

The need for Big Data

Let's look at some examples.

- Facebook currently stores 300 petabytes of data
- Facebook processes approximately 600 terabytes of data per day
- Facebook handles one billion users per day
- Facebook processes 2.7 billion 'Likes' per day.
- 300 million photographs uploaded every day.

The need for Big Data

- The United States National Security Agency (NSA)
- The NSA currently stores approximately five exabytes of data
- The NSA processes approximately 30 petabytes of data per day
- The NSA touches approximately 1.6 percent of all internet traffic per day (More than Google).
- Data such as web searches, phone calls, credit card transactions, health information is stored and analyzed.

The need for Big Data

- Google.
- Google processes 100 petabytes of information per day
- Google indexes 60 trillion web pages per day
- Google has greater than one billion unique searches per month
- Google processes 2.3 million searches per second.

The need for Big Data

No single hardware platform
can handle data sets this
large.

Big Data Requirements

To process this volume of data you need three things.

- The ability to store massive quantities of data
- The ability to extract and process elements of this data in a timely fashion.
- The ability for the system to scale as the volumes of data increase.

Examples of scalability requirements

LinkedIn grew from 37 million users in Q1 2009 to 450 million users in Q2 2016.

No single hardware platform can match these scalability requirements, no matter how much memory, CPU and storage is allocated to it.

History of Big Data systems.

Google developed this concept in the early 2000's after realizing that the web was growing too quickly for existing architectures to process it.

Google created two systems to address this problem.

- The Google File System to solve the problem of distributed storage
- The MapReduce programming architecture. to solve the problem of distributed processing.

History of Big Data systems.

The Apache foundation took Google's technologies and developed an open source version.

The Google File System became HDFS

The MapReduce architecture became Hadoop's Mapreduce

Hadoop is the name for the overall system architecture.

It includes HDFS, MapReduce and YARN.

- b

History of Big Data systems.

Hadoop 2.0 released by the Apache Foundation in 2013.

Hadoop 2.0 is a fundamental change in the architecture.

The resource manager YARN was split from the MapReduce system and created as its own subsystem.

History of Big Data systems.

Hadoop 2.0 released by the Apache Foundation in 2013.

Hadoop 2.0 is a fundamental change in the architecture.

The resource manager YARN was split from the MapReduce system and created as its own subsystem.

Hadoop Ecosystem

Hive

Hbase

Pig

Hadoop

Flume/Sqoop

Spark

Oozle



Hive

Provides an SQL interface to Hadoop

**The bridge to Hadoop for people who
don't have experience with OOP in
Java**



**A database management system on
top of Hadoop**

**Integrates with your application as if it
were a traditional database**



A data manipulation language

**Transforms unstructured data into a
structured format**

**Query the structured data using
interfaces like Hive.**



←

**A distributed computing engine used
along with Hadoop**

**Interactive shell used to quickly
process data**

**Many built-in libraries for machine
learning, NLP, stream processing, and
other tasks.**

Oozle

The diagram consists of a yellow rectangular box on the left containing the word 'Oozle'. To its right is a vertical line. Further right is the text 'Workflow scheduling engine'. An arrow points from the right side of the vertical line towards the 'Workflow scheduling engine' text.

Workflow scheduling engine

Flume/Sqoop

**Data transfer engine between
Hadoop and other types of systems.**

HDFS Design

- Each HDFS data node is a simple off-the-shelf system.
- HDFS is highly fault tolerant . Hardware failures are assumed.
- HDFS is designed for batch processing, HDFS is used for high throughput requirements rather than low latency requirements.
- Queries to HDFS are not usually done in real time.
- HDFS supports very large data set sizes.

HDFS Design

- Default block size for Hadoop is 64 Mb.
- Default blocksize for the Cloudera distribution is 128 Mb
- Files are split among multiple machines and disk storage in a cluster.
- One of the machines in a cluster is the master (name) node,
- The rest of them are data nodes.

HDFS Design

- The name node stores the meta data for all files stored in the HDFS.
- There is one name node per cluster and possibly a secondary name node.
- The name node stores all of the metadata for files.
- Consider a file like a book. The name node stores the index
- The data nodes store the actual pages.

HDFS Design

- The name node stores the meta data for all files stored in the HDFS.
- There is one name node per cluster and possibly a secondary name node.
- The name node stores all of the metadata for files.
- Consider a file like a book. The name node stores the index
- The data nodes store the actual pages.

HDFS Design

- The name node has two primary responsibilities.
 - First, it manages the HD file system. All requests from a client for data go to the name node first.
 - The name node stores the file system directory structure and all other metadata about each file.
 - The data node actually stores the data in blocks on its disk storage devices.
-
- b

HDFS Design

- A typical data file is a large text file (size in terabytes or petabytes).
- This file is broken up into chunks called blocks.
- Each block is stored on a different node in the hadoop cluster.
- Each block is the same size.
- Differently sized files are treated the same way.
- Storage is simplified. We only deal with blocks.
- A single unit for fault tolerance and replication.

HDFS Design

- The default block size in Hadoop is 64 MB (128 MB for Cloudera installations).
- Block size is a trade off between parallelism and overhead.
- Large block sizes means less processes working. Each process only works on one block
- Smaller block sizes means more network and processing overhead.
- Optimal blocksize depends on type of data and

HDFS Design

- The default block size in Hadoop is 64 MB (128 MB for Cloudera installations).
- Block size is a trade off between parallelism and overhead.
- Large block sizes means less processes working. Each process only works on one block
- Smaller block sizes means more network and processing overhead.
- Optimal blocksize depends on type of data and other considerations.
- Hadoop does its best to run all its tasks on the system that stores the block. This is data locality

HDFS Design

- The default block size in Hadoop is 64 MB (128 MB for Cloudera installations).
- Block size is a trade off between parallelism and overhead.
- Large block sizes means less processes working. Each process only works on one block
- Smaller block sizes means more network and processing overhead.
- Optimal blocksize depends on type of data and other considerations.
- Hadoop does its best to run all its tasks on the system that stores the block. This is data locality optimization

HDFS Design

- Hadoop nodes are generally implemented as commodity hardware.
- Commodity hardware can fail.
- We have to consider possibility of failure in the name node and data nodes.
- Some data in a block may get corrupted.
- A data node could crash completely, causing us to lose access to all the data stored.

HDFS Design

- Hadoop replicates every file and block stored in the HDFS.
- We define a replication factor to decide how data is replicated.
- Replication locations are also stored in the name node.
- Replication strategy needs to optimize two variables
- First, the redundancy factor. A higher redundancy factor means more security.
- Second, the write bandwidth. A higher redundancy factor also means a higher write time.

HDFS Design

- Nodes on different racks are further away than nodes on the same rack.
- A cluster of machines is made up of systems stored on different racks in a system room.
- Read/write bandwidth for inter rack nodes are lower than for intra rack nodes.
- Replication strategy is defined in `hdfs-site.xml` configuration file.

HDFS Design

- Hadoop then chooses another rack in a different location and writes the second replica to it.
- The third replica is on the new rack (but on a different node) as well,
- Replication strategy is defined in `hdfs-site.xml` configuration file.

HDFS Design

- A cluster of machines is made up of systems stored on different racks in a system room.
- Nodes on different racks are further away than nodes on the same rack.
- Read/write bandwidth for inter rack nodes are lower than for intra rack nodes.
- The default replication strategy is to replicate data on one node as close to the same rack as possible.

HDFS Design

- Hadoop then chooses another rack in a different location and writes the second replica to it.
- The third replica is on the new rack (but on a different node) as well,
- Replication strategy is defined in `hdfs-site.xml` configuration file.

HDFS Design

- Fault tolerance planning must also take into consideration failure of the name node.
- Name node failure means that all of the data in the cluster is permanently lost.
- With name node failure, all file/block mapping is gone.
- Two specific strategies for overcoming name node failure.
- First is the metadata files.
- Second is the secondary name node.

HDFS Design

- Two metadata files associated with the name node.
- First is the *fsimage* File system image.
- Second is the edit log.

HDFS Design

- The fsimage file holds a snapshot of the hadoop cluster's file system on startup.
- It contains the image of the file system before any writes have been performed.
- FSImage is loaded into memory on startup.

HDFS Design

- The edit log contains a log of all modifications to files in the cluster's file system since startup.
- Putting together the fsimage and the edit log can reconstruct the data in the hdfs cluster.
- Both files are by default saved to a local file system directory on the name node.
- Alternatively (or simultaneously) this data can be saved to a remote location over the network.
- Merging these two files can be computationally intensive.

HDFS Design

- The secondary name node contains a replica of the fsimage and edit log from the master.
- The secondary namenode merges the files together according to a specifically defined checkpoint.
- The newly merged fsimage is then copied back to the primary name node.
- The edits log on both nodes is then reset.

HDFS Design

- Two ways to define checkpointing.
- First is to checkpoint after a specific amount of time.
- Second is to checkpoint after a specific number of transactions are completed.
- The newly merged fsimage is then copied back to the primary name node.
- The edits log on both nodes is then reset.

HDFS Read and Write Operations

- To write a file in HDFS, a client needs to interact with the namenode master.
- The HDFS client sends a *create()* request on the DistributedFileSystem API.
- The API makes a Remote Procedure Call (RPC) to the namenode to create a new file in the namespace.
- The namenode performs various checks to make sure that the file doesn't already exist and that the client has valid create permissions.
- The namenode makes a record of the file if all checks pass.
- Otherwise, the namenode throws an IOException.

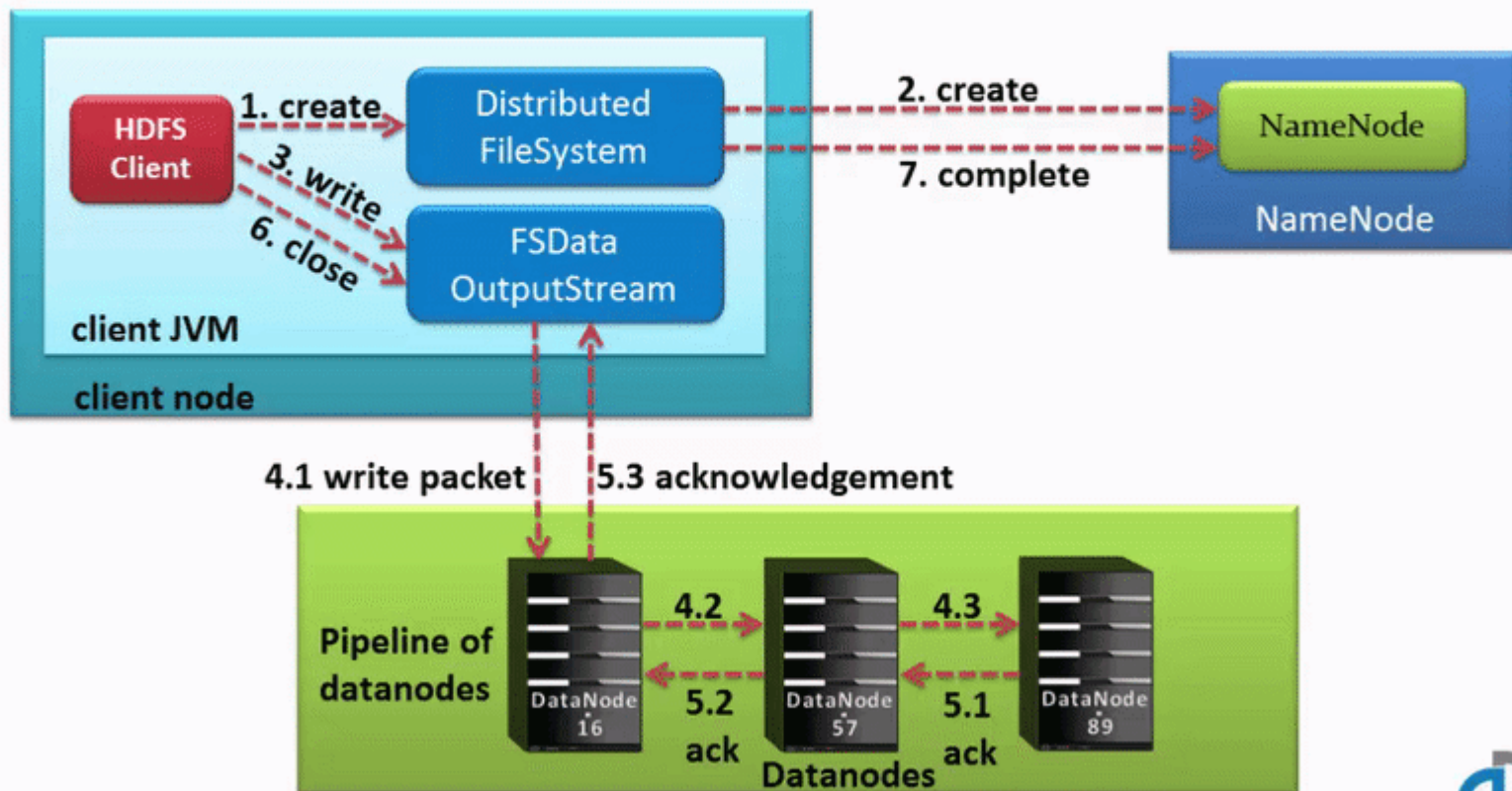
HDFS Read and Write Operations

- The API returns an *DFSDataOutputStream* object for the client to start writing data to.
- *DFSOutputStream* splits the client data into packets and writes it to an internal queue called the data queue.
- The data queue is consumed by the *DataStreamer* object which is responsible for asking the namenode to allocate new blocks.
- The list of datanodes form a pipeline.
- The *DataStreamer* streams the packets to the first datanode in the pipeline.

HDFS Read and Write Operations

- The first datanode stores the packet and then forwards it to the next datanode in the pipeline.
- Similarly, the second datanode stores the packet and forwards it to the third datanode in the pipeline.
- And so on until all datanodes in the pipeline have received and stored the packet.
- Once the client has finished writing data, the *close()* method is called on the stream object.
- The close flushes all remaining packets to the pipeline and waits for an acknowledgement.
- After acknowledgement, the client signals to the namenode that the file is complete.

HDFS Read and Write



HDFS Read and Write Operations

- To read a file from HDFS, a client needs to interact with the master namenode.
- The namenode verifies the clients access privileges.
- If checks pass, then the master returns the address of the slaves where the file is stored.
- The client then interacts directly with the respective datanodes to read the data.

HDFS Read and Write Operations

- The client starts by opening the file using the *open()* method on the *FileSystem* object, which is an instance of the *DistributedFileSystem* class.
- The *DistributedFileSystem* makes an RPC call to determine the location of the blocks for the first few blocks in the file.
- For each block, the namenode returns the addresses of the relevant datanodes. The addresses are sorted by proximity to the client.

HDFS Read and Write Operations

- The *DistributedFileSystem* returns an *FSDatInputStream* object to the client.
- *FSDatInputStream* wraps the *DFSInputStream* object.
- *DFSInputStream* manages the namenode and datanode I/O.
- The client calls the *read()* method on the stream.
- *DFSInputStream* connects the client to the closest datanode for the first block in the file.
- Data is streamed from the datanode back to the client. As a result, the client call call *read()* repeatedly on the stream.

HDFS Read and Write Operations

- If the *DFSInputStream* encounters an error when communicating with the datanode, it will then try the next closest datanode. It also remembers failed datanodes so that it won't attempt to retry them for later blocks.
- *DFSInputStream* verifies checksums for the data transferred to it.
- Corrupt blocks are reported to the namenode before it attempts to read the block from a different namenode.

HDFS Read and Write Operations

- When the client has finished reading the data, it calls a *close()* method on the stream object.

HDFS Federation

- Current HDFS architecture only allows for a single namespace for the entire cluster.
- Namespace is managed by a single namenode.
- These limitations are addressed by HDFS Federation

HDFS Federation

- Namespaces can only be scaled vertically on a single namenode.
- This limits the number of blocks, files and directories to the memory resources of the namenode.
- Throughput of the cluster are also limited to the namenodes.
-

HDFS Federation

- HDFS Federation uses multiple independent namenodes/namespaces.
- The namenodes are federated, in that they are independent of each other.
- The datanodes are common storage for all the namenodes.
- Each datanode registers with each namenode in the cluster.
- Datanodes send periodic heartbeats and block reports and handles commands from the namenodes.

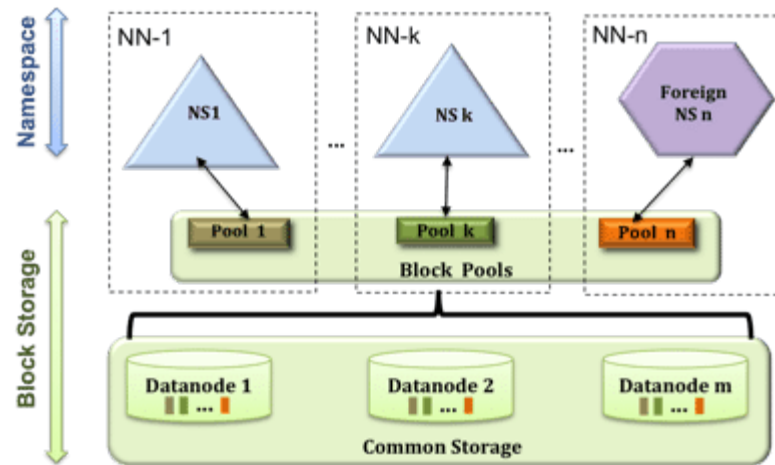
HDFS Federation

- A Block Pool is a set of blocks that belong to a single namespace. Datanodes store blocks for all the block pools in the cluster.
- It is managed independently of other block pools.
- This allows a namespace to generate Block IDs for new blocks without the need for coordination with the other namespaces.
- The failure of a namenode does not prevent the datanode from serving other namenodes in the cluster.

HDFS Federation

- A Namespace and its block pool together are called Namespace Volume.
- It is a self-contained unit of management. When a namenode/namespace is deleted, the corresponding block pool at the datanodes is deleted.
- Each namespace volume is upgraded as a unit, during cluster upgrade.

HDFS Federation



HDFS Federation

- **Key Benefits**
- Scalability and isolation
- Support for multiple namenodes horizontally scales the file system namespace.
- It separates namespace volumes for users and categories of applications and improves isolation.
-

HDFS Federation

- **Key Benefits**
 - Generic storage service
 - Block pool abstraction opens up the architecture for future innovation.
 - New file systems can be built on top of block storage.
 - New applications can be directly built on the block storage layer without the need to use a file system interface.
 - New block pool categories are also possible, different from the default block pool.

HDFS Installation


Hadoop Installation Modes



Standalone


Pseudo-distributed

Fully distributed




Standalone

The default mode for Hadoop
Runs on a single node
A single JVM process
YARN and HDFS do not run
Used to test Mapreduce programs



Pseudo-distributed

Runs on a single node
Uses two JVM processes to simulate a
distributed environment
HDFS for storage
YARN for managing tasks
Used as a fully-fledged test environment



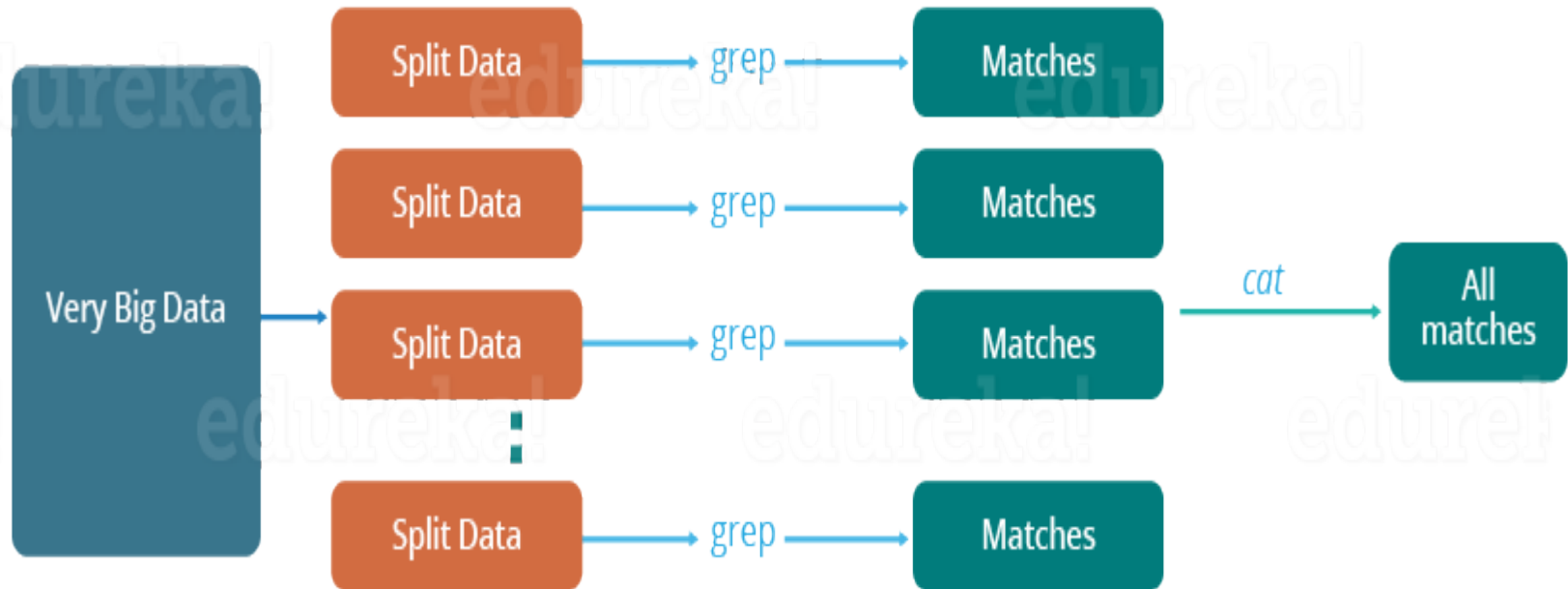
Fully distributed

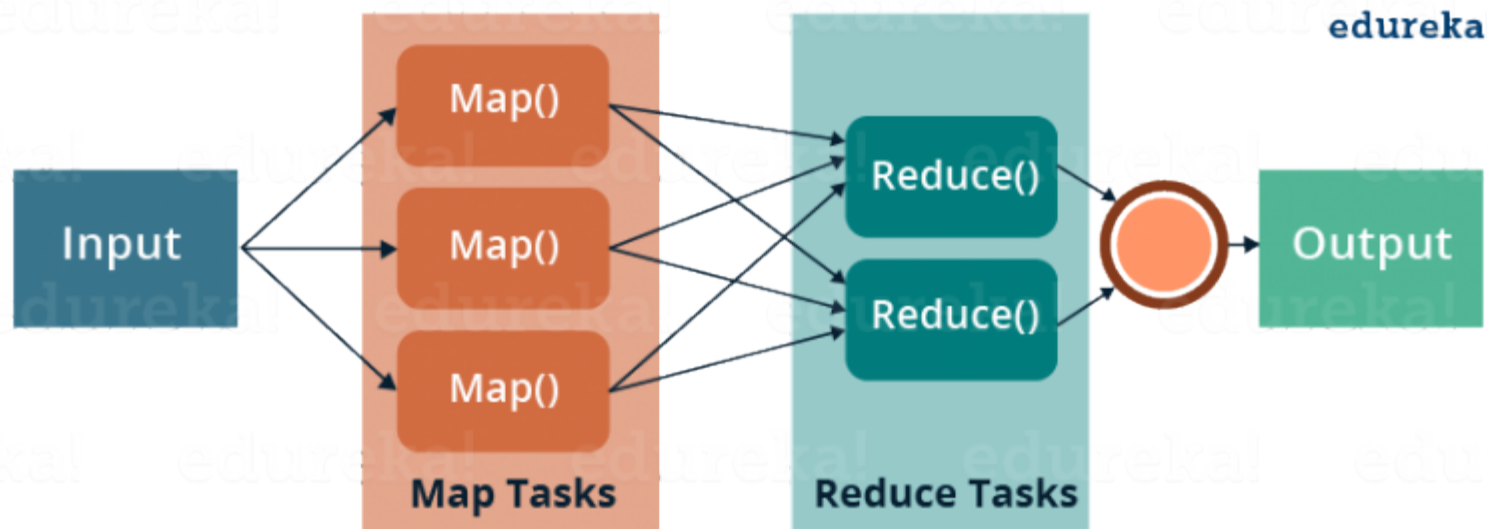
Runs on a cluster of machines, generally Linux systems in a data center or Virtual Machines in a cloud environment

Usually implemented via Cloudera, Hortonworks or MapR rather than manually.

The Traditional Way

edureka!





MapReduce

- Mapreduce jobs require running processes on many different machines.
- Mapreduce is a programming paradigm that allows processing on distributed systems.
- Modern systems generate millions of records of raw data.
- This data can be many petabytes in size.
- These sort of tasks are run in two phases, the *map* phase and the *reduce* phase.

MapReduce

- The map task divides a job into many different processes, each running on a different compute node.
- The reduce task takes the output of the many map tasks and runs some sort of reducing function on it to generate the desired output.
- The developer only needs to write a map and reduce function. All of the fault tolerance is handled by hadoop itself.
- Map output is in the form of *key/value* pairs.
- The reduce function takes these key/value pairs and runs a reducing function to get the desired output to the client,.

MapReduce

- In order to run a mapreduce job on the Hadoop cluster, the job must be compiled into a JAR file.
- This jar file is then copied to the designated Hadoop distribution directory for processing.
- A Mapreduce input directory needs to be created in the cluster's HDFS.
- To actually run a hadoop job, use the *hadoop jar* command.

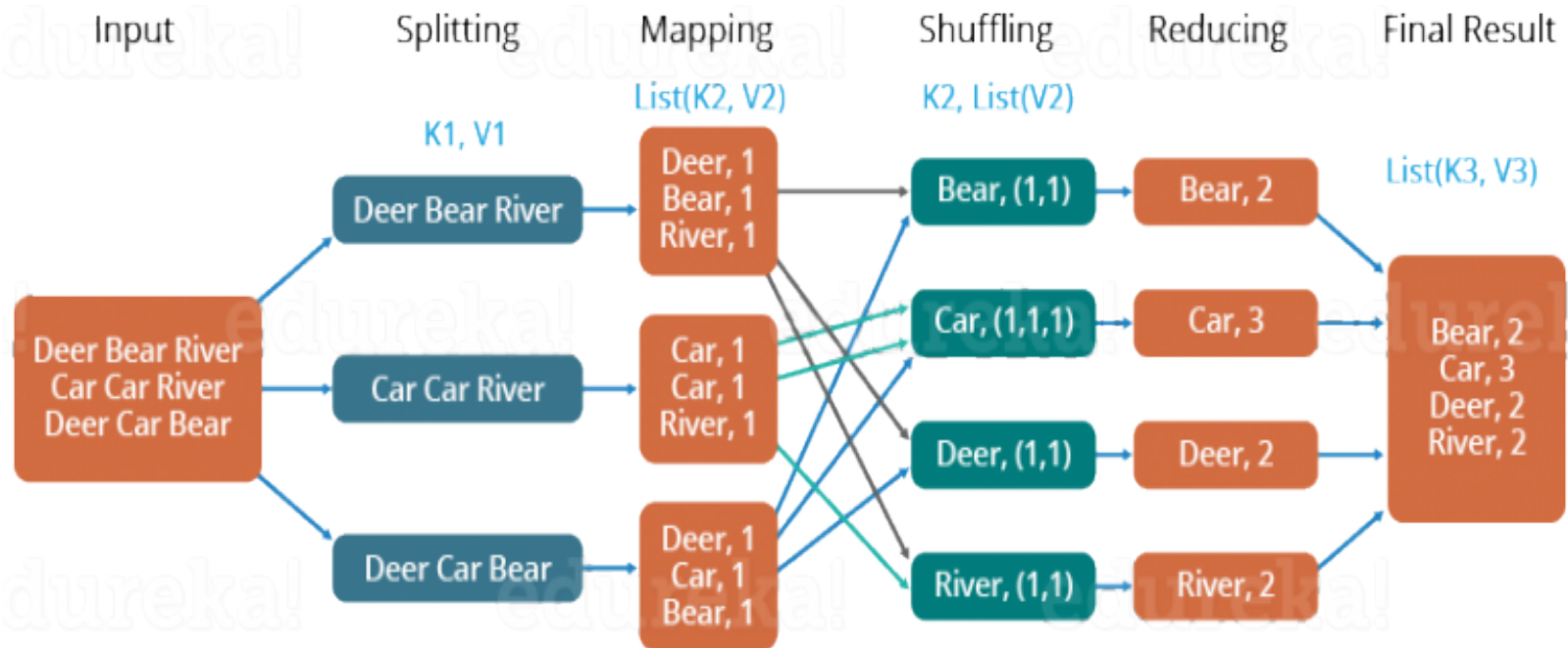
Mapreduce

- When a job has been submitted, you will see a job id starting with 'job_id<xxxxxx>'
- Using this job ID, you can monitor the results of the hadoop job .
- Hadoop provides a web based resource manager interface .
- The default URL is *http://localhost:8088*.

Mapreduce

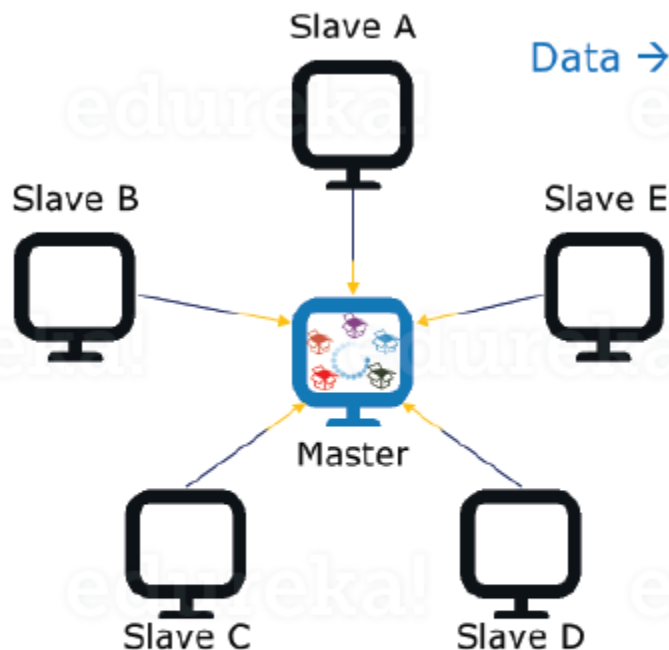
The Overall MapReduce Word Count Process

edureka!

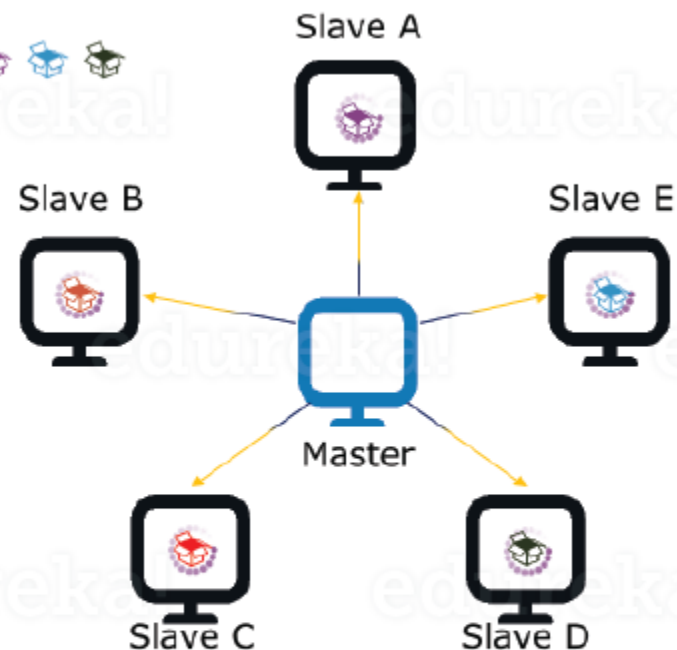


Mapreduce

edureka!

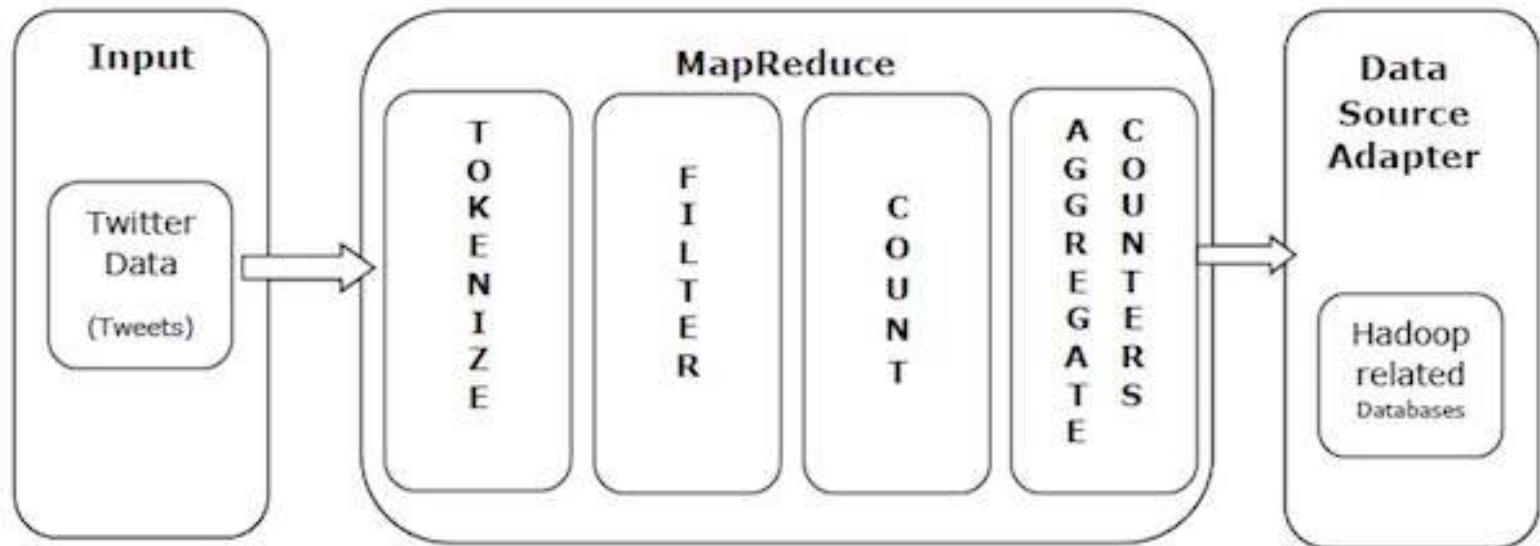


1. Moving data to the Processing Unit
(Traditional Approach)



2. Moving Processing Unit to the data
(MapReduce Approach)

Mapreduce Example



Mapreduce Advantages

- Parallel Processing.
- Data Locality

Mapreduce Optional Functions

- Partitioners
- Combiners

Mapreduce Optional Functions

- A partitioner works like a condition in processing an input dataset. The partition phase takes place after the Map phase and before the Reduce phase.
- The number of partitioners is equal to the number of reducers. That means a partitioner will divide the data according to the number of reducers. Therefore, the data passed from a single partitioner is processed by a single Reducer.

Mapreduce Optional Functions

- A partitioner partitions the key-value pairs of intermediate Map-outputs.
- It partitions the data using a user-defined condition, which works like a hash function.
- The total number of partitions is same as the number of Reducer tasks for the job.
- For example, if the input data is a list of employees by age and salary and gender, then we can use the partitioner to get maps by gender, i.e. one partition for males and one for females. Each partition is assigned to a reduce job.

Mapreduce Optional Functions

- A Combiner, also known as a semi-reducer, is an optional class that operates by accepting the inputs from the Map class and thereafter passing the output key-value pairs to the Reducer class.
- The main function of a Combiner is to summarize the map output records with the same key. The output (key-value collection) of the combiner will be sent over the network to the actual Reducer task as input.

Mapreduce Optional Functions

- A combiner does not have a predefined interface and it must implement the Reducer interface's `reduce()` method.
- A combiner operates on each map output key. It must have the same output key-value types as the Reducer class.
- A combiner can produce summary information from a large dataset because it replaces the original Map output.

Mapreduce Example

- Tokenize – Tokenizes the tweets into maps of tokens and writes them as key-value pairs.
- Filter – Filters unwanted words from the maps of tokens and writes the filtered maps as key-value pairs.
- Count – Generates a token counter per word.
- Aggregate Counters – Prepares an aggregate of similar counter values into small manageable units.

YARN

- YARN stands for Yet Another Resource Negotiator.
- In Hadoop Version 1, this was part of the MapReduce framework.
- Hadoop Version 2 split the resource management portion into its own subsystem.
- Yarn is used by Hadoop to schedule tasks for the cluster.

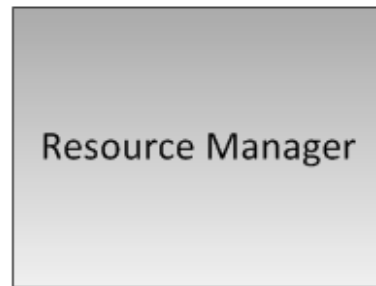
YARN

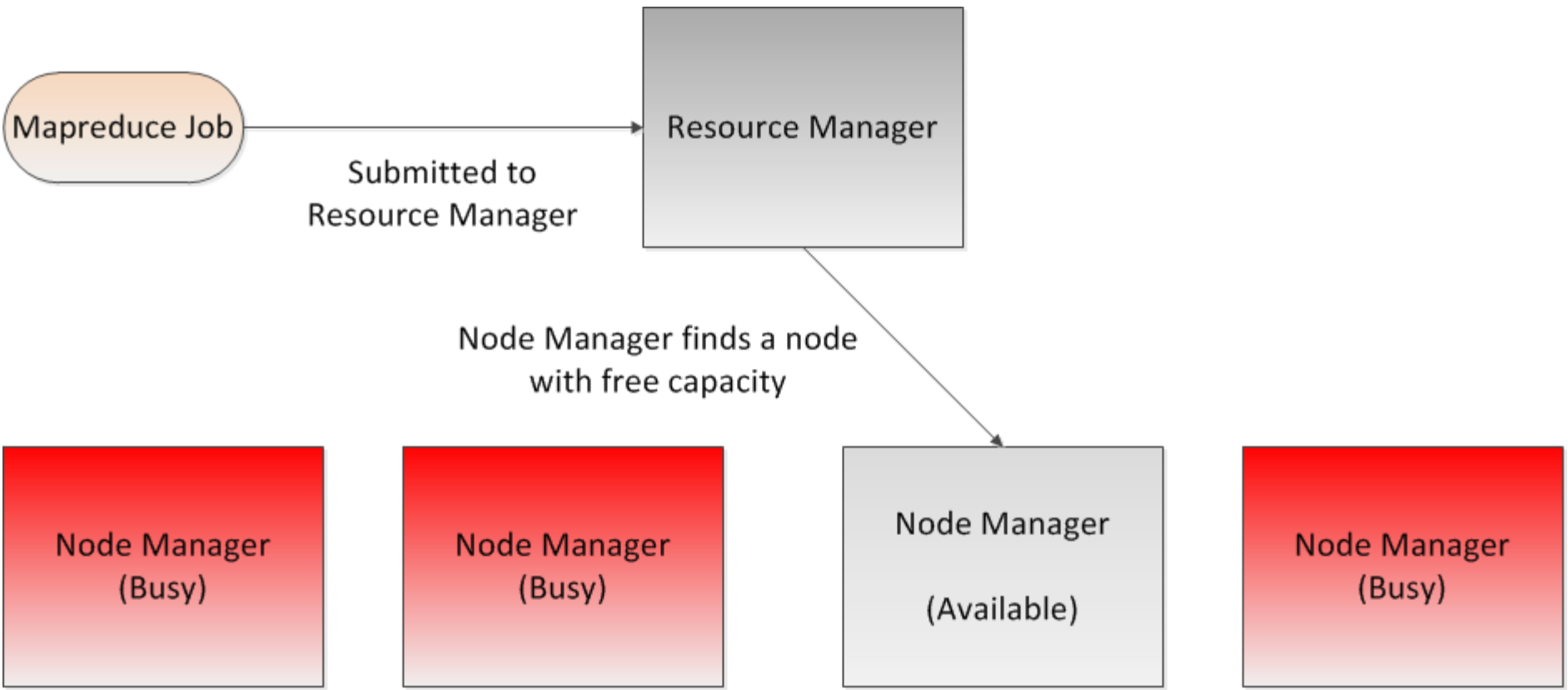
- YARN coordinates tasks running across the cluster.
- YARN keeps track of all tasks and reallocates them in case of a node failure.
- YARN consists of two components.
- First, the Resource Manager, which runs on a single master node.
- Second the Node Manager, which runs on all the nodes.
- These managers are implemented as UNIX daemon processes.

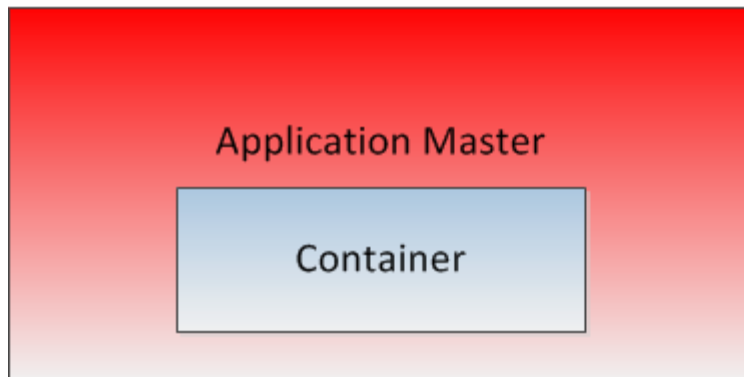
YARN

- The Resource Manager schedules tasks across all nodes.
- The Node manager keeps track of all individual jobs on that specific node.
- YARN consists of two components.
- First, the Resource Manager, which runs on a single master node.
- Second the Node Manager, which runs on all the nodes.
- Often, the node manager process is co-located with the data nodes.

Yet Another Resource Negotiator (YARN)





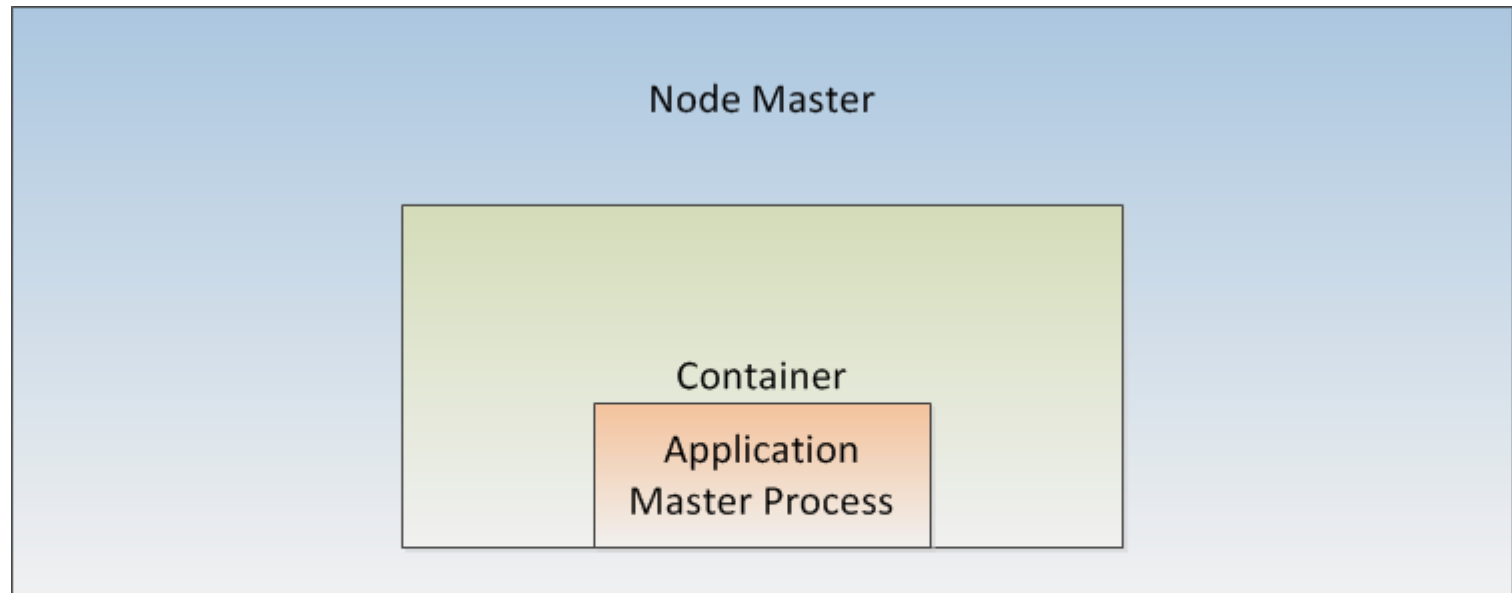


All processes on the node
are run inside a container

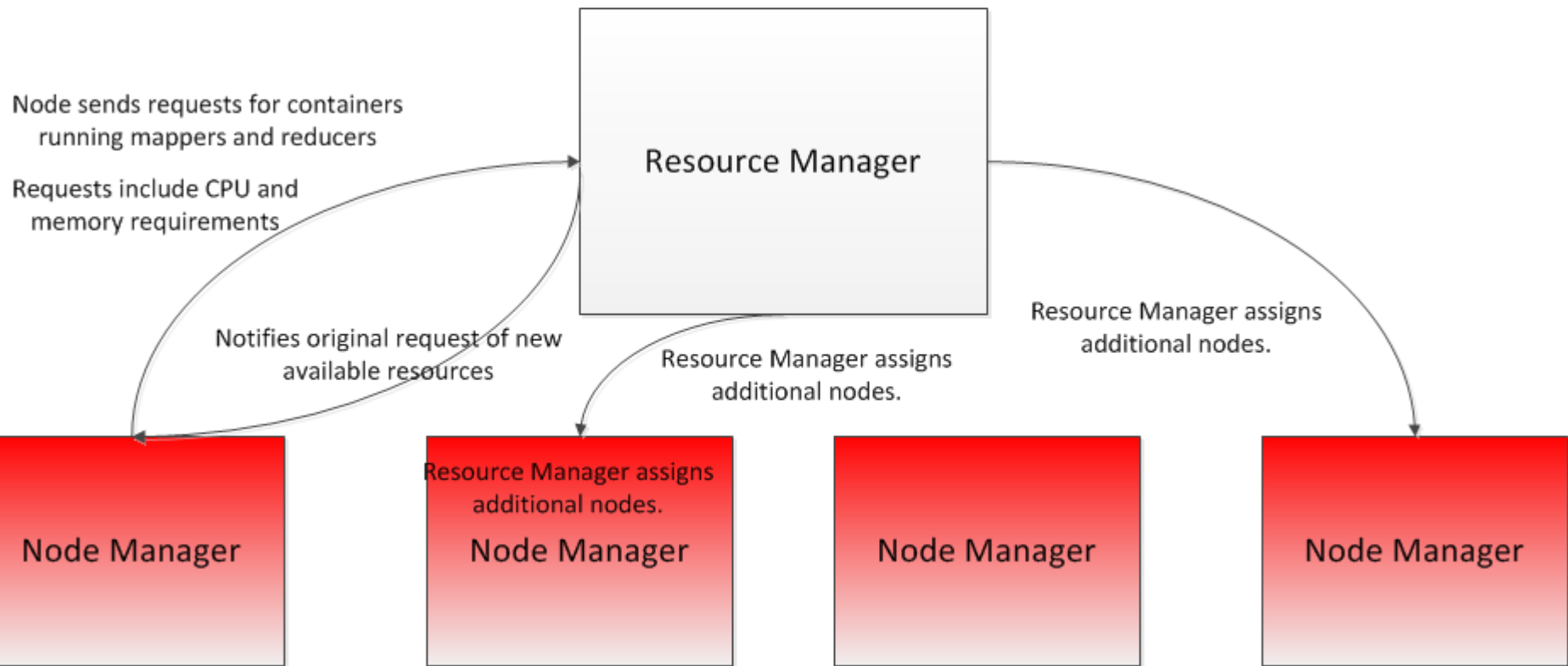
This is the logical unit containing
the resources the process needs,
i.e. Memory, CPU, etc.

A container runs a specific
application.

A node can have more than one
container.

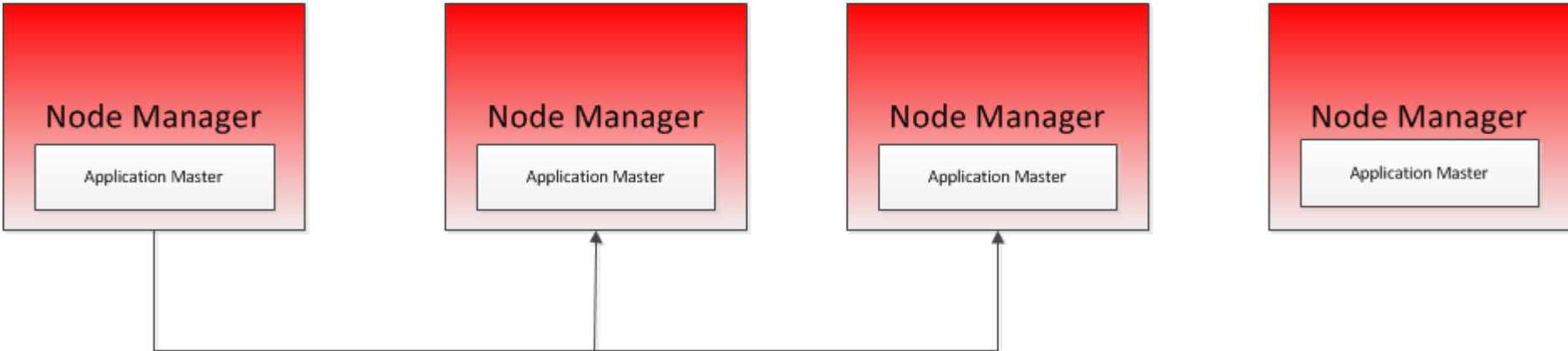


Performs the computation required for the task
If additional resources are required,
the application master makes the request





Resource Manager



Node Manager

Application Master

Node Manager

Application Master

Node Manager

Application Master

Node Manager

Application Master

Application Master on requesting node starts off
application masters on newly assigned nodes.

YARN

- All processes running in a node are run in a container.
- A container is a logical structure.
- A container is described by the resources it owns. For example, memory, CPU, storage, etc.
- A container runs a specific application assigned to it.
- A single node manager can support multiple containers.
- Once the container has been created, the Resource manager starts the application process within that

YARN

- If the application requires more resources to complete its task, it makes a request for more containers via the Node manager to the Resource Manager.
- Requests can include CPU or memory.
- The Resource Manager scans the cluster looking for compute nodes with the available resources.
- If it finds a suitable node, it requests the node manager to create a new container and assigns the extra work to it.
- Finally, the original application master making the request is notified of the additional resources.
- The application master then starts up new application masters on the newly available nodes.

YARN

- Most often, clusters have limited resources, so YARN will have to schedule work according to resource availability.
- When scheduling work, several considerations must be taken into account.
- First, write bandwidth must be limited.
- It is better to assign the map computation close to where the data lives.
- If the optimal node is busy, then we wait for the resources to become free.
- We do this using scheduling.
- Hadoop, by default, uses three types of scheduling

YARN

- The first type of scheduling is the *First In, First Out* (FIFO) scheduler.
- FIFO scheduling simply means that the first job submitted will take priority.
- FIFO scheduling can be inefficient, if Job A is submitted first, and takes significantly more time to complete than Job B, that means that we cannot run Job B first.
- FIFO scheduling can results in long wait times for jobs, therefore it is seldom used in practice.

YARN

- The next type of scheduling algorithm is the *capacity* scheduler.
- The capacity scheduler implements multiple queues
- Each queue is allocated a portion of the cluster's resources.
- Jobs can be submitted to specific queues.
- Within the queue itself, FIFO scheduling is used.
- Note that queues cannot poach each other's allocated resources.
- Capacity scheduling allows for parallelization of tasks.
- The advantage of capacity scheduling is that small jobs don't get stuck behind larger ones.
- The disadvantage is that you may have

YARN

- The last type of scheduler is the *fair* scheduler.
- In a fair scheduler, resources are always allocated proportionally to each job.
- There is no wait time for any job.
- The default policy for Hadoop is the capacity scheduler.
- The default policy can be set in yarn-policy.xml

Hadoop High Availability

- There are two ways to achieve High Availability in the cluster. They are:
- Using the Quorum Journal Manager(QJM)
- Using the NFS for the shared storage

Hadoop High Availability

- In Hadoop 2.0 onwards we have two solutions for HA.
- the High Availability feature where we can run redundant NameNodes in the same cluster out of which one will be active and other can be standby. In v2.0 only two redundant nameNodes are supported

Hadoop High Availability

- In v3.0 we can now add more than 2 redundant NameNodes.
- But only one NameNode can be active at all times.
- The active NameNode is responsible for all client operations in the cluster;
- the standby nodes are just another worker nodes but they also maintaining enough state information that in case of failure in the Active NameNode they can provide fast failover.

Hadoop High Availability

- To maintain this state and keep all the active and standby NameNodes synced, QJM comes into action.
- All the NameNodes communicate with a group of separate daemons called Journal Nodes (JNs).
- Active node logs all the modification to the majority of Journal Nodes as soon as they are done,

Hadoop High Availability

- The the Standby NameNodes are constantly watching Journal Nodes for these modifications. As soon as the modification is logged in the Journal Node, the Standby NameNodes applies these changes to its own namespace.#
- The standby NameNodes are now also up to-date in case of any failure.

Hadoop High Availability

- another precautionary measure is also taken, in case of failure of active NameNode.
- A Secondary Node will read all the logs and will make sure its namespace is up to date before taking the role of Active NameNode.

Hadoop High Availability

- In order to deploy an HA cluster, you should prepare the following:
- NameNode machines - the machines on which you run the Active and Standby NameNodes should have equivalent hardware to each other, and equivalent hardware to what would be used in a non-HA cluster.

Hadoop High Availability

- JournalNode machines - the machines on which you run the JournalNodes. The JournalNode daemon is relatively lightweight, so these daemons may reasonably be collocated on machines with other Hadoop daemons,
- for example NameNodes, the JobTracker, or the YARN ResourceManager.

Hadoop High Availability

-
- Note: There must be at least 3 JournalNode daemons, since edit log modifications must be written to a majority of Jns
- This will allow the system to tolerate the failure of a single machine.
- You may also run more than 3 JournalNodes, but in order to actually increase the number of failures the system can tolerate, you should run an odd number of JNs, (i.e. 3, 5, 7, etc.).

Hadoop High Availability

- Note that when running with N JournalNodes, the system can tolerate at most $(N - 1) / 2$ failures and continue to function normally.

Hadoop High Availability

- Similar to Federation configuration, HA configuration is backward compatible and allows existing single NameNode configurations to work without change.
- The new configuration is designed such that all the nodes in the cluster may have the same configuration without the need for deploying different configuration files to different machines based on the type of the node.

Hadoop High Availability

- Like HDFS Federation, HA clusters reuse the nameservice ID to identify a single HDFS instance that may in fact consist of multiple HA NameNodes.
- In addition, a new abstraction called NameNode ID is added with HA. Each distinct NameNode in the cluster has a different NameNode ID to distinguish it.
- To support a single configuration file for all of the NameNodes, the relevant configuration parameters are suffixed with the nameservice ID as well as the NameNode ID

Hadoop High Availability

- It is desirable for correctness of the system that only one NameNode be in the Active state at any given time.
- Importantly, when using the Quorum Journal Manager, only one NameNode will ever be allowed to write to the JournalNodes, so there is no potential for corrupting the file system metadata from a split-brain scenario.

Hadoop High Availability

- However, when a failover occurs, it is still possible that the previous Active NameNode could serve read requests to clients, which may be out of date until that NameNode shuts down when trying to write to the JournalNodes.
- For this reason, it is still desirable to configure some fencing methods even when using the Quorum Journal Manager.

Hadoop High Availability

- However, to improve the availability of the system in the event the fencing mechanisms fail, it is advisable to configure a fencing method which is guaranteed to return success as the last fencing method in the list. Note that if you choose to use no actual fencing methods, you still must configure something for this setting, for example “shell(/bin/true)”.

Hadoop High Availability

- The sshfence option SSHes to the target node and uses fuser to kill the process listening on the service's TCP port. In order for this fencing option to work, it must be able to SSH to the target node without providing a passphrase.

Hadoop High Availability

- shell - run an arbitrary shell command to fence the Active NameNode
- The shell fencing method runs an arbitrary shell command.

Hadoop High Availability

- shell - run an arbitrary shell command to fence the Active NameNode
- The shell fencing method runs an arbitrary shell command.

Hadoop High Availability

- After all of the necessary configuration options have been set, you must start the JournalNode daemons on the set of machines where they will run.
- This can be done by running the command “`hadoop-daemon.sh start journalnode`” and waiting for the daemon to start on each of the relevant machines.

Hadoop High Availability

- Once the JournalNodes have been started, one must initially synchronize the two HA NameNodes' on-disk metadata.
- If you are setting up a fresh HDFS cluster, you should first run the format command (`hdfs namenode -format`) on one of NameNodes.

Hadoop High Availability

- If you have already formatted the NameNode, or are converting a non-HA-enabled cluster to be HA-enabled, you should now copy over the contents of your NameNode metadata directories to the other, unformatted NameNode by running the command “hdfs namenode -bootstrapStandby” on the unformatted NameNode
-

Hadoop High Availability

- Running this command will also ensure that the JournalNodes (as configured by `dfs.namenode.shared.edits.dir`) contain sufficient edits transactions to be able to start both NameNodes.

Hadoop High Availability

- If you are converting a non-HA NameNode to be HA, you should run the command “hdfs namenode -initializeSharedEdits”, which will initialize the JournalNodes with the edits data from the local NameNode edits directories.
- At this point you may start both of your HA NameNodes as you normally would start a NameNode.

Hadoop High Availability

- Once your HA NameNodes are configured and started, you will have access to some additional commands to administer the HA HDFS cluster.
- Specifically, you should familiarize yourself with all of the subcommands of the “hdfs haadmin” command. Running this command without any additional arguments will display usage information.

Hadoop High Availability

- But we still have a problem.
- The above scenario only works if manual intervention is done to move to a secondary name node.
- We would really like to have an automatic failover system.
- This is where we introduce ZooKeeper.

ZooKeeper

- ZooKeeper is a distributed co-ordination service to manage large set of hosts.
- Co-ordinating and managing a service in a distributed environment is a complicated process. ZooKeeper solves this issue with its simple architecture and API.
- ZooKeeper allows developers to focus on core application logic without worrying about the distributed nature of the application.

ZooKeeper

- A distributed application can run on multiple systems in a network at a given time (simultaneously) by coordinating among themselves to complete a particular task in a fast and efficient manner.
- Normally, complex and time-consuming tasks, which will take hours to complete by a non-distributed application (running in a single system) can be done in minutes by a distributed application by using computing capabilities of all the system involved

ZooKeeper

- Apache ZooKeeper is a service used by a cluster to coordinate between themselves and maintain shared data with robust synchronization techniques.
- ZooKeeper is itself a distributed application providing services for writing a distributed application.

ZooKeeper

- The common services provided by ZooKeeper are as follows –
- Naming service – Identifying the nodes in a cluster by name. It is similar to DNS, but for nodes.
- Configuration management – Latest and up-to-date configuration information of the system for a joining node.

ZooKeeper

- Cluster management – Joining / leaving of a node in a cluster and node status at real time.
- Leader election – Electing a node as leader for coordination purpose.
- Locking and synchronization service – Locking the data while modifying it. This mechanism helps you in automatic fail recovery while connecting other distributed applications like Apache HBase.
- Highly reliable data registry – Availability of data even when one or a few nodes are down

ZooKeeper

- Distributed applications offer a lot of benefits, but they throw a few complex and hard-to-crack challenges as well.
- ZooKeeper framework provides a complete mechanism to overcome all the challenges. Race condition and deadlock are handled using fail-safe synchronization approach.
- Another main drawback is inconsistency of data, which ZooKeeper resolves with atomicity.

ZooKeeper

- Here are the benefits of using ZooKeeper –
- Simple distributed coordination process
- Synchronization – Mutual exclusion and co-operation between server processes. This process helps in Apache HBase for configuration management.
- Ordered Messages

ZooKeeper

- Serialization – Encode the data according to specific rules. Ensure your application runs consistently. This approach can be used in MapReduce to coordinate queues to execute running threads.
- Reliability
- Atomicity – Data transfer either succeed or fail completely, but no transaction is partial.

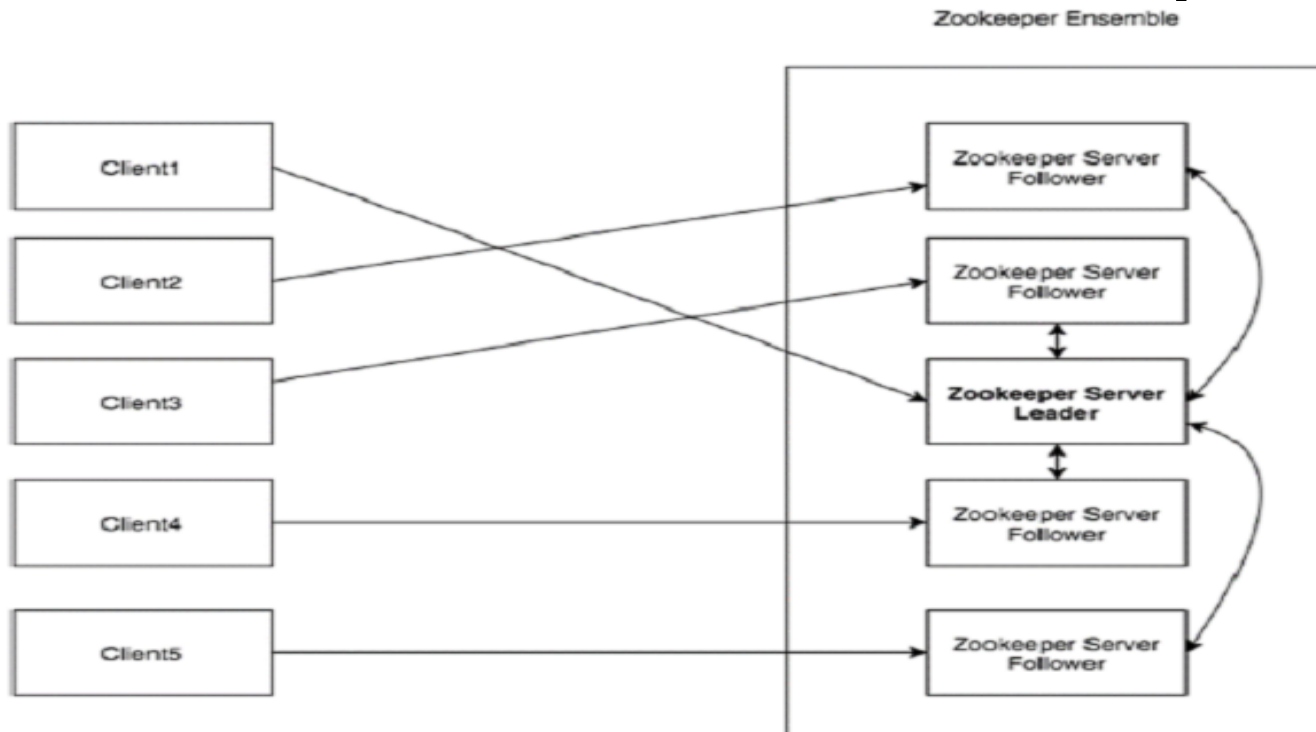
ZooKeeper

- Cluster management – Joining / leaving of a node in a cluster and node status at real time.
- Leader election – Electing a node as leader for coordination purpose.

ZooKeeper

- Locking and synchronization service – Locking the data while modifying it. This mechanism helps you in automatic fail recovery while connecting other distributed applications like Apache HBase.
- Highly reliable data registry – Availability of data even when one or a few nodes are down

Archicecture of ZooKeeper



ZooKeeper

- Clients are one of the nodes in our distributed application cluster, that access information from the server. For a particular time interval, every client sends a message to the server to let the sever know that the client is alive.
- Similarly, the server sends an acknowledgement when a client connects. If there is no response from the connected server, the client automatically redirects the message to another server.

ZooKeeper

- A Server is one of the nodes in our ZooKeeper ensemble, provides all the services to clients. Gives acknowledgement to client to inform that the server is alive.
- An Ensemble is a Group of ZooKeeper servers. The minimum number of nodes that is required to form an ensemble is 3.

ZooKeeper

- A Leader is a server node which performs automatic recovery if any of the connected node failed.
Leaders are elected on service startup.
- A Follower is a server node which follows leader instruction.

ZooKeeper

- So, how does ZooKeeper help us with Hadoop cluster management?
- Here are some of the things the Hadoop uses ZooKeeper for:
- The ZKFailoverController (ZKFC) is a new component which is a ZooKeeper client which also monitors and manages the state of the NameNode. Each of the machines which runs a NameNode also runs a ZKFC

ZooKeeper

- The ZKFC is responsible for:
- Health monitoring - the ZKFC pings its local NameNode on a periodic basis with a health-check command. So long as the NameNode responds in a timely fashion with a healthy status, the ZKFC considers the node healthy. If the node has crashed, frozen, or otherwise entered an unhealthy state, the health monitor will mark it as unhealthy.

ZooKeeper

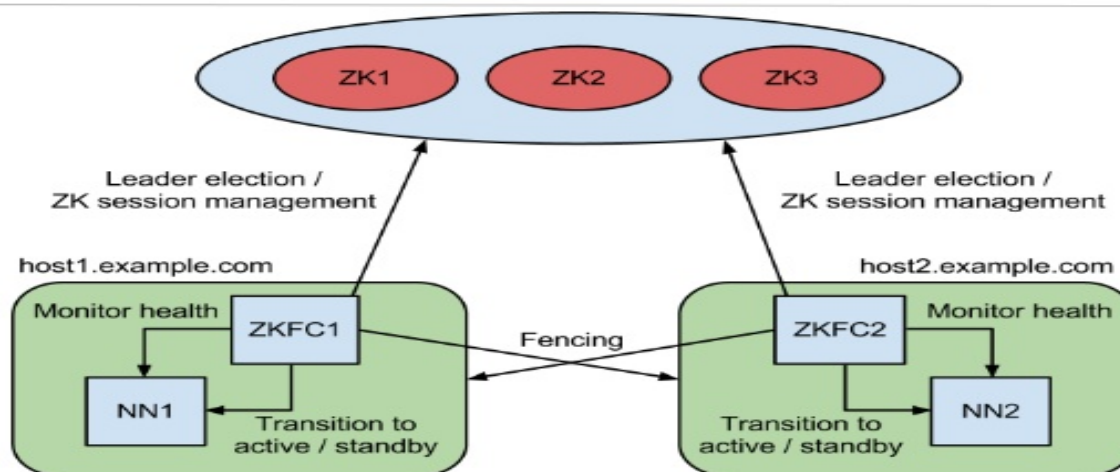
- The ZKFC is responsible for:
- ZooKeeper session management - when the local NameNode is healthy, the ZKFC holds a session open in ZooKeeper. If the local NameNode is active, it also holds a special "lock" znode. This lock uses ZooKeeper's support for "ephemeral" nodes; if the session expires, the lock node will be automatically deleted.

ZooKeeper

- The ZKFC is responsible for:
- ZooKeeper-based election - if the local NameNode is healthy, and the ZKFC sees that no other node currently holds the lock znode, it will itself try to acquire the lock. If it succeeds, then it has "won the election", and is responsible for running a failover to make its local NameNode active.

ZooKeeper

Automatic Failover Architecture



ZooKeeper&

- In a typical HA cluster, two separate machines are configured as NameNodes.
- At any point in time, exactly one of the NameNodes is in an Active state, and the other is in a Standby state.
- The Active NameNode is responsible for all client operations in the cluster, while the Standby is simply acting as a slave, maintaining enough state to provide a fast failover if necessary.
- r.

ZooKeeper

- In order for the Standby Namenode to keep its state synchronized with the Active Namenode, both nodes communicate with a group of separate daemons called JournalNodes (JNs).
- When any namespace modification is performed by the Active node, it durably logs a record of the modification to a majority of these JNs. The Standby node is reads these edits from the JNs and apply to its own name space

ZooKeeper

- In the event of a failover, the Standby will ensure that it has read all of the edits from the JournalNodes before promoting itself to the Active state. This ensures that the namespace state is fully synchronized before a failover occurs.
- It is vital for an HA cluster that only one of the NameNodes is Active at a time. ZooKeeper has been used to avoid split brain scenario so that name node state is not getting diverged due to failover

Hadoop Monitoring

- We need to be able to collect metrics.
- Collecting Namenode metrics
- Collecting DataNode metrics
- Collecting HDFS metrics.

Hadoop Monitoring

- The Namenode has an HTTP RESTful API
- Usually located at `https://<namenode>:50070`
- You can also browse to `http://<namenode>:50070/jmx`

Hadoop Monitoring

- Each datanode also has an HTTP RESTful API
- Usually can be found at:
- <http://<datanode>:50070/dfshealth.html>
- #tab-datanode
- Also, can use the JMX API to collect data.
- The ResourceManager exposes all the MapReduce counters for each job.
- Default port for these metrics is at 8088 on the resource manager.

Hadoop Monitoring

- Many third party tools are available
- Apache Ambari
- Cloudera Manager
- Both tools are free.
-

Apache Spark

- Apache Spark is a lightning-fast cluster computing designed for fast computation.
- It was built on top of Hadoop MapReduce and it extends the MapReduce model to efficiently use more types of computations which includes Interactive Queries and Stream Processing.
-

Apache Spark

-
- Industries are using Hadoop extensively to analyze their data sets.
- The reason is that Hadoop framework is based on a simple programming model (MapReduce) and it enables a computing solution that is scalable, flexible, fault-tolerant and cost effective.
- Here, the main concern is to maintain speed in processing large datasets in terms of waiting time between queries and waiting time to run the program

Apache Spark

- Spark was introduced by Apache Software Foundation for speeding up the Hadoop computational computing software process.
- Against common belief, Spark is not a modified version of Hadoop and is not, really, dependent on Hadoop because it has its own cluster management. Hadoop is just one of the ways to implement Spark.

Apache Spark

- Spark uses Hadoop in two ways – one is storage and second is processing. Since Spark has its own cluster management computation, it uses Hadoop for storage purpose only.
- The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.

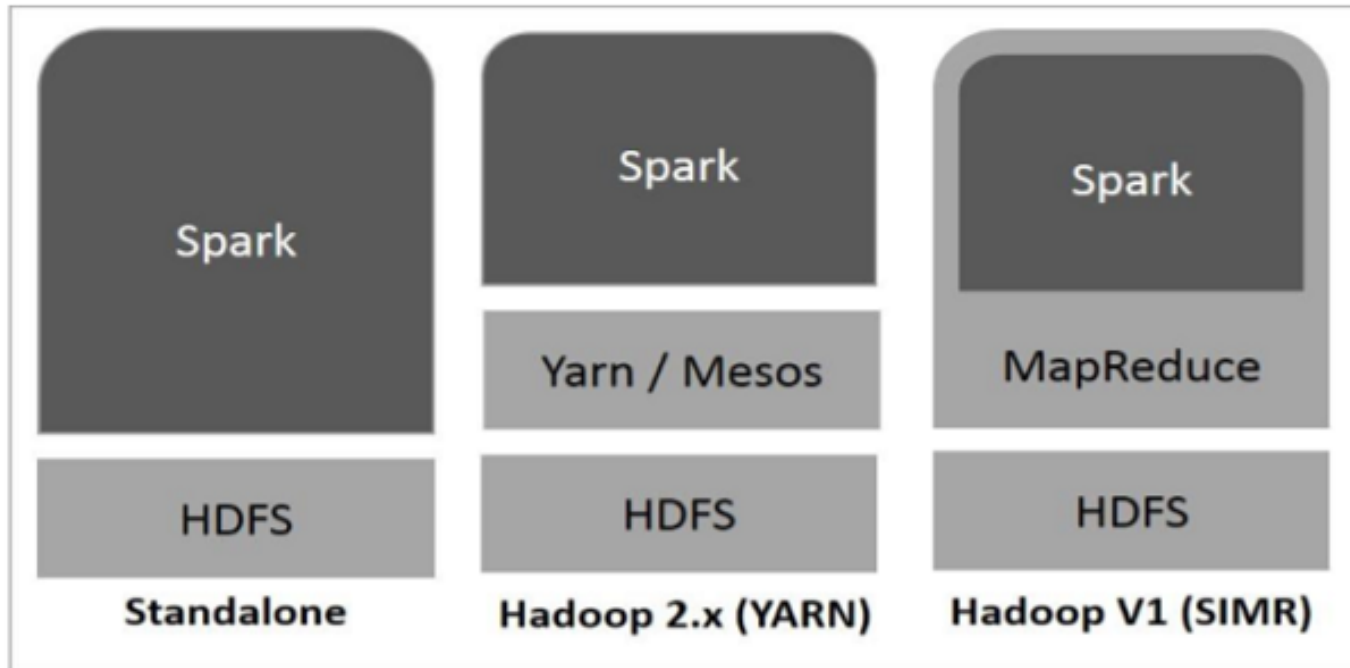
Apache Spark

- Apache Spark has following features.
- Speed – Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk.
- This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.

Apache Spark

- Supports multiple languages – Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.
- Advanced Analytics – Spark not only supports 'Map' and 'reduce'. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms

Apache Spark



Apache Spark

- There are three ways of Spark deployment as explained below.
- Standalone – Spark Standalone deployment means Spark occupies the place on top of HDFS(Hadoop Distributed File System) and space is allocated for HDFS, explicitly. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.

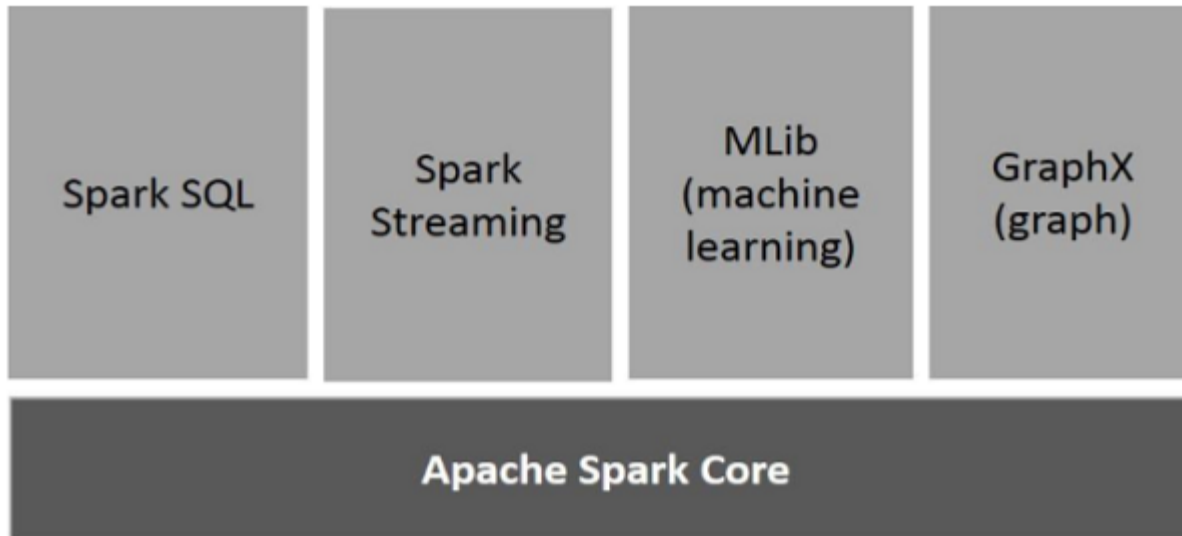
Apache Spark

- Hadoop Yarn – Hadoop Yarn deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.

Apache Spark

- Spark in MapReduce (SIMR) – Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

Apache Spark



Apache Spark

- Features of Spark include:
- Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Apache Spark

- Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. Let us first discuss how MapReduce operations take place and why they are not so efficient.

Apache Spark

- MapReduce is widely adopted for processing and generating large datasets with a parallel, distributed algorithm on a cluster.
- It allows users to write parallel computations, using a set of high-level operators, without having to worry about work distribution and fault tolerance.

Apache Spark

- Unfortunately, in most current frameworks, the only way to reuse data between computations (Ex – between two MapReduce jobs) is to write it to an external stable storage system (Ex – HDFS).
- Although this framework provides numerous abstractions for accessing a cluster's computational resources, users still want more.

Apache Spark

- Both Iterative and Interactive applications require faster data sharing across parallel jobs.
- Data sharing is slow in MapReduce due to replication, serialization, and disk IO.
- Regarding the storage system, most Hadoop applications spend more than 90% of their time doing HDFS read-write operations.

Apache Pig

- Apache Pig is a dataflow language for expressing data analysis and infrastructure processes
- Pig uses both HDFS (to do file I/O) and MapReduce (to run jobs)
- Pig is extensible through user defined functions that can be written in Java or other languages.

Apache Pig

- Pig has three components
- The first is the Pig Latin language.
- Pig Latin is a high level scripting language
- Pig Latin doesn't need any metadata or schemas
- All PigLatin statements translated into a series of MapReduce jobs.

Apache Pig

- Pig has three components.
- The second component is Grunt.
- Grunt is the interactive shell for Pig.

Apache Pig

- The Pig Latin language is a language for expressing data analytics and infrastructure processes.
- Supports many types of traditional data operations, such as join, sort, filter and others.
- Simplifies joining data and chaining jobs together

Apache Pig

- Hcatalog can be used with Pig to offload location and metadata information requirements from Pig.
- We no longer need Pig scripts to handle the data locations.
- The dataflow model with Hcatalog looks like this:
- Load (Hcatalog)
- Transform (Pig)
- Dump or Store (Pig)

Apache Pig

- Pig Latin Scripts describe a Directed Acyclic Graph
Where the edges are dataflow and the nodes are data operations.

Apache Pig

```
grunt> LOGS = load 'sample.log'
grunt> LEVELS = FOREACH LOGS GENERATE (
                                REGEX_EXTRACT($0,'(INFO|DEBUG|WARN|ERROR|
FATAL)',1) AS LOGLEVELS
grunt> DUMP LOGLEVEL
```

Apache Pig

- The Pig interpreter evaluates each statement.
- If the statement is valid, Pig adds it to the 'Plan' which is built by the interpreter..
- The interpreter will not execute any statements until it finds a DUMP or STORE command.

Apache Pig

- Pig uses the concept of a 'bag' as its relational structure.
- A bag is a collection of unordered tuples.
- Each tuple corresponds to a row in a relational table.
- Unlike a standard SQL relation, tuples don't have to have the same data types or sizes

Apache Pig

- Pig has three data structures
- The first is the bag
- The second is the tuple
- The third is the map (a collection of key/value pairs).

Apache Pig

```
logevents = LOAD 'input/mylog' AS (date:chararray,  
level:chararray,code:int, message:chararray)  
Severe = FILTER logevents BY (level = 'severe' and code >=500  
Grouped = GROUP severe BY code  
STORE grouped INTO 'output/severeevents'
```


Apache Pig

```
e1 = LOAD 'pig/input/File1' USING PigStorage(',') AS  
(name:chararray,age:int,l
```

```
salary:double)
```

```
f = FOREACH e1 GENERATE age,salary;
```

```
DESCRIBE f;
```

```
DUMP f;
```

```
zip:int,
```

Apache Pig

```
Employees = LOAD 'pig/input/File1' USING PigStorage(',')  
              AS(name:chararray,age:int, zip:int, salary:double);  
sorted = ORDER employees by Salary;
```

```
Employees = LOAD 'pig/input/File1' USING PigStorage(',')  
              AS (name:chararray, age:int, zip:int, salary:double);  
Agegroup = GROUP employees by age;  
li = LIMIT agegroup 100;
```

Apache Pig

```
e1 = LOAD 'pig/input/File1' USING PigStorage(',')  
      AS (name:chararray,age:int,zip:int,salary:double)  
e2 = LOAD 'pig/input/File2' USING PigStorage(',')  
      AS (name:chararray, phone:chararray);  
e3 = JOIN e1 by name, e2 by name  
DESCRIBE e3;  
DUMP e3;
```

Apache Pig

- We can use illustrate, explain, and describe to help us debug pig latin scripts
- Suggest using local mode to test the script before running it on the cluster
- Local node is slow, but there is no waiting for a slot for the job to run
- Logs for local mode appear on your screen instead of on the remote task node.
- Local mode runs all in your local process so you can attach a debugger to it.

Apache Hive

- Hive is a data warehouse layer build on top of Hadoop
- Allows you define a structure for your unstructured Big Data
- Simplifies analysis and queries with an SQL like language called HiveSQL

Apache Hive

- Hive is not a relational database
- Hive uses a database to store metadata but all data that Hive processes is stored in the HDFS.
- Hive is not designed for OLTP
- Hive runs on Hadoop (a batch processing system)
- Hive latency is generally high
- Hive is not suited for real-time queries and row-level updates.

Apache Hive

- Hive takes multi structured data and gives back a view and a structure which can make sense to a business analyst.
- Hive supports uses such as
 - Ad-hoc queries
 - Summarization
 - Data analysis

Apache Hive

- The Hive architecture includes :
- The Hive command line interface
- The Metastore, which stores schema information and provides structure to stored data
- The Hive QL, which supports query processing, compiling and optimizing

Apache Hive

- Hive is a good choice when you want to query the data
- When you need an answer to a specific question
- If you are already familiar with SQL
- Pig is a good choice when you want to perform Extract , Transform and Load (ETL)
- When you want to prepare data for easier analysis
- When you have a long series of steps to do.

Apache Hive

- The HiveQL is similar to other SQLs
- It uses standard relational database concepts such as tables, rows, columns and schema)
- Supports multi-table insert via your code. It accesses 'Big Data' via tables
- Converts SQL queries into MapReduce jobs
- Also supports plugging custom MapReduce scripts into queries

Apache Hive

- A Hive table consists of:
- Data: typically a file or group of files stored on the HDFS.
- Schema: in the form of metadata stored in a relational database
- Schema and data are separate
- A schema can be defined for existing data
- Data can be added or removed independently.
- Hive can be pointed at existing data.
- You must have a schema defined if you want to point Hive to existing data. In the HDFS.

Apache Hive

```
Hive> CREATE TABLE mytable (name string, age int)
      ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ';'
      STORED AS TEXTFILE;
```

Apache Hive

```
Hive> LOAD DATA LOCAL INPATH  
      '/tmp/customers.csv' OVERWRITE INTO TABLE customers;
```

```
Hive> LOAD DATA INPATH  
      'user/train/customers.csv' OVERWRITE INTO TABLE customers
```

```
INSERT INTO birthdays SELECT firstName, lastName, birthday FROM  
      customers where birthday IS NOT NULL;
```