Page:
Date: 18/12/21

Big Data Analytics / Technology (BDT)

* Big Data Technologies and Hadoop
- Introduction to Hadoop
- What is Big Data ?
  - Big data is any data which cannot be processed using traditional / single node approach of problem solving. (single node / single machine)
  - Big data is amout of data which forces you to
  - Beyond the hype :
    - Data Collection - collect data
    - Data Aggregation - combine
    - Data insights - helps to do something

Imp. * The 5 Vs :

1) Volume - How big is my data
2) Velocity - How fast is my data
3) Variety - Is there a lot of variety
4) Veracity - How accurate is my data
5) Value - How valueable is my data

* Big Data Sources :
  - Big data is very time sensitive -
  - stock level analysis
  - System can process 46 - billion records - past data

* Industries using Big Data ..
  - Telcom domain
  - Defence
  - security
  - stock market
  - sports & education

Excel → postgres → hive

Hadoop made data storage very cheap.

For storing - NAS - Network Application Server / NFS - N/W file server.

* Big Data challenges :
  - storage compute : In terms of CPU & RAM
  - Data sources : where to get the data
  - collect data from stock market / Gathering
  - cost associated with collecting data

* Three pillars of Big Data
  1. Data science - Algorithms
  2. Hadoop
  3. Cloud - AWS, Asure

Present - Analyst - Figure out outcomes, Analising after completion

Future - Scientist - Predict the data, take decesion of forward

Analyst - Gives accurate data

Scientist - Gives propability (only prediction)

* Solutions to Big Data problems : Hadoop but not only one

GPFS, HDFS are counterpart of hadoop.

- HT-condor - Also makes big data easy

* Various Big Data technologies :

HDFS - Hadoop distributed file system

GPFS - General parallel file system

AWS S3 - counter part of HDFS

* Big Data Adoption :

Industry wide adoption is a common trend coming into the market.

Reasons for adoption.

1. Explosion is data generation & collection
2. ~~Explosion~~ Greater insights via - / Actions

a. Data discovery

b. Analytics discovery

c. TOOLS & technology discovery d. Infrastructure discovery.

HT- condor - scheduling mechanism

* Recommendations for Big Data Adoptions :
- Define strategy
- Discovery of existing assets
- Build analytics on top of larger data sets for richer values.

* Data storage strategies & techniques have changed to adopt to the 3 Vs of Big Data :
Data repository :
1. Databases
2. Data warehouses & data marts
3. Data Lakes - store into file & structure not fixed
   - Revolutionised the data sharing & reuse practices
   - Data was priviously cured & stored.
   - With Data lakes, the data is stored & then cured on demand

Data curation - Management of data. It is related to metadata than data.
There is s/w created for data curation

* Data Life cycle :
1. Creation
   a. Data entry b. External Aquisition
   c. Capture from devices
2. Storage : a. Security & Backup Recovery
3. Usage : View, process, modify & share
4. Archival : Archiere & Available for use
5. Destruction : Purge

Page :
Date : 13/12/21

* Big Data & Hadoop platform
  Hadoop distros & versions - Apache Hadoop

Imp
* HDFS Architecture :
  Hadoop distributed file system.

* Design Assumption & goals :

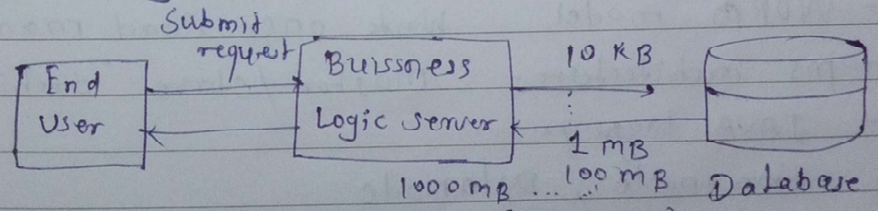
1) 1. Hardware Failure : In terms of node, network, disk, It take cares of hardware failure (HDFS)
   clustered o/w - It has multiple nodes which gives ability to run multiple nodes.

2) 2. streaming data Access : All files stored into HDD which is fast and this data is available by streme over the network & it get's superfast. Data is flow node by node. data set.

3) 3. Large data sets : If HDFS has very large files/ then performance will be high. And it is (hadoop) basically designed for large data set i.e. Think hadoop like elephant.

4) Simple coherency model : Two threads Running things parallelly. no people write in same location.

5) Data locality - How data is copied & where it will placed. Moving data from one location to application



Data Locality (General)

Vertical Scaling - Scale-up
Horizontal scaling - Scale-out

6) Portability across Heterogeneous H/W & S/W.
   HDFS is portable - it will run anywhere.

Camlin

HDFS has - one NN, multiple secondary NN

Node Types : Nothing But processes / Machine

1) NameNode : manages the namespace for HDFS i.e. name of the file which is stored in name node

NameNode (NN)

2) Secondary NameNode : Acts as a backup for Active Namenode - If first NN down this will helpfull

3) JournalNode - Helps with the migration of edit logs. If NameNode changes in namespace that will comes here.

4) DataNode - Holds data blocks - One dataNode running on a single machine.

* Data Dumping *

Block size = 256 MB

file size = 1024 MB

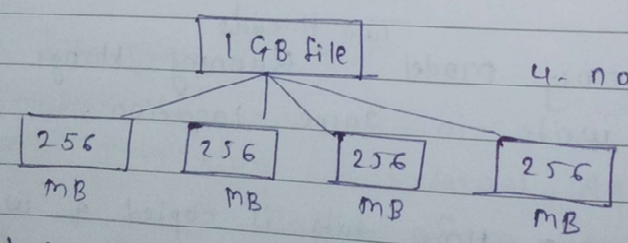
No. of block = file size / block size

= 1024 / 256 MB = 4 nodes

If no. of nodes / blocks increses time will be decreses.

i.e Block size = 128 No. of blocks = 8

So, time will be less for a one node.



* What is HDFS ? - HDFS Architecture

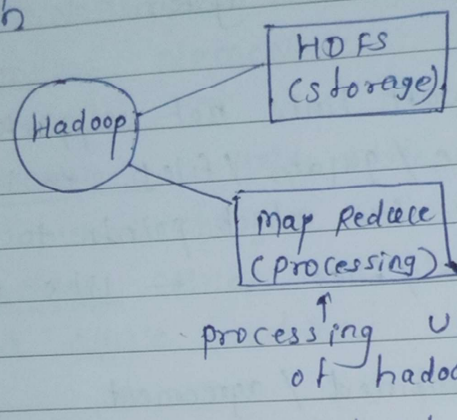
- WORM model : Write once Read many e.g CD.
- MS architecture - master / slave architecture
- JAVA language
- Namenode, DataNode
- Cannot edit file but can append

Replication :- If one or any node do

By defaut Replication factor : 3

Hadoop :

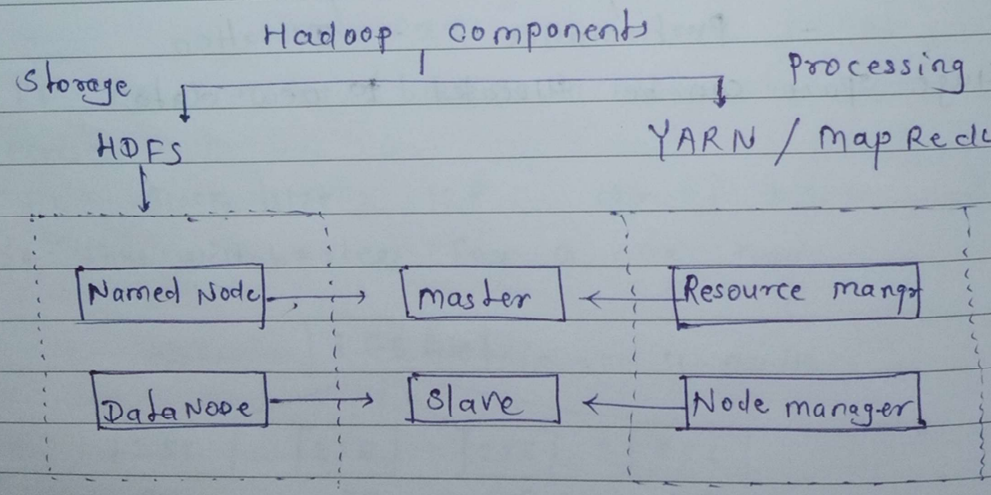
Hadoop is a framework that allows us to store & process large data sets in parallel & distributed fasion



→ Allows to dump any kind of data accross the cluster

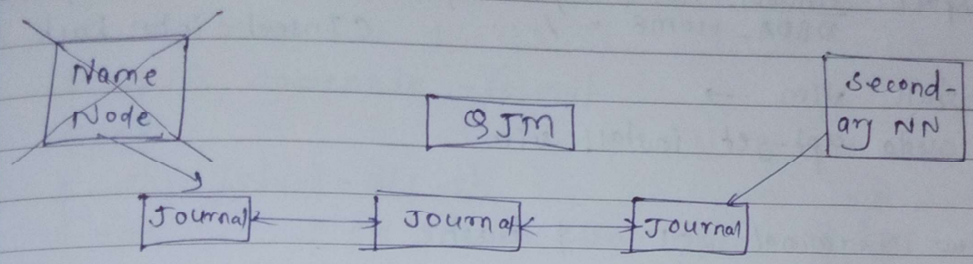
→ Allows parallel processing of the data stored in HDFS

HDFS : Hadoop distributed file system allows to store structured, unstructured or semistructured data.



... to complete command (Autocomplete)

QJM : Quorum Journal Manager (QJM)



- If one journal nodes are failed it will transfer request to another one
- This are very lightweight (Journal nodes)

→ Download hadoop - 3.3.1
download ssh →
open terminal - $ ssh-keygen ← press enter thrise
$ ssh-copy-id gayatri@localhost
before this install ssh -
$ sudo apt install openssh-server ~~open~~ enter
$ openssh-client -y
$ ssh-copy-id gayatri@localhost
$ ssh gayatri@localhost

On Terminal - $ cd
$ mkdir HADOOP_HOME / DBDA_HOME
$ cd DBDA_HOME/
$ cp ../Downloads/hadoop-3.3.1.tar.gz .
$ ls
$ tar xvzf hadoop-3.3.1.tar.gz
$ ls
$ cd hadoop-3.3.1/
$ pwd
copy o/p of pwd cmd & use it in export

$ htop - running processes

Date: 21/12/21

* Map Reduce : We can process big data with the help of map Reduce.
It is a way to execute job on cluster.

- Parallel Programming : for increse throughput.

1) scale-out Vs scale-up - incresing CPU, RAM, processing speed increses this is scale-up
Add more clusters / processors / nodes this is scale-out.

2) Resource Utilisation :
a. CPU utilisation ← may be idle
b. memory ← busy than CPU
c. Hard-disk IOPs ← Busiest but slowest

3) CPU Idle time :
It is complete task fastly & go back to idle state.
By hdfs we can reduce CPU idle time

4) Disk I/P O/P bottlenecks

5) Map-Reduce benefits :
a. Independant of resources
b. Near linear increase in throughput
c. Lesser context changing overhead.

- What is map-Reduce?
- Programming paradigm / Framework / concept
-

- Alternative to the MR framework :
a. HT-condor b. spark c. Hive d. Pig

- Why MapReduce :
- simple logic
- can easily handle huge amount of data.
- Parellel execution - Linear growth in scale-out

/ - Abosolute path

hdfs never overwrite any of the folder & try it will get exception

* Architecture : Different processes :-

start HDFS & YARN from console :-

Different processes -

* YARN - Yet Another Resource Negotiator

It is consider as a brain of hadoop ecosystem

Used to perform all processing activities by allocating resources & scheduling algorithms.

Two major components it has :

(i) Resource Manager - Receives the processing Req. - passes the parts of requests to corresponding Node-managers.

(ii) Nodemanagers - Installed on every DataNode - Responcible for execution of task on every DataNode

$ ls sbin/ ← Admin level commands for superuser

* Start YARN & see YARN User Interface over browser (localhost:8088/cluster)

$ ls -ltr etc/hadoop/mapred-site.xml

$ vi etc/hadoop/mapred-site.xml

- edit this file

$ sbin/start-yarn.sh ← start yarn

Browse - localhost:8088 ← YARN - UI

$ sbin/start-dfs.sh

$ jps

$ hdfs dfs -put LICENSE.txt

$ cd /assignments $ ls ← Create this folder into DBDA-HOME & paste jar & runner.sh file

$ cat runner.sh

$ hadoop jar $HADOOP_HOME/assignments.....

→ o/p → job.successfull true ← paste last two line of runner.sh file

hdfs df -ls /output

→ show 2 items - -SUCCESS & part-r-00000

netstat -pant | grep 9000 ←
ps afx | grep (process_id) ← To see running processes
kill -9 (proce_id) ← To kill process

Replication factor :— It is only have files not directory

$ cd.. ← Go to bin directory
$ cd bin/
$ ./hdfs dfs -ls / ← show no. of files & directory present at hdfs
$ ./hdfs dfs -ls /output ← show files present in output directory
$ ./hdfs dfs -cat /output/part-r-00000
↳ see the contents of part-r-00000 file which is into /output directory — which show wordcount

* MapReduce — It is about key-value pair
mapper Input ← processor
← It converts into key value pair

e.g. and → (and, 1) and - key
the → (the, 1) 1 - value
↓
sorted mapper output
(and, 1) ← sort the data
(the, 1)
↓
* Network * ← signify mapper & reducer running over network
↓
Reducer copy without combiner ← combine all data
↓
Reducer Input without .
(and → 1, 1, 1, ...) combine no. of 'ands'
(the → 1, 1, 1, ...) combine no. of 'the's'

F2 - for rename folder

↓

Reducer ← Creating sum of all this elements

↓

Reducer output ← No. of 'and' i.e. 10
(and, 10) ← No of 'the' i.e. 8
(the, 8)

In this MapReduce data size increses when when we

file → import → maven ← type
Select - existing maven project ← next
Browse - directory - select pom.xml

* maven - All dependant liabrary download from internet & place into system

Right click → pom.xml → run as → clean maven
— — → run as → maven install

<dependancy> ← Tell maven which dependant liabrary required
⋮
</dependancy>
Hadoop-common ← Use all jar which are downloadd

open Terminal → open folder → pom.xml
$ mvn clean install

Page :
Date : 28/12/21

HIVE : Hadoop Based data-warehouse.

Hive is a engine which converts SQL to Map-Reduce & give result.

* Alternatives to Hive :

1. Cloudera Impala
2. Apache Drill
3. Spark SQL
4. Apache phoenix
5. Presto from Facebook - (unstructured-data)

* What is Hive ?

- created by Facebook
- It is data-ware house package build on top of Hadoop
- Majorly used for data analytics
- Targeted for users comfortable with SQL
- It has HiveQL instead on SQL → HQL.
- Hive doesnt restrict to primary key constraint.

- It is used for managing & querying structured data
- Abstracts complexity of Hadoop - make simple big-data journey
- Known for batch processing :
- It provides metadata

* Hive Vs RDBMS :-

Schema on Read Vs schema on Write

sudo pm-suspend ← for sleep ubuntu

Page :
Date :

$ HADOOP_HOME/bin/hadoop fs -chmod g+w /tmp
$ cd $HIVE_HOME
$ HIVE_HOME/bin/schematool -dbType derby -initSchema
.... schemaTool completed

$ $HIVE_HOME
$ ./bin/hive ← Go to hive prompt
hive> show databases ;
exit; / ctrl+d ← for out to hive

* When we not give absolute path for destination / for put it will consider as - . (home directory)
/user/Hive ← Home dir for all hive components
$ hdfs dfs -ls /wer/hive ← It is home directory of all or warehouse
/user/hive/warehouse - The directory where warehouses located

* HiveQL Data types :- Primitive Data types
Boolean, TimeSTAMP, DATE
TINYINT - 1Y, SMALLINT - 1S, INT - 1
BIGINT - 1L, FLOAT → 1.0, Double - 1.0
DECIMAL - 1.0, STRING - 'a', "a", VARCHAR - 'a', "a"

Complex - Data types
ARRAY - array(1,2) — Same type
MAP - map('a', 1, 'b', 2) → key - primitive, value any type
STRUCT - Struct('a', 1, 1.0) → diff type
UNION - Create_union(1, 'a', 63)

* Hive Table :-
Managed Table - Managed by hive (data)
External Tables - Data managed by external entity.

Camlin

SERDE - Need to par...

1] Managed Tables : - Hive will manage the data
- Data will be in warehouse dir
- e.g. create table tname (col1 string);

$ ps afx | grep -i hive ← To check hive is start somewhere else.
cd $ $HIVE_HOME
$ ./bin/hive
hive > show databases ; ← First time it will take time
> create database cdac ;
> dfs -ls / ;
> dfs -ls /user/hive/warehouse ; ← show created db
> use cdac ;
> show tables ;
> create table simple_score_managed (col1 integer, col2 string);
> dfs -ls /user/hive/warehouse/cdac.db
→ show tables within this.
> show create table simple_score_managed ;
> ↳ show details of table / default values
> desc simple_score_managed ;

* Default Parameters :-
- ROW FORMAT SERDE - Serialiser / Deserialiser
- Input FORMAT - For insert
- Output FORMAT - For select
- Location - Unique location
- By default - Database is - Default
- TBLPROPERTIES - Keep timestamp when table last edited / modified.

- Drop will delete the meta as well as data.

Insert
insert into
values (1, 'dbda');

By default field seperator is - ^A (ctrl-A)

Page:
Date:

hive > dfs -ls /user/hive
> dfs -cat /user/
> select * from table_name;

* External Tables : User controls creation & deletion.
- location is specified at table create time.

e.g.
Create External Table my_ext_table (coln1 string)
location '/user/ .... ';

- Hive does not move it into warehouse dir.
- Drop table will delete only the metadata. Actual data will remain intact.

> create external table my_ext_table(col1 string) location
'user/gayatri/my_ext_dir';
> show create table my_ext_table;
- To insert data use files & put
- If want data from external / data coming from external sources use <u>managed table</u>.

> set hive.print.headers = true;
> set hive.cli;

Schema on Read :- Hive doesn't check data is present or not in table while Read.

Page :
Date : 29/12/21

Partitions and Buckets

- Placed data into correct location
- Hive organises data into partitions
- Data is sliced, so partitions will help fasten the query
- Tables or partitions can be further sub-devided into buckets.
- Partitioning & bucketing are ways for data management.
- Partition works on - folders
  Bucketing works on - files.

* Partitions - Data is partitioned upon columns.
- Data may be partitioned on multiple dimentions
- At filesystem, partitions are nested dirs.
- The columns over which we partition are called as partition columns, and they basically do not exists into the data, but exist onto the data-path location.

e.g.

1. Create table logs (ts BIGINT, line STRING)
   Partitioned by (dt string, country string);
   — ts :- timestamp dt : date

2. Load data local inpath ' ← create partition dir
   into table logs
   Partition (dt='2001-01-01', country='US');

3. show partition logs; — To view partitions.
4. Select ts, dt, line from logs
   where country='US';

Page :
Date :

* Bucketing : Imposes extra structure on table.
It is done on perticular column -
a. Enables efficient queries esp join
b. make sampling more efficient.
- Bucketing happens only on the columns presents / inside the file.

e.g.
Create table bucket_users (id INT, name STRING)
clustered by (id) into 4 Buckets;

Insert overwrite bucketed_users
select * from users;

* Storage Formats : Two type of formats :
a) Row Format b) File Format

* Default Delimiters :
Default row - ctrl+A

* Importing Data - 1. CTAS : Create table as select.
e.g. insert overwrite table table_name
select col1, col2 . . . from source_table;
2. Using static partition :
3. Using Dynamic partition insert

Parquet files are very compressed format, it takes very less storage.

ctrl + l

Page :
Date :

* Querying Data - Joins :
Table 1 - select * from sales ;
Table 2 - select * from things ;
> select sales. *, things. *
> from sales JOIN things ON (sales. id = things. id);

Outer Joins :- 1) Left Outer Join → Add Null - No comm
2) Right Outer join -
3) Full Outer join -

- Join will run MapReduce side because it is running on localmachine

* Views :-
Create View sales_View as . . .
- View are virtual tables defined by select.
- View cannot used to update

* UDF : What are UDF's
User defined functions -
1) Regular UDF - operates on a single row & produces single row as output.
2) UDAF - User Defined aggregate functions
operates on multiple rows & produces single row as output.
3) UDTF - User defined Table functions :
operates on a single row & produces multiple row as output.

UDF - use when hive - inbuild functions won't work

Page :
Date : 30/12/21

* HBase *

- NoSQL Database (Non-Relational)
- column oriented
- Distributed
- HBase is opensource implementation of Big table
- BigTable is sparse, distributed, persistant multi-dimensional sorted map
- Works on top of HDFS
- Contributed by facebook, cloudera, Hortonworks etc

HDFS - max Data size = 30 + PB (Peta byte)
HBase - = 1 PB
H-Base = Hadoop-Base

* HBase Vs RDBMS

Features	RDBMS	HBase
1) Data structure →	Row oriented	Column oriented
2) Transaction →	Multi-Row ACID	Single row only
3) Query Lang →	SQL	NoSQL
4) Security →	Authentication	HDFS + Authentication
5) Indexed →	on any column	only on one column (row-key)
6) max data size →	TBs	~1 PB
7) Throughput →	1k queries/sec	million queries/sec

* HBase ACID Properties :

1) Hbase is not ACID compliant, because it does not support Isolation.
- A : Atomic - updates single record automatically.
- C : Consistent - No PK / FK relations
- I : Isolation - No guarantee changes sequentially.
- D : Durable - Changes are durable in HDFS.

* Hbase Table :

① column family - col1, col2, col3 ...
(ii) Rowkey : sorted, identifies rows, helps to retrieve.

Camlin

* Data model - Definitions
- Timestamp is added on every row
- Rowkey : Each row has a key column called Rowkey, Datatype of Rowkey can be anything, long, string, serialized data structures, Tables are sorted by Rowkey
- {columnfamily1 : col1 }
  colnfamily column qualifier
- column can create at the time of insersion automatically.
- compulsory to specify colnfamily at the time create table.

* Entity Definitions - Regions :
- Tables are partitioned Horizontally this called Regions.
- Regions are the units that get distributed over the Hbase cluster
- Regions are created as the data gets inserted into the table

* HBase locking : Updates are atomic -
- maintained by managing the timestamp
- concurrency - No of users inserted, updated, deleted

* HBase Architecture : components
1) HBase master : maintain Regionservers, Assigning Region-server, Recovering from Region server failures.
2) Region servers - carry Regions, Provide client's read/Write request, manage region spilts.
3) Zookeeper - maintains address of cluster master, manages hbase:meta, Provide co-ordination
* Region server list - conf/regionservers
  Site specific Config - conf/hbase-site.xml
- Retrieval time is same for any amount of data because of sorting on Hashmap table.

Cap theorem - HBase → CP RDBMS → CA
Mongo → CP Cassendra - AP

* HBase Access methods :
- HBase shell written in - Ruby lang. with cmds
- Web interface - port - 60010
- Java API - Native API for HBase
- REST API - Work on port 8080
  $ hbase rest start -p <port-No>
- Thrift - like a tool help to write application logic into languages - Enable access from any other lang.
- Hive / pig for analytics.

* HBase CAP Theorem :
C : consistency - consistence across multi node reads.
A : Availability - Not guarantee 100% uptime
each req. not get responce
P : Partition Tolerance
- HBase is tolerant over partitions
- If one partition fails others can manage.

* Installation - HBase

Date: 02/01/2022

* APACHE SPARK *

* Spark ? - spark is a programming framework or cluster computing platform
- Data cannot be stored, just compute
- Spark extends the MapReduce model
- In-memory operations and opensource.

* Why spark ? And its comparison
- Getting faster result than hive
- Used in cibil score, stock market

* spark software stack -
spark core - Basic functionality, RDD definition
Yarn, mesos, kubernates

* Components - spark SQL - project shark
- spark streaming - Enables processing of live streams of data.
- ML Lib - M/c learning functionality
prebuilt ML algorithms.
- Graphx -

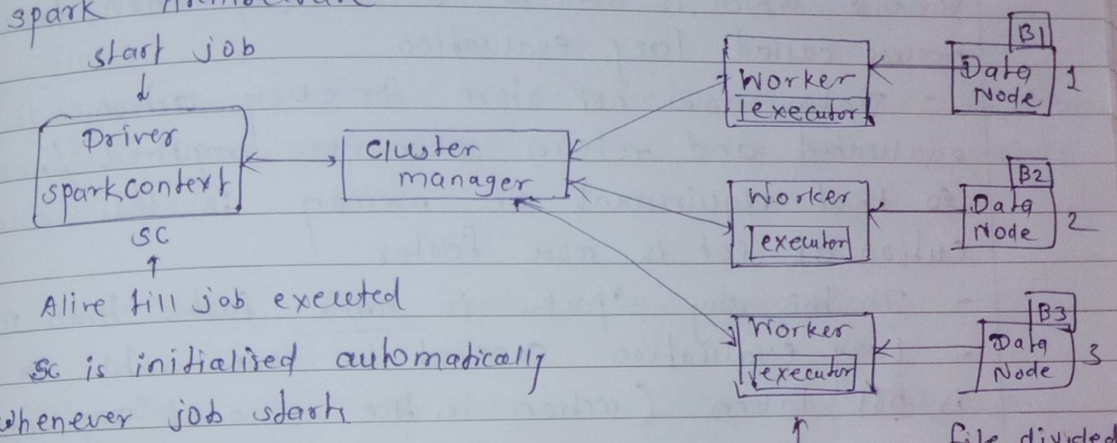
* cluster Managers - spark can be installed with-
a. standalone cluster manager (comes by default)
b. YARN (need to be configured
c. mesos - like resource manager framework
d. kubernetes - need to be configured (used on cluster)
These all are cluster manager
- spark versions - spark 3.x & spark 2.4
We use - spark - 3.2

* Modes of Installation - For standalone cluster spark
1. Local - Threaded mode
2. pseudo-distributed - Master & slaves runs as spark JVM
3. Distributed - M.f.s runs over a cluster.

start-worker ← for single slave - on localhost (single node)
start-workers ← more than one slave on cluster (multiple node)

RDD : Resilient Distributed Datasets
RDD's are immutable distributed collection of datasets.
Resilient - Hard to kill, Always existing.
Partitions may or may not be on a different node.
spits into multiple partitions

spark Architecture



Alive till job executed
SC is initialised automatically whenever job starts

e.g. Scala> val rdd1 = SC.Parallelize-
(Array("Pandas", "I like pandas"))

rdd1 - RDD type distributed variable
SC - spark context
> rdd1. ← see No. of operations we can do
e.g. > rdd1.count

Hadoop - mapReduce - Need to know what to do & how to do any operation
Hadoop - spark - Need to know what to do only spark will take care of how to do that.

* Spark does not store data on disk, it just utilises memory for that operation / processing and return result whenever needed.
Disk interference with input output is not required data hands on only RAM that why it is fast.

MapReduce - Increses no. of input/output oper^n with disk here slower than spark & takes too much time

* RDD Operations:

1] Transformation:

- Return a new RDD eg map, filter etc
- Lazy evaluation :- Whenver any action taken by it will check to operations variable which are empty, again check RDD var and through that collect actual data from file or any input & get back to RDD variable which is sent to action (operation) taken this process called lazy evaluation.
  - Data does not store for every action it just evaluated and return whenever required / needed so that requirement of memory is less and execution of job is more faster
  - In this way Spark is 100x faster than MapReduce
  - Lazy evaluation cannot be done until an action is not taken (action in the sense of any operation)

2] Action :- Return a result to driver program or write it or print it to storage

- eg count(), first(), collect()
- Action forces evaluation of transformations

- Action & transformation identified by return type -
  i.e. If return type is RDD ← Transformation
  If return type is long, ... ← Action

Transformation ← does not need data for execution
Action ← needs data for execution eg print()

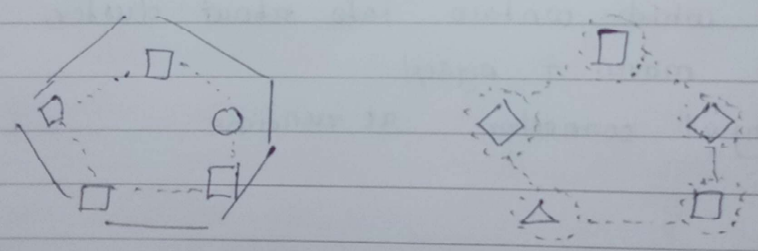
$ spark-submit --master local 01_Pyspark_Initialise.py
↳ local mode
$ spark-submit --master spark://gayatri- copy from
8080:7077 001_Pyspark_Initialise.py
↳ standalone mode
$ spark-submit --master yarn 001_pyspark_Initialise.py
↳ yarn cluster

Run commads - Deployment cmds from file

* KAFKA *

* Systems - An assembled code or machinery to perform a perticular job.

1) Monolithic - A single service with all responcibilities.
   e.g. Libre office (office, excel, powerpoint.)
2) micro-services - multiple machines with all responcibilities running in a distributed fasion.
   e.g.



* Messaging systems :-
  - Reliablity
  - These systems are typically queues.

Need: Decouple your system
- Introduce fault-tolerance
- Increase reliability
- Increase throughput - e.g Bike assembly

shif 4

Diff. messaging systems -

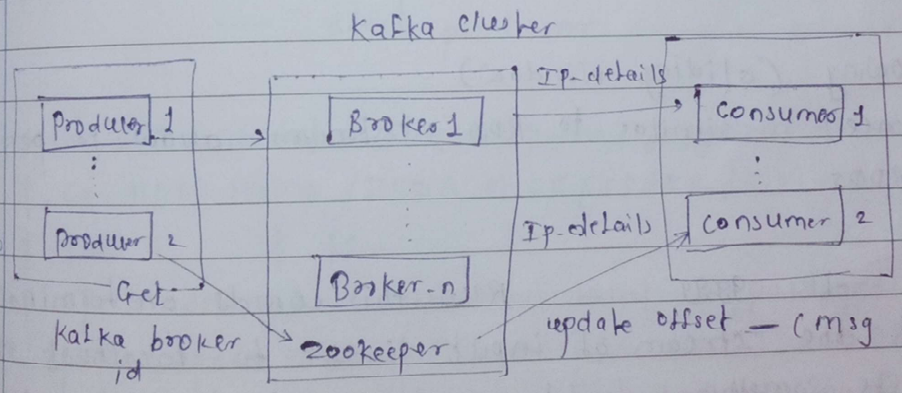
1) kafka 2) Amazon kinesis 3) RabbitMq (mesg que)
4) ActiveMq (messaging queue) 5) pulsar.

- Advantages of kafka - 1) Exactly once semantics
2) Guaranteed delivery 3) ordered delivery
4) storage of messages.

* Kafka Terminology :

1) kafka cluster - Provide pub-sub messaging.
2) kafka broker - Distributed s/w which contribute to cluster
3) Topic - Named queue to store messages
4) Producer - Generate msg & put onto kafka topic
5) consumer - Receives msg pulls the msg from kafka topic

* Kafka Architecture :-



cluster - Retends msg (saved) upto 7- days after that remove it.
Kafka - Multiple producer → Consumer Group.

* Kafka Installation : - stop h-base because kafka also works with zookeeper at same port.
- Here log files used to redirects errors
- so create mkdir /logs directory

Date: 07/01/2022

SPARK STREAMING

* What is streaming ?
- Real time data - potentially unlimited data
- Runs 24x7 - Typically operations on streams
- filter, Aggregates like count, avg, min, max etc.

e.g. zoom meeting, Netflix
- spark is used for structured streaming

* Output modes for streaming :
1) complete mode - Write the entire table to external device.
2) Append mode - only new rows are written
3) Update mode - only updated rows are written.

- Spark is micro batching streaming used on structured data

* Windowing (sliding window)
* Persistence - similar to RDD, Dstreams allow to persist the RDDs
-

* $ nc -lk 9999 - Run this comds on terminal to start the stream of input lines to localhost:9999
> Write something on terminal that data will be streamed onto console. (or any file format which want to write the stream data)

$ pwd | nc -lk 9999 - store cmd o/p to stream of data file

stream - df.readstream()
df.writestream
.outputmode("complete")
format("console") ← show it on console If want to store then pass file path here.

Page :
Date :

RDD Vs Dataframe Vs Dataset.

1) RDD - Resilient Distributed Datasets.
- Fundamental Data structure of spark
- Each dataset in RDD divided into logical partitions
- RDD is immutable distributed collection
- RDD is partitioned across nodes & processed in parallel
- RDD makes faster & efficient data processing.
- RDD is basic, low level, primary user API provided by spark.

2) Dataframe - from : spark 1.3
- Unlike RDD, Data is organised named columns, like table in a relational db.
- Dataframes in spark have their execution automatically optimized by a query optimizer called catalist.
- Catalist optimizer understands the structure of data with dataframes, it makes speed up computation
- Dataframe is a primary high level extraction of RDD

3) Dataset : It is an extension to dataframe, provides type-safe, object-oriented interface
- Dataframes & Datasets APIs are built on top of Spark SQL Engine

errors	SQL/RDD	Dataframe	Datasets
syntax	Run-time	compile time	compile time
Analysis	Run-time	Run-time	compile time

Page :
Date : 11/01/22

Apache Airflow :

- Platform to programatically author, schedule & monitor workflows.
- Workflow is a sequence of tasks that processes a set of data
- 4 key principles - Scalability, Dynamicity, Extensible, Elegant

stage :

1) stat & fetch the data from source
2) Flattern XMLs
3) Run GroupBy Queries
4) Dump into DB

Username - admin
Passwd - @gauw196

* install Airflow on Anaconda

- Download & install anaconda
- follow the installation guide given.
- follow steps to install airflow.
- create airflow-env ← environment
- open pycharm editor ← setting ← project ← interpreter '+' — type 'apache-airflow' — 2.2.3 version install packages ok
- open terminal - cd airflow/
- $ ls -l $ mkdir dags
- $ vi airflow.cfg ← configuration file use whenever need to modify some default cont^n

eg change time interval
min_file_process_interval = 30

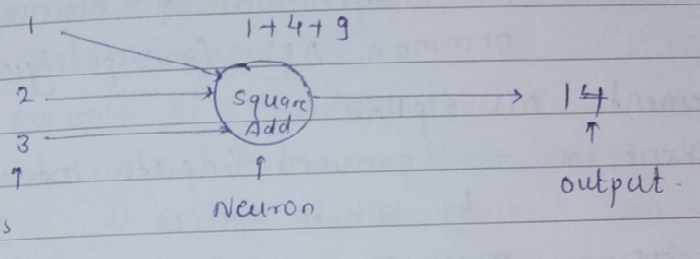
$ conda activate airflow-env ← go into env
$ airflow webserver --port 8080 ← start webserver
- open new terminal - $ airflow scheduler
- open browser - localhost:8080 ← see dags

* [Every Dataframe having Dataset of type Row.

Page :
Date : 13/01/22

SPARK ML

* Neuron - smallest unit specialised to transmit inform throughout the system.
- Number of i.e infinite number of neurons presents



* Data Types :
1. scalar - single value component
2. Vectors - multi value - sparse & dense vector
3. Text
4. Rows - DF data type by default (from spark.sql)

* Summerizer - Provides vectors column summary statistics.
summerize(count, min, max ..

* Basic stats - corelation .show (truncate = False)
Gives entire data instead (data.....)

* Take dataset from kaggle - using spark

Data Types in mLib :-
1] Local vector - Has Integer-typed, 0-based indices and double-typed values, stored on a single machine
Two types of vector - 1) sparse 2) Dense.
1) Dense vector - It is backed by double array.
2) Sparse vector - It is backed by two parallel arrays
i) Indices & values.

e.g.
(1.0, 2.0, 3.0) ← vector ← dense vector → [1.0, 2.0, 3.0]
→ sparse vector format → (3, [0, 2], [1.0, 3.0])
3 - is the size of vector

Date :

* ML Pipline : A sequence of Algorithms.

1) Dataframe
2) Estimator - implements fit()
3) Transformer - implements transform()
4) Pipline - combination of Estimator & Transformer
5) Parameter - All transformers & estimators share common API for specifying parameters

* Implements ML-Pipline
- Tokenizer - convert inputs into lowercase & split by white spaces.
- HashingTF - maps sequence of terms freq. using hashing tricks

* co-relation - corelation betn two numerical variables
- Using spark we can calculate the co-relation betn two series of data.
- Spark ML support two type of co-relation
  - Pearson's corelation
  - Spearman's correlation
- sparkML provides ability to calculate pairwise correlations among many series.
- We can use heatmap to show graphical co-relation

* Vectors : are datatype containing floats

Dense vector :

1	0	10	0	0	13	0	20

Disk space increased

sparse vector :

- It can be
1) - Dictionary
2) - List of tuples
3) - It can be two sorted lists containing indices and values.

0	1
2	10
5	13
7	20

Less disk space required

Vectors.sparse(10, {1:10, 2:20, 3:30})

10 ← size of array

values - Dictionary

↳ (size, [indexes], [data])

print ↑ like this