Chandrawanshi Mangesh Shivaji
CS555 1801CS16 Mshandrao.

Page No.
Date / /

Q.1) (c) the first Mapper has finished processing it input.

Q.2) (c) Shuffle and Sort

Q.3) (b) False. [Mappers & Reducers may generate any no. of key-value pairs (including 0)]

Q.4) (b) setup(): once, cleanup(): once

Q.5) (e) File divided into fixed size blocks, stored on multiple datanodes. Each block is replicated 3 times by default. HDFS guarantees that different blocks from same file are never on same datanode

Q.6) (c) hadoop jar MyTar.jar MyDriverClass inputdir outputdir

Q.7) (c) Namenode

Q.8) (c) Online transaction processing (OLTP) for an ecommerce website.

Q.9) (c) Storing the data partitions

Q.10) (d) WordCount Runs Successfully.

Chandrawanshi Mangesh Shivaji.
CS555 1801CS16 MShandrav

Page No.
Date / /

Q.11) →

(a) →

(i) Given,
Bigdata analytics exam course midsem exam IIT Patna.
Task: Count occurences for each word.

Mapper Object ⇒ 1 (∵ Task is only one) and also we are not creating parallel threads for mapping.

Reducer Object ⇒ 1 (There is no requirement for additional reducers and no extra threads created).

(ii)
Call to map() ⇒ 1.
(As it is a single line of text, map() reads input line-by-line).

calls to reduce() ⇒ 7.
map() generates following <key, value> pairs as intermediate op after shuffle & sort.
< analytics, [1] >
< Bigdata, [1] >
< course, [1] >
< exam, [1, 1] >
< IIT, [1] >
< midsem, [1] >
< patna, [1] >.

7 unique keys and reduce() is required for each one of them.

Chandrawanshi Mangesh Shivaji
180ICS16 CS555 @Chandrau

Page No.
Date / /

(b) → Given, Cluster.
1 namenode, (metadata) 5 datanodes (actual data)

Upload size = 128 GiB.
Replication factor = 3.
Size of total upload = 3 × 128 GiB = 384 GiB

Average amt. of data on each datanode
= $\left(\frac{384}{5}\right)$ = 76.8 GiB (Assuming equal sharing).

Also, 384 GiB = 384 × 1024 MB.
Block size = 64 MB.

Total Blocks = $\frac{384 \times 1024}{64}$ = 6144 blocks

Blocks on each datanode = $\left(\frac{6144}{5}\right)$

= 1228 block each + 4 blocks remaining on any 4 datanodes.

Final Storage:
1229 blocks on 4 datanode
1228 blocks on 1 datanode

= (1229 × 64) MB in 4 datanodes
+ (1228 × 64) MB in 1 datanode

= 76.8125 GB (on 4 datanodes)
76.75 GB (on 1 datanode).

Chandrawanshi Mangesh Shivaji 18015516
CS555 Mchandrau

Page No.
Date / /

Q.12) → The characteristic of BigData can be explained using the 3-Vs :-

1) Volume : Big Data indicates huge Volume of data that is being generated on daily-basis from various sources like social media platforms, business processes or
2) Velocity : The speed at which data is being created in real time. Clickstreams, High-frequency trading, every other data exchange is done within microseconds
3) Variety : It refers to the types of data generated & stored as big data. It can be structured, unstru-ctured & even semi-structured.

Benefits associated with Big-Data :-

1) Examining the real-time data allows us to make better decisions.
2) Based on analysis businesses can build customer centric solutions.
3) Big Data Technologies allow to scale & provide flexibility to store data before processing it.

Risks involved with Big Data :-

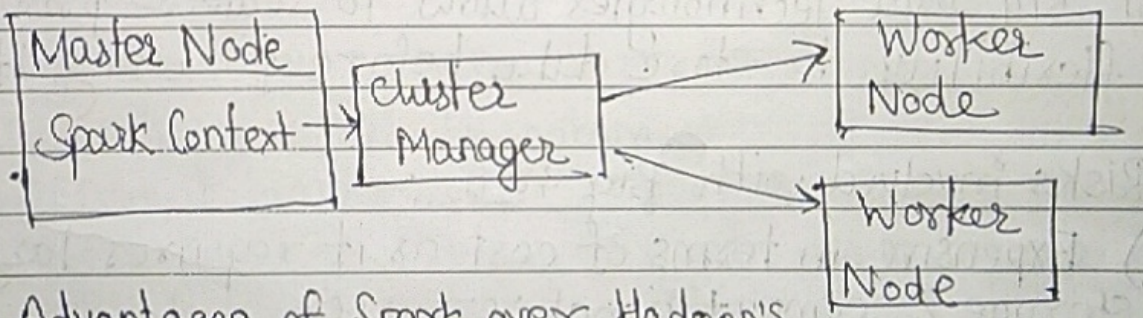
1) Expensive in terms of cost as it requires large storage & computational resources
2) Privacy issues w.r.t. data collected from public domains.
3) Due to complex nature, well trained professionals are required to maintain security of data along with integrity & usability

Q.13) → The different component of spark cluster are as :

a) Master Node :- It contains the driver program which runs the application. It creates a spark context to allow interacting with spark functionalities.

(b) Cluster Manager :- Spark Context with Cluster manager splits the jobs into multiple tasks & then these tasks are sent to various nodes to be executed. Cluster Manager basically allocates these nodes for the spark context.

(c) Worker Nodes : The Nodes which basically executes the task on the partitional RDDs in the nodes itself & then return the result back to spark context



Advantages of Spark over Hadoop's Map Reduce :-

1) In Map-Reduce, data was maintained & then mapped and reduced to produce results. Spark allows to use in-memory processing (No time spent in moving data in and out of disk). Can be 100 times faster.
2) Stream Processing (continuous IP & OP)
3) Presence of RDDs (caching, distributed processing)
4) Lazy Evaluation (done only when required).

Name: Chandrawanshi Mangesh Shivaji.
Roll No.: 1801CS16.
CS555 Big Data Chandrawanshi.

Page No.
Date / /

Q.14) → Components of Hadoop Distributed File System (HDFS) :-

HDFS is the primary or major component of Hadoop Ecosystem and is responsible for storing large data-sets of structured and unstructured data across various nodes and thereby maintaining the metadata in the form of log files.
It consists of two core components i.e.

1. Name Node :- It is the primary node which contains metadata (data about data) requiring comparatively fewer resources than the data nodes.

2. Data Node :- It stores the actual data. These data nodes are comodity hardware in the distributed environment. It helps in making Hadoop cost effective.

HDFS maintains all the co-ordination between the clusters and hardware, thus working at the heart of the system.

Q.15)

Rack Awareness : The name node has the features of finding the closest data node for faster performance for that name node holds the ids of all Racks present in the Hadoop cluster This concept of choosing the closest data node for serving a purpose is Rack Awareness.

Considering the replication factor is 8, the Rack Awareness Algorithm says that the first replica of a block will be stored on a local rack and the next two on a remote rack, but on a differ -ent datanode within that remote rack. If we have more replicas, they will be placed on random datanodes provided not more than 2 replicas reside on the same rack if possible.

Advantages :-

(I) To improve network performance
Switch is used to communicate betⁿ nodes on different racks. Rack awareness helps to reduce write traffic. Greater Read performance due to use of bandwidth of multiple racks.

(II) To prevent loss of data
As no replica is stored on similar racks, tolera nce against switch failure and power failure is increased.

Chandrawanshi Mangesh Shivaji 1801cs16.
CS555 Mchandrav

Page No.
Date / /

Q.16) ⇒ Main components of YARN architecture.

Client : Submits Map-Reduce Jobs.

Resource Manager : Ultimate authority controlling allocation of resources. Two main components:

* Scheduler :- Allocates resources to various apps running in cluster ; acc. to constraints such as queue capacities and user limits. Shedule is made to perform the functionality.

* Application Manager :- It is responsible for accep-ting the application and negotiating the first container from the resource manager. Restarts application manager container in case of failure.

Node Manager : It is the per machine slave which is responsible for application launch & reporting usage of resources to Resource Manager.

Application Master : Framework specific entity to execute and monitor component tasks

Containers :- Collection of Physical Resources such as RAM, CPU cores & disk on a single node. Invoked by Container Launch Context (CLC).

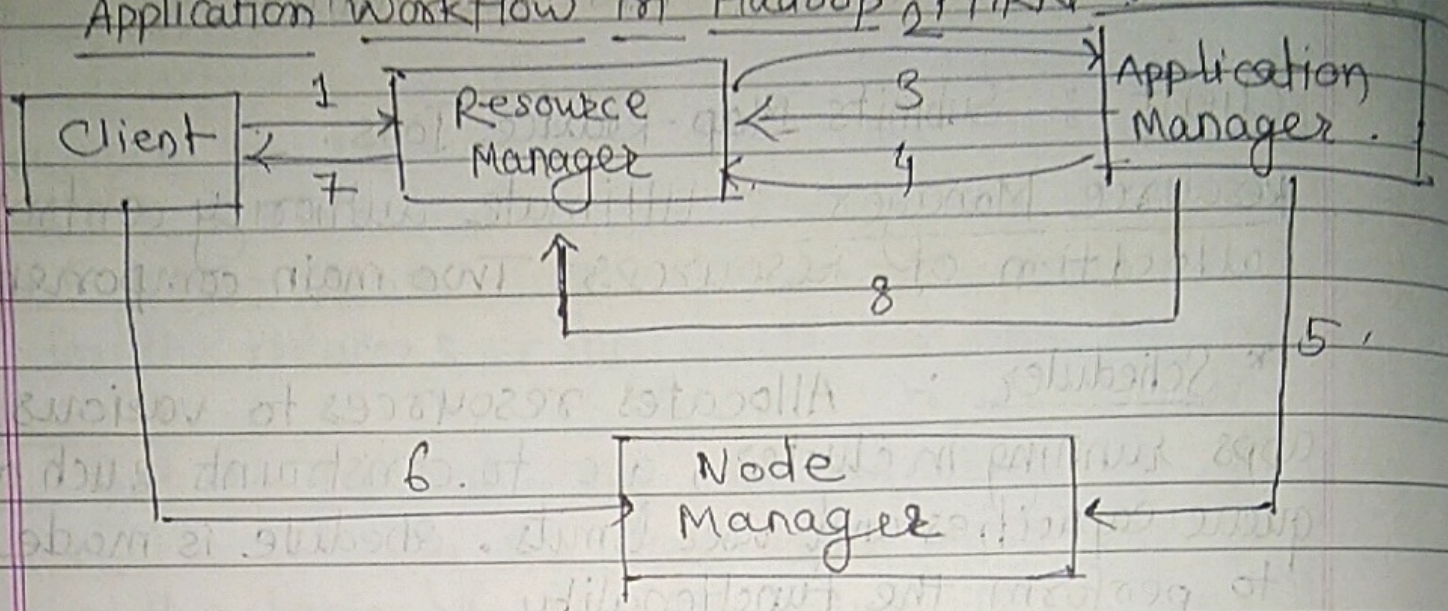
Chandrawanshi Mangesh Shivaji 1801CS16

CS555 Mchandrav.

Page No.

Date / /

Application Workflow in Hadoop 2 YARN :



YARN is the architectural center of Hadoop that allows multiple data processing engines such as interactive SQL, real time streaming, data science and batch processing to handle data stored in a single platform unlocking an entirely new approach to analytics. It improves the Hadoop framework in following ways.

Multitenancy :- Multiple access engines

Cluster utilization, Scalability, Compatibility.

Q.17) ⇒ Apache Kafka include five core APIs :

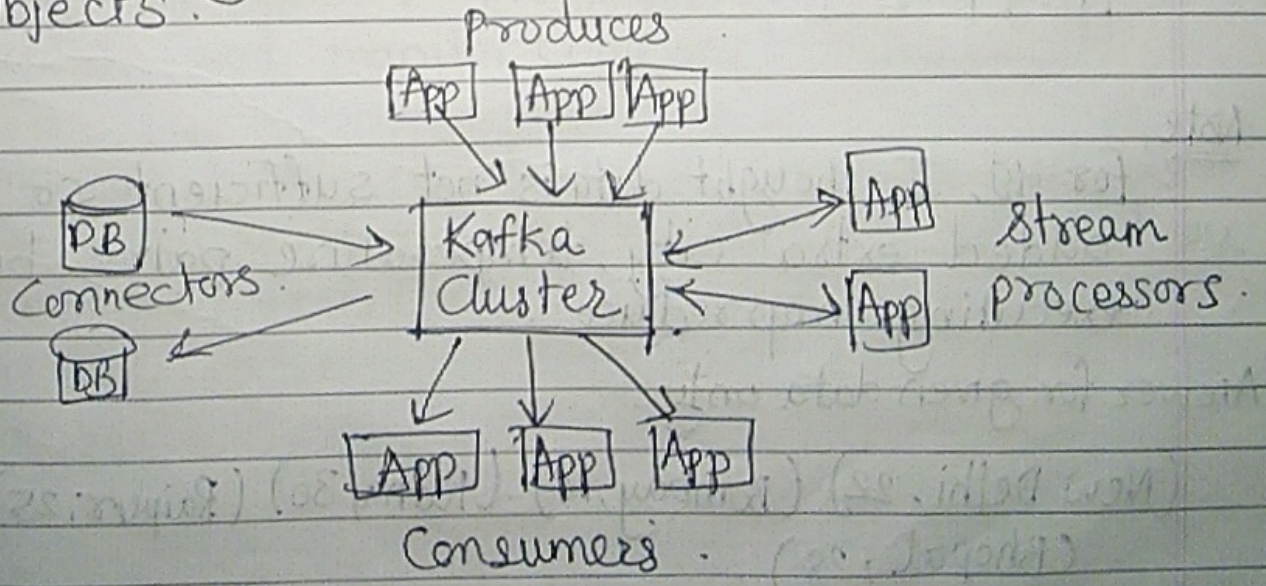
* Producer API : The producer API allows applications to send streams of data to topics in kafka cluster

* Consumer API : The consumer API allows application to read streams of data from topics in the kafka cluster.

* Streams API : The streams API allows transforming streams of data from input topics to output topics.

* Connect API : The connect API allows implementing connectors that continually pull from some source system or application into kafka or push from kafka into some sink system or application

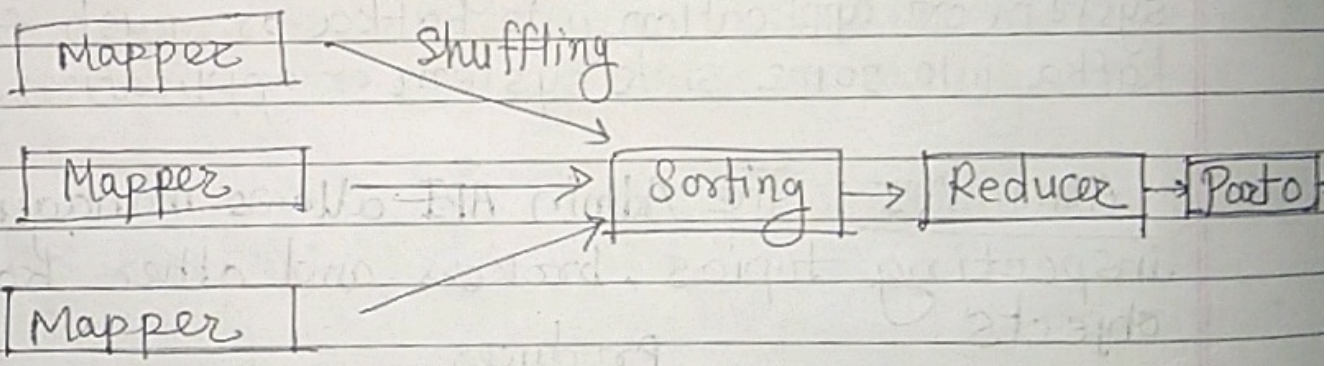
* Admin API : The Admin API allows managing and inspecting topics, brokers and other kafka objects.



Q.18) ⇒

Shuffle :- Shuffling helps to carry data from mapper to required Reducer with the help of HTTP. the framework call for application partition of output in all mappers. This process is necessary for the reducers or they would not have any input.

Sort :- In this phase, the output of the mapper that is actually the key-value pairs will be sorted on the basis of its key-value. It is done to easily distinguish when a new reduce task should start. When a new key in the sorted input data is different from the previous, then a new reducer task starts.



Note

For 19, I thought data is not sufficient so I added extra city, temperature pairs before executing mapreduce.

Answer for given data only.

(New Delhi, 22) (Bombay, 16) (Patna, 30) (Raipur, 25) (Bhopal, 28).

Q.19) ⇒

Assume you have 5 files and each file consists of two key/value pairs as in two columns in each file - a city name. ____ }

It is important to notice that each file may consist of the data for same city multiple times. Now, out of this data, we need to calculate the max temperature for each city accross these 5 files. The Mapreduce framework will divide it into 5 map tasks and each map task will perform data functions on one of the five files and returns max temperature for each city.

Intermediate results from mapper for each of the files.

File 1	(New Delhi, 22) (Bombay, 15) (Patna, 30) (Raipur, 25) (Bhopal, 28).
File 2	(New Delhi, 25) (Bombay, 16) (Patna, 32) (Raipur, 12) (Bhopal, 15)
File 3	(New Delhi, 23) (Bombay, 23) (Patna, 24) (Raipur, 34) (Bhopal, 18)
File 4	(New Delhi, 31) (Bombay, 28) (Patna, 26) (Raipur, 20) (Bhopal, 24)
File 5	(New Delhi, 24) (Bombay, 20) (Patna, 31) (Raipur, 18) (Bhopal, 26).

These are then passed to the reducer job, where the input from all files is combined to output a single value.

Final result (New Delhi, 31) (Bombay, 28) (Patna, 32) (Raipur, 34) (Bhopal, 28)

These calculations are performed instantly & extremly efficiently on large datasets

Chandrawanshi Mangesh Shivaji
CS555 1801CS16 @chandrau

Page No.
Date / /

Q.20)

(i) Fetch list of questions that have atleast 10 answers.

⇒

```
map (key, value) {
    for each record in value :
        if record ∈ Ans :
            emit (record[2], 1)        (∵ 3rd column is QuesID)
}

combine (QuesID, value) {
    for each v in value :
        sum += v
    emit (QuesID, sum)
}

reduce (QuesID, value) {
    for each v in value :
        sum += v
    if sum >= 10 :
        emit (QuesID, sum)
}
```

(ii) Fetch the date on which first answer was posted.

⇒

```
map (key, value) {

  for each record in value:
      if record ∈ Ans:
          emit (1, record[4])   (∵ 5th column is date)
}

Combine (key, value)
{
  minDate = min (records in value)
  emit (key, minDate)
}

Reduce (key, value)
{
  minDate = min (records in value)
  emit (key, minDate)
}.
```

Q.21) Resilient Distributed Datasets :- (RDDs)

It is a fundamental data structure of spark. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDD can contain any type of Python, Java, or Scala objects, including user defined classes. Formally an RDD is read-only, partitioned collection of records.

Properties

① Immutable, partitioned collection of records
② Built through coarse grained transformations (map, join,)
③ Can be cached for efficient reuse.

Transformations and Actions are RDD operations :-

1) Transformations on RDD is performed to build RDDs through deterministic operations on other RDDs.
2) These include map, join, filter
3) These are lazy in nature. (Only executed on action call)

Action :

1) Action on RDDs are performed to return value or export data.
2) It includes count, collect, save.
3) It triggers execution

* Spark Transformations : map, filter, flatMap, mapPartitions, mapPartitionsWithIndex, sample, ∪, ∩, distinct, pipe

* Spark Actions :- reduce, collect, count, first take, saveAsTextFile, countByKey, foreach

Chandrawanshi Mangesh Shivaji
1801CS16 CS555

Page No.
Date / /

Q.22) ⇒ Given,
Total Number of Mappers = a.
Total Number of Reducers = b.

A mapreduce job with a mappers and b reducers involves upto a * b distinct copy operations. since each mapper may have intermediate output going to every reducer.

There will b output files generated after running the program.

As, there are c number of unique words, Each of the b output files will contain exactly $\frac{c}{b}$ <key, value> pairs. (total pairs $= \frac{c}{b} \times b = c$)

Eg. In a word count example, a key, value pair is obtained from every word found. If we had 100 words, then 100 key, value pairs will be generated from the mappers to the reduces.

Q.23) →

★ Pseudo-Code (Algorithm using the Map-Reduce)

map (key, record):

(I) output [record (SUPPLIER), record (PRICE)]

(II) reduce (SUPPLIER, List of PRICE):

(III) emit [SUPPLIER, AVG (PRICE)].

★ SQL Query : SELECT AVG(Price)
FROM DATA
GROUP BY SUPPLIER,

Consider following example table.

ProductID	Supplier	Price
P1	S1	20
P2	S2	30
P1	S3	25
P3	S1	25
P4	S1	40
P2	S3	25
P4	S2	50.

↓ map.

(I)
S1, 20
S2, 30
S3, 25
S1, 25
S1, 40
S3, 25
S2, 50

Shuffle →

(II)
S1 (20, 25, 40)
S2 (30, 50)
S3 (25, 25).

Chandrawanshi Mangesh Shivaji CS555
1801CS16 @Bhandrau.

Page No.
Date / /

(III)

reduce →

S1	28.33
S2	40
S3	25

```
Mapper (key, value) {
    values = value.split(".")
    supplier = values[1]
    price = values[2]
    output (supplier, price)
}
```

0	1	2
ProductID	Supplier	Price

"key" corresponds to supplier.

```
Reducer (key, values) {
    sum = 0
    num = 0
    for value in values
        sum += value
        num += 1
    avg = sum/num
    output (key, avg)
}
```

Q.24) ⇒

(a) K-means Algorithm using Hadoop's MapReduce.

Pseudo Code :-

```
Mapper < key, value >
{       setup(context)
        c = getCentroids(context)
        map(key, value) {
            for each point p in value :
                min_dist, idx = min_{i=0}^{k-1} (dist(c[i], P))
                emit(c[i], P)
        }
}.

Reducer < centroid, pointslist >.
{       reduce(centroid, pointslist)
        {   newCenter = 0
            for P in list
                newCenter += P
            newCenter /= len(list)
            emit(centroid, newCenter)
        }.
}

main()
{
    centroIds = initRandomcentroid(dataset, K)
    stop = false
    count = 0
```

```
setContext(centroids):
while(!stop)
{
    job = prepareJob(context)
    job.waitforCompletion(True)
    count+=1
    stop = (noChange centroids(Context)
            || count >=10)
}
centroids = getCentroids(context)
print(centroids)
}.
```

(b). K-means for given dataset.

Choosing initial centroids as : 8, 31, 40.

Map-phase output : Euclidean Distance.

(8,8), (8,14), (8,3), (8,12), (31,21), (8,3), (8,10)
(31,30), (31,31), (40,48), (31,31), (8,6), (31,31),
(40,44), (31,32), (8,12), (40,44), (40,44), (31,32)
(8,7), (40,37), (31,25), (40,36), (8,13), (8,14),
(31,33), (31,31), (8,1), (8,4), (31,31).

Reducer phase :

For 8 → 8, 14, 12, 3, 3, 10, 6, 12, 7, 13, 14, 1, 4.

new centroid = $\frac{\Sigma x_i}{n}$ = 8.23 ≈ 8.

Chandrawanshi Mangesh Shivaji
CS555 1801CS16 @Mkandraw

Page No.
Date / /

For 31 → 21, 30, 31, 31, 31, 32, 32, 25, 33, 31, 31

new centroid = $\frac{\Sigma x_i}{n}$ = 29.81 ≈ 30

For 40 → 48, 44, 44, 44, 37, 36.

new centroid = $\frac{\Sigma x_i}{n}$ = 42.16 ≈ 42

Final Output.

Sr.No	Clustex Center	Points	No. of Points
1.	8	8,14,3,12,3,10,6,12, 7,13,14,1,4	13
2.	30	31,30,31,31,31,32, 32,25,33,31,31	11
3.	42	48,44,44,44,37,36	6.