HADOOP Course

Sunday, April 24, 2022 5:27 PM

Big Data :
Data - just processed information
How much data is a big data ?
-> 10kb /10mb/10gb/10tb

People say there are 3v's some say 10v's of big data which help us in determining whether a data is big data or not ?
But there are only 2 factors which can help us determine whether the data is big or not :

1. Volume/Quantity
2. Processing Speed

BIG DATA FORMAL DEFINITION :

What is Bigdata?
==========
Some says when dataset is large (or) volume is huge then data is termed as "BIGDATA".

Some says when we are not able to store the data on laptop then it is termed as "BIGDATA"

We get many such definitions regarding "BIGDATA"

FORMAL DEFINITION GIVEN BY IBM
==========

Any data which is characterized by 3v's is termed as "BIGDATA"

They are
1)volume
2)variety
3)velocity

1)Volume
======
Volume of data should be Large, It should be some Terabytes or petabytes
A single System(Machine) is incapable of handling it
Example
=======
Facebook users upload more than 900 million photos a day. Huge volume of data, traditional system incapable of handling them.

2) Variety
=======
Data can be of any type
a)Structured (RDBMS Databases, Oracle, Mysql)
b)Semi structured(Csv, Xml, Json)
c)un-structured(Audio, Video, Image, Log files)

it is not like traditional "DBMS" where data we get in a structured manner can be of any type as shown above.

3)Velocity:
=========
Speed at which data is coming is termed to be "Velocity".
In simple words the speed at which "Ingesting data", "processing data" and "retriving data"(response) is termed as "Velocity"

Example
==========
Remember our Facebook example? 250 billion images may seem like a lot. But if you want your mind blown,
consider this: Facebook users upload more than 900 million photos a day.
A day. So that 250 billion number from last year will seem like a drop in the bucket in a few months.

Velocity is the measure of how fast the data is coming in.
Facebook has to handle a tsunami of photographs every day.
It has to ingest it all, process it, file it, and somehow, later, be able to retrieve it.

According to Ibm Formal definition , the Dataset with above 3 characteristics is termed as "Bigdata."

There might be 4v's,5v's so-on.....

4v's are somewhat relevant and lets talk about 4th v:
==========
3v's are same as above discussed and 4th 'V' is "Veracity"

Veracity:
==========
Quality of the data that is being analyzed is termed as "Veracity".

Low veracity:
==========
We can find meaningless, poor quality data like
a)we can find lot of null values
b)age might be in negative

Such Low veracity data doesn't contribute any meaning-full insight.

High Veracity:
==========
On the other hand High veracity data contribute meaning-full insights

Example:
=======
a)High veracity data set would be "" data from a medical experiment or trial "", which gives us a meaning-full insight of an "Experiment"

What's big data, how to decide?
-> Lets say we have 100mb data , small file, then we wrote a small program to process this data, and it took 5mins to complete this task.
Volume - 100mb; time - 5 mins
- is this big data? NO
Since business has no problem with 5mins and the volume is also small that you have no problem storing it.

But now volume - 1000mb; time - 50mins
- no this is also not a big data, since both factors are good for us. (time depends on business req.)

Now volume - 10gb; time - 600mins/10hrs
- no this is not a big data

Now vol- 1tb; time - 1000 hours/40 days
- yes this is big data coz business does not agree with the processing time.

So, big data is a problem not a tech, that the data is so big that we have problem with the storage and processing time.
There are quite a few solution for this problem available : Hadoop, Spark, Flink, Hydra, Disco, MPP.

Ecosystems :
Initial version of Hadoop only had two things - 1.HDFS 2.MapReduce (developed from googles two paper GFS, google file system; for dist. storage and MapReduce; for dist. processing)

A. **HDFS** - Allows us to store data in a distributed way.
An ability to store data into multiple nodes(machines) of a cluster by dividing data into pieces(blocks).
Example - if I have a cluster of 4 nodes (machines are called nodes)

ClusterA - [] [] [] []
N1 n2 n3 n4
1tb 1tb 1tb 1tb = total memory of cluster : 4tb; 64 gb ram; 32 core processors

In Hadoop 2, block size is by default 128mb, so if we want to store a 1gb file in our hadoop clusterA it would mean that our file would be divided into blocks of 128mb.
So, 1Gb = 1024mb -> in size of 128 mb; Number of blocks => 1024/128 blocks = 8 blocks (B0, B1, B2,……,B7)
How would these blocks be stored in our cluster ? --> B0 may be stored in N1, B1 in N3 and so on

N1	N2	N3	N4
B0, B3	B2,B5	B1,B7	B4,B6

-> This means that HDFS will take care of how to store blocks in nodes in a cluster

HDFS is like a distributed hard disk , where we don't care how data is stored.

B. **MapReduce** - A framework which can help you to process the data on the nodes(in a cluster).
Typically you write a java program/python code - convert to jar file - execute jar file - it will start processing the data inside the cluster on nodes.
It does two things - processes the data, and it can also help you to negotiate the resources in Hadoop 1.x, but since writing a java program formapreduce is a nightmare so in hadoop 2.x we use various ecosystem components to make our task easier.

so ppl starting making ecosystem component(hive, pig[], scoop[data transfer], oozia[scheduler/seduce a job], airflow, etc,30+)

HADOOP ARCHITECTURE (HDFS) :

Hadoop 1.x -

In a real time production cluster, we'll have few masters and many slaves. Below is a 8 node cluster :

Master	Master	Master
Slave Slave	Slave Slave Slave	Slave Slave Slave

Masters should be powerful while slave can be a generic machine.
In all the slave machines hadoop would be installed, with hadoop some supporting services will also come called "daemons".
Daemons must be up and running, so that hadoop should be functional, Daemons are :

1. Name Node {In Master Cluster}
2. Secondary Name Node {In Master Cluster}
3. Job Tracker (only in hadoop 1.x; has been replaced in later version) {In Master Cluster}

4. Task Tracker (only in hadoop 1.x; has been replaced in later version) {In Slave Cluster}
5. Data Node {In Slave Cluster}

Master will contain - Name Node, Secondary Name Node, Job Tracker
Slave - Task Tracker, Data Node

All the slave nodes together form a HDFS -> which will help us in storing the data by dividing them into blocks of max block size of 128mb(default size-hadoop1.x-64mb, 2.x+3.x -128mb)

STORING THE DATA

-> If we have 1gb file, data.txt and want to store them in Hadoop, one way is to write a command : hadoop fs -put data.txt \tmp\data\1 (hadoop fs -put 'file name' 'path')
data.txt -> 1024/128mb -> 8 blocks -> b0,b1,b2,...,b7 -> these blocks would be saved to different slave nodes randomly.

-> Since the file is divided into blocks and saved to different slave nodes what is the use of the path that we gave in input command ? - this path is only a logical path and not an actual address where the file would be stored

- this logical path would be maintained by Name Node(NN)

-This Name Node contains a register/dictionary which would contain the information which block is stored in which slave node (e.g. b0:n1,b1:n3,b2:n4,...) , steps :
1. our put command will talk to the name node about where to keep the file ,
2. so the name node checks/does lookup on the register on the status of the cluster(slave nodes) and tell which slave nodes to put the data in,
3. now put command will break the files into blocks and place them in the slave nodes communicated by name node.

-> where the blocks will be maintained and stored physically in slave nodes by Data Node(DN). Now Data Node transmits the info about the blocks position to Name Node and Name Node updates its register on the basis of the info provided.

Over the time hardware failure may start in the datanode(since datanode are just generic machines that are not as good/expensive as the master nodes), and data may get lost so to tackle this issue Hadoop has a concept of data replication/duplicaton , i.e., duplicate the data in some other location as well, and with default hadoop comes with a duplication factor of 3.
Data duplication happens immediately as soon as the data comes inside the slave node, it starts replicating.

6
Data duplication/replication :

Slave0	Slave1	Slave2	Slave3	Slave4
B0	B1	B2	B3	B4
B22,b42	B21,b32	B31,b41	B01, b122	B02, b11
Data Node 1	DN2	DN3	DN4	DN5

- SLAVE NODES
- Data Packet stored primarily

- Data backup via data replication/duplication

- this concept takes care of no data loss in hadoop.

-> These slave nodes or data nodes(DN) constantly ping Name Node(NN), giving the info that I'm alive to let the NN in loop, this is known as Heart Beat by default its in every 3 seconds but as soon as dn stops this update to nn, nn will wait for some time and if still if there is no ping, it would assume that this dn (lets say dn1) is dead/down and declare it as dead/down.
Now nn know that dn1 is down, not it will check where is the copy of the data stored in dn1 is kept (there would be two replicas more), lets say those are dn4 and dn5.
Since, we now have only 2 replicas of the data that's stored in dn1(data packet - b0) but our duplication factor = 3 (i.e we need to have 3 replicas of a particular data), NN will now order either of dn4, dn5 to make one more replica in some other dn, lets say in dn2.
What if the dn that went down , dn1, now comes online ? -> now we would have 4 replicas of the same data, so NN will get this info as dn1 would start pinging the NN again, but NN would not consider the data in dn1, and the data blocks inside dn1 will be invalidated, so the new dn, dn2 dn4 dn5, would be considered.

-> This means that HDFS will take care of how to store blocks in nodes in a cluster

- SLAVE NODES
- Data Packet stored primarily

- Data backup via data replication/duplication

RETRIEVEING THE DATA

We will use this command, in order to fetch the data : hadoop fs -get \tmp\data\1 (hadoop fs -get 'path')

-> now since the data is divided and stored as blocks inside the hdfs, this command would first go to the NN and it would get the info of where the blocks have been stored in which slave nodes via the register and return it to our "get" client (cmmd), keep in mind NN will not participate in data transfer, it will just pass the info and then the get client will go to each Slave/DN to fetch the blocks.

PROCESSING THE DATA (DATA PROCESSING)

After storing data we would want to process data, which can be done via by some program like a Spark/Scala code, Hive query or a Map Reduce Job.

Let's assume we wrote a MapReduce Job in java to process the data.

M-R Job (.java) -> bundled into .jar (lets say WC.jar) -> this .jar file contains : Mapper Program, Reducer Program, Driver Program (Main Java program)

MyMapper	MyReducer	MyDriver

Slave0	Slave1	Slave2	Slave3	Slave4
B0	B1	B2	B3	B4
B22,b42	B21,b32	B31,b41	B01, b122	B02, b11

cmd :
first stored the data using -> hadoop fs -put "file name" "path"

Then, use this command -> hadoop jar WC.jar MyDriver \temp\data\1 \tmp\data\count (hadoop jar 'jar file name' 'driver name' 'input path' 'output path')
{output path - where you want the data to be stored}

Q. How this above command/job would be executed ?
SOL.

- This above command will submit the information regarding the job in Job Tracker(JT)
- Then Job Tracker will got to NN and seek the info about where the data is stored, the input path, upon which the submitted jo b is to be executed
- Now the NN will go to its register and get the info about where and in which Data Node(DN) the data is stored and pass this i nfo to Job Tracker (JT)
- The JT will send the .jar file to all the DN where the related data is stored.

  Difference between Big Data Env and Normal Data Processing - here we are moving the program (.jar file) to the place where data is stored but in a normal env. We would be sending our data to the program itself (eg - uploading a pic to remove bg from the pic, logic present in server and there you upload it ad process the image).
  i.e. Hadoop tries its level best to not to move data as much as possible.

- In each Slave Node we would have a Data Node(DN) which will be storing the data, and also a Task Tracker(TT) which will initi ate the job and execute the .jar file that we sent via JT
- Now our jar file contains 3 programs - Mapper, Reducer and Driver, driver is just a helper class, so assuming Mapper is executed first
- Mapper would be executed in all the Slave Nodes, or the Data Nodes and the output of it all will be combined and stored in an other location/Machine.
- From where the Reducer program would be executed on that combined data.
- Every Task Tracker(TT) will have pre defined slots , Map Slots and Reducer Slots, TT will have the progress of how much perce ntage of the job is done in the Hadoop UI.

Speculative Execution : If some job gets stuck at 99%, the JT will kill the TT which got stuck and will provide that task to some other TT, and the new TT would be considered.

-> In Hadoop we're doing horizontal scaling, i.e. adding more number of nodes/ clusters and not adding more machines just adding more space and all , but we can do vertical scaling from time to time as well.

2nd Video :

Name Node VS Secondary Name Node

In Hadoop it is advisable to store few large files as much as possible instead of many small files.
Why? -> because more the number of files more the no. of entries in the register of the NN, and it would become unnecessarily heavy e.g. one 1 gb file will have lower no. if entries in the NN register than 1024 files of 1mb, so therefore its not advisable to store small files in Hadoop.

So, because register is important for our hadoop NN should have a large RAM(~256 gb).

This register is nothing but a file containing meta data, and there are two types of registers in the NN :

- fs Image (Read Only in NN)
- Edit Log

Every time the NN is started/refreshed it would create these two files, and will load the previous data of these two files in the NN RAM and then both the files are empty.

Edit Log - It would contain the log or information of each and every command executed in hadoop (all the cmd related to -fs, i.e. data/file store remove, etc; not every cmd basically)
With time this log would grow really big.

Now since edit log is very big and whenever the NN is refreshed it would take the previous data in both fs Image and Edit Log and do some processing to calculate lets say the number of files present in the hadoop cluster., so it would take a long time to do this calculation, so therefore we have Secondary Name Node (SNN).

So, we have two files fs Image and Edit Log :

- On the first run both the files are empty

Fs Image	Edit Log
0	0

- Then as the process starts, edit log will start saving entries in it, lets say the number of files added, deleted, etc from the hadoop cluster

Fs Image	Edit Log
0	10 added (+)

- Then at some point of time, Check Point in Hadoop(default interval is 1 hr/ or 1 million transactions), the SNN will fetch the fs Image and Edit Log file from the NN and do some processing/calculation on its data.
  - The fs Image will contain the data of how many files were present and edit log will be used to see how many files were added or deleted from the last CheckPoint.

Fs Image	Edit Log
10	0

  Meanwhile there would be a new Edit Log created in the NN and the ongoing transaction would be noted down there.

- Now this information of fs Image will be sent to NN

- Lets say now the NN before the second check point looks like this :

Fs Image	Edit Log
10	2 added(+), 3 removed(-)

- Now, this data will be copied to SNN, meanwhlie a new edit log would be started in NN, now the SNN would do the cacl of how many files are present in current hadoop cluster using edit log's info

Fs Image	Edit Log
10	2 added(+), 3 removed(-)

Fs Image = 10+2-3 =9 files

- So, this fs Image gets copied to our NN and it looks like this :

Fs Image	Edit Log
9	……

so this way SNN does our task easier and now the NN would not have to do the long calculation every time its restarted since SNN does our task in between.

-> There's one more use of SNN, in Hadoop 1.x, that if the NN fails/ gets offline we would have the recent fs Image of the last check point stored in our SNN.

You can change the default CheckPoint interval using this configuration - dfs.namenode.checkpoint.peroid / dfs.namenode.checkpoint.transaction
this config is present in hdfsSite.xml (there are many site files containing many configuration)

Limitation of Hadoop 1.x :

- NN is a single point of failure, i.e. the data that NN contains is stored only in NN and if NN fails that data is also gone.
- Only able to accept Map Reduce Job framework, even if you write Hive Query you had to think it in terms of mapper phase and reducer phase, not able to run Spark.
- Map Reduce was a resource negotiator. M-R is a framework to process the data and shouldn't be used as a resource negotiator i.e. if I had a clauster and wanted to submit a hadoop job Then where the job should be schedul8ed, who should schedule it etc were all monitored by M-R, it would talk to Job Tracker and so on, but M-R was the main thing here.