copies of the same block are not stored in the same datanode.
* Advantage of metadata – Saves time for seeking file blocks before processing.
* Hence NameNode contains-

1. Metadata
2. Configuration file
3. Physical addresses of nodes(IP address)
4. Edit log

Edit log will be useful at the time of recovery. Single point of failure (SPOF) is at namenode only. That means if namenode used is down, total cluster will be down.

Steps to be taken if namenode is down.

1. Replace hardware
2. Restore all related backed-up data
3. Restart the edit-log service (this will recognize the entire system configuration)

If any new node is added to the cluster or any node is removed, that info has to be updated in RACK AWARENESS table present in namenode.

Edit log will contain details like-

Timestamps of when the file was created, when the file was edited or replaced or removed.

**DATANODE-**

As per order of namenode, datanodes simply saves file blocks.

Once blocks are stored, it sends acknowledgement to namenode.

**PROCESS TO PROCESS DATA IN HDFS:**

**JOB TRACKER-**

* It is responsible for processing.
* Verifies metadata, sys conf from namenode about the I/P file.
* Divide the job into multiple tasks (smaller works).

Eg. If file her 3 blocks- 3 tasks are required as

Task 1 for B1 block

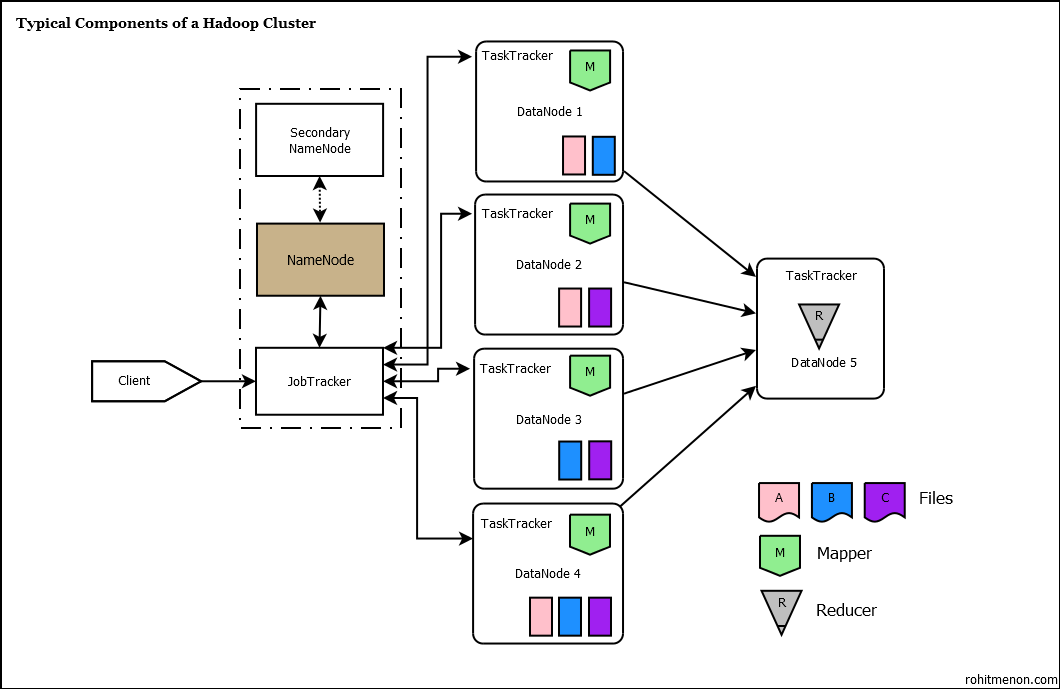
Task 2 for B2 and Task 3 for B3

* It has to take decision for work assignment of each task.

Eg. Task 1 is for B1 block.

B1 is available in 1,3 and 5 datanodes. Among 1,3 and 5 only one machine has to be selected for processing B1.

**Architecture while processing data using MAPREDUCE:**



**Decision criteria used by Jobtracker**:

1. High configuration.
2. Nearest location.
3. Which datanode is free(not busy with any task)
4. If all are free

Health thru-put ratio.

1. If health is good of all nodes, random selection is done.
2. If all are busy

Non-localized processing will be done.

**Fault Tolerance:**

Eg, if B2 task is assigned to system 2

If system 2 is down,

Task will be reinitiated and assigned to system 1 or system 4 because still replicas of block B2 is available in those systems.

Before selecting sys1 or sys4, again job tracker will apply decision criteria because previously sys1 or sys4 might be busy but may not be busy now.

**Workload balancing after work assignment:**

Job tracker estimated that the task on block B1 will be finished by sys3 in 5 mins, sys5 in 30 mins and by sys1 in 40 mins.

Now sys3 is preferred but lets say it is very busy.

Hence job tracker will assign the task to sys5.

After completion of 3mins processing on B1 by sys5, lets say sys3 got free.

Now job tracker will terminate the task at sys5 and reassign to sys3.

**Health Monitoring Service:**

Task tracker (TT) send heartbeat(signal) per 10 seconds to the job tracker (JT). This cannot be configured, its by default.

Job tracker decision interval = 10 mins by default (this can be configured)

Eg. TT heartbeat interval = 10 secs

JT decision interval = 2 mins

So from a healthy machine JT has to receive (2\*60)/10 = 12 beats in a decision interval.

If number of beats are < 12 -🡪 bad health of datanode

If number of beats are = 0 -🡪 datanode is dead

**TASK TRACKER:**

As per order of job tracker, TT will process specified blocks and send acknowledgement to job tracker.

Note : if task tracker is down, datanode is active for storage of data.

If datanode is down, complete machine is down.

**SECONDARY NAMENODE:**

The Secondary NameNode is not-a failover NameNode!

– It performs memory/intensive administrative functions for the NameNode

– NameNode keeps information about files and blocks (the metadata) in memory

– NameNode writes metadata changes to an edit log

– Secondary NameNode periodically combines a prior snapshot of the file system metadata and edit log into a new snapshot

– New snapshot is transmited back to the NameNode! Secondary NameNode should run on a separate machine in a large installation

– It requires as much RAM as the NameNode

The **fsimage** file contains a file system metadata snapshot

– It is not-updated at every write

– This would be very slow! When an HDFS client performs a write operation, it is recorded in the

Primary NameNode’s edit log

– The edits file

– The NameNode’s in/memory representation of the file system metadata is also updated. Applying all changes in the edits file during a NameNode restart could take a long time.

– The file could also grow to be huge

The Secondary NameNode periodically checkpoints-the NameNode’s in-memory file system data

1. Tells the NameNode to roll its edits file

2. Retrieves fsimage and edits from the NameNode

3. Loads fsimage into memory and applies the changes from the

edits file

4. Creates a new, consolidated fsimage file

5. Sends the new fsimage file back to the primary NameNode

6. The NameNode replaces the old fsimage file with the new one, replaces the old edits file with the new one it created in step 1 ! By default, checkpointing occurs once an hour or for every 1,000,000

transactions, whichever occurs sooner

**If Namenode is down:**

Storage ---🡪 not possible

Process---🡪

New job submission - not possible

Running jobs – would not run

Queued jobs – would not run

If Job Tracker is down:

Storage ---🡪 possible

Process---🡪

New job submission - not possible

Running jobs – would not run

Queued jobs – would not run

Hence if namenode is down, storage operations will not happen,

New job submissions cannot be done,

Running jobs will be terminated,

Queued jobs will not be released.

Expectations:

At least running jobs need to continue processing.

This is possible with the help of Secondary Namenode (SNN).

SNN will hold the metadata of files (not all the files) on which job processing is applied and file block address, the H/W configuration of machine etc.

So even though NN is down, JT can apply fault tolerance.

After completion of job execution, results will be stored in HDFS.

Now SNN will handle this storage responsibility and update metadata of result file in SNN. This metadata will be reflected in NN once edit log is restarted and NN is brought up.

NOTE: From CDH4 onwards, SNN is eliminated. Instead **STANDBY** node which is 100 in sync with original namenode is used.

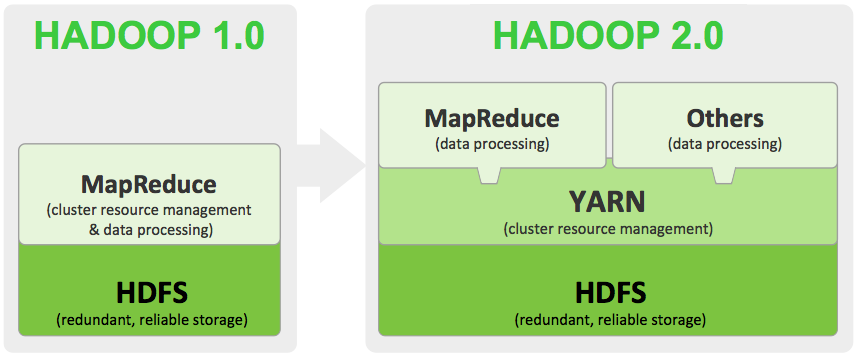
**Introduction of Hadoop 2/MapReduce2 and YARN architecture:**

The architecture discussed so far is the basic architecture used by Hadoop 1 version or MapReduce1 (MRv1).

Here, the JobTracker is used to create and assign tasks to data nodes, which can become a resource bottleneck when the cluster scales out far enough (usually around 4,000 clusters).

Resource management and job scheduling was performed by Job Tracker alone because of which non-batch applications cannot be run on the Hadoop 1

With CDH4 onward, the Apache Hadoop component introduced two new terms for Hadoop users to wonder about: MR2 and [YARN](http://archive.cloudera.com/cdh4/cdh/4/hadoop/hadoop-yarn/hadoop-yarn-site/YARN.html).



**What is YARN?**

YARN stands for Yet-Another-Resource-Negotiator . It is a new framework that facilitates writing arbitrary distributed processing frameworks and applications.

YARN is architecture for distribution cluster. Hadoop2 (MRv2) is using YARN for resource management and programming model which support parallel processing that we known as MapReduce is same as Hadoop1 (MRv1).

YARN provides the daemons and APIs necessary to develop generic distributed applications of any kind, handles and schedules resource requests (such as memory and CPU) from such applications, and supervises their execution.

YARN’s execution model is more generic than the earlier MapReduce implementation. YARN can run applications that do not follow the MapReduce model, unlike the original Apache Hadoop MapReduce (also called MR1).

**What is MR2?**

With the advent of YARN, there is no longer a single JobTracker to run jobs and a TaskTracker to run tasks of the jobs.

The old MR1 framework was rewritten to run within a submitted application on top of YARN.

This application was christened MR2, or MapReduce version 2.

It is the familiar MapReduce execution underneath, except that each job now controls its own destiny via its own ApplicationMaster taking care of execution flow.

It is a more isolated and scalable model than the MR1 system where a singular JobTracker does all the resource management, scheduling and task monitoring work.

MR2 and a new proof-of-concept application called the DistributedShell are the first two applications using the YARN API in CDH4.

**YARN Architecture in detail:**

**YARN daemons**

– ResourceManager – one per cluster

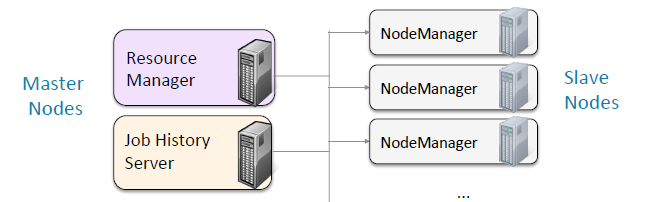
– Initiates application startup, schedules resource usage on slave nodes

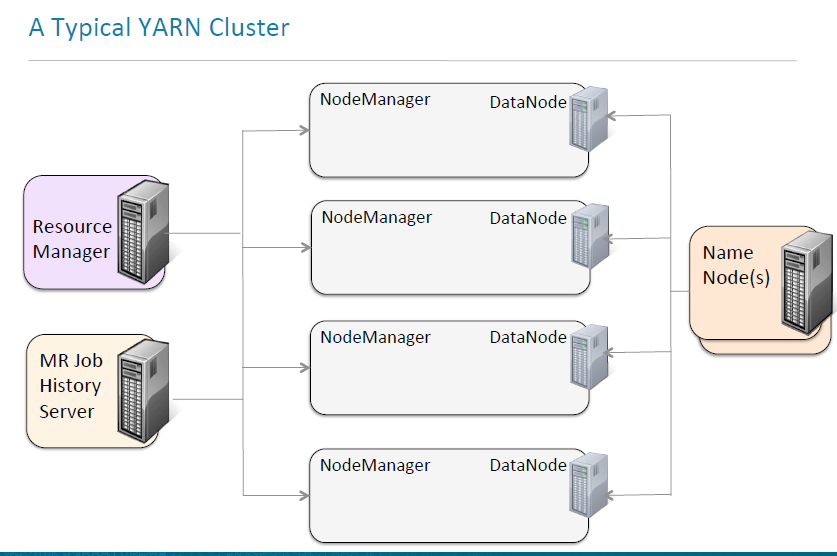
– NodeManager – one per slave node

– Starts application processes, manages resources on slave nodes

– JobHistoryServer – one per cluster

– Archives jobs’ metrics and metadata





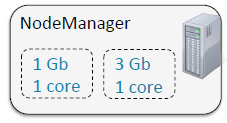
**Running an Application in YARN!**

**Containers**

– Allocated by the ResourceManager

– Require a certain amount of resources (memory, CPU) on a slave node

– Applications run in one or more containers



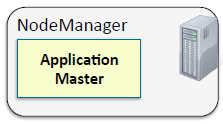
**Application Master**

– One per application

– Framework/application specific (MRAppMaster for MapReduce)

– Runs in a container

– Requests more containers to run application tasks



**MapReduce Application Terminology**

**Job**

– A Mapper, a Reducer, and a list of inputs to process

– Also called an applica4on&

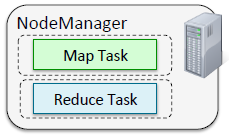
**Task**

– An individual unit of work

– A job is broken down into many tasks

– Each task is either a Map task or a Reduce task

– Runs in a container on a slave node

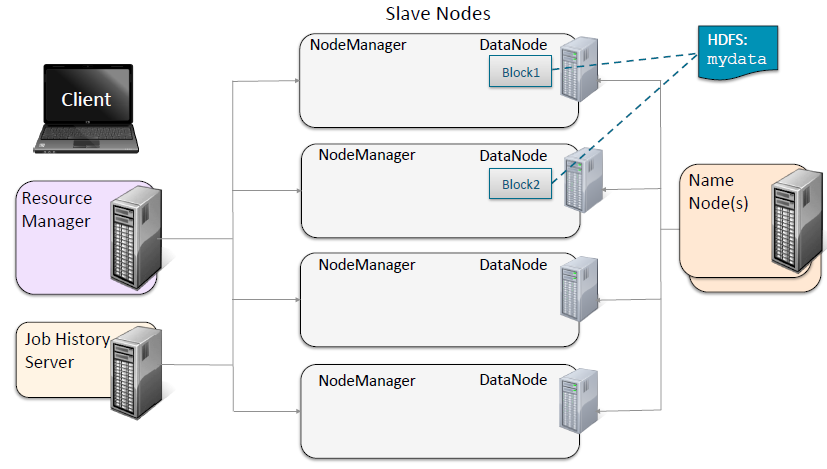


**Client**

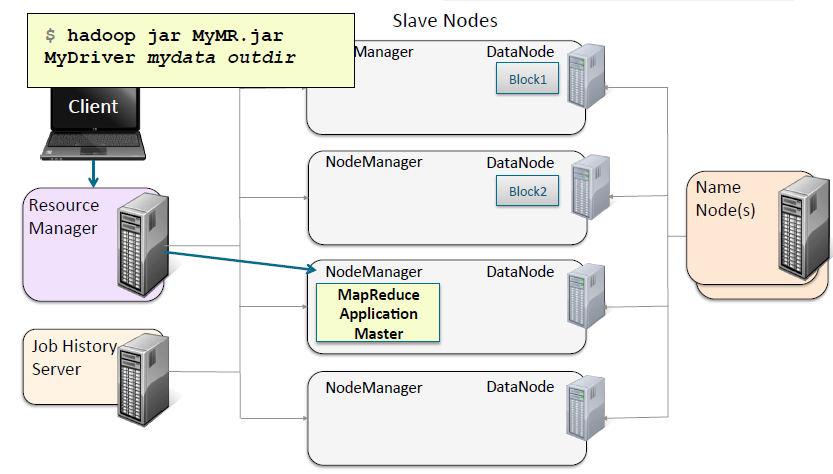
– The program that submits the job to the ResourceManager

– Sometimes refers to the machine it runs on

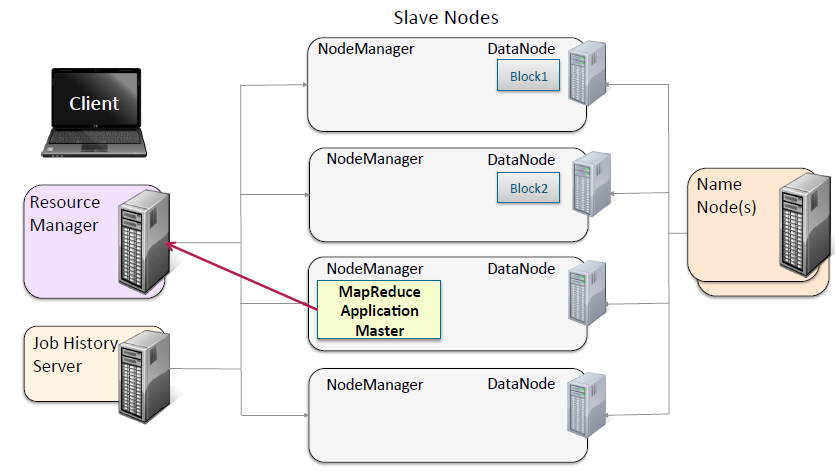
**Running a Job on a YARN Cluster:**



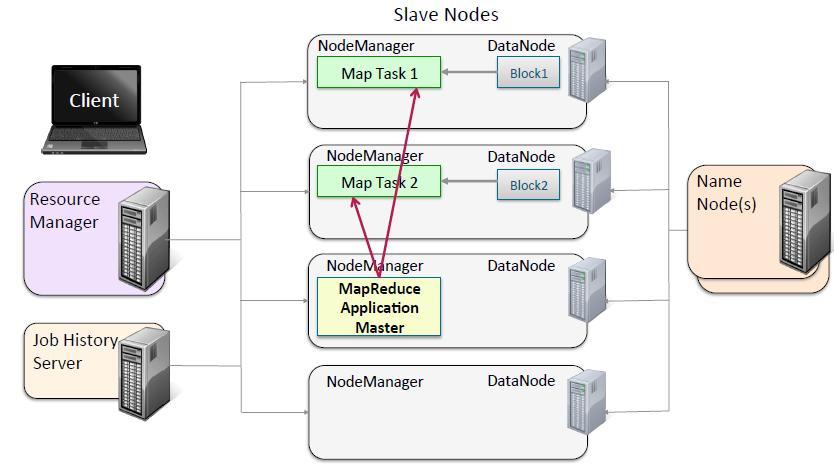
**Running a Job on a YARN Cluster: Job Submission**



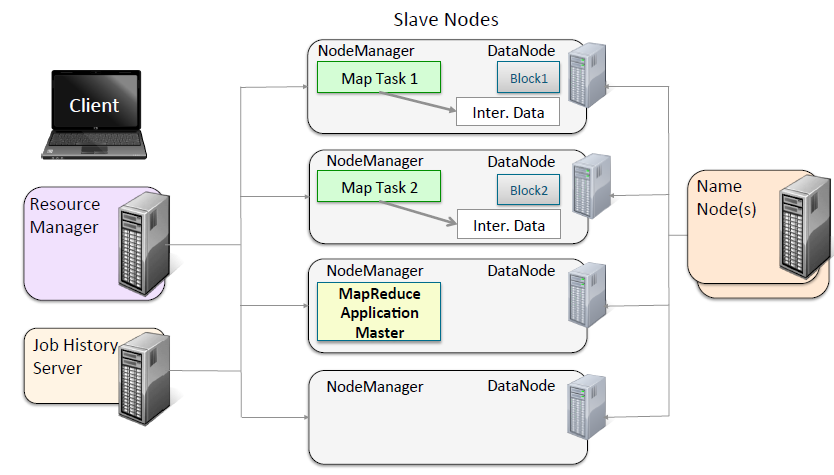
**Running a Job on a YARN Cluster: Map Task Resource Request**



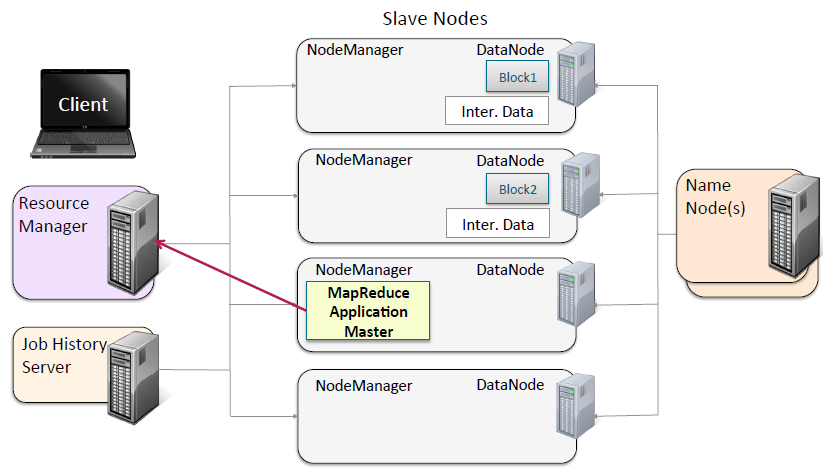
**Running a Job on a YARN Cluster: Map Task Launch**



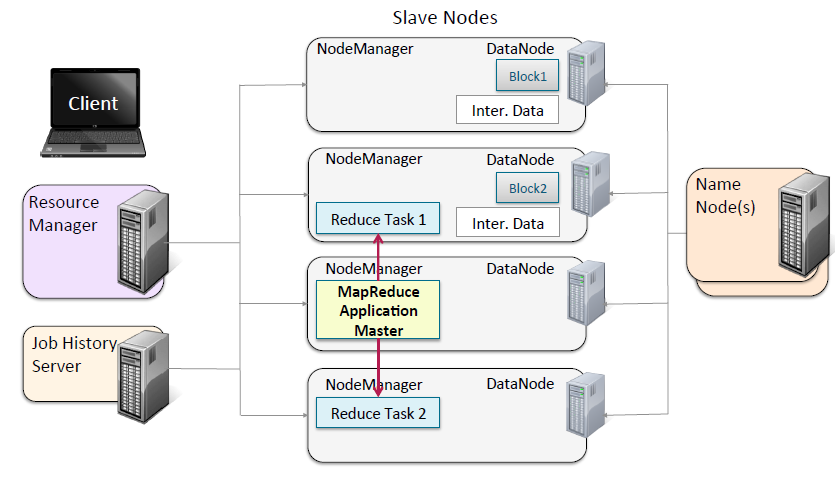
**Running a Job on a YARN Cluster: Intermediate Data**



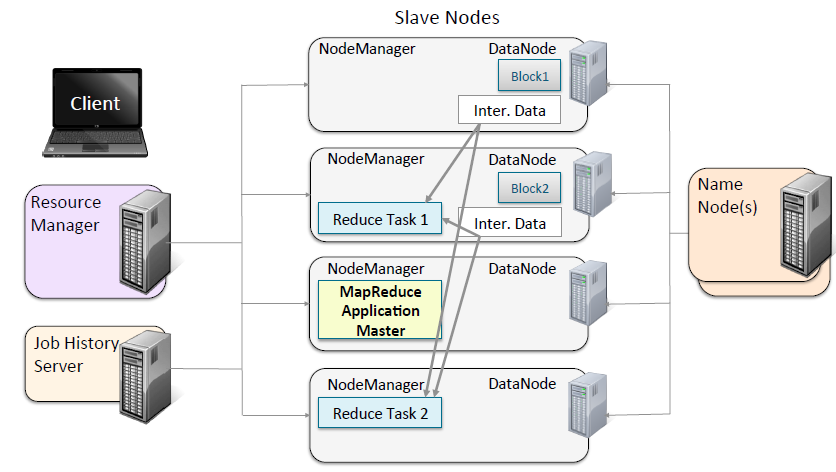
**Running a Job on a YARN Cluster: Reduce Task Resource Request**



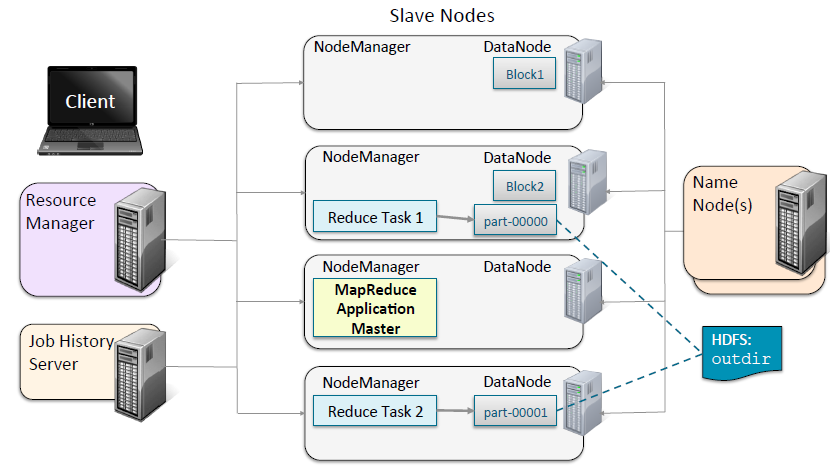
**Running a Job on a YARN Cluster: Reduce Task Launch**



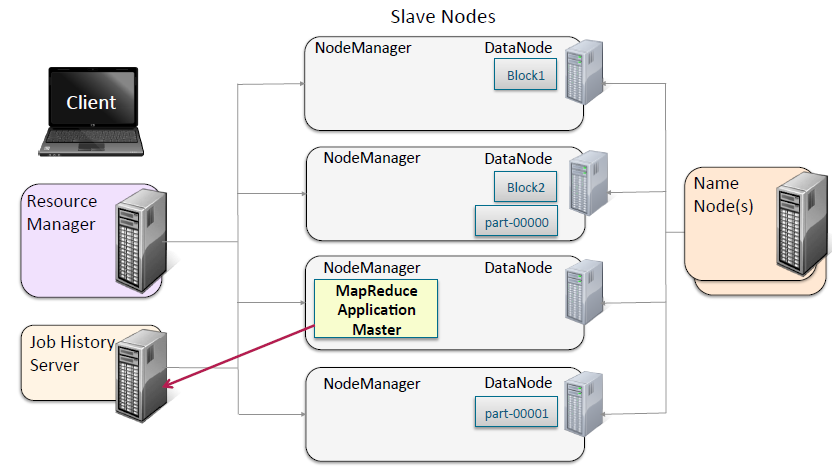
**Running a Job on a YARN Cluster: Shuffle and Sort**



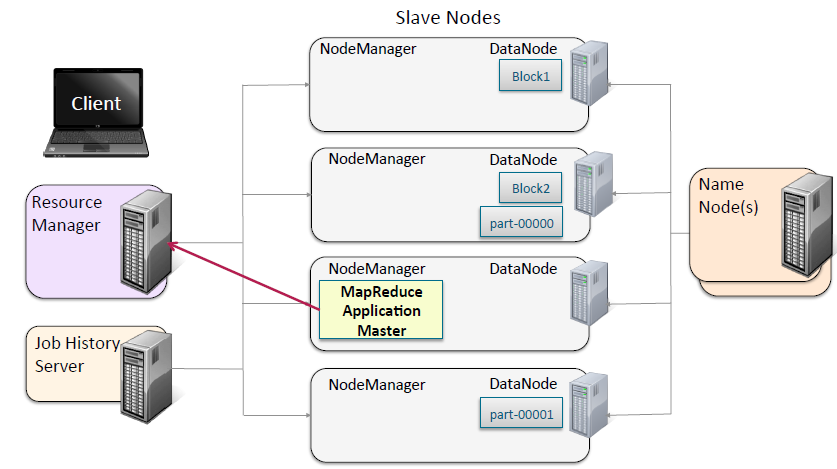
**Running a Job on a YARN Cluster: Job Output**



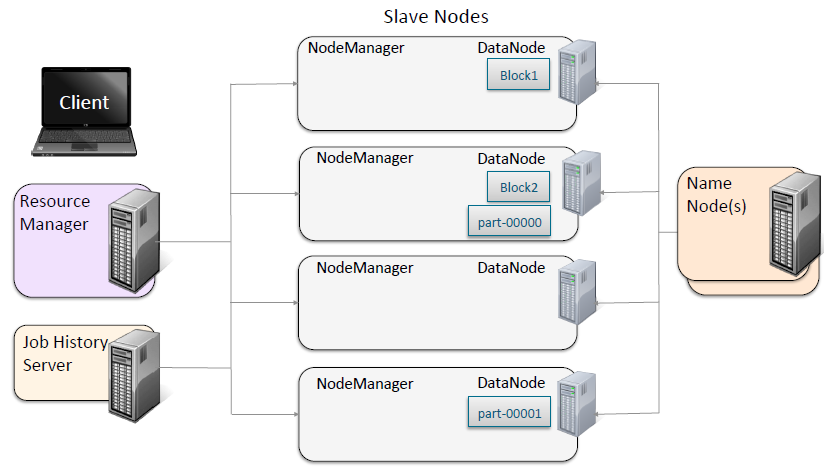
**Running a Job on a YARN Cluster: Job History**



**Running a Job on a YARN Cluster: Job Completion**



**Running a Job on a YARN Cluster: Resources Released**



**ResourceManager: Things to Know**

What the ResourceManager does:

– Manages nodes

– Tracks heartbeats from NodeManagers

– Runs a scheduler

– Determines how resources are allocated

– Manages containers

– Handles ApplicationMasters’ requests for resources

– Deallocates containers when they expire or when the application completes

– Manages ApplicationMasters

– Creates a container for ApplicationMasters and tracks heartbeats

– Manages cluster/level security

**NodeManager Things to Know**

What the NodeManager does:

– Communicates with the ResourceManager

– Registers and provides info on node resources

– Sends heartbeats and container status

– Manages processes in containers

– Launches ApplicationMasters on request from the ResourceManager

– Launches application processes on request from ApplicationMasters

– Monitors resource usage by containers; kills runaway processes

– Provides logging services to applications

– Aggregates logs for an application and saves them to HDFS

– Runs auxiliary services

– Maintains node level security

**Local Resources**

MapReduce tasks need access to local resource files

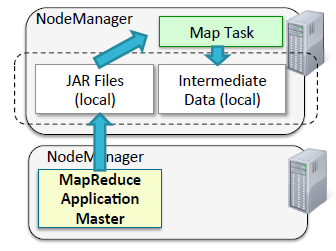
– Stored on slave node local disk

Used for

– Map task output (intermediate data)

– Distributed cache

– e.g. the JAR file containing the program binary to run



Auxiliary Services

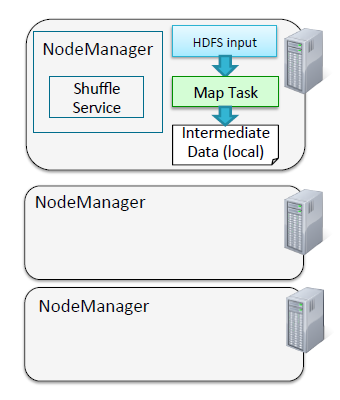
NodeManagers can be configured to run auxiliary services

– Persistent applications that provide services to applications

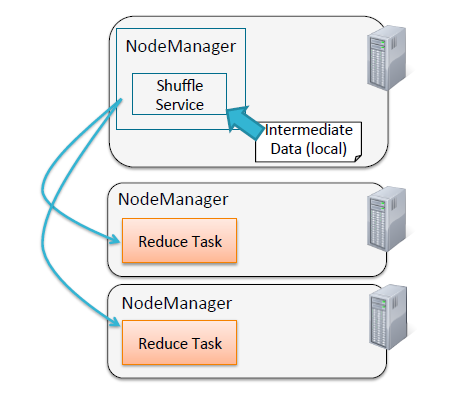
– Run in the NodeManager’s JVM

Used by MapReduce for shuffle and sort

**MapReduce Shuffle Service**: **Stage 1**



**MapReduce Shuffle Service: Stage 2**



**Cluster Resource Allocation**

**Application Master**

– Negotiates with the Resource Manager to obtain containers on behalf of the job

– Presents containers to Node Managers

**Resource Manager**

– Grants containers

– Performs cluster scheduling

**Node Manager**

– Manages life/cycle of containers

– Launches Map and Reduce tasks in containers

– Monitors resource consumption

**Requesting Resources**

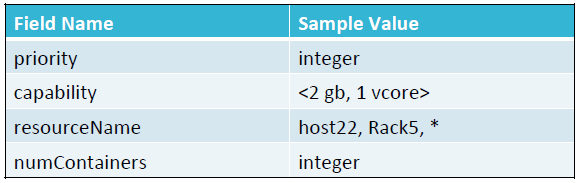
A resource request is a fine grained request for memory and CPU sent to the Resource Manager to be fulfilled

If the request is successful, a container is granted

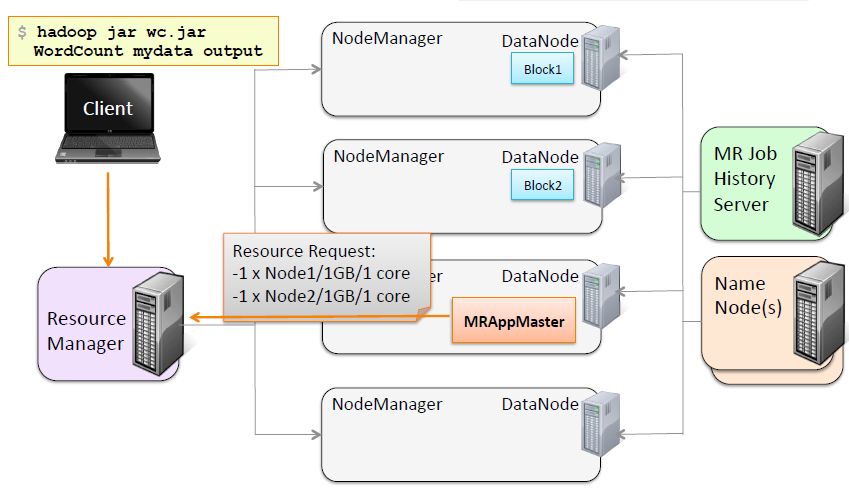
A resource request is composed of several fields that specify

– The amount of a given resource required

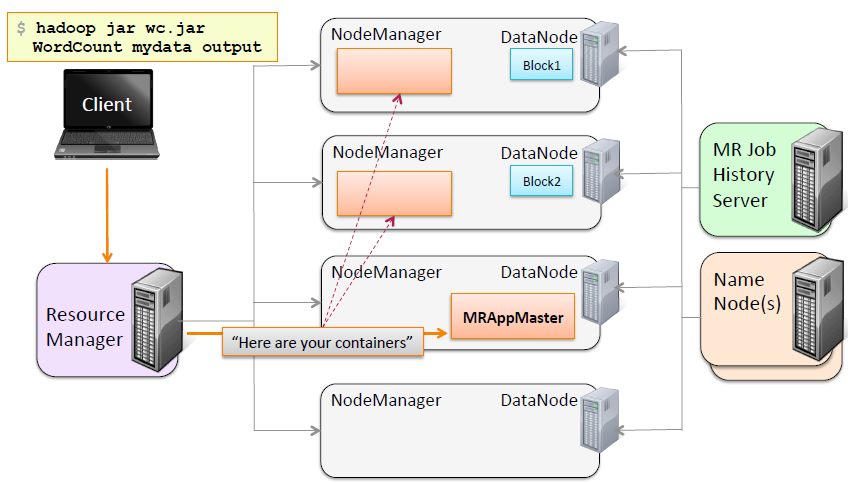
– Data locality information, i.e. preferred node or rack on which to run



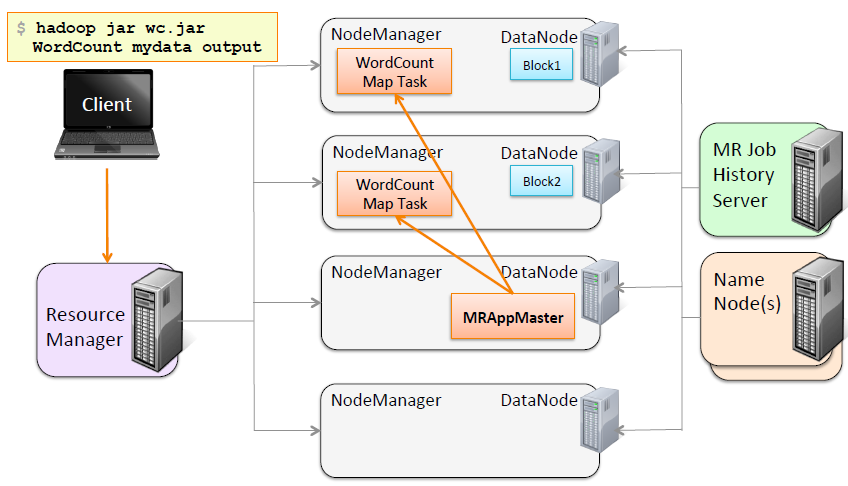
**Requesting Resources for Map Tasks**



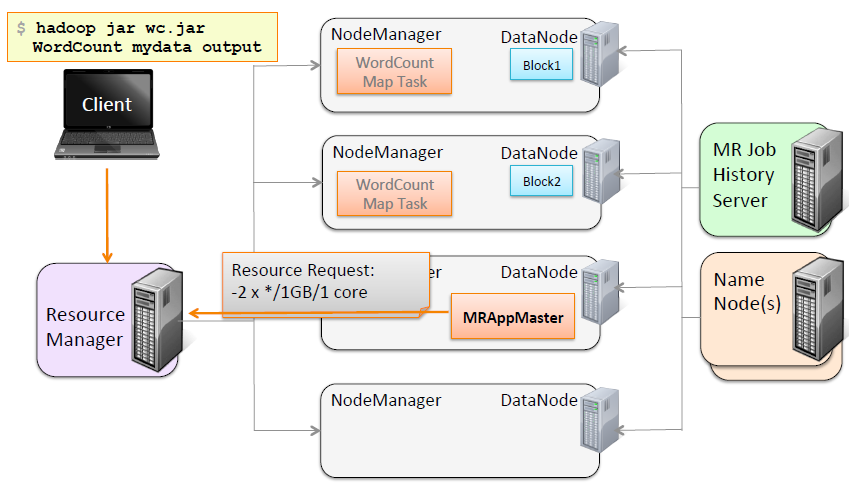
**Granting Containers for Map Tasks**



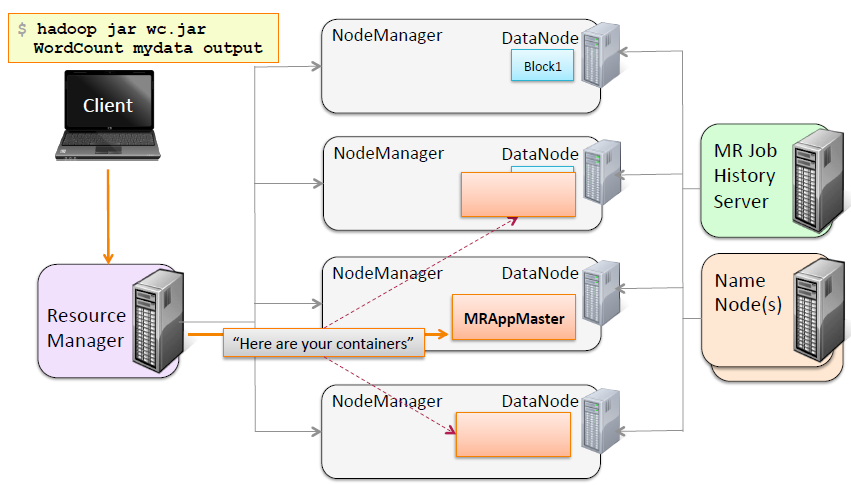
**Allocating the Map Task Containers**



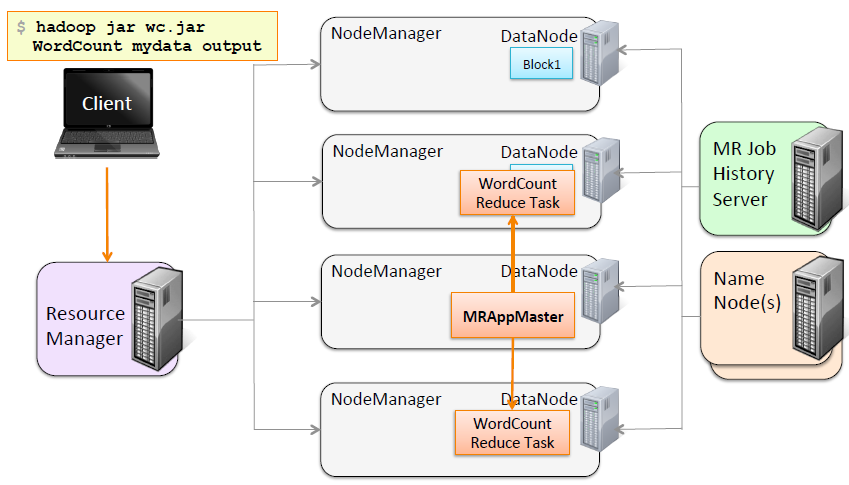
**Requesting Resources for Reduce Tasks**



**Granting Containers for Reduce Tasks**



**Allocating the Reduce Task Containers**



**Failure Recovery**

Fault Tolerance

What happens when something fails?

**Task (Container)**

The ApplicationMaster will reattempt tasks that complete with exceptions or stop responding (4 times by default)

– Applications with too many failed tasks are considered failed

**ApplicationMaster**

– If application fails or if ApplicationMaster stops sending heartbeats, the ResourceManager will reattempt the whole application (2 times by default)

– ApplicationMaster optional setting: Job recovery

– If false, all tasks will re/run

– If true, ApplicationMaster retrieves state of tasks when it restarts; only incomplete tasks will be rerun

**NodeManager**

– If the NodeManager stops sending heartbeats to ResourceManager, it is removed from list of active nodes

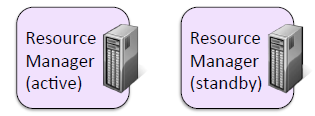
– Tasks on the node will be treated as failed by the ApplicationMaster

– If the ApplicationMaster node fails, it will be treated as a failed application

**ResourceManager**

– No applications or tasks can be launched if ResourceManager is unavailable

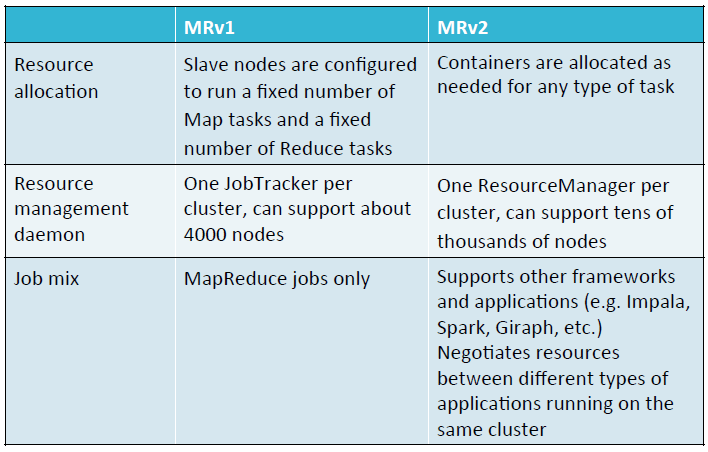
– Can be configured with high availability (HA)



**Summary**

YARN is a generic platform for any form of distributed application to run on, while MR2 is one such distributed application that runs the MapReduce framework on top of YARN.

**MapReduce v1 vs. MapReduce v2 on YARN**



**Hadoop Managers**

This includes Cloudera manager, Ambari and Hortonworks. Main purpose of these is

end to end administration for Hadoop Cluster

* Deploy, manage, and monitor your cluster