1.

a. Describe the significance of t-test.

Ans. t-test will be performed when the standard deviation is unknown. It is a parametric test.

b. Write the R code for two sample t-test

Ans.
```
nsmk<-lung[lung$Smoker==0,]
smk<-lung[lung$Smoker==1,]
res<-t.test(smk,nsmk)
res
```

c. What is Machine Learning?

Ans. Machine Learning is getting of knowledge by study, experience or being taught.

d. Differentiate between Supervised learning, Unsupervised Learning?

Ans. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output. Unsupervised learning is where you only have input data (X) and no corresponding output variables.

e. What HDFS contains?

Ans. Name node
Data node
Secondary name node

f. Describe Map Reduce?

Ans. **MapReduce** job usually splits the input data-set into independent chunks which are processed by the **map** tasks in a completely parallel manner. The framework sorts the outputs of the **maps**, which are then input to the **reduce** tasks.

g. Significance of Secondary Name Node in HDFS?

Ans. Periodic merge of Transaction log
Does the check pointing.

h. Write applications of Map Reduce?

- distributed grep
- distributed sort

i. What Hadoop eco system contains?

Ans. Hadoop Distributed File System(HDFS)
MapReduce

j. Significance of i)Job Tracker ii)Task Tracker

Ans. **Job Tracker** - Schedules jobs and tracks the assign jobs to Task tracker.
Task Tracker - Tracks the task and reports status to Job Tracker.

k. Describe Fail over and Fencing?

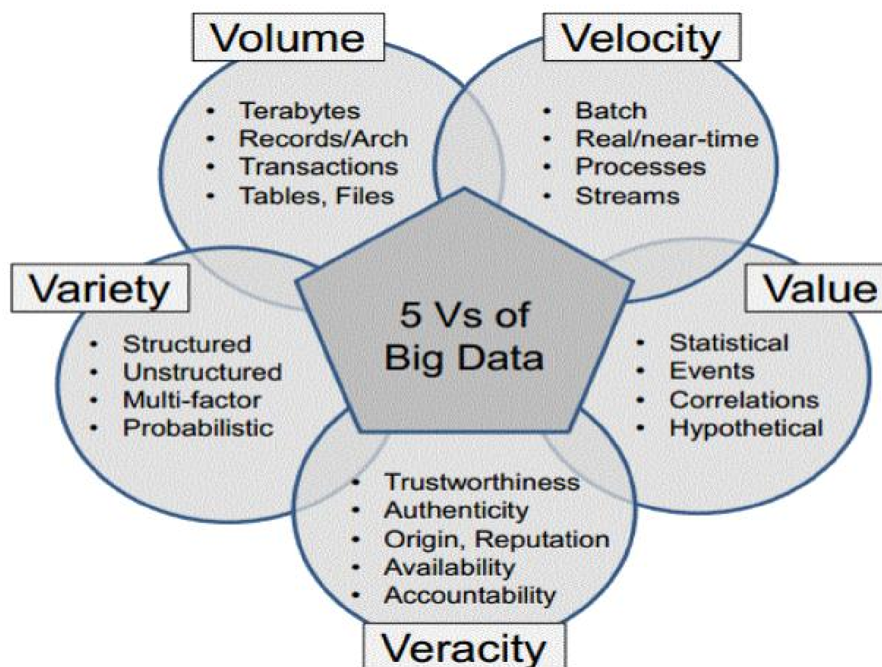
The transition from the active namenode to the standby is managed by a new entity in the system called the *failover controller*.
The HA implementation goes to great lengths to ensure that the previously active namenode is prevented from doing any damage and causing corruption—a method known as *fencing*.

l. What is YARN ?

Ans. YARN stands for **Yet Another Resource Negotiator**. It's a framework introduced in 2010 by a group Yahoo!. This is considered as next generation of MapReduce. YARN is not specific for MapReduce. It can be used for running any application.

UNIT-1

2 a. Explain the characteristics of Big Data



- **Volume:** The Big word in Big data itself defines the volume. At present the data existing is in petabytes (10^{15}) and is supposed to increase to zettabytes (10^{21}) in nearby future. Data volume measures the amount of data available to an organization, which does not necessarily have to own all of it as long as it can access it.
- **Velocity:** Velocity in Big data is a concept which deals with the speed of the data coming from various sources. This characteristic is not being limited to the speed of incoming data but also speed at which the data flows and aggregated.

Variety: Data variety is a measure of the richness of the data representation text, images video, audio, etc. Data being produced is not of single category as it not only includes the traditional data but also the semi structured data from various resources like web Pages, Web Log Files, social media sites, e-mail, documents.

Value: Data value measures the usefulness of data in making decisions.Data science is exploratory and useful in getting to know the data, but "analytic science" encompasses the predictive power of big data. User can run certain queries against the data stored and thus can deduct important results from the filtered data obtained and can also rank it according to the dimensions they require. These reports help these people to find the business trends according to which they can change their strategies.

Veracity: it refers to the messiness or trustworthiness of the data. Today quality and accuracy of data are less controllable (hash tags, abbreviations, typos and colloquial speech) but technology now allows us to deal with it. How to find high-quality data from the vast collections of data that are out there on the Web.

Complexity: Complexity measures the degree of interconnectedness (possibly very large) and interdependence in big data structures such that a small change (or combination of small changes) in one or a few elements can yield very large changes or a small change that ripple across or cascade through the system and substantially affect its behavior, or no change at all

2b. Describe the applications of Big Data?

Amazon.com handles millions of back-end operations and have 7.8 TB, 18.5 TB, and 24.7 TB Databases.

Walmart is estimated to store more than 2.5 PB Data for handling 1 million transactions per hour.

The Large Hadron Collider (LHC) generates 25 PB data before replication and 200 PB Data after replication.

Sloan Digital Sky Survey ,continuing at a rate of about 200 GB per night and has more than 140 TB of information.

Utah Data Center for Cyber Security stores Yottabytes (10^{24}).

3 What is Hypothesis Testing? Explain the following terms with examples?

a)Null Hypothesis: The hypothesis to be tested is called the null hypothesis and given the symbol H0.

The null hypothesis states that there is no difference between a hypothesized population mean and a sample mean.

It is the status quo hypothesis.

For example, if we were to test the hypothesis that college freshmen study 20 hours per week, we would express our

null hypothesis as:
H0 : $\mu = 20$

b)Alternative Hypothesis: We test the null hypothesis against an alternative hypothesis, which is given the symbol Ha. The alternative hypothesis is often the hypothesis that you believe yourself! It includes the outcomes not covered by the null hypothesis.

In this example, our alternative hypothesis would express that freshmen do not study 20 hours per week:

The result of a hypothesis test:
'Reject H_0 in favour of H_A' OR 'Do not reject H_0'

c) Degrees of Freedom: Degrees of freedom are essentially the number of samples that have the 'freedom' to change without affecting the sample mean.

d) P value: you will receive a p-value as part of youroutput.
The p-values is the likelihood of observing that particuluar sample value if the null hypothesis were true.
Therefore, if the p-value is smaller than your significance level, you can reject the null hypothesis.

e) How to calculate t test value?

1. define null and alternative hypothesis.
2. take alpha.
3. calculate degrees of freedom.
4. state decision rule.
5.Calculate test-statistic.
6. Take result
7.conclusion.

f)Type- 1 error & Type-2 error:

A Type I error is when we reject the null hypothesis when it is true

Type II error is when we do not reject the null hypothesis, even when it is false.

UNIT-2

4 a. i) Apply the Hierarchical clustering using Single Linkage method for the following data, construct Hierarchical Tree.
ii) Write R code for Hierarchical clustering using single linkage method for the following

	BA	**FI**	**MI**	**NA**	**RM**	**TO**
BA	0	662	877	255	412	996

FI	662	0	295	468	268	400
MI	877	295	0	754	564	138
NA	255	468	754	0	219	869
RM	412	268	564	219	0	669
TO	996	400	138	869	669	0

Problem.:

i) Let's now see a simple example: a hierarchical clustering of distances in kilometers between some Italian cities. The method used is single-linkage.

ii) **Input distance matrix** (L = 0 for all the clusters):

	BA	FI	MI	NA	RM	TO
BA	0	662	877	255	412	996
FI	662	0	295	468	268	400
MI	877	295	0	754	564	138
NA	255	468	754	0	219	869
RM	412	268	564	219	0	669
TO	996	400	138	869	669	0

iii) The nearest pair of cities is MI and TO, at distance 138. These are merged into a single cluster called "MI/TO". The level of the new cluster is L(MI/TO) = 138 and the new sequence number is m = 1.

iv) Then we compute the distance from this new compound object to all other objects. In single link clustering the rule is that the distance from the compound object to another object is equal to the shortest distance from any member of the cluster to the outside object. So the distance from "MI/TO" to RM is chosen to be 564, which is the distance from MI to RM, and so on.

v)

vi) After merging MI with TO we obtain the following matrix:

	BA	FI	MI/TO	NA	RM
BA	0	662	877	255	412
FI	662	0	295	468	268
MI/TO	877	295	0	754	564
NA	255	468	754	0	219
RM	412	268	564	219	0

vii) min d(i,j) = d(NA,RM) = 219 => merge NA and RM into a new cluster called NA/RM

viii) L(NA/RM) = 219

ix) m = 2

x)

	BA	FI	MI/TO	NA/RM
BA	0	662	877	255
FI	662	0	295	268
MI/TO	877	295	0	564
NA/RM	255	268	564	0

xi) min d(i,j) = d(BA,NA/RM) = 255 => merge BA and NA/RM into a new cluster called BA/NA/RM

xii) L(BA/NA/RM) = 255

xiii) m = 3

xiv)

	BA/NA/RM	FI	MI/TO
BA/NA/RM	0	268	564
FI	268	0	295
MI/TO	564	295	0

xv) min d(i,j) = d(BA/NA/RM,FI) = 268 => merge BA/NA/RM and FI into a new cluster called BA/FI/NA/RM

xvi) L(BA/FI/NA/RM) = 268

xvii) m = 4

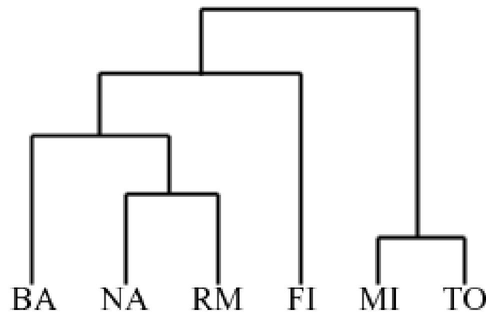
xviii)

	BA/FI/NA/RM	MI/TO
BA/FI/NA/RM	0	295
MI/TO	295	0

xix) Finally, we merge the last two clusters at level 295.

xx)

xxi) The process is summarized by the following hierarchical tree:



xxii)

ii) R-code for hierarchical clustering for single linkage method:

```
a <- c(0,662,877,255,412,996)
b <-  c(662,0,295,468,268,400)
c <-  c(877,295,0,754,564,138)
d <-  c(255,468,754,0,219,869)
```

```
e <-  c(412,268,564,219,0,669)
f <-  c(996,400,138,869,669,0)
p <- rbind (a,b,c,d,e,f)
p1 <-  matrix(p, 6, 6, dimnames=list(c("a","b","c","d","e","f")), c("a","b","c","d","e","f"))
p1 <- as.data.frame(p1)
m <- apply(p1,2,mean)
n <- apply(p1,2,sd)
z <- scale(p1,m,n)
dist1 <- dist(z)
hcs <- hclust(dist1,method="single")
plot(hcs, hang= -1)
```

5 a. Write the R code for cluster analysis on iris data set using K-means algorithm
iris dataset(Sepal Length, Sepal Width, Petal Length, Petal Width, Species)

R-code for k-means algorithm using iris dataset:

```
r1<-iris
names(r1)
r2<-r1[,-5]
names(r2)
r3<-kmeans(r2,3)
r3
table(iris$Species,r3$cluster)
plot(iris$Petal.Length,iris$Petal.Width,col=r3$cluster)
```

b. Write the R code for cluster analysis on Lung Capacity data set using K-medoids algorithm.
LungCapacity data set (Gender, Height, Smoker, Exercise, Age, Lung Capacity)

R-code for k-medoids algorithm using K-medoids algorithm:

```
pamkm<-Lung Capacity
names(pamkm)
pamkm1<-pamkm[,-1]
names(pamkm1)
library(cluster)
t<-pam(pamkm1,3)
t
table(iris$Species,t$clustering)
plot(t)
layout((matrix(c(1,2),1,2)))
plot(t)
```

UNIT-3

6 a. Explain HDFS concepts in detail

HDFS Concepts:

- Blocks
- Namenodes and Datanodes
- Block Caching
- HDFS Federation
- HDFS High Availability
- Fail over and Fencing

1.Blocks

- File Blocks
  - 64MB(default), 128 (Recommended)

2.Namenodes and Datanodes

DataNode is slave machine in Hadoop cluster running the data node daemon.

A Master called the Name Node .

The Name Node keep track of the file metadata –which files are running in the system and how each file is broken down into the Name Node to keep the metadata current.

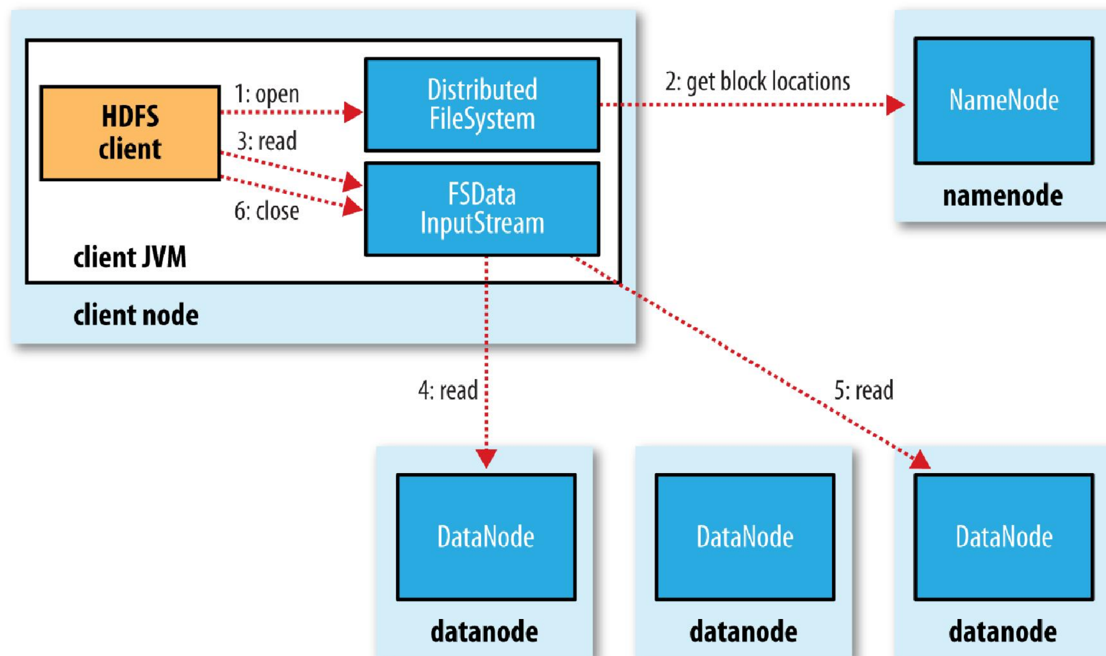
3. Block Caching

- A Datanode reads blocks from disk.
- Frequently accessed files the blocks may be explicitly cached in the datanode's memory.
- By default , block is cached only one Datanode's memory.

4. HDFS Federation

- The Namenode keeps a reference to every file and blocks in the file system in memory.
- It means , on very large clusters with many files, memory becomes the limiting factor for scaling.
- How much memory does a Name Node need ?
- HDFS Federation introduced in the 2.x release series , allows .

b. Explain the anatomy of how data read from HDFS



- The client opens the file it wishes to read by calling open() on the File System object, which for HDFS is an instance of DistributedFileSystem (step-1)
- DistributedFileSystem calls the name node, using remote procedure calls (RPCs), to determine the locations of the first few blocks in the file (step 2).
- For each block, the name node returns the addresses of the data nodes that have a copy of that block.
- If the client is itself a data node the client will read from the local data node if that data node hosts a copy of the block
- The Distributed File System returns an FSDataInputStream to the client for it to read data from. FSDataInputStream in turn wraps a DFSInputStream, which manages the data node and name node I/O.
- The client then calls read() on the stream (step 3). DFSInputStream, which has stored the data node addresses for the first few blocks in the file, then connects to the first (closest) data node for the first block in the file.
- Data is streamed from the data node back to the client, which calls read() repeatedly on the stream (step 4).
- When the end of the block is reached, DFSInputStream will close the connection to the data node, then find the best data node for the next block (step 5).
- This happens transparently to the client, which from its point of view is just reading a continuous stream.
- Blocks are read in order, with the DFSInputStream opening new connections to data nodes as the client reads through the stream.
- It will also call the name node to retrieve the data node locations for the next batch of blocks as needed.
- When the client has finished reading, it calls close() on the FSDataInputStream (step 6).
- During reading, if the DFSInputStream encounters an error while communicating with a data node,

- It will try the next closest one for that block. It will also remember data nodes that have failed so that it doesn't needlessly retry them for later blocks.
- The DFSInput Stream also verifies checksums for the data transferred to it from the data node.
- If a corrupted block is found, the DFSInputStream attempts to read a replica of the block from another data node; it also reports the corrupted block to the name node.

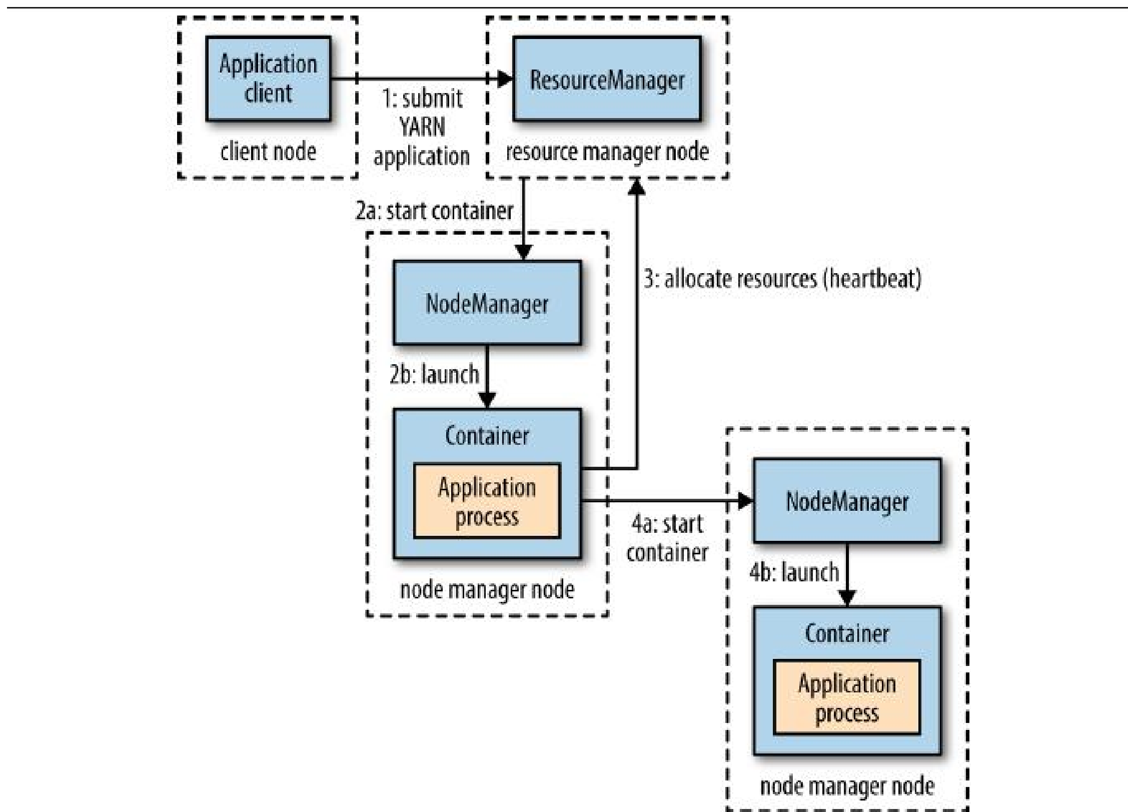
7 a. Explain the components of YARN.

- The problem is solved by splitting the responsibility of JobTracker (in Classic MapReduce) to different components. Because of which, there are more entities involved in YARN (compared to Classic MR). The entities in YARN are as follows;
  - Client: which submits the MapReduce job
  - Resource Manager: which manages the use of resources across the cluster. It creates new containers for Map and Reduce processes.
  - Node Manager: In every new container created by Resource Manager, a Node Manager process will be run which oversees the containers running on the cluster nodes. It doesn't matter if the container is created for Map or Reduce or any other process. Node Manager ensures that the application does not use more resources than what it is allocated with.
  - Application Master: which negotiates with the Resource Manager for resources and runs the application-specific process (Map or Reduce tasks) in those clusters. The Application Master & the MapReduce tasks run in containers that are scheduled by the resource manager and managed by the node manager.

b. Explain how YARN runs an application on HDFS?

- YARN provides its core services via two types of long-running daemon: a *resource manager (one per cluster) to manage the use of resources across the cluster, and node managers running on all the nodes in the cluster to launch and monitor containers.*
- A container executes an application-specific process with a constrained set of resources (memory, CPU, and so on).
- To run an application on YARN, a client contacts the resource manager and asks it to run an *application master process (step 1).*
- The resource manager then finds a node manager that can launch the application master in a container (steps 2a and 2b).
- Precisely what the application master does once it is running depends on the application. It could simply run a computation in the container it is running in and return the result to the client. Or it could request more containers from the resource managers (step 3), and use them to run a distributed computation (steps 4a and 4b).
- YARN itself does not provide any way for the parts of the application (client, master, process) to communicate with one another.
- Most nontrivial YARN applications use some form of remote

communication (such as Hadoop's RPC layer) to pass status updates and results back to the client, but these are specific to the application.



Resource Requests:

- YARN has a flexible model for making resource requests. A request for a set of containers can express the amount of computer resources required for each container (memory and CPU), as well as locality constraints for the containers in that request.
- Locality is critical in ensuring that distributed data processing algorithms use the cluster bandwidth efficiently, so YARN allows an application to specify locality constraints for the containers it is requesting. Locality constraints can be used to request a container on a specific node or rack, or anywhere on the cluster (off-rack).
- Sometimes the locality constraint cannot be met, in which case either no allocation is made or, optionally, the constraint can be loosened. For example, if a specific node was requested but it is not possible to start a container on it (because other containers are running on it), then YARN will try to start a container on a node in the same rack, or,if that's not possible, on any node in the cluster.
- In the common case of launching a container to process an HDFS block (to run a map task in MapReduce, say), the application will request a container on one of the nodes hosting the block's three replicas, or on a node in one of the racks hosting the replicas,or, failing that, on any node in the cluster.
- A YARN application can make resource requests at any time while it is running. For example, an application can make all of its requests up front, or it can take a more

dynamic approach whereby it requests more resources dynamically to meet the changing needs of the application.

Application Lifespan :

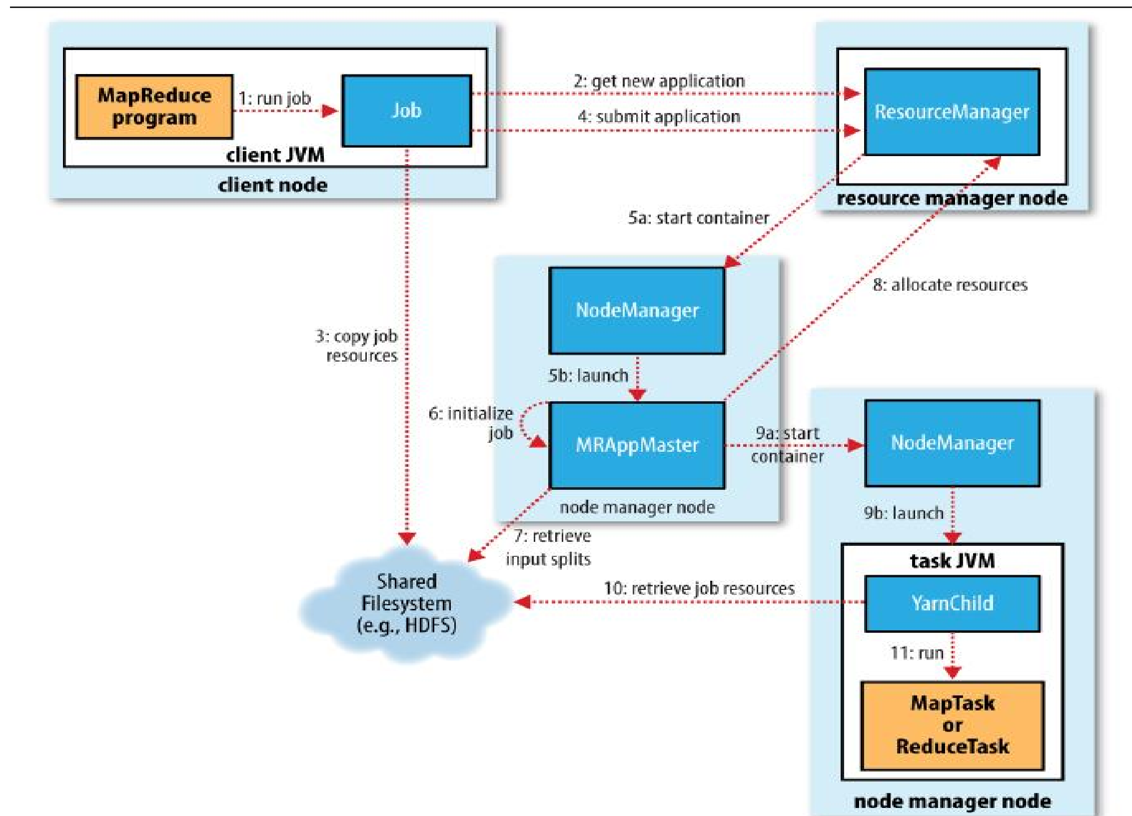
- The lifespan of a YARN application can vary dramatically: from a short-lived application of a few seconds to a long-running application that runs for days or even months.
- Rather than look at how long the application runs for, it's useful to categorize applications in terms of how they map to the jobs that users run.
- The simplest case is one application per user job, which is the approach that MapReduce takes.
- The second model is to run one application per workflow or user session of (possibly unrelated) jobs.
- This approach can be more efficient than the first, since containers can be reused between jobs, and there is also the potential to cache intermediate data between jobs.
- Spark is an example that uses this model.
- The third model is a long-running application that is shared by different users. Such an application often acts in some kind of coordination role.
- For example, Apache Slider has a long-running application master for launching other applications on the cluster.
- This approach is also used by Impala to provide a proxy application that the Impala daemons communicate with to request cluster resources.
- The "always on" application master means that users have very lowlatency responses to their queries since the overhead of starting a new application master is avoided.

Building YARN Applications :

- Writing a YARN application from scratch is fairly involved, but in many cases is not necessary, as it is often possible to use an existing application that fits the bill.
- For example,if you are interested in running a directed acyclic graph (DAG) of jobs, then Spark or Tez is appropriate; or for stream processing, Spark, Samza, or Storm works.
- There are a couple of projects that simplify the process of building a YARN application.
- Apache Slider, mentioned earlier, makes it possible to run existing distributed applications on YARN.
- Users can run their own instances of an application (such as HBase) on a cluster, independently of other users, which means that different users can run different versions of the same application.
- Slider provides controls to change the number of nodes an application is running on, and to suspend then resume a running application.
- Apache Twill is similar to Slider, but in addition provides a simple programming model for developing distributed applications on YARN.
- Twill allows you to define cluster processes as an extension of a Java Runnable, then runs them in YARN containers on the cluster.
- Twill also provides support for, among other things, real-time logging (log events from runnables are streamed back to the client) and command messages (sent from the client to runnables).
- In cases where none of these options are sufficient—such as an application that has complex scheduling requirements—then the *distributed shell application that is a part* of the YARN project itself serves as an example of how to write a YARN application.
- It demonstrates how to use YARN's client APIs to handle communication between the client or application master and the YARN daemons.

UNIT-4

8 Explain how HDFS runs a MapReduce job?



- Job Submission
- Job Initialization
- Task Assignment
- Task Execution
- Progress and Updates
- Job Completion
- Failures

Job Submission:

- The submit() method on Job creates an internal Job Submitter instance and calls submit Job Internal() on it.
- Having submitted the job, waitForCompletion() polls the job's progress once per second and reports the progress to the console if it has changed since the last report.

- When the job completes successfully, the job counters are displayed. Otherwise, the error that caused the job to fail is logged to the console.
- The job submission process implemented by JobSubmitter does the following:

1. Asks the resource manager for a new application ID, used for the MapReduce jobID (step 2).
2. Checks the output specification of the job. For example, if the output directory has not been specified or it already exists, the job is not submitted and an error is thrown to the MapReduce program.
3. Computes the input splits for the job. If the splits cannot be computed, the job is not submitted and an error is thrown to the MapReduce program.
4. Copies the resources needed to run the job, including the job JAR file, the configuration file, and the computed input splits, to the shared filesystem in a directory named after the job ID (step 3). The job JAR is copied with a high replication factor so that there are lots of copies across the cluster for the node managers to access when they run tasks for the job.
5. Submits the job by calling submitApplication() on the resource manager(step 4).

Job Initialization:

- The application master must decide how to run the tasks that make up the MapReduce job. If the job is small, the application master may choose to run the tasks in the same JVM as itself.
- Such a job is said to be *uberized, or* run as an *uber task.*
- What qualifies as a small job? By default, a small job is one that has less than 10 mappers, only one reducer, and an input size that is less than the size of one HDFS block.
- Finally, before any tasks can be run, the application master calls the setupJob() method on the OutputCommitter. It will create the final output directory for the job and the temporary working space for the task output.

Task Assignment:

- If the job does not qualify for running as an uber task, then the application master requests containers for all the map and reduce tasks in the job from the resource manager(step 8).
- Requests for map tasks are made first and with a higher priority than those for reduce tasks, since all the map tasks must complete before the sort phase of the reduce can start.
- Requests for reduce tasks are not made until 5% of map tasks have completed.
- Reduce tasks can run anywhere in the cluster, but requests for map tasks have data locality constraints that the scheduler tries to honor.
- Requests also specify memory requirements and CPUs for tasks.
- By default, each map and reduce task is allocated 1,024 MB of memory and one virtual core.
- The values are configurable on a per-job basis via the following properties:
  mapreduce.map.memory.mb,
  mapreduce.reduce.memory.mb,
  mapreduce.map.cpu.vcores

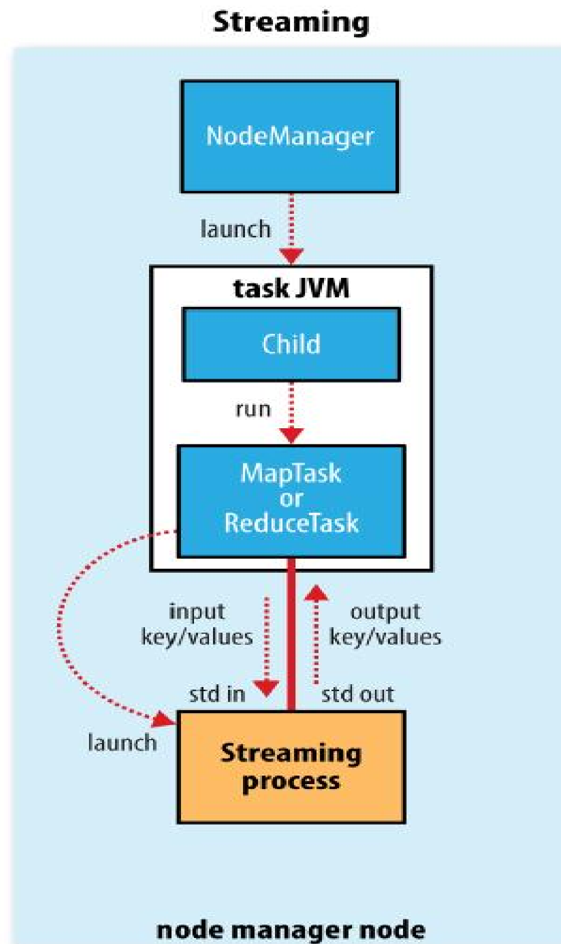
Task Execution:

- Once a task has been assigned resources for a container on a particular node by the resource manager's scheduler, the application master starts the container by contacting the node manager (steps 9a and 9b).
- The task is executed by a Java application whose main class is YarnChild. Before it can run the task, it localizes the resources that the task needs, including the job

configuration and JAR file, and any files from the distributed cache . Finally, it runs the map or reduce task (step 11).

- The YarnChild runs in a dedicated JVM, so that any bugs in the user-defined map and reduce functions (or even in YarnChild) don't affect the node manager—by causing it to crash or hang.

The relationship of the Streaming executable to the node manager and the task container



Progress and Status Updates:

- MapReduce jobs are long-running batch jobs, taking anything from tens of seconds to hours to run.
- A job and each of its tasks have a *status,* which includes such things as the state of the job or task the progress of maps and reduces, the values of the job's counters, and a status message or description.
- When a task is running, it keeps track of its *progress.* For map tasks, this is the proportion of the input that has been processed. For reduce tasks, it's a little more complex.

Job Completion:

- When the application master receives a notification that the last task for a job is complete, it changes the status for the job to "successful."
- when the Job polls for status, it learns that the job has completed successfully, so it prints a message to tell the user and then returns from the waitForCompletion() method. Job statistics and counters are printed to the console at this point.

- Finally, on job completion, the application master and the task containers clean up their working state (so intermediate output is deleted), and the OutputCommitter's commit Job() method is called. Job information is archived by the job history server to enable later interrogation by users if desired.

  Failures:

  In the real world, user code is buggy, processes crash, and machines fail. One of the major benefits of using Hadoop is its ability to handle such failures and allow your job to complete successfully.

  We need to consider the failure of any of the following entities:

  Task Failure: The most common occurrence of this failure is when user code in the map or reduce task throws a runtime exception. If this happens, the task JVM reports the error back to its parent application master before it exits. The error ultimately makes it into the user logs. The application master marks the task attempt as *failed*, and frees up the container so its resources are available for another task.

  Application Master Failure: YARN imposes a limit for the maximum number of attempts for any YARN application master running on the cluster, Individual applications may not exceed this limit.The limit is set by yarn.resourcemanager.am .max-attempts and defaults to 2, If you want to increase the number of MapReduce application master attempts, you will have to increase the YARN setting on the cluster. An application master sends periodic heartbeats to the resource manager, and in the event of application master failure, the resource manager will detect the failure and start a new instance of the master running in a new container (managed by a node manager).

  Node Manager Failure: If a node manager fails by crashing or running very slowly, it will stop sending heartbeats to the resource manager (or send them very infrequently). The resource manager will notice a node manager that has stopped sending heartbeats if it hasn't received one for 10 minutes (this is configured, in milliseconds, via the yarn.resourcemanager.nm.liveness-monitor.expiry-interval-ms property) and remove it from its pool of nodes to schedule containers on.

  Resource Manager Failure: Failure of the resource manager is serious, because without it, neither jobs nor task containers can be launched. In the default configuration, the resource manager is a single point of failure, since in the (unlikely) event of machine failure, all running jobs fail—and can't be recovered.

  To achieve high availability (HA), it is necessary to run a air of resource managers in an active-standby configuration. If the active resource manager fails, then the standby can take over without a significant interruption to the client.

9 a. Explain the features of Map Reduce.

features:

- Counters
- Speculative Execution
- Distributed Cache

Counters:

• Instrument Job's metrics – Gather statistics – Quality control – confirm what was expected – Diagnostics

• Framework provides a set of built-in metrics – For example bytes processed for input and output

• User can create new counters – Number of records consumed – Number of errors or warnings
• Counters are divided into groups
• Tracks total, mapper and reducer counts.

Built-in Counters:

Maintains and sums up counts

Several groups for built-in counters – Job Counters – documents number of map and reduce tasks launched, number of failed tasks – File System Counters – number of bytes read and written – Map-Reduce Framework – mapper, reducer, combiner input and output records counts, time and memory statistics

Distributed Cache:

- A mechanism to distribute files
- Make them available to MapReduce task code
- yarn command provides several options to add distributed files
- Can also use Java API directly
- Supports – Simple text files – Jars – Archives: zip, tar, tgz/tar.gz

b. How different failures are handled by HDFS eco system.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The project includes these modules:

Hadoop Common: The common utilities that support the other Hadoop modules.

Hadoop Distributed File System (HDFS™): A distributed file system that provides high-throughput access to application data.

Hadoop YARN: A framework for job scheduling and cluster resource management.

Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.