Rs = 120/-

HADOOP

CLASS ROOM NOTES

Kelly Technologies

Flat No: 212, 2nd Floor, Annapurna Block,
Aditya Enclave, Ameerpet, Hyderabad, AP.

Ph No: 040 6462 6789, 0998 570 6789
E-mail: info@kellytechno.com, www.kellytechno.com.

Hadoop

By Mr. GOPAL KRISHNA

* Inspired by Google Bigtable and mapreduce papers circa 2004.
* Created by DougCutting.
* Originally built to support distribution for Nutch Engine.
* Named after a Stuffled elephant.

BigData :-

Big Data is about the growing challenge that organizations face as they deal with large and fast-growing sources of data or information that also present a complex range of analysis and use problem. These can include:

* Having a computing infrastructure that can ingest, validate, and analyze high volumes (size and/or rate) of data.
* Assessing mixed data (Structured and unstructured) from multiple sources.
* Dealing with unpredictable content with no appare schema or structure.
* Enabling real-time or near-real-time collection, analysis and answers.

Kelly Technologies
Flat No. 212, 2nd Floor,
Annapuma Block, Aditya Enclave,
Ameerpet, Hyderabad-500 016.
Ph: 040-6462 6789, 968 570 6789

BigData technologies describe a new general of technologies and architectures, designed to economica extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis.

The data comes from everywhere: sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records, and cell phone GPS signals to name a few. this data is BigData.

three vs of BigData :-

Volume describes the amount of data generated by organizations or individuals.

Variety describes structured and unstructured data, such as text, sensor data, audio, video, click streams, log files and more.

Velocity describes the frequency at which data is generated, captured and shared.

BigData - Hadoop :-

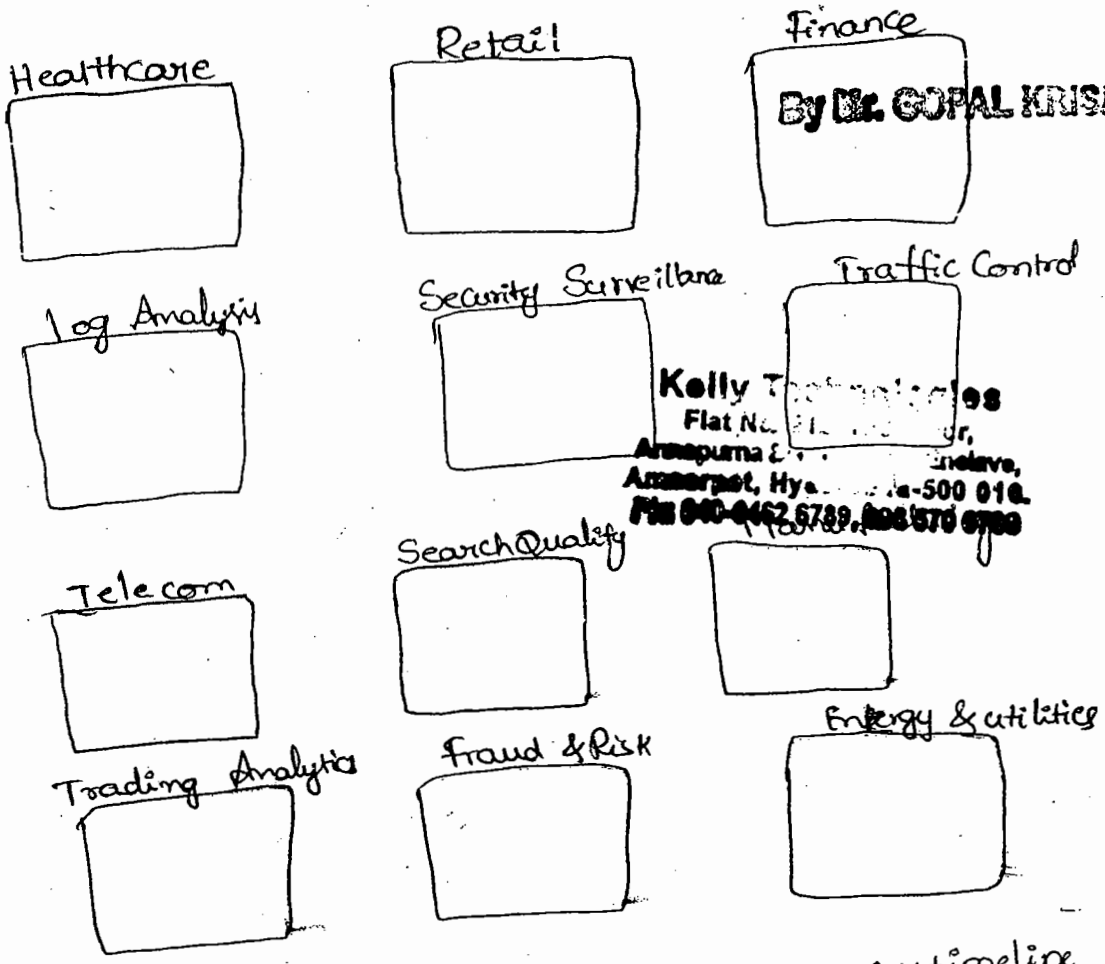
Hadoop is the Apache open source software framework for working with BigData. It was Derived from Google technology and put to practice by yahoo and others. But, Bigdata is too varied and complex for a single one-size-fits-all solution.

while Hadoop has surely captured the greatest name recognition, it is just one of three classes of technologies well suited to storing and managing BigData.

Hadoop is software framework, which means it includes no. of components that were specifically designed to solve large-scale distributed data storage, analysis and retrieval tasks.

need of BigData :-

* Google process 20 PB a day
* wayback machine has 3 PB + 100 TB/month.
* facebook has 2.5 PB of user data + 15 TB/day.
* eBay has 6.5 PB of user data + 50 TB/day.



Hadoop for the research users a quick timeline of how things have progressed :-

2004 :- Initial versions of what is now HDFS (Hadoop distributed file System) and mapReduce implemented by DougCutting & Mike Cafarella.

Dec 2005 :- Nutch ported to the new framework. Hadoop runs reliably on 20 nodes.

Jan 2006 :- Doug Cutting joins yahoo!

Feb 2006 :- Apache Hadoop project official support the standalone development of mapReduce and HDFS. Adoption of Hadoop by yahoo! Grid team.

April 2006 :- sort benchmark (10GB/node) run on 188 nodes in 47.0

May 2006 :- yahoo! setup a hadoop research cluster - 300 nodes sort benchmark run on 500 nodes in 42 hours [better hardware than April benchmark] set of machines.

2006 :- Research cluster reaches 600 nodes.

2006 :- sort benchmark run on 20 nodes in 1.8 hrs
100 nodes in 3.3 hrs
500 nodes in 5.2 hrs
900 nodes in 7.8 hrs.

Jan 2007 :- Research cluster reaches 900 nodes.

Apr 2007 :- Research clusters - two clusters of 1000 nodes.

Apr 2008 :- won the 1-Terabyte sort benchmark in 209 seconds on 900 nodes.

Oct 2008 :- Loading 10 Tera Bytes of data per day onto the research clusters.

march 2009 :- 17 clusters with a total of 24,000 nodes

April 2009 :- won the minute sort by sorting 500GB in 59 seconds (on 1400 nodes) and the 100-terabyte sort in 173 minutes [on 3,400 nodes].

Example for a statistical Analysis of ONE MINUTE internet search :-

- Google Receives over 2,000,000 search Queries
- Facebook receive 34,722 "Likes".
- consumers spend $ 272,070 on web shopping.
- Apple receives 47,000 Apps downloads.
- 370,00 minutes of calls on skype;
- 98,000 posts on tweets,
- 20,000 posts on Tumblr,
- 13,000 hours of music streaming on Pandora,
- 12,000 new ads on Craigslist,
- 6,600 pictures uploaded to FLICKr
- 1,500 new blog posts,
- 600 new youTube videos.

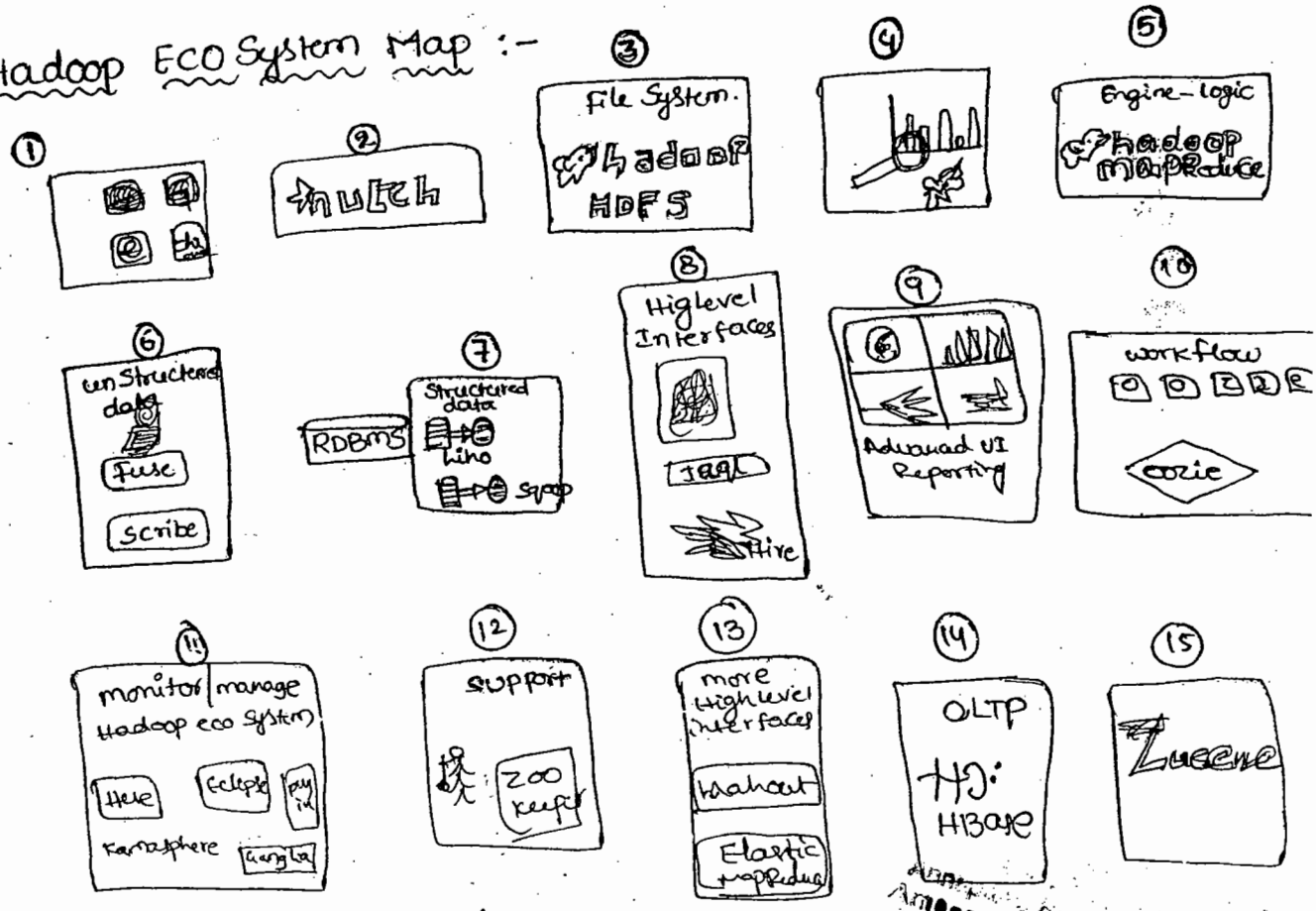
Hadoop's History :-

→ Hadoop was created by Doug Cutting & Michael J. Cafarella. Doug, who was working at yahoo at the time, named it after his son's toy elephant. It was originally developed to support distribution for the Nutch search engine project.

→ Hadoop is based on work done by Google in the late 1990s / early 2000s.

specifically, on papers describing the Google file System (GFS) published in 2003, and MapReduce published in 2004.

Hadoop ECO System Map :-



① Large data on the web.

② Nutch built to crawl this web data.

③ Large volume of data had to saved - HDFS introduced.

④ How to use this data ? Report

⑤ MapReduce Framework built for coding & running analytics

⑥ unstructured data - weblogs, click streams, Apache logs, Server logs - fuse, webdav, chukwa, flume and scribe.

⑦ sqoop and Hiho for loading data into HDFS - RDBMS data.

⑧ High level interfaces required over low level map reduce programming - Hive, Pig, Jaql.

⑨ BI tools with advanced UI Reporting.

⑩ workflow tools over Map-Reduce processes and High level languages - Oozie.

⑪ monitor & manage hadoop, run jobs/hive, view HDFS - high level view - Hue, Karmasphere, eclipse plugin, cacti, ganglia.

⑫ Support frameworks - Avro(Serialization), Zookeeper(Coordination).

⑬ More High level interfaces/uses - Mahout, Elastic MapReduce

⑭ OLTP also possible in HBase.

⑮ Lucene is a text search engine library written in Java.

Different ECOSystems of Hadoop :-

* Hadoop is best known for MapReduce and it's Distributed file System (HDFS, renamed from NDFS)

[Note:- NDFS is also used for a projects that fall under the umbrella of infrastructure for distributed computing and large scale data processing]

HDFS :- if you want 4000+ computers to work on your data, then you'd better spread your data across 4000+ computers. HDFS does this for you. HDFS has a few moving parts. The Datanodes store your data, and the Name Node keeps track of where stuff is stored. There are other pieces, but you have enough to get started.

MapReduce :- This is the programming model for Hadoop. There are two phases, not surprisingly called map and Reduce. TO impress your friends tell them there is sort & Shuffle between the map and Reduce phase. The JobTracker manages the 4000+ components of your mapReduce job. The task trackers take orders from the JobTracker. if you like Java then code in Java. if you like SQL or other non-Java languages you are still in Luck, you can use a utility called Hadoop Streaming.

Hadoop Streaming :- A utility to enable mapReduce code in any language : C, Perl, Python, C++, Bash etc The examples include a python mapper and an AWK reducer.

Hive and Hue :- if you like SQL, you will be delighted to hear that you can write SQL and Hive convert it to a mapReduce job. No, you don't get a full ANSI-SQL environment, but you do get 4000 notes and multi-petabyte scalability. Hue gives you a browser based graphical interface to do your Hive work.

Pig :- A Higher-level programming environment to do mapReduce coding. The Pig Language is called Pig Latin you may find the naming Conventions Some what unconventional, but you get incredible price-performar and high availability.

Sqoop :- provides bi-directional data transfer between Hadoop and your favorite relational database.

Oozie :- Manages Hadoop workflow. This does not replace your Scheduler or Bpm tooling, but it does provide if-then-else branching and control

Hadoop JOBS.

HBase:- A super-scalable key-value store. It works very much like a persistent hash-map (for python fans think dictionary). It is not a relational database despite the name HBase.

Flume:- A real time loader for streaming your data into Hadoop. It stores data in HDFS and HBase. You will want to get started with Flume, which improves on the original Flume.

mahout:- machine learning for Hadoop. used for predictive analytics and other advanced analysis.

Fuse:- makes the HDFS system look like a regular filesystem so you can use ls, rm, cd, and others on HDFS data.

zookeeper:- used to manage Synchronization for the cluster. you won't be working much with zookeeper, but it is working hard for you. If u think you need to write a program that uses zookeeper you are either very, very, smart and cloud be a commitee for an Apache project, or you are about to have a very bad day.

How Big is 'BigData'?

with time, data volume is growing exponentially. Earlier we used to talk about MegaBytes or GigaBytes. But time has arrived when we talk about data volume in terms of terabytes, petabytes and also zettaBytes! Global data volume was around 1.82 ZB in 2011 and is expected to be 7.9 ZB in 2015. It is also known that the global information doubles in every two years!

How analysis of BigData is useful for organizations? 5

Effective analysis of Big Data provides a lot of business advantage as organizations will learn which areas to focus on and which areas are less important. BigData analysis provides some early key indicators that can prevent the company from a huge loss or help in grasping a great opportunity with open hands! A precise analysis of BigData helps in decision making For instance, now a days people rely so much on Facebook and Twitter before buying any product or service.

HDFS and mapReduce features in Hadoop versions:

Feature	1.x	0.22	2.x
secure Authentication	yes	NO	yes
old configuration Names	yes	deprecated	deprecated
New Configuration Names	NO	yes	yes
old mapReduce API	yes	yes	yes
New MapReduce API	yes [with some missing libraries]	yes	yes
MapReduce 1 runtime [classic]	yes	yes	NO
MapReduce 2 runtime [YARN]	NO	NO	yes
HDFS Federation	NO	NO	yes
HDFS high availability	NO	NO	yes

⊛ what is Hadoop ?

- Hadoop is analysis for Bigdata.
- Hadoop is an open source framework for creating distributed applications that process huge amount of data.

One definition of huge

Ex:- 25,000 machines
more than 10 clusters
3 PB of data (compressed, unreplicated)
700+ users
10,000+ jobs/week.

mother definition

- Hadoop is a open source, Distributed, Batch processing and fault-tolerance system which is capable of storing huge amount of data [TB, PB, zettabytes etc] along with processing on the same amount of data.
- Hadoop easy to use parallel programming model.
- Hadoop framework consists two main core components:

① HDFS — it is responsible for storing massive amount of data on the cluster.

② mapReduce — It is responsible for processing massive amount of data on the cluster.

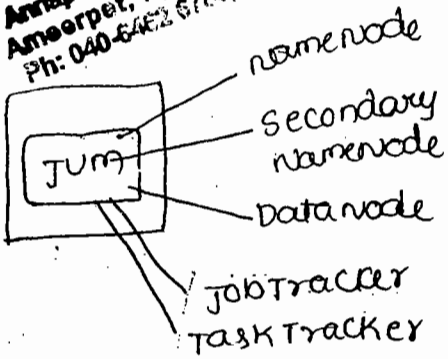
- HDFS & mapReduce capabilities are it's Kernel for Hadoop.

Hadoop modes of Installation :-

1. stand alone mode
2. pseudo distributed mode [labwork]
3. Fully Distributed mode [Realtime]

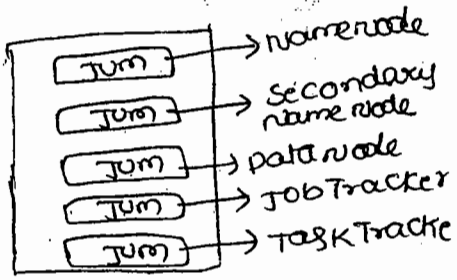
1. Stand alone mode :← Development Test Debug.

- single machine
- no daemons are running
- Everything runs in single jvm.
- standard os storage.
- Good for development and test with small data, but will not catch all errors.



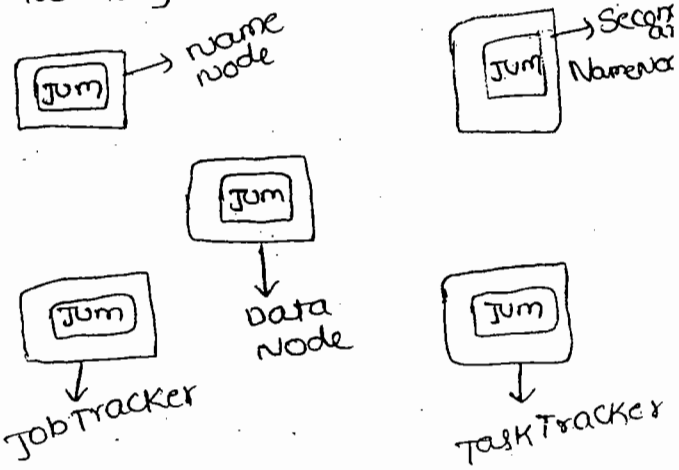
2. pseudo distributed mode :- [multiple jvm's in Single node]

- Single machine but cluster is simulated.
- Daemons run in Separate jvm.
- Separate jvm's in Single node.
- Good for Development & Debugging.



3. Fully Distributed mode :- [All components run in separate nodes]

- Run Hadoop on cluster of machines
- Daemons run
- production environment
- Good for staging & production.



What is Distributed File System :-

- System that permanently store data.
- Support concurrency, distribution, replication access to file and remote servers.
- divided into logical units (files, shards, chunks, blocks)
- DFS's are working n/w based approach because of DFS's are more complex than Regular disk file systems.

Ex:- file system tolerate node failure without suffering data loss.

Hadoop Distributed File System :-

- Hadoop is a distributed file system and it uses to store bulk amounts of data like terabytes (or) even petaBytes.
- HDFS support high throughput mechanism for accessing this large amount information.
- In HDFS files are stored in sequential redundant manner over the multiple machines and this guaranteed the following ones.
  1. Durability to failure
  2. High availability to very parallel applications.

Example: NFS (Network file system)

- NFS gives remote access to a single logical volume stored on a single machine.
- NFS server can visible a portion of it's local files System to external clients and also the client can mount this remote file system directly into their own Linux file system, and interact with it as though it were part of the local drive.

Advantage of NFS

- It is transparency [That is clients do not need to be particularly aware that they are working on files stored remotely]

Advantages of HDFS :-

1. HDFS store large amount of information.
2. HDFS is simple and robust coherency model.
3. That is it should store reliability.
4. HDFS is scalable and fast access to this information and it also possible to serve large number of clients by simply adding more machines to the cluster.
5. HDFS should integrate well with Hadoop mapReduce, allowing data to be read and computed upon locally when possible.
6. HDFS providing streaming read performance.
7. Data will be written to the HDFS once and then read several times.
8. The overhead of cashing is helps the data should simply be re-read from HDFS source.
9. Fault-tolerance by detecting faults and applying quick, automatic recovery.
10. processing logic close to the data, rather than the data close to the processing logic.
11. portability across heterogeneous commodity hardware and operating Systems.
12. Economy by distributing data and processing across clusters of commodity personal computers.
13. Efficiency by distributing data and logic to process it in parallel on nodes where data is located.
14. Reliability by automatically maintaining multiple copies of data and automatically redeploying processing logic in the event of failures.

Disadvantages of HDFS :-

In distributed file system, it is limited in it's power. The files in an NFS volume all reside on a single machine. This will create some problems.

① It does not gives any reliability guarantees if that machine goes down.

Ex:- By replacing the files to other machine.

② All the clients must go to this machine to retrieve their data. This can overload the server if a large no. of clients must be handled.

③ clients need to copy the data to their local machines before they can operate on it.

Goals of HDFS :-

① Very large distributed file System :- 10K nodes, 100 million files, 10 PB.

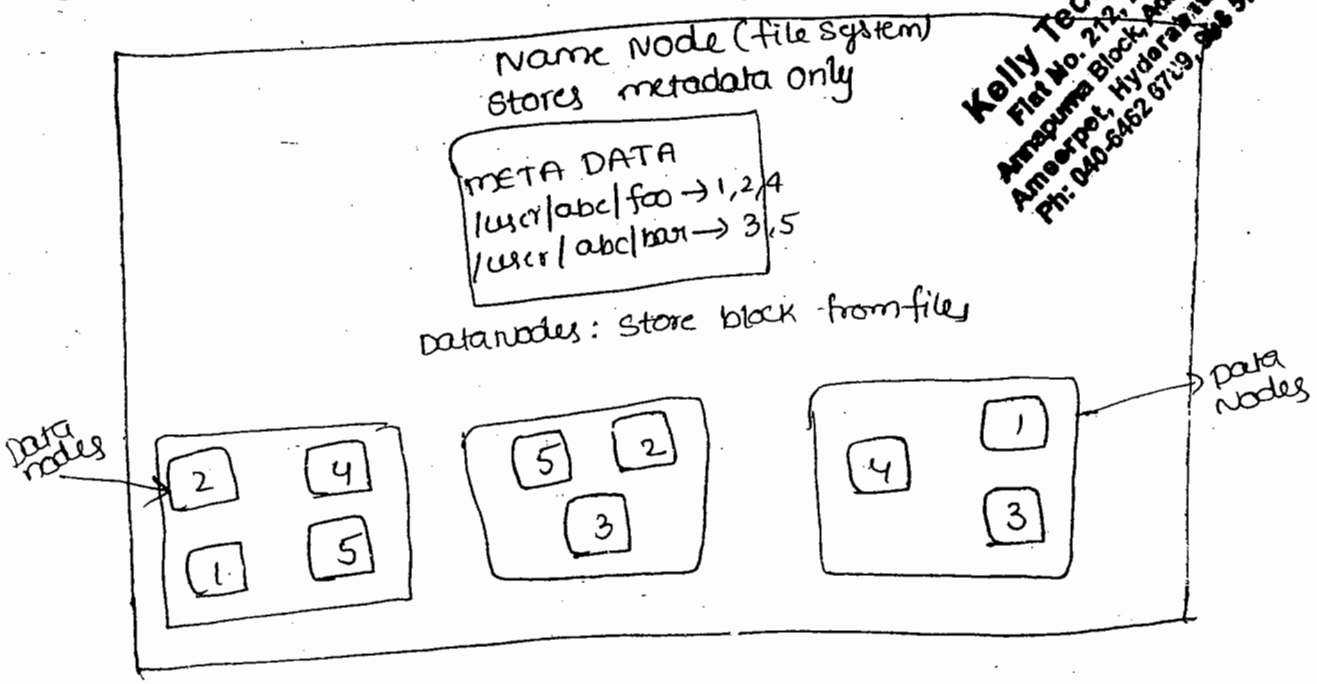
② Assumes commodity hardware : files are replicated to handle hardware failure. Detect failures and recover from them.

③ optimized for batch processing :- Data locations exposed so that computations can move to where data resides. It provides very high aggregate bandwidth.

HDFS is a block Structured file System :-

Block :- Block is the minimum unit of data that is said in HDFS, which is typically 64 MB by default. However we can increase the blocksize in the multiples of 64 MB.

- Each file is broken into blocks of a fixed size and these blocks are stored across a cluster of one or more machines with data storage capacity.
- Individual machines in the cluster are called the Data nodes.
- A file can be made of several blocks and not necessarily stored on the same machine.
- The target machine chose each block randomly on a block-by-block basis.
- So access permission to a file may need the cooperation of multiple machines and it supports file size for larger than a single machine DFS.
- Individual files sometimes need large space than a single hard drive could hold. if several machines must be involved in the serving of a file then a file could be rendered unavailable by the loss of any one of those machines. HDFS combats this problem by replicating each block across a no. of machines (by default 3)



• In the above figure the DataNodes represents multiple files with replication factor of 2 and the NameNode maps the filenames onto the block ids.

• In block structured file systems commonly use a block size on the order of 4 or 8 KB.

• The default block size in HDFS is 64 KB. This permits HDFS to decrease the amount of metadata storage required per file.

• In HDFS block structured file system, all the information's are handled by single machine called NameNode.

• The NameNode stores all the metadata for the file system. All the informations like tracks file names, permissions, and the locations of each block of each files etc can be stored in the main memory of the NameNode machine and it permits fast access to the metadata.

• The allow open a file the client first contacts the NameNode and access a list of locations for the block that comprise the files and these locations identify the DataNodes which hold each block.

• clients then read file data directly from the DataNode servers in parallel. So the NameNode is not directly involved in the bulk data transfer keeping it's overhead to a minimum.

• NameNode information is should be preserved even if the NameNode machine fails.

• NameNode failure is more severe for the cluster than DataNode failure.

• when individual Data nodes may crash and the entire cluster will continue to operate, the loss of the NameNode will render the cluster in accessible until it is manually restored.

Features of HDFS :-

HDFS is file system designed for storing have some key characterstics. They are

1. Support for very large files.
2. Commodity Hardware
3. Streaming data access
4. High-latency data access
5. lots of small files
6. multiple writers, arbitrary file modifications
7. moving computation is than moving data.

① Support for very large files :-

Files that are support hundreds of megabytes, Giga Bytes Tera Bytes in size.

Hadoop clusters are running today that store PB, ZB of data. ↓ set of Nodes.

② Commodity Hardware :-

- HDFS requires a commodity Hardware [The H/w which is been available for most of the vendors and Hadoop does not require high configuration Hardware, expensive s/w to be part of it's basic installation.
- HDFS is always working without a noticeble interruption to the users. [In the face of failures]
- For the commodity Hardware chance of node failure across the cluster is high, at least for large clusters.

3) Streaming data access :-

- HDFS is the most efficient data processing pattern. That pattern is write once/read many times.
- HDFS will follow streaming data access [Sequential flow]

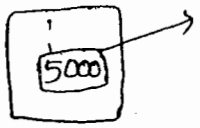
Ex:-

file Data
1
2
3
:
1,00,000

→ search for 6564 record.

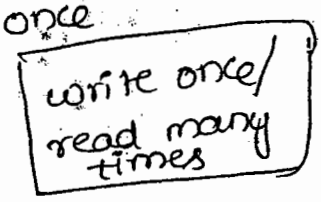
→ Search come from 1 to 6564 Sequentially (no random, index access)

Ex:- updation

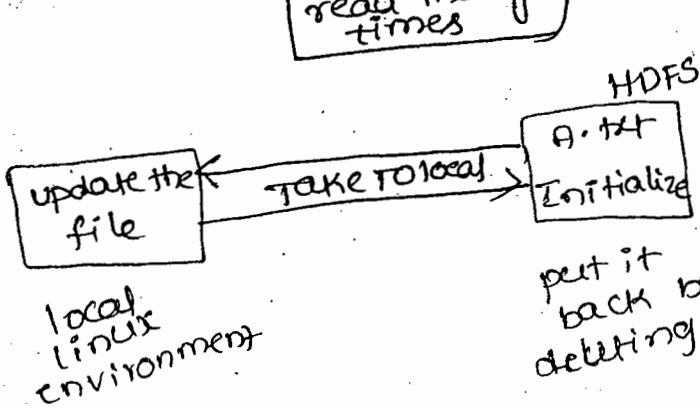


if we can update the 500 record. It is not possible here. Take into local, update the file.

- Sequential flow means, when we can save the file on HDFS file, we can read the data (or) write the data in the sequential fashion only.
- HDFS having the feature of read many times and write once



Ex:-



once we can store a file on top of HDFS file, we can not override that file, instant we can take the file into local environment, if we modify the content and we can put the same file on top of the HDFS file. By deleting whole HDFS file.

④ High-latency data access :-

- Generally applications that require low-latency access data.
- As we are using in HDFS, used to very large amount of data. Because to take more time - this may be at the expense of latency.

Ex :-

Hadoop (Sequential Access)

hive > ↓ high latency — 12 min

RDBMS (Randomly Access)

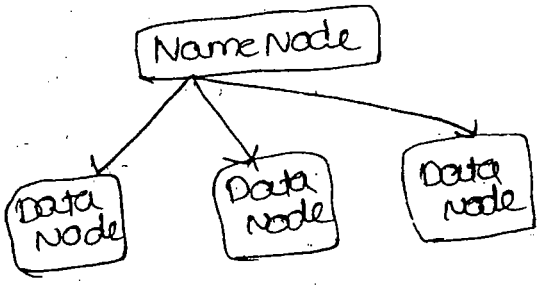
SQL > ↓ low latency — 5 min

Here HDFS is used to large amount of data But to take more time.

In the tens of milliseconds range, will not work well with HDFS.

⑤ lots of small files :-

- NameNode is responsible for maintaining the metadata information of the hadoop file system.
- hadoop file system have no. of files in the amount of memory on the NameNode.



⑥ multiple writers, arbitrary file modifications :-

- Files in HDFS may be written to by single writer.
- writers are always made at the end of the file.
- There is no support for multiple writers (or) modifications at arbitrary offset in the file.

[These might be supported in the feature, but they are likely to be relatively]

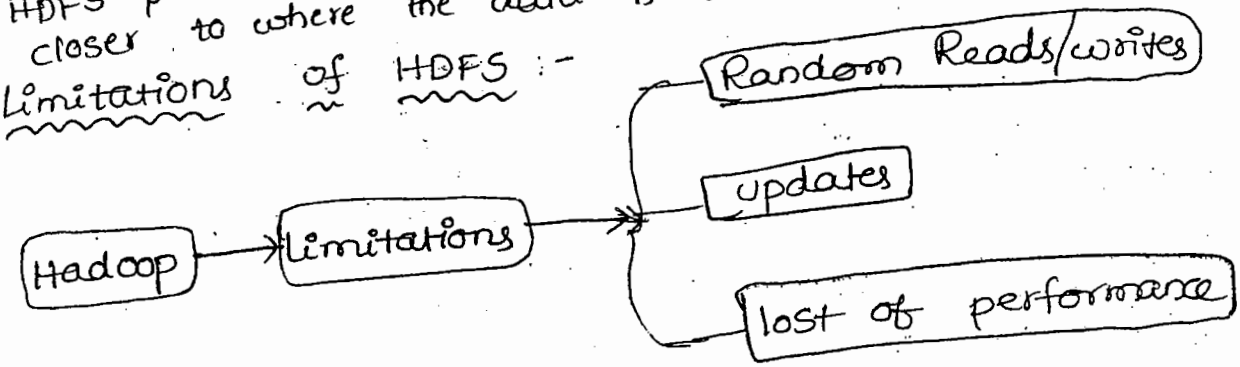
⑦ moving computation is easy than moving data :-

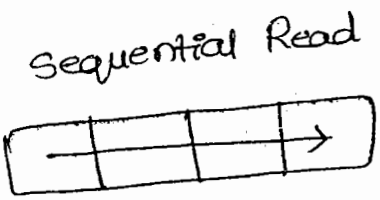
In Traditional distributed system data from SAN (Shared Access Network) device will be copied to.

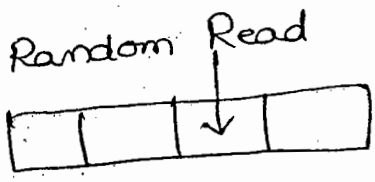
- A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge.
- The assumption is that it is often better to migrate the Computation closer to where the data is located rather than moving the data to where the application is running.

HDFS provides interfaces for application to move themselves closer to where the data is located.

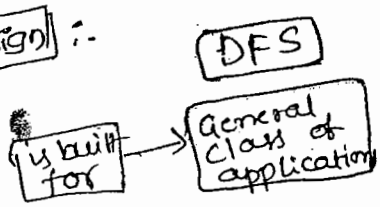
Limitations of HDFS :-

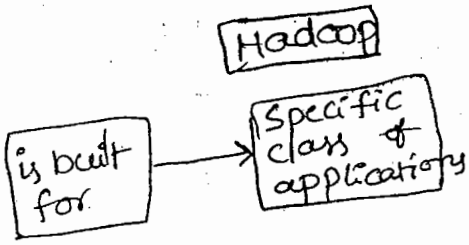






esign :-





Hadoop Architecture...

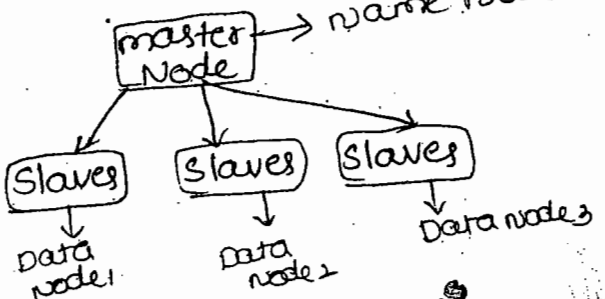
→ Hadoop Architecture will be classified into 5 daemons.

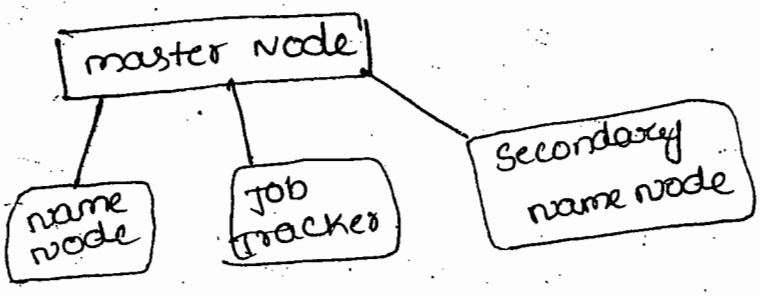
1. Namenode and it's functionality.
2. Datanode and it's functionality.
3. JobTracker and it's functionality.
4. TaskTracker and it's functionality.
5. Secondary Namenode and it's functionality.

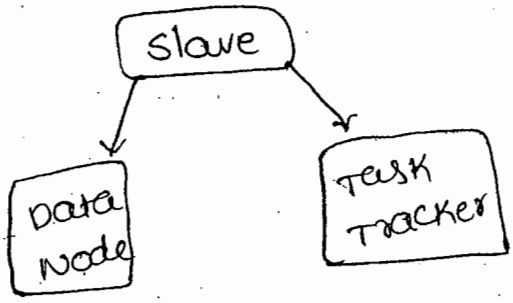
By Mr. GOPAL KRISHNA

→ Hadoop architecture follows master-slaves architecture.

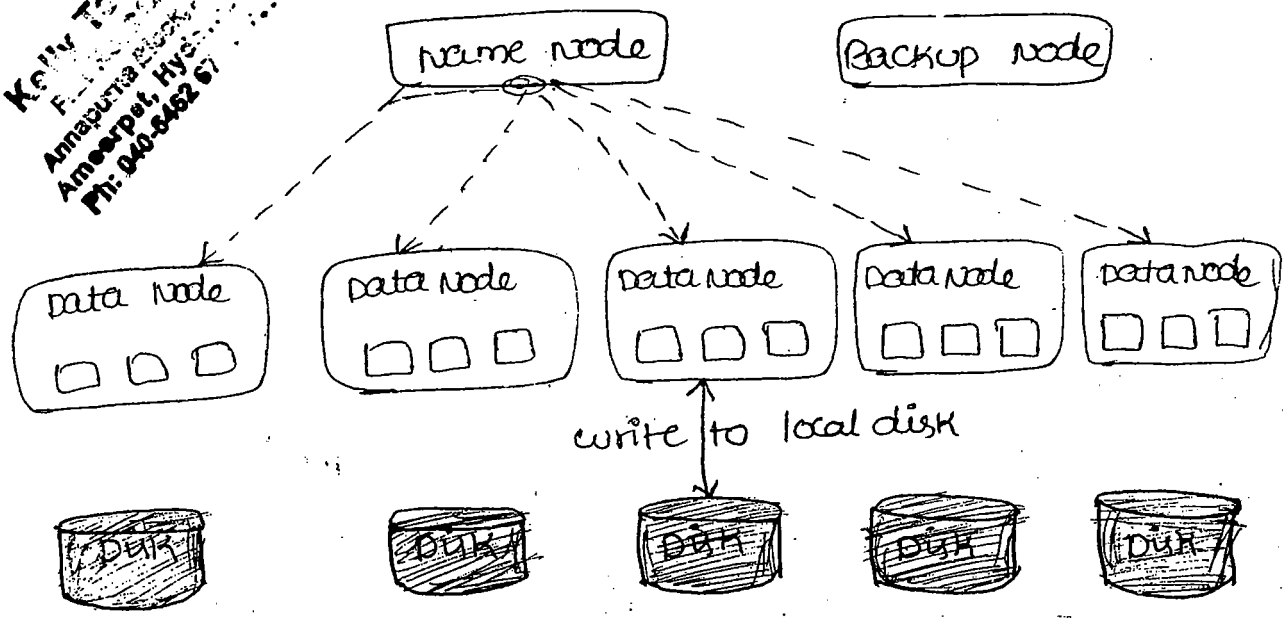
Data → raw data





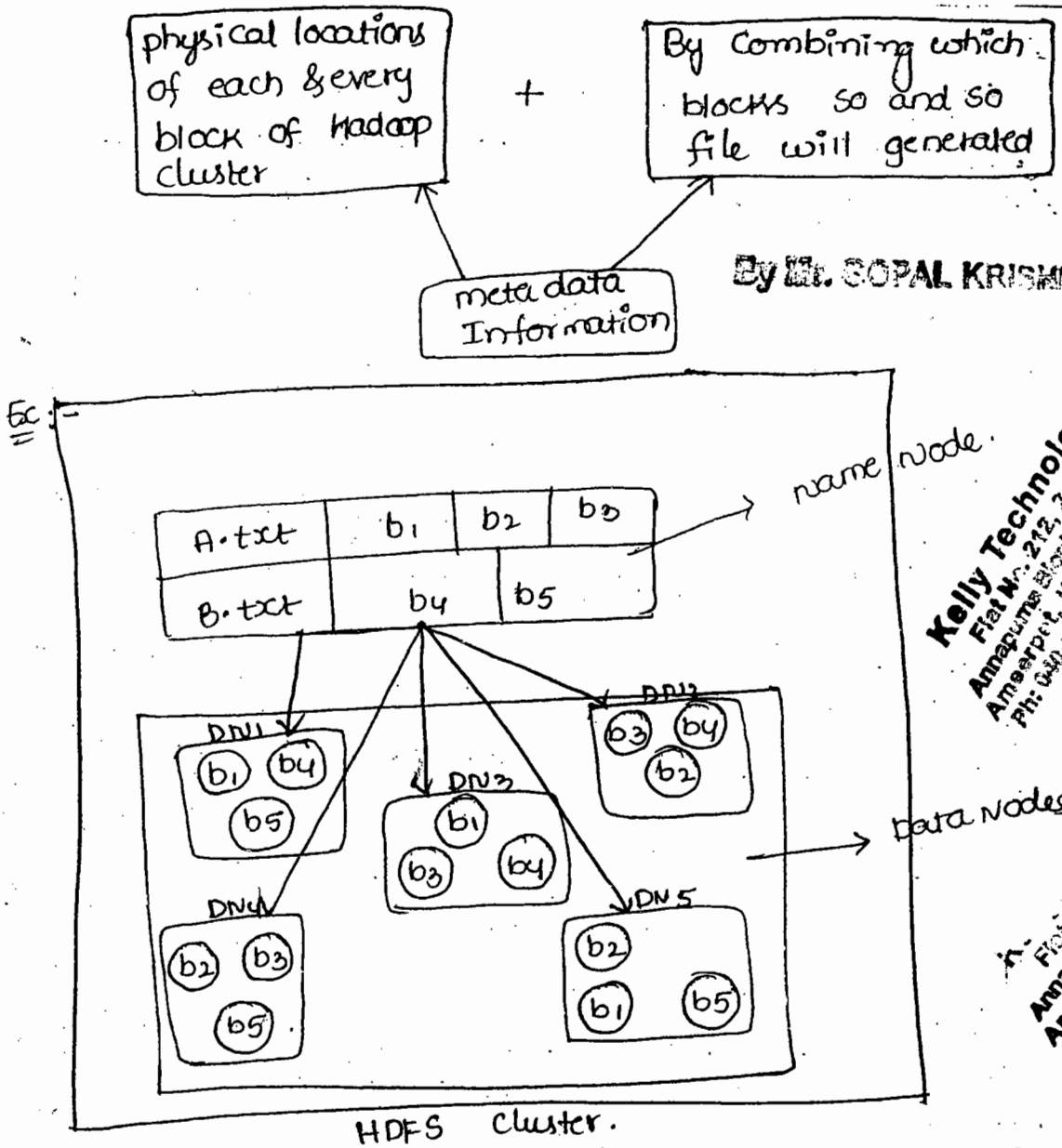


HDFS Architecture



NameNode :-

- The master node in Hadoop architecture it is treated as NameNode.
- NameNode is responsible for maintaining the metadata information of the Hadoop file system.
  - information → process the data
  - metadata → data about data.
- NameNode maintains the file System. The file System have metadata for all the files & directories. This information is stored persistantly on the local disk.
  - persistantly → permanently.
- NameNode maintain the file System NameSpace.
- The NameNode executes the NameSpace file, The file operations like opening, closing & renaming files & directories.
- The NameNode will only update two important permanent files of Hadoop file System called the
  - fs name space Image
  - edit log.
- NameSpace access to files by clients.



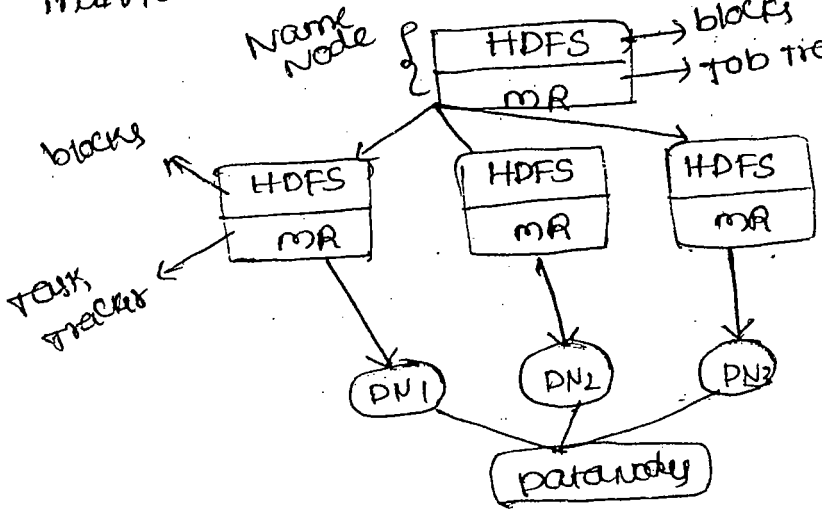
- A client access the file system on behalf of user by communicating with the NameNode.
- The client access file system to a portable operating system Interface (POSIX). So, the user code does not to know about the NameNode.
- Assign Blocks to Datanodes.
- Keeps track of live nodes (through heart beats)
- Limitates re-replication in case of data node loss.
- Block metadata is held in memory.
  - will run out of memory when too many files exist.
- Is a single point of failure in the System
  - some solutions exist.

② Data Node :-

- Data Node is a place hold of the data i.e actual data in datanodes only in the form of HDFS blocks. [By default each block size is 64mB] (not the metadata.)
- Data Nodes are store and retrieve blocks, reporting to Namenode.
- A client access the file system on behalf of the user by communicating with the Data Nodes.
- uses underlaying regular FS for Storage (eg: ext3)
  - Takes care of distribution of blocks across disks
  - Don't use RAID
  - more disks means more IO throughput.
- Does not Know about the rest of the cluster (Shared nothing)

③ Job Tracker :-

- Job Tracker is one of the s/w daemons for Hadoop Architecture.
- Job Tracker is responsible for scheduling & Rescheduling the tasks in the form of mapReduce jobs.
- Job Tracker is also getting the acknowledgement [response] back from the Task Tracker.
- Generally JobTracker will recide on top of the Namenode.
- JobTracker manages the mapreduce jobs, distributes individual tasks to machines running the Task Tracker.



④ Task Tracker :-

- Task Tracker is responsible for instantiating & monitoring individual map & reduce works.
- Task Tracker is also known as s/w daemon for Hadoop architecture.
- Task Tracker is primarily responsible for executing the tasks assign by the JobTracker in the form of MRJOBS.
- Generally Tasktracker will recides on top of the data nodes.

Note :-

① JobTracker & Task Tracker are the two important daemons of Hadoop architecture, which are completely responsible for the processing of the data by the means of mapReduce programming.

⑤ Secondary NameNode :-

- Secondary NameNode is deprecated.
- It performs periodic checkpoints of the namespace and helps keep the size of file containing log of HDFS modifications within certain limits at the NameNod
- It is replaced by checkpoint node.
- Secondary namenode will acts as separate physical file.
- whenever the primary node is down in Hadoop architecture, the Secondary namenode will come into picture.
- we can never ever call a Secondary NameNode as the replica our primary Node.
- Secondary NameNode is responsible ✗ only read the fsImage space & edit log.

> what is the checkpoint mechanism?

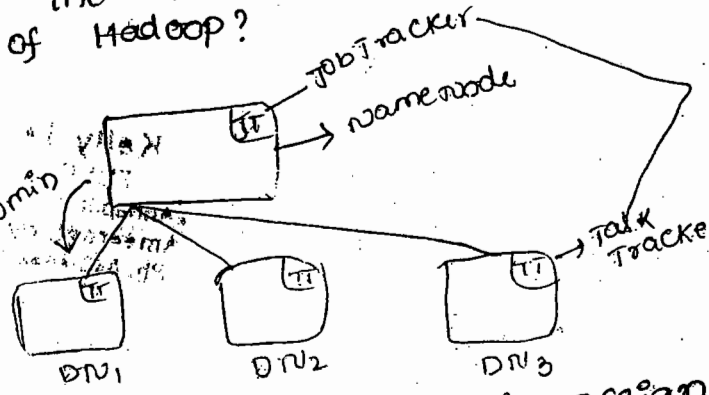
• Through the checkpoint mechanism only Hadoop cluster will make sure that all the metadata information of Hadoop file system will get updated in the two persistance files

* fsnameSpaceImage
* editLog.

• The checkpoint mechanism can be configured either an hourly, weekly, daily.

The checkpoint node periodically creates checkpoints of the namespace.

> what is the Heart Beat mechanism & Speculative execution of Hadoop?



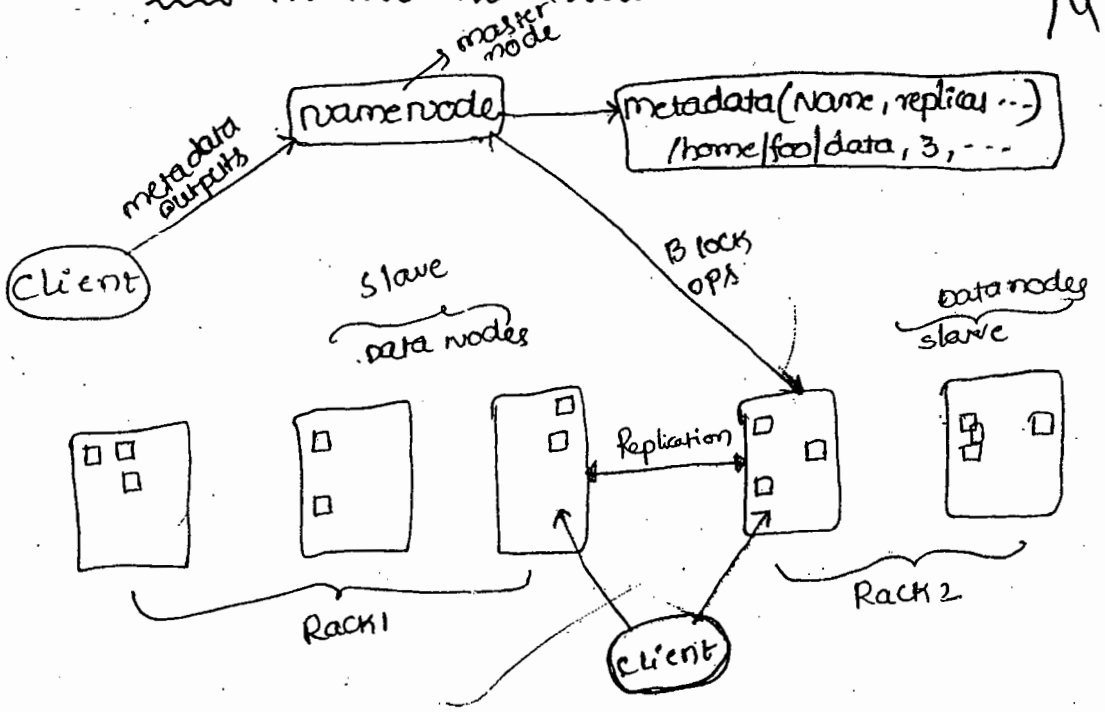
when a particular JobTracker is assign the NameNode will excepts a kind of acknowledgement from the Datanodes [TaskTracker] and regular interval bases. This is known as Heart Beat mechanism

if a Datanode fails to response within 10 min time the namenode will mark that node either it is dead, running slow (or) not functionality slow.

Immediately the JobTracker will assign the same task to some other Idlenodes with hadoop cluster. This is known as Speculative execution of Hadoop.

By the means of Speculative execution of Hadoop only the end user will not feel any delay will getting the response back from Hadoop processing

Data Storage in HDFS



- HDFS Architecture is a master-slave architecture.
- In HDFS cluster consists of master node is a NameNode
- NameNode manages the file system name space regulates access to by clients.
- In HDFS, file System NameSpace allows user data to be stored in files. Internally file is split into Blocks. Blocks are stored into set of Datanodes.
- NameNode executes the file system namespace operations like opening, closing & renaming files & directories. It also determines the mapping of blocks to Datanodes.
- The Datanodes are responsible for serving read & write requests from the file System clients.
- The Datanodes also perform block creation, deletion and replication upon instruction from the NameNode.

Introduction about blocks :-

- HDFS is designed to support very large datasets.
- HDFS supports write once/read many times semantics on files.
- In HDFS Data is split into blocks and distributed across multiple Data nodes in the cluster.
- Each block is typically 64MB (or) 128 MB in size.
- Each block is replicated multiple times. By default replication factor of block is 3 times. Replicas are stored on different data nodes.
- HDFS utilizes the local file system to store each HDFS block as a separate file.
- HDFS blocksize can not be compared with the traditional file system block size.

Data Replication :-

- An application can specify the no. of replicas of a file.
- The replication factor can be specified at file creation time and can be changed later.
- The replication factor are configurable per hdfs-site.xml file.
- Default replication factor of file - 3

no. of copies of a file = Replication factor of that file

hdfs-site.xml

```
<name> dfs.replication </name>
<value> 3 </value>
```

variable specifies how many times each HDFS block should be replicated.

- The placement of the replicas is critical to HDFS reliability and performance.
- optimizing replica placement distinguishes HDFS from other distributed file systems.
- Rack-aware replica placement
  goal : improve reliability, availability and N/w bandwidth utilization.
- Research topic.
- many racks, communication between racks are through switches.
- Network bandwidth between machines on the same rack is greater than those in different racks.
- NameNode determines the rackid for each datanode.
- Replicas are typically placed on unique racks
  - Simple but non-optimal
  - writes are expensive
  - Replication factor is 3
  - Another research topic ?
- Replicas are placed: one on a node in a local rack, one on a different node in the local rac one on a node in a different rack.
- 1/3 of the replica on a node, 2/3 on a rack and 1/3 distributed evenly across remaining racks.

Replica selection :-

- Replica selection for READ operation: HDFS tries to minimi the bandwidth consumption and latency.
- if there is a replica on the Reader node then that is prefered.
- HDFS cluster may span multiple data centers : replica in the local data center is prefered over the remote one.

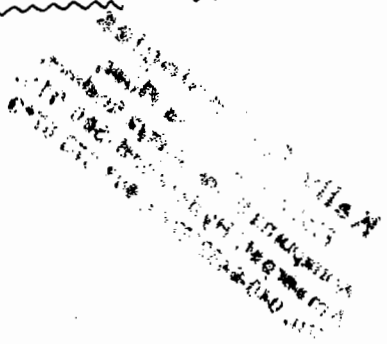
→ HDFS by interacting with the interfaces have two best approaches. They are

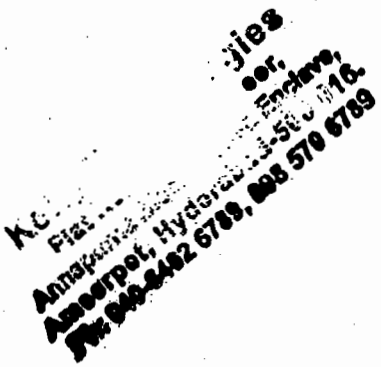
1. Commandline Interface.
2. JavaBased Approach.

1) Command line Interface :-

- Commandline Interface is one of the simplest and many developers, the most familiar.
- Commandline Interface is the interactive Shell.

2) Java Based Approach ;-





The persistance of file System metadata :-

- In the HDFS, namenode is responsible for the metadata. metadata have file System namespace & edit log.
- namenode will only update these files.
  1. fsImage
  2. Edit log.

fsImage :-

- the fsimage file is a persistent checkpoint of the file System metadata.
- However, it is not updated for every file System write operation, Since writing out the fsImage file, which can grow to be gigabytes in size, would be very slow.
- This does not compromise resilience.

when Namenode fails :-

- if the namenode fails, then the latest State of it's metadat can be reconstructed by loading the fsImage from disk into memory, then applying each of the operations in the edit log.
- In fact, this is precisely what the namenode does when it starts up [Learn about Safe mode]

edit Log :-

- when a client, file System client performs a write operation (Such as creating or moving a file) It is first recorded in Edit log.
- The namenode also has in memory representation of the file System metadata, which it updates after the editlog has been modified.

: Edit log to permanently record every change that occurs to file System metadata.

When a client request a data :-

- The in-memory metadata is used to serve read requests.
- The editlog is flushed and synced after every write before a success code is returned to the client.
- For Namenodes that write to multiple directories, the write must be flushed and synced to every copy before returning successfully.
- This ensures that no operation is lost due to machine failure.

Ex:- * Creating the new file in HDFS.

* Name Node to insert a record into the editlog.

* Similarly, changing the replication factor of a file causes a new record to be inserted into the Editlog.

* The namenode uses a file in it's localhost, OS fileSystem to store the editlog.

Note:

- HDFS Namespace is stored by the Namenode. The namenode uses a transaction log called the editlog.
- HDFS Namespace including the mapping of blocks to files and file System properties is stored in a file called FSImage.
- FSImage is stored as file in the namenodes local file System too.
- The NameNode keeps an image of the entire file System namespace and file Block map in memory.

Safemode :- whenever cluster is starting up in Hadoop certain things will done by the NameNode.

Some thing

1. loading all system related configuration files.
2. check for the satisfactory replication for the data.
3. All system related dependent files.

- while doing these above all operations. The NameNode is HDFS puts on read-only-mode [HDFS can not be reached at that moment] this stage is known as safemode ON
- After doing all these stuff automatically safemode will come out of safemode ON → OFF which means that HDFS will be accessable mode.
- But sometimes safemode will not be turned into OFF mode. At that point of time Hadoop adminstra to run the below command.

By Mr. GOPAL KRISHNA

→ hadoop dfsadmin -safemode LEAVE ↵
safemode is in OFF

→ hadoop dfsadmin -safemode ENTER ↵
safemode is in ON.

→ hadoop dfsadmin -safemode get ↵
Safemode is OFF

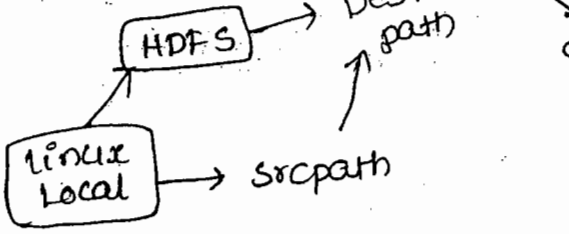
→ hadoop dfsadmin -report ↵

looks the GUI mode ↓
http://127.0.0.1:50070/ ↵

cluster health report command.

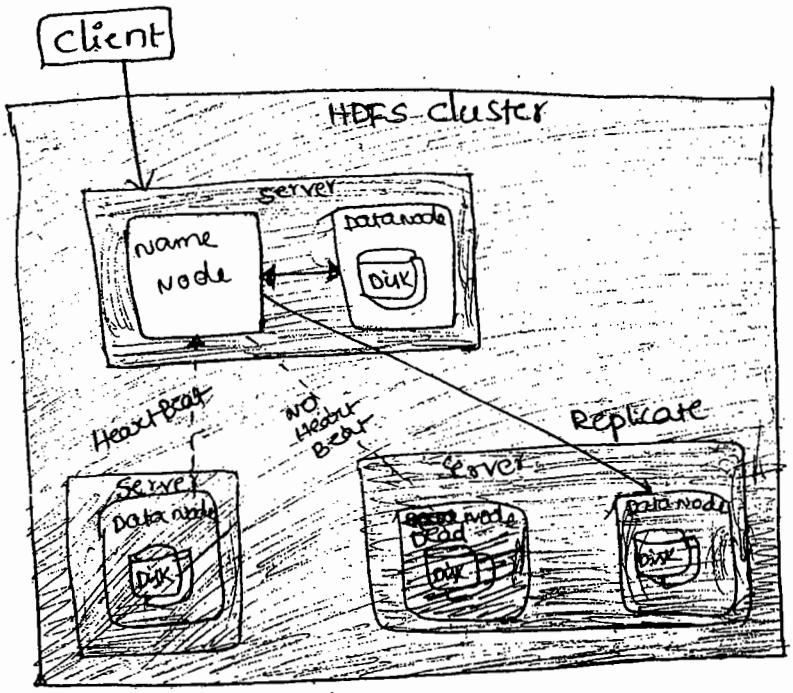
Hadoop file System.

- hadoop fs is indicating the compiler to interact with linux local environment to HDFS environment.
- hadoop fs is not support the -vi command. Because HDFS is writeonce.
- hadoop fs is support -touchz command.
- hadoop fs looks only HDFS directories & file but not for localdirectories & files.
- In HDFS file overloading is not possible.
- if the one of the file moving one environment to another environment there have only default previllages.
- we can not create file on top of HDFS.
- we can create the file on local directory.



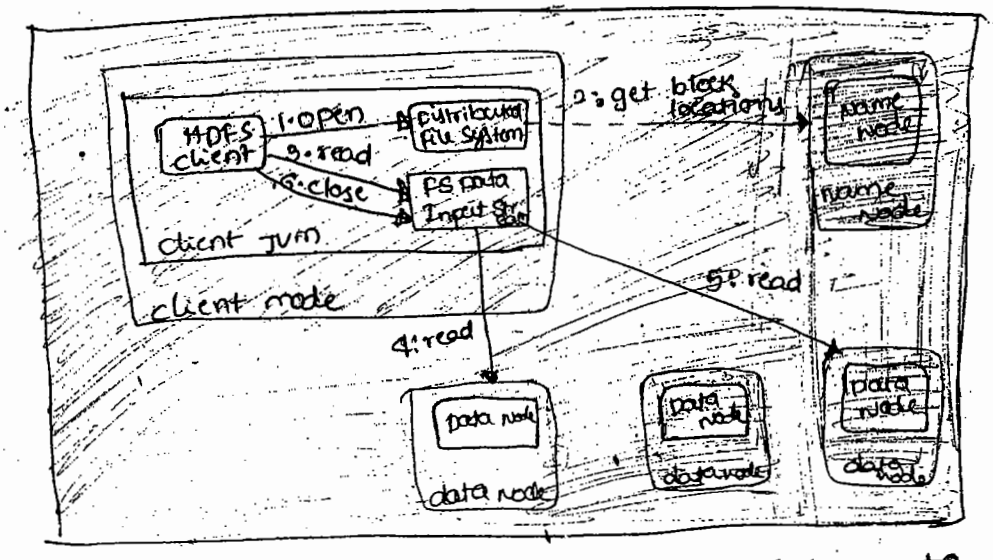
- hadoop fs does not support hardlinks or softlinks.
- hadoop fs does not implement user quotas.
- Error information is sent to stderr & output is sent to stdout.

HeartBeat mechanism of Hadoop cluster



HDFS Debugging Steps :-

Client Reading Data 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 DFS.
- DFS calls the namenode, using RPC, to determine the locations of the blocks for the first few blocks in the file. For each block, the namenode

returns the addresses of the datanodes that have a copy that block and the datanodes are stored according to their proximity to the client.

- The DFS returns a FSDataInputStream (an input stream that supports file seeks) to the client for it to read data from.

The client then calls read() on the stream.

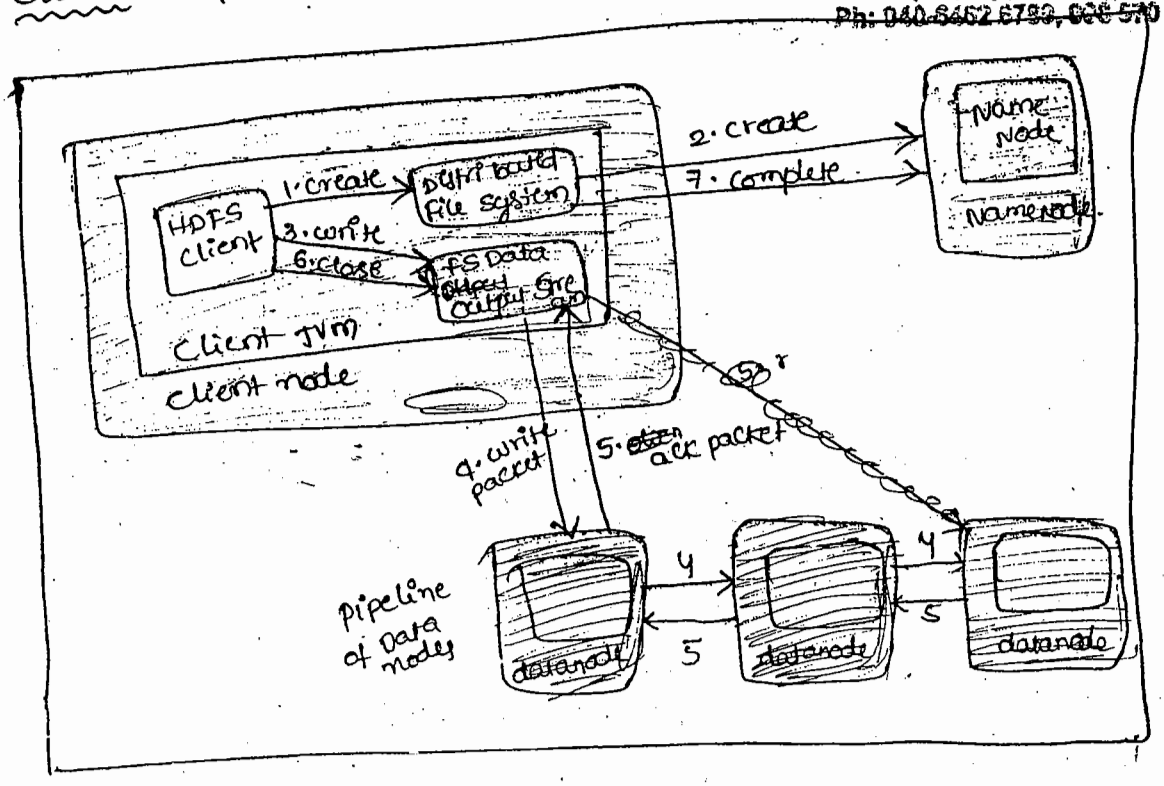
- DFSStream connects to the first (closest) datanode for the first block in the file.

Data is streamed from the datanode back to the client, which calls read() repeatedly on the stream.

when the end of the block is reached, DFSInputStream will close the connection to the datanode, then find the best datanode for the next block.

when the client has finished reading, it calls close() on the FSDataInputStream.

) Client writing Data to HDFS



- The client creates the file by calling create() on DFS.
- DFS makes an RPC call to the namenode to create a new file in the filesystem's namespace, with no blocks associated with it.
- The namenode performs various checks to make sure the file doesn't already exist, and then the DFS returns a FSDataOutputStream for the client to start writing data to.
- As the client writes data, DFS output Stream splits it into packets, which it writes to an internal queue, called the data queue.
- The Data queue is consumed by the DataStreamer whose responsibility it is to ask the namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas.

The list of datanodes forms a pipeline.

- The DataStreamer streams the packets to the first datanode in the pipeline, which stores the packet and forwards it to the second datanode in the pipeline. Similarly, the second datanode stores the packet and forwards it to the third (and last) data node in the pipeline.

DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowledged by datanodes, called the ack queue.

A packet is removed from the ackqueue only when it has been acknowledged by all the datanodes in the pipeline.

what if a datanode fails while it is being written to it.

3) The pipeline is closed.

b) Any packets in the ackqueue are added to the front of the data queue so that datanodes that are downstream from the failed node will not miss any packets.

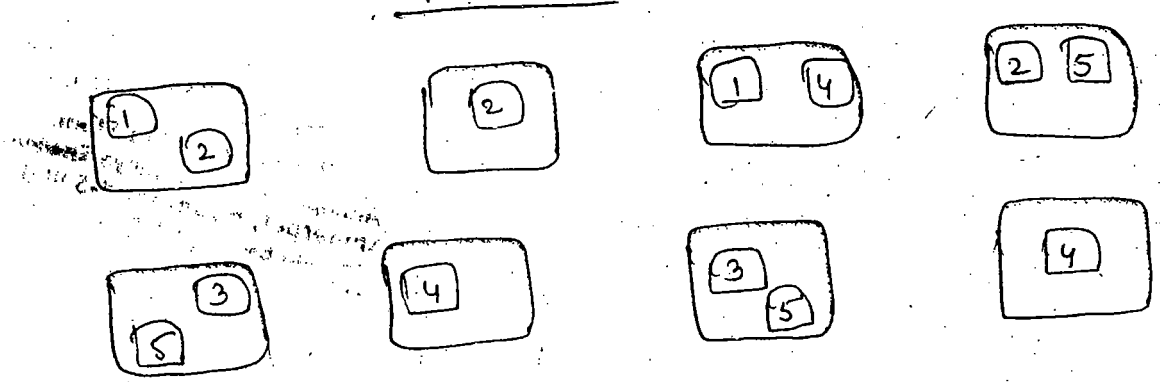
) The partial block on the failed datanode will be deleted if the failed datanode recovers later on the failed data node is removed from the pipeline and the remainder of the block's data is written to the two good datanodes in the pipeline.

) The namenode notices that the block is under-replicated, and it arranges for a further replica to be created on another node.

eplica placement

Block Replication

namenode (Filename, numReplicas, block-id ---
/users/root/data/part-0, r:2, {1,3}, ----
/users/root/data/part-1, r:3, {2,4,5} ----



• The placement of replicas is critical to HDFS reliability and performance.

• The purpose of rack-aware replica placement policy is to improve 3 different ways. They are

① Readability
② availability
③ N/w bandwidth utilization.

• communication between two nodes in different racks has to go through switches. In most cases, network bandwidth between machines in the same rack is greater than bandwidth between machines in different racks.

• HDFS placement policy is to put one replica on one node in the local rack, another on a node in different rack, and last on a different node in the remote rack.

Replica Section :-

• TO minimize global bandwidth consumption and read latency.

• HDFS tries to satisfy a read request from a replica that is closes to the reader.

• if there exists a replica on the same rack as the reader node, then that replica is preferred to satisfy the read request.

• if angg/HDFS cluster spans multiple data centres, then a replica that is resident in the local data center is preferred over any remote replica.

Replication pipelining :-

- Data is pipelined from Datanode to next Datanode.

For example :-

- when a client is writing data to an HDFS file. It's data is first written to a localfile.
- Suppose HDFS file has a replication factor is 3.
- The client retrieves a list of datanodes from the Namenode.
- The list of datanodes will host a replica of that block.
- The client then flushes the data blocks to the first Datanode.
- The first Datanode starts receiving the data in small portions (4 KB) writes each portion to it's local repository and transfers the portion to the second datanode in the list.
- The second Datanode starts receiving each portion of the data block, writes each portion to it's the repository and then flushes that portion to the 3rd datanode.
- Finally 3rd Datanode writes the data to it's the local repository.

Thus a Datanode can be receiving data from the previous one in the pipline and at the sametime forwarding data to the next one in the pipeline.

Communication protocols :-

- All HDFS communication protocols are layered on top of the TCP/IP protocol.
- A client establishes a connection to a configurable TCP port on the NameNode machine. It talks the client protocal with the NameNode.
- The Datanodes talk to the NameNode using the DataNode protocol.

- A Remote Procedure call (RPC) abstraction wraps both the client protocol and the Datanode protocol.
- By design, NameNode never initiates any RPCs. Instead, it only responds to RPC requests issued by Datanodes (or) clients.

Robustness :-

- The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are
  1. NameNode failures
  2. DataNode failures
  3. N/w partitions.

Data Disk failure, Heartbeats & Re-Replication :-

- Each Datanode sends a Heart-beat message to the namenode periodically.
- A N/w partition can cause a subset of Datanodes to lose connectivity with the NameNode.
- The NameNode detects this condition by the absense of a Heartbeat message.
- The NameNode marks Datanodes without recent Heartbeats as dead datanode is not available to HDFS any more.
- Datanode death may cause the replication factor of some blocks to fall below their specified value.
- The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary.
- The necessity for re-replication may arise due to many reasons. (1) Datanode may become unavailable, a replica may become corrupted, a harddisk on a datanode may fail, or the replication factor of a file may be increased.

Cluster Rebalancing :-

- The HDFS architecture is compatible with data rebalancing schemes.
- A Scheme might automatically move data from one data node to another Datanode.
- if the freespace on a Datanode falls below a certain thereshold.
- if In the event of Sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster. These types of data rebalancing schemes are not yet implemented.

Data Integrity :-

- Data integrity is possible that a block of data fetched from a datanode arrives corrupted. This corruption can occur because of faults in storage device, N/w faults, or buggy S/w.
- The HDFS client S/w implements checksum checking on the contents of HDFS files.
- when a client creates an HDFS file, it computes a checksum of each block of the file and stores these checksums in a separate hidden file in the same HDFS namespace.
- when a client retrieves file contents it verifies that the data it received from each datanode matches the checksum stored in the associated checksum file.
- if not, then the client can opt to retrieve that block from another Datanode that has a replica of that block.

metadata disk failure :-

- FSImage & Edit log are central data structures of HDFS. A corruption of these files can cause the HDFS instance to be non-functional.
- For this reason, the NameNode can be configured to support maintaing multiple copies of FSImage & Editlog.
- The NameNode will update too important persistance files of HDFS.

① FSImage ② Editlog.

to get updated synchronously.

- Synchronous updating of multiple copies of the FSImage & Editlog may degrade the rate of namespace transactions per second that a NameNode can support.
- However, this degradation is acceptable because even though HDFS applications are very data intensive in nature, they are not metadata intensive.
- when a NameNode restarts, it selects the latest consistent FSImage & Editlog to use.
- The NameNode machine is a single point of failure for an HDFS cluster.
- if the NameNode machine fails, manual intervention is necessary. Currently automatic restart and failover of the NameNode software to another machine is not supported.

Snapshots :-

- Snapshots support storing a copy of data at a particular instant of time.
- one usage of the snapshot feature may be to roll back a corrupted HDFS instance to a previously known good point in time.
- HDFS does not currently support snapshots but will in a future release.

Data Organization

① Data Blocks :-

- HDFS is designed to support very large files.
- HDFS supports write once & read many times Semantics on files.
- A typical blocksize used by HDFS is 64mB. Thus an HDFS file is chopped up into 64mB chunks, and if possible, each chunk will reside on a different DataNode.

② Staging :-

- A client request to create file does not reach the NameNode immediately.
- In fact, Initially the HDFS client caches the file data into a temporary localfile.
- Application writers are transparently redirected to this temporary local file.
- when the localfile accumulates data worth over one HDFS blocksize, the client contacts the NameNode
- The NameNode inserts the file name into the file System hierarchy and allocates a data block for it.

- The NameNode responds to the client request with the identify of the Datanode and the destination data block.
- Then the client flushes the block of data from the local temporary file to the specified Datanode.
- when a file is closed, the remaining un-flushed data in the temporary localfile is transferred to the Datanode.
- The client then tells the NameNode that the file is closed. At this point, the NameNode commits the file creation operation into a persistent store
- if the NameNode dies before the file is closed, the file is lost.

The above approach has been adopted after careful consideration of target applications that run on HDFS.

- These applications need streaming writes to files. if a client writes to a remote file directly without any client side buffering, the N/w speed and the congestion in the N/w impacts throughput considerbly.
- This approach is not without precedent. Earlier DFS's [ex:- AFS have used client side coaching to improve performance]

[POSIX → requirement has been relaxed to achieve higher performance of data uploads].

Accessibility

- HDFS can be accessed from applications in many different ways.
- HDFS provides a JAVA API for applications to use.
- HTTP Browser can also be used to browse the files of an HDFS instance.
- coork is in progress to expose HDFS through the WebDAV protocol.

FS Shell :-

- HDFS allows user data to be organized in the form of files and directories. It provides a command line Interface called FS Shell; user interact with the data in HDFS.
- The Syntax of this command set is similar to other shells (e.g. bash, csh) that users are already familiar with.
- FS shell is targeted for applications need a scripting language to interact with the stored data.

DFS Admin :-

- The DFS Admin command set is used for administering an HDFS cluster. These are commands that are used only by an HDFS Administrator.

Browser Interface :-

- A typical HDFS install configures a webserver to expose the HDFS namespace through a configurable TCP port.
- This allows a user to navigate the HDFS namespace and view the contents of its files using a web browser.

Space Reclamation [File deletes & undeletes] :-

- when a file is deleted by a user or an application it is not immediately removed from HDFS.
- HDFS first renames it to a file in the /Trash directory.
- The file can be restored quickly as long as it remains in /trash.
- A file remains in /trash for a configurable amount of time. After the expiry of its life in /trash, the namenode deletes the file from the HDFS namespace
- the deletion file causes the blocks associated with the file to be freed.

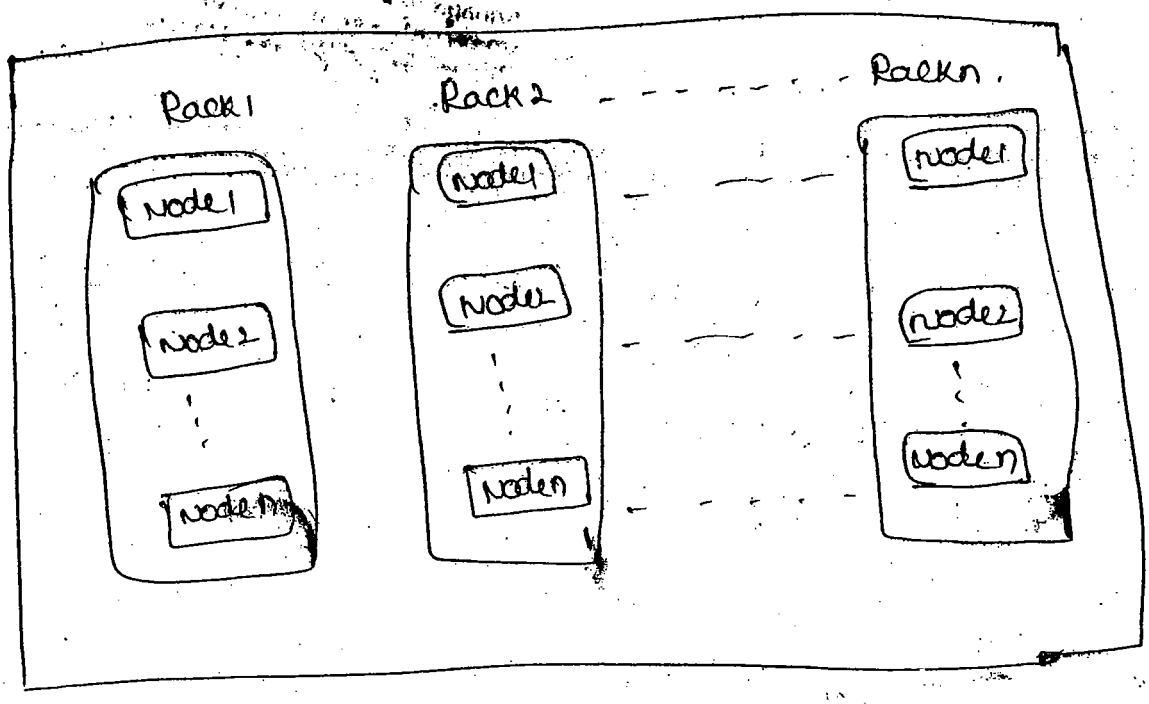
[Note:- There could be an appreciable time delay b/w the time a file is deleted by a user & the time of the corresponding increase in free space in HDFS.]

- A user can undelete a file after deleting it as long as it remains in the /trash directory. if a user wants to undelete a file that he/she has deleted, he/she can navigate the /trash directory and retrieve the file The /trash directory contains only the latest copy of the file that was deleted.
- The /trash directory is just like any other directory with one special feature: HDFS applies specified policies to automatically deleted files from this directory.
- The current default policy is to delete files from /trash that are more than 6 hours old.
- In the future, this policy will be configurable through a well defined Interface.

Decrease Replication factor :-

- When the replication factor of files is reduced, the NameNode selects excess replicas that can be deleted.
- The next Heartbeat transfers this information to the Datanode.
- The Datanode then removes the corresponding blocks and the corresponding free space appears in the cluster.
- Once again, there might be time delay between the completion of the Set Replication API call and the appearance of free space in the cluster.

Hadoop cluster :- (Set of machines)



Linux Commands

File/Directory Basics

① mkdir → making the directory
Ex:- mkdir anitha ↵

② cd → change the directory.
Ex:- cd anitha ↵
/anitha #

③ clear → clear the commands.

④ pwd → present working directory.

⑤ date → display the date.

⑥ who am i → display the user name.

⑦ rm → Delete files.

⑧ ls → List files

⑨ cp → copy files
Ex:- cp sample.txt sample1.txt
(sample.txt → src path; sample1.txt → destination path.)
(or) → cp -r dir1 dir2
→ cp -r dir1/*.* dir2 (or)
→ cp -r dir1/*.txt dir2

⑩ mv → renaming the files
Ex:- mv sample.txt input.txt ↵
(sample.txt → src path; input.txt → renaming path.)

⑪ ln → link files
Ex:- ln sample.txt ↵
ln: creating hard link ./sample.txt

⑫ rmdir → remove the directory
Ex:- rmdir anitha ↵
delete the directory.

File viewing

1) Cat → View files

Ex: cat input.txt ↵
This is a Hadoop Note Book.

2) less → Page through files.

Ex: less input.txt ↵
This is a Hadoop Note Book.
input.txt (END)

3) head → View file begining [1st 10 records displayed]

Ex: head input.txt.

4) Tail → View file ending last 10 records display.

Ex: tail input.txt.

5) nl → number lines

nl input.txt ↵

6) od → View binary data.

od input.txt ↵

7) xxd → View hexa binary data.

xxd input.txt

8) gv → View Post Script / PDF files.

9) xdvi → View Tex dvi files.

File Creation & Editing

1) emacs → Text editor
2) pico → Text editor
3) vi → Text editor
4) umask → Set default file protections.
5) Soffice → Edit / word / Excel / power point docs.
6) abiword → Edit word documents.
7) gnumeric → Edit excel documents.

File properties

① Stat → Display file attributes.

Ex: Stat input.txt ↵

File: - --- Device: --
Size: --- - accessu: ---

② WC → Count bytes/words/lines

Ex: WC input.txt ↵

1 1 18

③ du → measure disk usage.

Ex: du ↵

20.

④ file → Identity file types.

Ex: file input.txt ↵

⑤ touch → change file stamps.

Ex: touch input.txt.

⑥ chown → change file owner.

Ex: chown 777 input.txt

⑦ chgrp → change file group.

Ex: chgrp 43 input.txt

ll ↵

rw-r--r-- 1 777 utmp 18

⑧ chmod → change file protections.

Ex: chmod 777 input.txt

ll

rwx rwx rwx - - -

⑨ chattr → change advanced file attributes.

⑩ lsattr → list advanced file attributes.

File location

① find → locate files.

Ex:- find ↵
./sample1.txt
./input1.txt.

② slocate → locate files via index.

③ which → locate commands

④ whereis → locate standard files.

Ex: whereis <srcpath>

File Text manipulation

① grep → search text for matching lines.

grep hadoop sample.txt

② cut → Extract columns

③ paste → append columns

④ tr → translate characters

⑤ sort → sort lines

⑥ uniq → locate identical lines

⑦ tee → copy stdin to a file and to stdout simultaneously.

File compression

① gzip → compress files (GNU zip)

② compress → compress files (unix)

③ bzip2 → compress files (Bzip2)

④ zip → compress files (windows zip)

File Comparison

① diff → compare files line by line.
diff input.txt sample.txt ↵

② comm → compare sorted files.
comm input.txt sample.txt ↵

③ cmp → compare files byte by byte.
cmp input.txt sample.txt ↵

④ md5sum → Compute check sums.

Disks and File Systems

df → show free disk space
mount → make a disk accessible
fsck → check a disk for errors
sync → Flush disk caches.

printing

lpr → print files
lpq → view print queue
lprm → remove print jobs.

Spelling operations

look → lookup spelling
aspell → check spelling interactively
spell → check spelling in Batch.

Backups and Remote Storage

mt → control a tape device
dump → Backup a disk
restore → Restore a dump
tar → Read/write tape archives.
cd record → Burn a CD.
rsync → mirror a set of files.

Audio & Video

grip → play CDs & ripmp3s
xmms → play Audio files.

Processes

1. ps → List all processes
2. w → list user's processes
3. uptime → view the system load
4. top → monitor processes
5. xload → monitor system load
6. free → Display free memory
7. Kill → Terminate process
8. nice → set process priorities
9. renice → change process priorities

Scheduling Jobs

1. Sleep → wait for some time
2. watch → Run programs at set intervals
3. at → schedule a job.
4. crontab → schedule repeated jobs.

Hosts

1. uname → print system information
2. hostname → print the system's hostname
3. ifconfig → Set/display N/w information
4. host → lookup DNS
5. whois → lookup Domain registrants
6. ping → check if host is reachable
7. traceroute → view N/w path to host.

Networking

1. SSh → securely log into remote hosts
2. telnet → log into remote hosts.
3. scp → securely copy files between hosts.
4. sftp → " " "
5. ftp → copy files b/w hosts.
6. evolution → GUI email client
7. mutt → Textbased email client
8. mail → minimal email client
9. mozilla → webbrowser
10. lynx → Text only web-browser
11. wget → Retrieve webpages to disk.
12. Srln → Read usenet news.
13. gaim → Instant messaging/IRC
14. talk → Linux/unix chat.
15. write → send msgs to terminal
16. mesg → prohibit talk/write.

HDFS Blocks :-

- Blocks are traditionally either 64MB or 128 MB.
  - Default is 64 MB.
- The motivation is to minimize the cost of seeks as compared to transfer rate.
  - Time to transfer' > Time to seek.
- For example, let say
  - seektime = 10ms
  - Transfer rate = 100 mb/s.
- To achieve seek time of 1% transfer rate.
  - Blocksize will need to be = 100MB

→ what is the difference between FS shell & DFS shell?

File System
FS Shell

① FS relates to a generic file system which can point to any file systems like local, HDFS -- etc.

② FS shell is invoked by bin/hadoop fs. All the FS shell commands take path URI as arguments.

③ The URI format is scheme://authority/path. The scheme & authority are optional. if not specified, the default scheme specified in the configuration is used.

④ For the HDFS the scheme is hdfs, and for local file system the scheme is file.

⑤ An HDFS file or directory such as /parent/child can be specified as hdfs://namenodehost/parent/child or simply as (/parent/child)

⑥ most of the commands in FS shell behave like corresponding UNIX commands

Distributed file system
DFS Shell

① DFS is very specific to HDFS.

② The HDFS shell is invoked by bin/hadoop dfs. All the HDFS shell commands take path URI's as arguments.

③ The URI format is scheme://authority/path. The scheme & authority are optional. if not specified, the default scheme specified in the configuration is used.

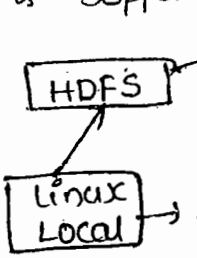
④ For the HDFS the scheme is hdfs and for local file system the scheme is file.

⑤ An HDFS file or directory such as /parent/child can be specified as hdfs://namenode:namenodeport/parent/child. (or) simply as /parent/child

⑥ most of the commands in HDFS shell behave like corresponding UNIX commands.

Hadoop file System (hadoop fs) :-

- hadoop fs indicating the compiler to interact with Linux Local environment to HDFS environment.
- hadoop fs is not support the -vi command. Because HDFS is write once.
- hadoop fs is support for -touchz command.



- hadoop fs looks the only HDFS directory but not Local directory.
- we can not create a file on top of HDFS.
- we can create the file on local.
- we can not update a file on top of HDFS. we can updations in local, after that file is put into the HDFS.

- Hadoop fs is not does not support hardlinks (or) softlinks.
- hadoop fs does not implement user quotas.
- Error information is sent to stderr & output is sent to stdout.

Display detailed help for a command.

hadoop fs -help <command-name>

HDFS Shell Commands

:S shell :-

1. cat → content display (or) View the content
2. chgrp
3. chmod
4. chown
5. copyfromlocal
6. copyToLocal
7. cp
8. du
9. dus
10. expunge
11. get
12. getmerge
13. ls
14. lsr
15. mkdir
16. movefromLocal
17. mv
18. put
19. rm
20. rmr
21. setrep
22. stat
23. tail
24. test
25. text
26. touchz
27. count

user commands

1. archive
2. distcp
3. fs
4. fsck
5. fetchdt
6. Jar
7. Job
8. pipes
9. queue
10. version
11. CLASS NAME
12. classpath

Administration Commands

1. balancer
2. daemon log
3. datanode
4. dfsadmin
5. Jobtracker
6. namenode
7. secondary namenode
8. taskTracker
9. safemode

Shell commands:

① mkdir → make the directory.

Ex:- hadoop fs -mkdir /anitha ↵ → It is just created on HDFS.

hadoop fs -mkdir anitha ↵ → It is created on default hdfs path [/user/root]

② help → display help for all commands

Ex:- hadoop fs -help ↵

③ ls → list the file display stats, for a directory displays immediate children.

Ex:- hadoop fs -ls ↵ if path is not specified.

hadoop fs -ls <directory name> <path>

④ lsr → Recursively list the contents that match the specified file pattern.

Ex:- hadoop fs -lsr <path>

⑤ du → show the amount of space, if we are using the files in HDFS

Ex:- hadoop fs -du <path> <file/dir>

note:- Add -h option to display in human-readable format instead of bytes.

Ex:- hadoop fs -du -h /somedir ↵

206.3K /somedir

=

⑥ du → show the amount of space, in bytes, used by the files that match the specified file pattern.

Ex: hadoop fs -du

hadoop fs -du <path>
↓ file (or) directory.

⑦ df → show the capacity, free and used space of the file system.

note: if the file system has multiple partitions, and no path to a particular partition is specified then the status of the partitions will be shown.

hadoop fs -df <path>

⑧ cp: copy files that match pattern <src> to <destination>

note: when copying multiple files, the <dest> must be a directory.

Ex: hadoop fs -cp anitha/input.txt /anitha
<src path> <dest path>
must be HDFS paths.

⑨ mv → move files that match the specified file pattern <src> to <dest>

note:- when moving multiple files, the destination must be a directory.

hadoop fs -mv <srcpath> <dest path>
must be HDFS paths.

(10) rm → Delete the files and particular non-empty directories

Ex: hadoop fs -rm <path>

(11) rmr → remove directories recursively.

Ex: hadoop fs -rmr <path>
↳ directory.

(12) Count → Count the no. of directories, files and bytes under the paths that match the specified file pattern.

Note: The output columns are Dir-Count, File-Count, Content-Size, File-Name.

Ex: hadoop fs -Count <path>
↳ directory.

Note: The output columns with -q are QUOTA, QUOTA, REMAINING_QUOTA, SPACE_QUOTA, REMAINING_SPACE_QU
DIR_COUNT, FILE_COUNT, CONTENT_SIZE, FILE_NAME

Ex: hadoop fs -count -q <path>
↳ directory.

(13) put → copy single src (or) multiple srcs from local file syste to the destination file system. Also reads input from stdin and writes to destination file system.

Ex: hadoop fs -put <src path> <dest path>
(or) ↓ ↓
copyFromLocal local file HDFS dir

multiple files from Local to HDFS

Ex: hadoop fs -put *.* <dest path>
↓ HDFS path.

only txt files from Local to HDFS

Ex: hadoop fs -put *.txt <dest path>
↓ ↓
local path. HDFS path

(14) **get → copy files to Local file System**

syn: hadoop fs -get <src> <dest>
(or) copyToLocal
<src> → HDFS path
<dest> → local path

multiple files from HDFS to Local

syn: hadoop fs -get /anitha/** [<dest path>
<src path>
local path (or) . → current directory.

(15) **expunge → empty the trash**

syn: hadoop fs -expunge ↵

(16) **moveFromLocal → move (copy) single src (or) multiple src's from localfile system to destination file system**

syn: hadoop fs -moveFromLocal <localsrc> <dest>

(17) **getmerge → takes a source directory and destination file as input and concatenates file in src into the destination local file.**

syn: hadoop fs -getmerge <Src path> <local destination path>
<Src path> → HDFS path
<local destination path> → local path.

note:- addnl can be set to enable adding a newline character at the end of each file.

syn: hadoop fs -getmerge <srcpath> (or) <localpath> <addnl>
<srcpath> → compulsory Hadoop directory
<localpath> → local directory.

(18) text → Takes a source file and outputs the file in text format.
The allowed formats are zip & TextRead Input Stream.

syn:- hadoop fs -text <src path>

37

(19) touchz → Creates a file of zero length (or) size.

syn:- hadoop fs -touchz <path>

(20) test →
-e check to see if the file exists.
returns 0 if true.
-z check to see if the file is zero length
returns 0 if true.
-d check return 1 if path is directory
else return 0.

syn:- hadoop fs -test -ezd <pathname>
hadoop fs -test -e <path>
hadoop fs -test -z <path>
hadoop fs -test -d <path>

(21) Stat → Returns the stat information on the path.

syn:- hadoop fs -stat <pathname> (local & HDFS.)

(22) Tail → Displays last Kilobyte of the file to stdout
-f option can be used as in unix.

syn:- hadoop fs -tail <pathname>
↓ HDFS path.

(23) Setrep → changes the replication factor of a file.
-R option is for recursively increasing the replication factor of files within a directory.

syn:- hadoop fs setrep [-w 3 -R] /user/root/dir
↓ -R
↓ <HDFS pathname>

(24) chgrp → change group association of files. with -R make the change recursively through the directory structure.

The user must be the owner of files (or) else a super-user.

Syn:- hadoop fs -chgrp -R (Group URI) ↵

hadoop fs -chgrp -R group anitha ↵ (or) hadoop fs -chgrp group filename ↵

anitha → directory name

(25) chmod → change the permissions of files. with -R make the change recursively through the directory structure. The user must be the owner of files or else a super-user.

hadoop fs -chmod 744 -R dirname

Permissions

Syn:- u - user who owns the file
g - group that owns the file
o - others
a - all
r - read the file, w - write or edit the file
x - execute or run the file as a program

Numeric permissions:

Chmod can also be attributed by using Numeric permissions.

400 → read by owner
040 → read by group
004 → read by anybody (other)
200 → write by owner
020 → write by group
002 → write by anybody
100 → execute by owner
010 → execute by group
001 → execute by anybody.

Ex:- -rwx-rwx-r-x 3 hope 123 Feb 03 15:36 file.txt

file → owner, group, every one, owner, size, mod date, file name

Ex:- hadoop fs -chmod 777 /tmp ↵ path

(26) chown → change the owner of files. with -R make the change recursively through the directory structure. The user must be the owner of the file (or) else a super-user.

Ex:- hadoop fs -chown hope file.txt ↵ file name.

Give permissions to owner to user hope for the file file.txt.

chown -R hope hdfs dir name

chown -R hope work.

Given chown permissions to hope for all files in the work directory.

User Commands

① archive :-

Hadoop stores the small files inefficiently such as each file get stored in a block & Namenode has to keep the metadata information in memory. so with this reason most of the Namenode memory will get eat up by this small files only which results in a wastage of memory.

To avoid the same problem we use hadoop archives (or) har files (.har is the extension for all the archive files).

When creating archive directory the input is converted to mapreduce jobs, so we can call hadoop archives as a input for our mapreduce programming.

* Hadoop archives are special format archives.
- Hadoop archive maps to a file system directory.
- Hadoop archive always has a *.har extension.
- Hadoop archive directory contains metadata [in the form of -index and -masterindex) and data (part-*) files

-index → contain the files that are part of archive and location with in the part files.

Syn:- Create the archive file :-

hadoop archive -archiveName Name -p <src> <dest>
↑ parent path.

E:- hadoop archive -archiveName foo.har -p /Anitha /myarchive

hadoop fs -ls /myarchive ↵
hadoop fs -lsr /myarchive/foo.har ↵
hadoop fs -cat /myarchive/foo.har/part-0 ↵

② Distcp :- Distributed copy.

- The distcp command is tool used for large inter and intra cluster copying.
- we have multiple hadoop clusters are running. we want to transfer several terabytes of data one cluster to another.

Ex:- hadoop clusters are loaded with terabytes of data.

- It will take forever to transfer terabytes of data from one cluster to another.
- Distributed (or) parallel copying of data can be good solution for this & that is what distcp does.
- Distcp runs mapreduce job to transfer your data from one cluster to another.

Syn:- `hadoop distcp <srcpath> <dest path>`

```
hadoop distcp hdfs://nn1:8020/foo/bar \
              hdfs://nn2:8020/bar/foo ↵
```

Kelly Technologies
Flat No. 212, 2nd Floor,
Annapurna Block, Aditya Enclave,
Ameerpet, Hyderabad-500 016.
Ph: 040-6462 6789, 966 570 6789

multiple src list :-

```
hadoop distcp hdfs://nn1:8020/foo/bar \
              hdfs://nn1:8020/foo/b \
              hdfs://nn2:8020/bar/foo ↵
```

From file using :-

```
hadoop distcp -f hdfs://nn1:8020/srclist \
                 hdfs://nn2:8020/bar/foo ↵
```

③ fs → Runs a generic file system user client

Syn :- `hadoop fs [Generic-options] [command-options]`

command options can be found at Hadoop shell guide.

④ jar → Runs a jar file. users can bundle their mapreduce code in a jar file and execute it using this command.

The Streaming jobs are run this command.

Hadoop streaming is a utility that comes with the hadoop distribution.

Syn:- hadoop jar <jar name>

Ex:- hadoop jar <wordcount.jar>

⑤ job → This command to interact with mapreduce jobs

hadoop job [Generic options]

⑥ pipes →

MapReduce

Introduction :-

- mapReduce published in 2004 by google.
- Hadoop can run mapreduce programs written in various languages like Java, Ruby, python and C++.
- mapReduce is a parallel programming model for processing the huge amount of data.
- mapReduce making the structured data and out of some unstructured data .. etc.
- mapReduce provides { automatic parallelization & distribution
  Fault-tolerance, I/O scheduling
  monitoring and status updates }

MapReduce Overview :-

- Applications processing data on Hadoop are written using the mapReduce paradigm.
- A 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. Typically both the input and the output of the job are stored in a file System. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.
- mapReduce applications specify the input/output locations and supply map and reduce functions via implementation of appropriate Hadoop interfaces such as mapper and Reducer. These, and other parameters comprise the Job configuration. The Hadoop Job client then submits the Job (jar/executable, etc) and configuration to the JobTracker, which then assumes the responsibility of distributing the software/configuration to the

slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

- The map/Reduce framework operates exclusively on <Key, value> pairs - that is, the framework views the input to the job as a set of <Key, value> pairs and produces a set of <Key, value> pairs as the output of the job, conceivably of different types.

what is mapReduce:

- map/Reduce
  - programming model from LISP (Ex: functional style programming)
  - and other functional language.
- many problems can be phrased this way.
- Easy to distribute across nodes
- Nice retry/failure Semantics.
- sort/merge based distributed computing.
- The underlaying system takes care of the partitioning of the input data, scheduling the program's execution across several machines, handing machine failures, and managing required inter-machine communication.
- Computational processing occurs on both
  - → unstructured data : file system
  - → structured data : database.
- Tried and tested in production
- many Implementation options.
- Fault-tolerant, reliable, and supports thousands of nodes and petabytes of data.

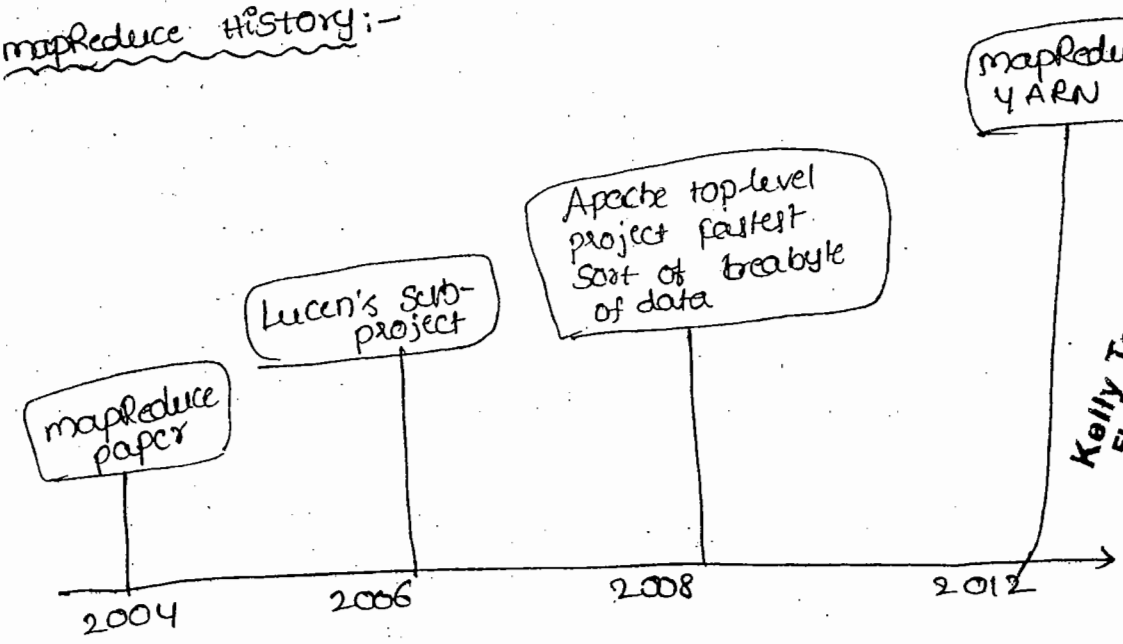
motivation for mapReduce (why):-

- large scale data processing.
- want to use 1000's of cpus.
- But don't want hassle of managing things.
- mapReduce architecture provides
  - automatic parallelization & distribution
  - Fault-tolerance
  - I/O scheduling
  - monitoring & status updates.

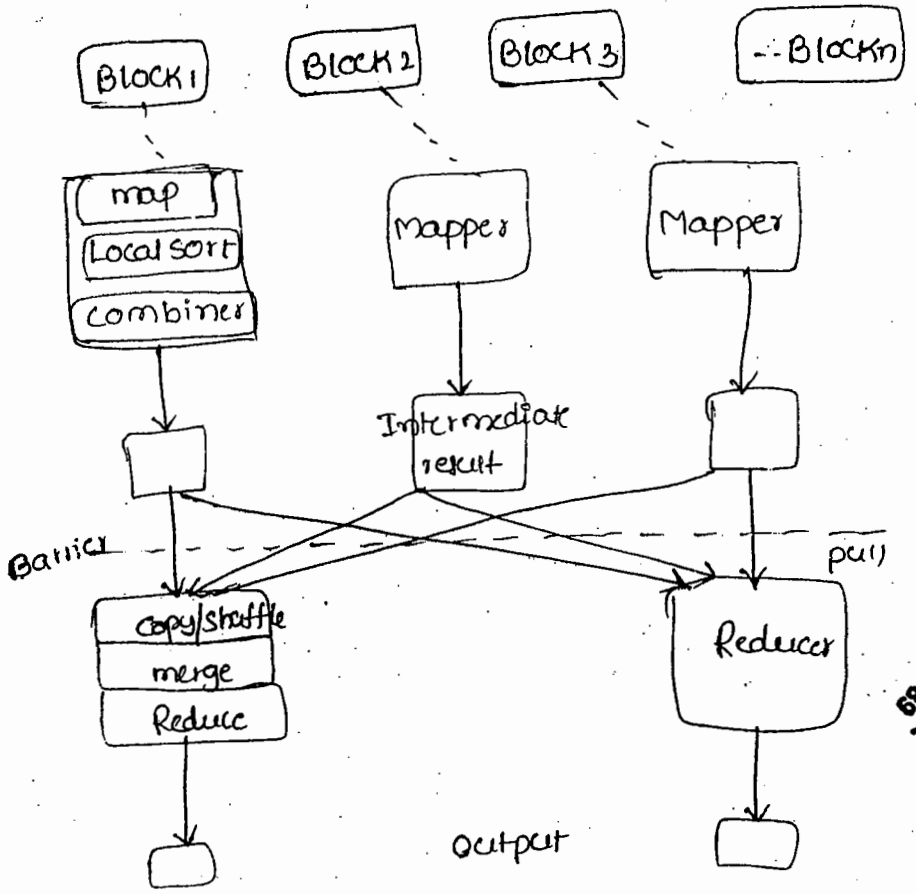
mapReduce limitations:-

- Can not control the order in which the maps or reductions are run.
- For maximum parallelism, you need maps and Reducers to not depend on data generated in the same mapReduce JOB (i.e Stateless).
- A database with an index will always be faster than a mapreduce JOB on unindexed data.
- Reduce operations do not take place untill all maps are complete (or have failed then been skipped).
- General assumption that the output of Reduce is smaller than the input to map; large data source used to generate smaller final values.

mapReduce history:-



mapReduce Architecture:



- mapReduce Architecture is a master-slave architecture.
- mapReduce is a programming model only. we can expect the parallel processing in Hadoop which means that if an input file coming into Hadoop file as part of processing of the same file.
- Input will get divided into multiple chunks and each & every chunk will be processed in different nodes.
- The whole process of mapReduce will be controlled by
  1. JobTracker
  2. TaskTracker.
- JobTracker will assign the tasks in the form of the mapReduce Jobs to all the available JobTrackers.
- TaskTrackers has to act up on the assigned tasks, and they have send the progress information to the JobTracker on Timely basis.

- Incase of failures (w.r.t tasks) → JobTracker will reassign the task to some other idle available TaskTrackers.

→ It consists of two phases: map and Reduce.

MapReduce: the mapper :-

- when the map function starts producing output, It is not simply written to disk. The process is more involved, and takes advantage of buffering writes in memory and doing some presorting for efficiency reasons.
- Each map task has a circular memory buffer that it writes the output to. The buffer is 100 MB by default, it can change based on the size. when the contents of the buffer reaches a certain thershold size 80% a background thread will start to spill the contents to disk during this time, the map will block until the spill is complete.

Before it writes to disk, data is partitions corresponding to the reducers. Each partition, Sort by key, and if there is a combiner function, it is run on the output of the sort. So there is less data to write to local disk and transfer to the Reducer.

The mapper reads data in the form of key/value pairs and it outputs zero or more key/value pairs.

mapReduce Flow → The Mapper

Each of the portions (Record Reader

mapper

partitioner

Reducer

..etc).

mapReduce : the Reducer

- The mapoutput file is sitting on the local disk of machine. The reduce task needs the map output for it's particular partition from several tasks across the cluster.
- The map tasks may finish at different times, so the reduce task starts copying their outputs as soon as each completes. This is known as the copy phase of the reduce task, by default 5 threads can copy, this we can change by property.
- when all the map outputs have been copied, the reduce task moves into the sort phase. For example, if there were 50 map outputs, and the merge factor was 10. then there would be 5 rounds. Each round would merge 10 files into one, so at the end there would be five intermediate ~~results~~ files. final round that merges these 5 files into a single sorted file and move to the reduce phase. The output of this phase is is written directly to the output file System, typically HDFS.

In the above diagram, After the map phase is over, all the intermediate values for a given intermediate key are combined together into a list. This list is given to a Reducer:

- There may be a single Reducer, multiple Reducers.
- This is specified as part of the job configuration.
- All values associated with a particular intermediate key are guaranteed to go the same Reducer.
- The intermediate key, and their value lists, are passed to the Reducer in sorted key order
- This step is known as the 'Suffle & Sort'.

- The Reducer outputs zero or more final key/value pairs.
  - these are written to HDFS
  - In practice, the Reducer usually emits a single key/value pair for each input key.

The mapReduce flow: Reducers to outputs.

- Each of the portions (Reducer, Recordwriter, outputfile).

mapReduce Different programming model :-

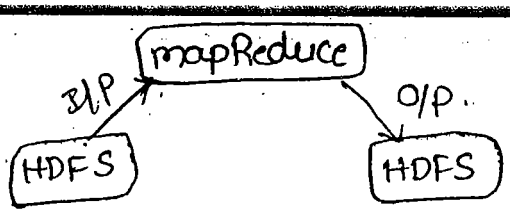
① Different phases of mapReduce algorithm

② Different data types in mapReduce.

Different phases of mapReduce algorithm :-

- mapReduce works by breaking the processing into 3 phases.
  - ① mapper phase
  - ② sort & shuffle phase [logical layer]
  - ③ Reduce phase.

- In mapReduce each phase as key-value pairs as input and output functions.
  - input and output functions → mapper functions
  - key-value pairs as → Reducer functions.



- mapReduce will expect the input in the form of (Key, value) pairs from HDFS layer only. once it is done with the processing it will produce the o/p again on top of HDFS in the form of (Key, value) pair.

phase	input	output
mapper	(K,V)	(K,V)
sort & shuffle	(K, V)	(K, list(V))
Reducer	(K, list(V))	(K,V)

Example for mapReduce :-

Input.txt

Hadoop is a Bigdata analysis storage & processing for the data. Hadoop is having good market now. good demand will give good openings.

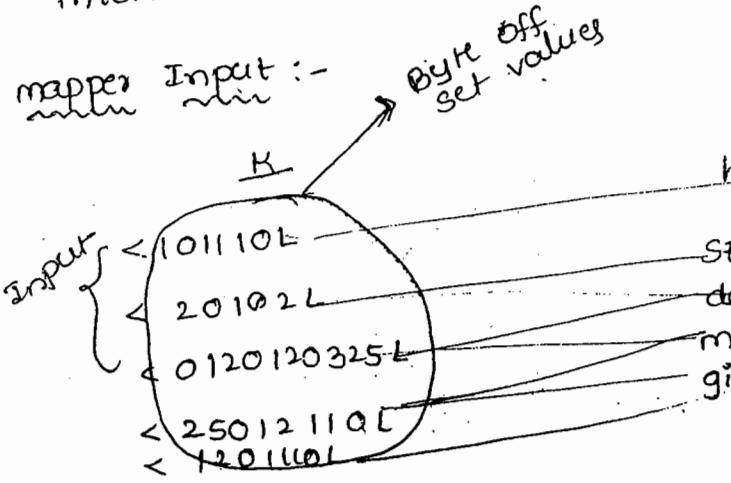
Explanation :-

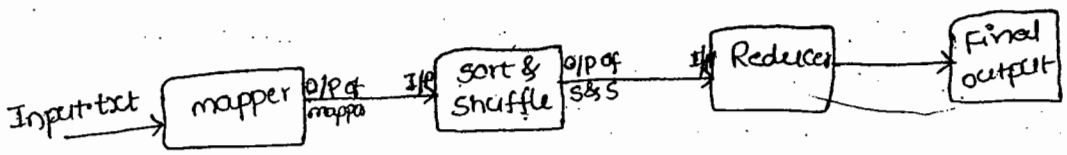
1) map phase :- map phase will expect input & output form.

- map phase will works input of Key, value pairs. (completely written by the system in the mapper phase)
- is inherently parallel
- each list element processed independently

- In the map phase Key value is in the form of Byte off set values.
- A list of data elements are provided to mapper function called the map() which transforms input data to an intermediate output data element.

mapper Input :-





mapper output :-

- mapper output will be displayed in the form of (K,V) pairs.
- In this mapper output value is always 1.

K	V
hadoop	1
is	1
a	1
bigdata	1
⋮	
opening	1

	Input	output
mapper	K,V	K,V
sort & shuffle	K,V	K, list(V)
Reducer	K, list(V)	(K,V)

Sort & Shuffle :-

- In this phase mapper output as take the Sort & Shuffle phase input.

Sort :- Sort is used to list the inputs in sorted order.

Shuffle :- mapReduce makes the guarantee that the input to every reducer is sorted by key. It is known as the shuffle.

Sort & Shuffle Input

K	V
hadoop	1
is	1
a	1
Bigdata	1
⋮	
openings	1

Sort & Shuffle output

K	list(values)
hadoop	1, 1
is	1, 1
Bigdata	1
⋮	
openings	1

Reducer :- is inherentaly sequential. (unless processing multiple lists)

- A Reducer function recieves an iterator input values from an output list. It combines these values together returning a single output value.

Reducer Input

K	list(values)
hadoop	1, 1
is	1, 1
Bigdata	1
⋮	
openings	1

Reducer output

K	V
hadoop	2
is	2
Bigdata	1
⋮	
openings	1

Different Data types in mapReduce:

Normal Language	mapReduce
Int	Intwritable
float	floatwritable
double	Doublewritable
long	Longwritable
String	Text
boolean	Booleanwritable

How to write basic mapReduce Algorithm:-

- In any mapReduce program with irrespective of Business logic will divided into 3 phases.
  1. Driver code → configuration level details
  2. mapper code → Business logic
  3. Reducer code → final output.

Driver code:-

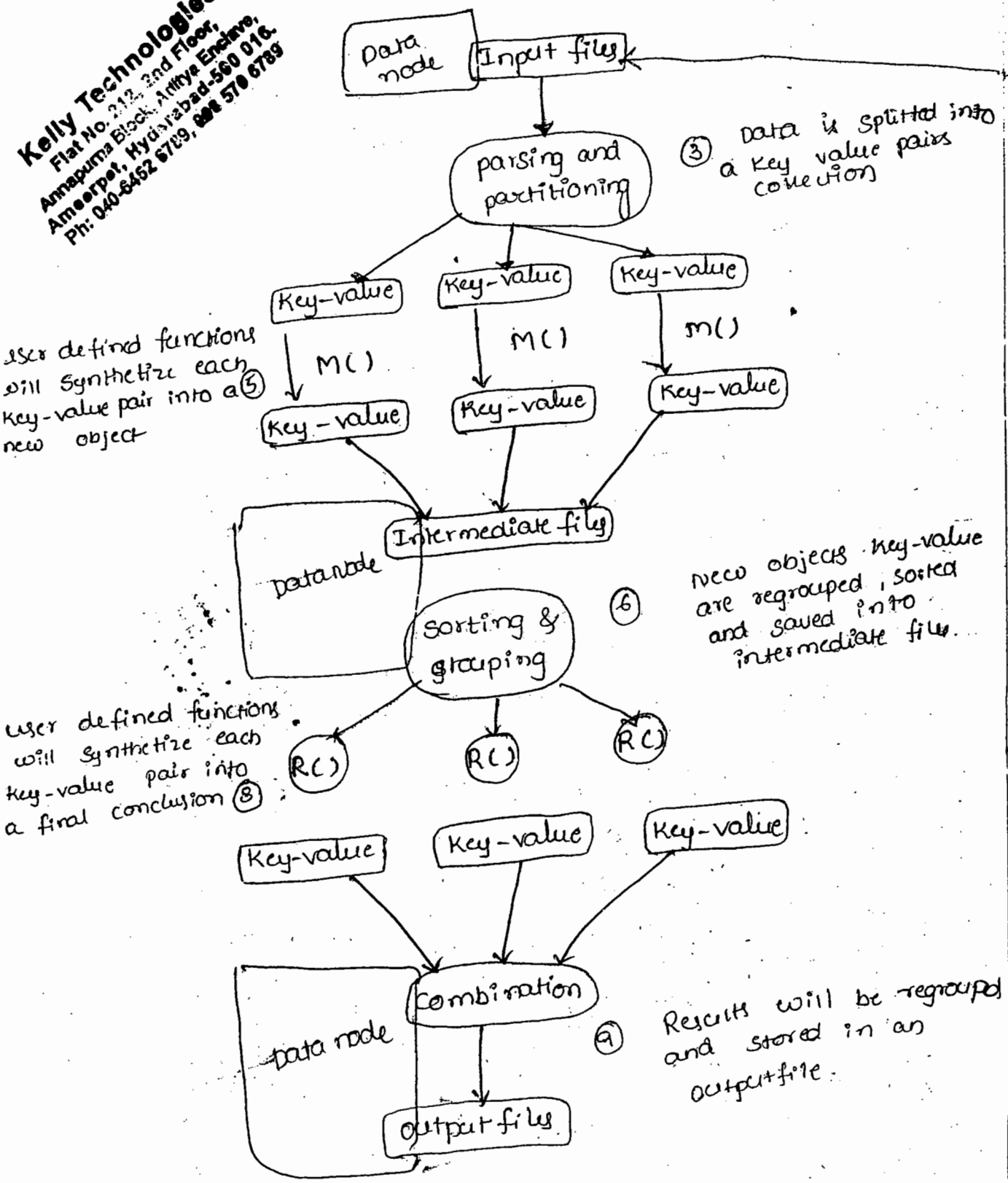
- Configuration level details with respect to Job, JAR Creation .. etc.
- mapper, Reducer class level details.
- final output key, value data type details.
- Input and output HDFS paths.

mapper - Reducer
one → one

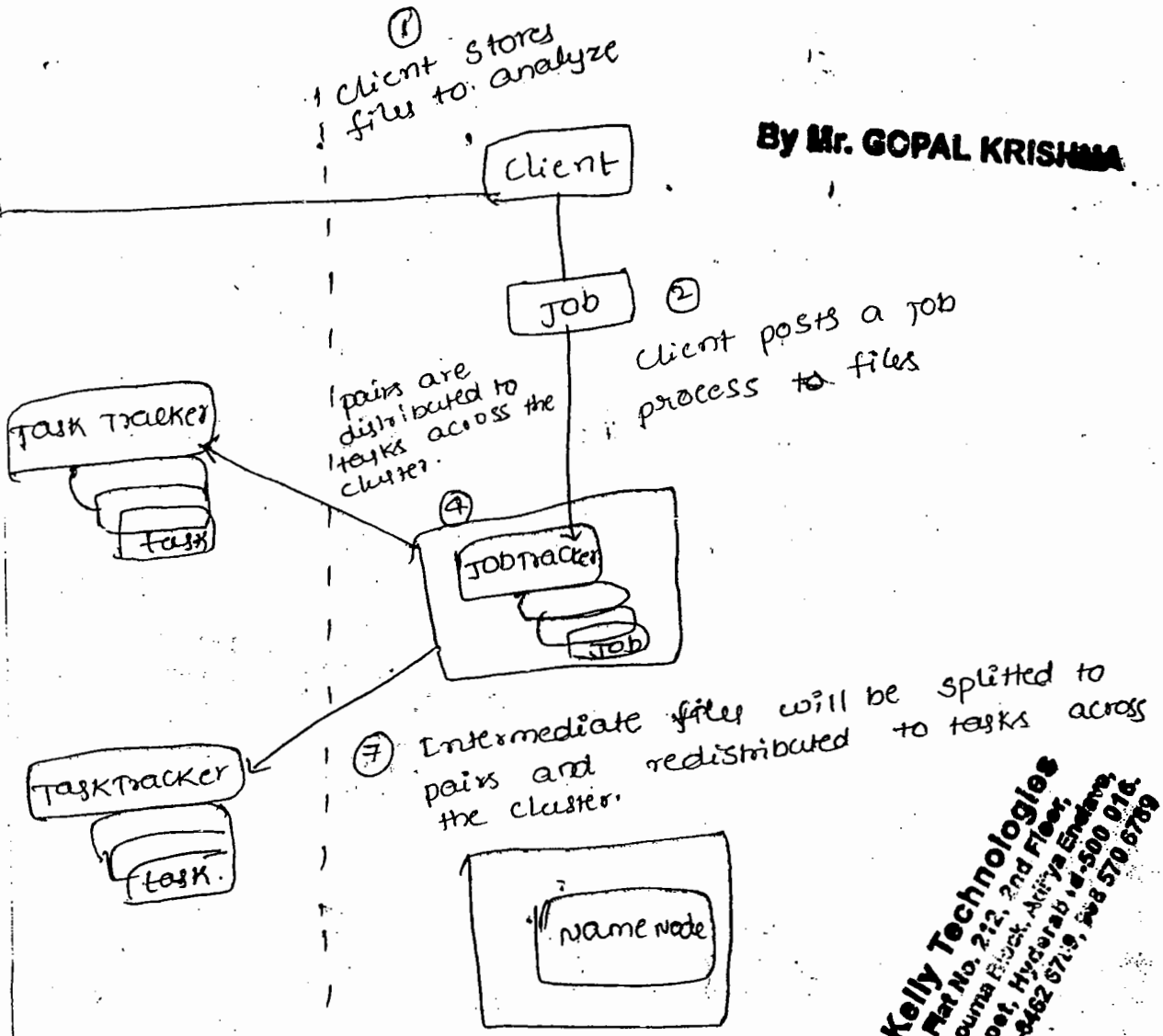
many → one

mapper
mapper
mapper
→ Reducer

Hadoop MapReduce life cycle.



over multiple slavenodes



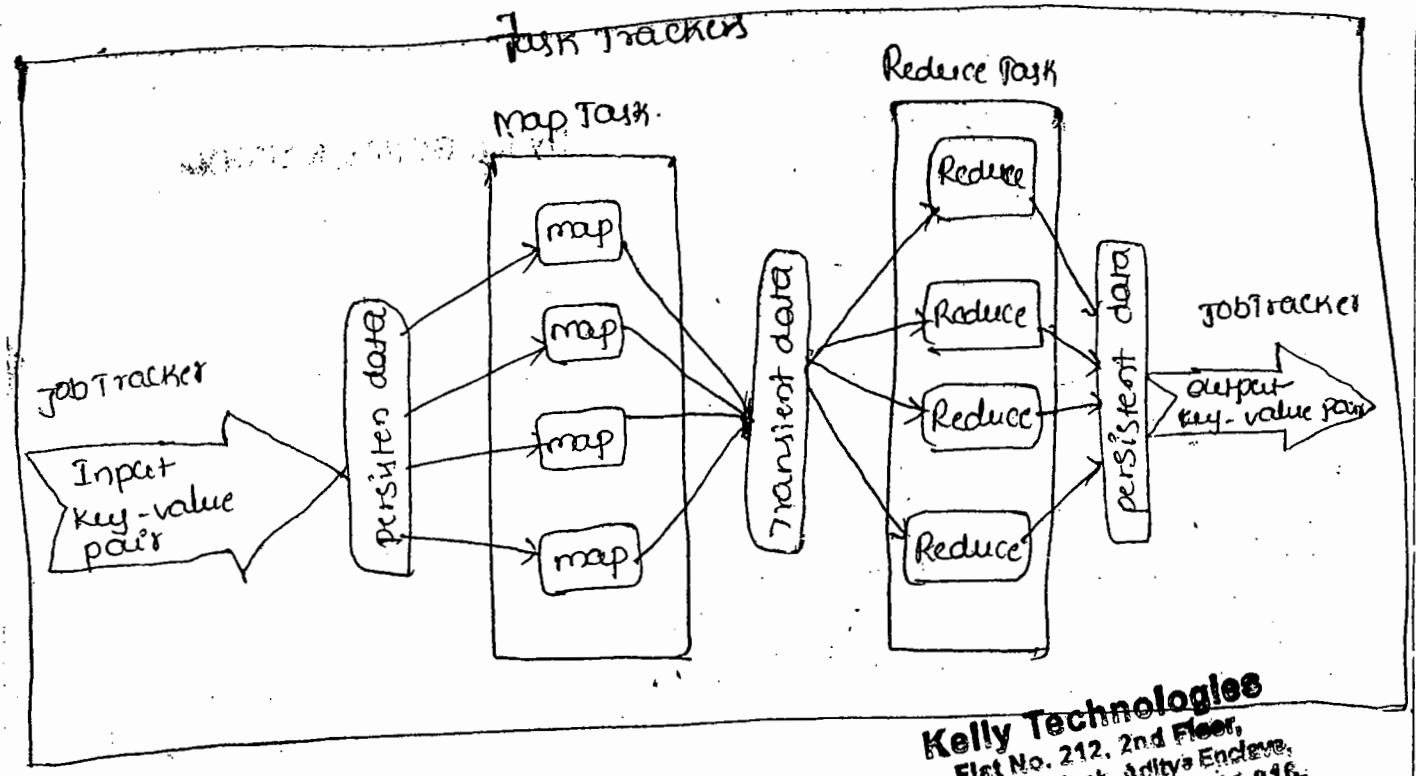
over the master node.

Record Reader is a class. Each input Format provides it's own Record Reader implementation.

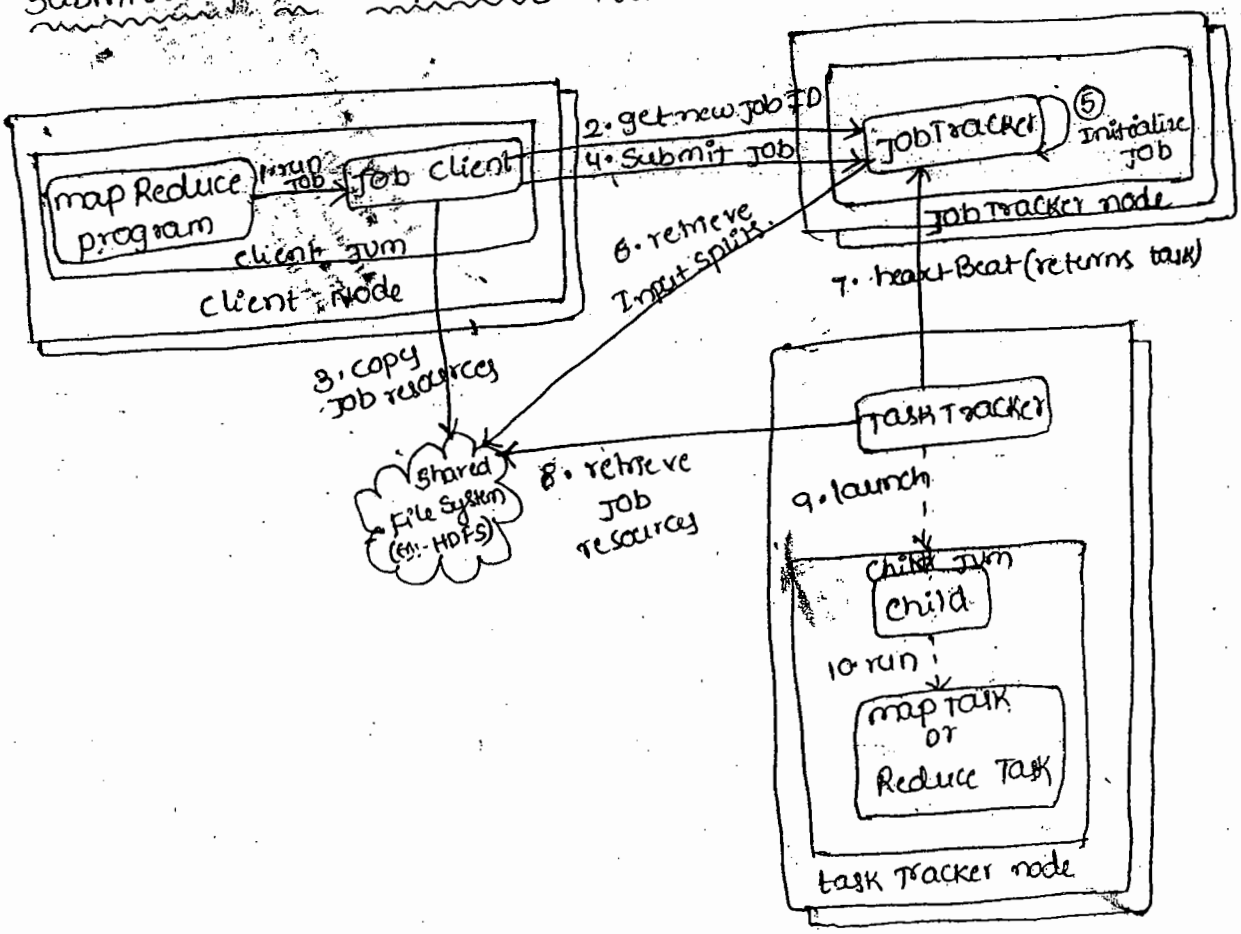
LineRecordReader :- Reads a line from a text file.

Key value Record Reader :- used by key value Text Input Format.

map Reduce execution framework.



Submitting a mapreduce job :-



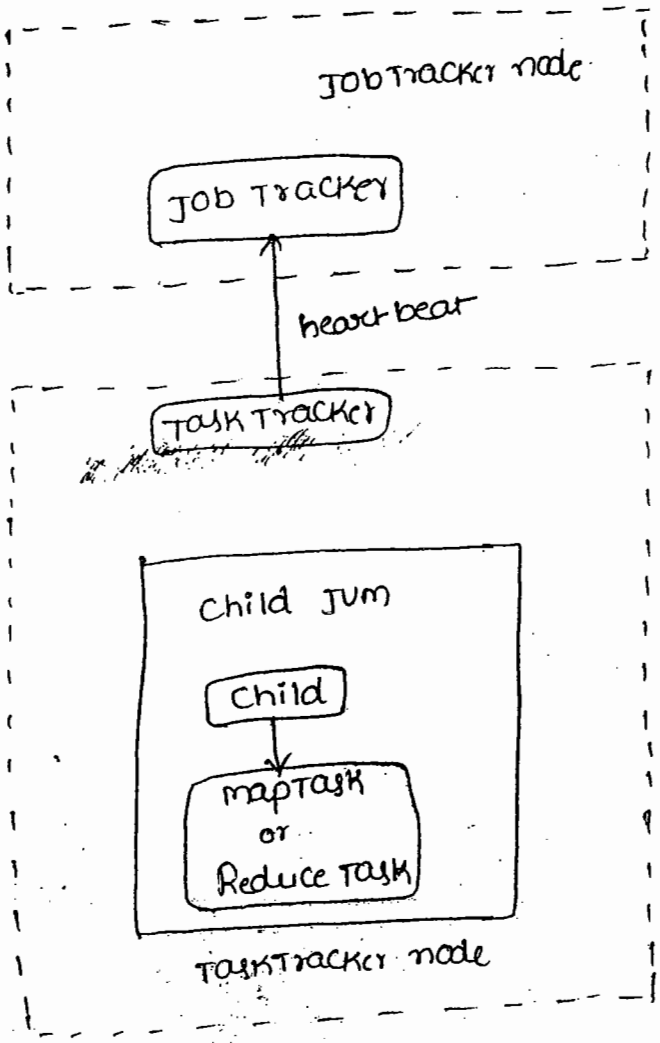
- whenever client submits the input file to the mapreduce in the form of JobConf object (.JAR file). JobTracker will receive the file to the available TaskTrackers in the hadoop cluster.

- By default every TaskTracker supports two map tasks and two reduce tasks. So it will accepts the tasks by JobTracker and it will perform the maptasks first and sends the progress information back to the JobTracker by the means of heart Beat.

- Based up on the progress information sent by the TaskTracker to the JobTracker, JobTracker will initiate the next phase called Reducer (provided if the TaskTracker has completed map phase @100%).

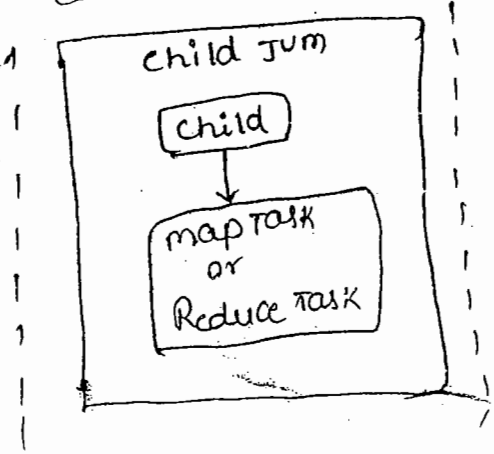
- once JobTracker receives all the Reducer outputs from all the TaskTrackers to whom it has assigned the task then it is called Job complete.

Fault - Tolerance



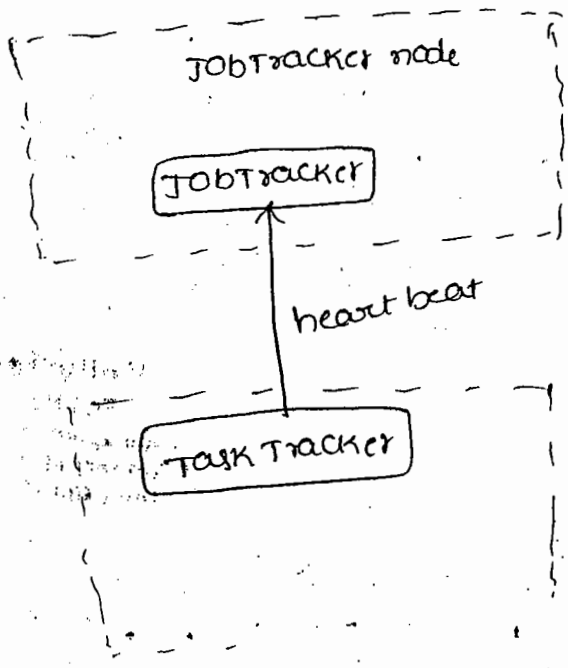
① Task Failure:

- if a child task fails, the child JVM reports to the TaskTracker before it exists. Attempt is marked failed, freeing up slot for another task.
- if the child task hangs, it is killed. JobTracker reschedules the task on another machine.
- if task continues to fail, Job is failed.



② TaskTracker failure:

- JobTracker receives no heartbeat.
- Removes TaskTracker from pool of TaskTrackers to Schedule task on.

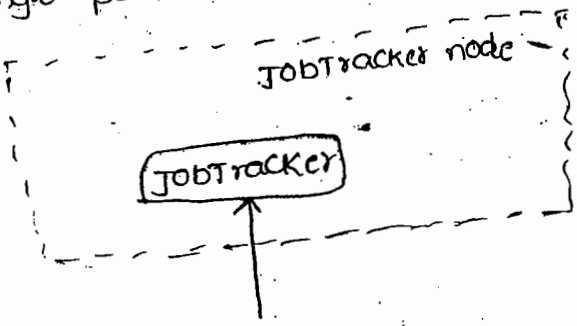


→ if at all JobTracker does not recieve heartbeat report from anyone of the TaskTracker it considers the particular tasktracker is down and it assign the same task to some other idle replicated node.

② TaskTracker fails.

③ JobTracker failure

- Single point of failure. Job fails.



③ JobTracker fails.

→ if at all genuinely there is a flow in the logic written in both mapper & reducer if at all we are getting corrupted / bad records in all these cases task will fail.

In this case jobtracker will try to execute the same task 4 times by default.

In the 4 attempts the same task is been failing then Jobtracker will mark the entire job as failed [as we can say job is complete when all the tasks are completed successfully).

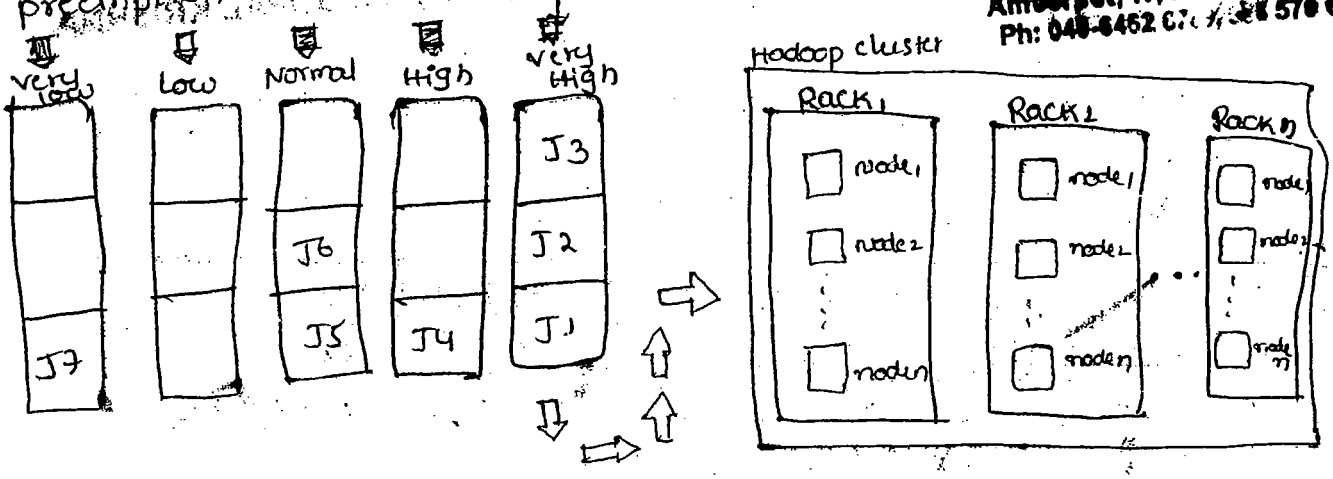
Scheduling :-

1. FIFO Scheduler (with priorities)
2. Fair Scheduler
3. Capacity Scheduler.

long jobs monopolize cluster
short jobs can hold back and have no guarantees on execution time

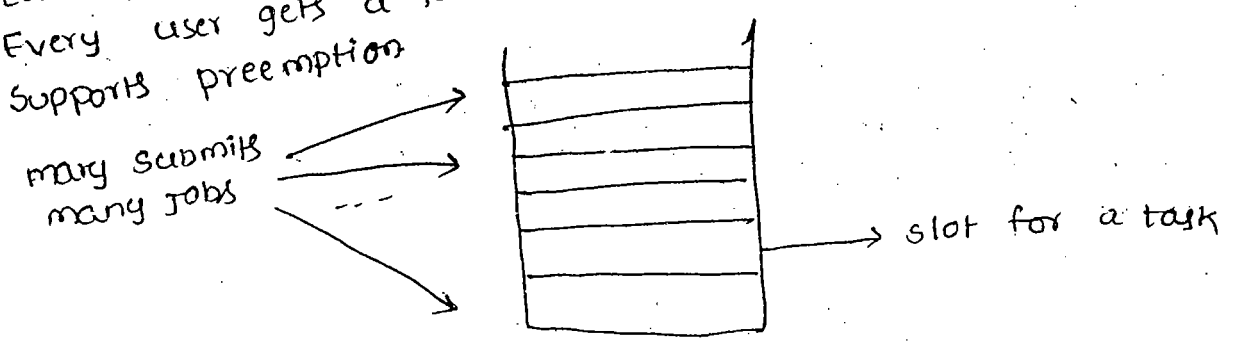
) FIFO scheduler (with priorities):-

- Not suitable for shared production-level cluster
- Each job uses the whole cluster, so jobs wait their turn.
- can set priorities for the jobs in the queue (5 queues with priorities)
- preemption is not supported.



) Fair scheduler :-

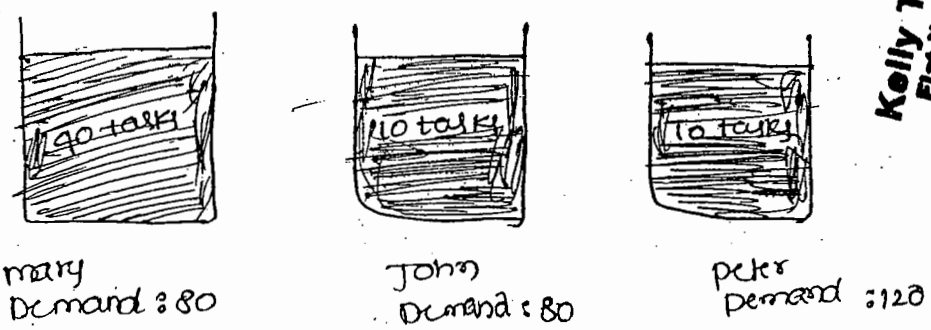
- Jobs are assigned to pools (1 pool per user by default)
- A user job pool is a number of slots assigned for tasks for that user.
- Each pool gets the same no. of tasks slots by default.
- Every user gets a fair share of the cluster capacity overtime.
- Supports preemption



- multiple users can run jobs on the cluster at the same time.
- Example:
  - mary, John, peter submit jobs that demand 80, 80, and 120 tasks respectively.
  - say the cluster has a limit to allocate 60 tasks at most.
  - Default behaviour: Distribute task fairly among 3 users (each get 20).

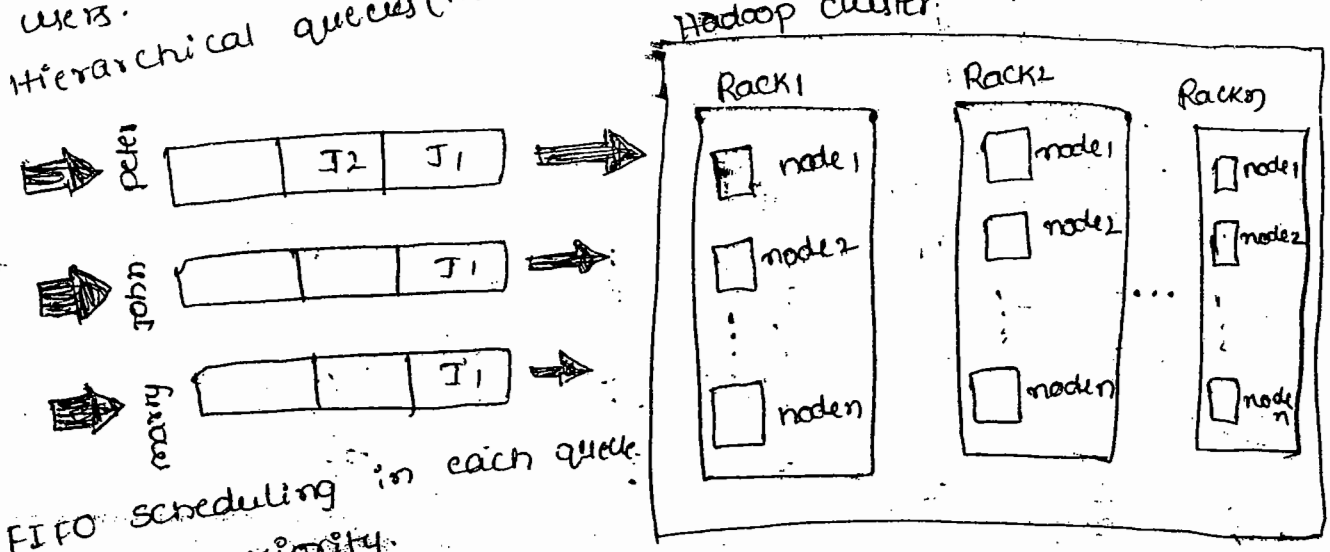
maximum amount of tasks that can be allocated in this cluster: 60

- minimum share can be set for a pool
- In the previous example, say mary has a minimum share of 40.
- mary would be allocated 40, then the rest is distributed evenly to other pools.



③ Capacity Scheduler :-

- Similar goal to Fair scheduler: Distribute Jobs fairly among users.
- works with queues instead of pools.
- A user thinks he has the cluster to himself with FIFO scheduling, but actually is sharing resources with other users.
- Hierarchical queues (mimic an organization)



- FIFO scheduling in each queue
- Supports priority

Task execution - Improving Performance :-

Speculative execution :

- Job execution is time sensitive
- Hadoop detects slow running tasks and launches another, equivalent task as backup.
- The output from the first of these tasks to finish is used.

Task JVM Reuse :-

- Tasks run in their own JVMs for isolation.
- Starting up a JVM is relatively expensive if jobs are short.
- In these cases when jobs have many short tasks, improve performance by configuring JVM reuse.

Joining datasets in MapReduce JOBS :-

① map side joins
② Reduce Side joins.

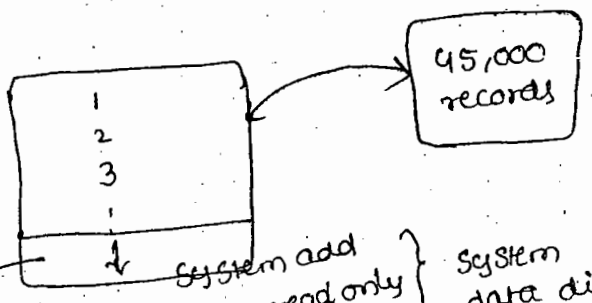
map-side joins :-

- map-side join is done in the map phase and done in memory.
- The most common problems with map-side joins are out of memory exceptions on slave nodes.
- map-side join is faster because join operation is done in memory.
- Replicate a relatively smaller input source to the cluster. put the replicated data set into a local hash table.
- Join a relatively larger input source with each local hash table.
- mapper do mapper-side join.

Disadvantage :-

- All the data should be sorted order.
- Side dist.

Ex :-

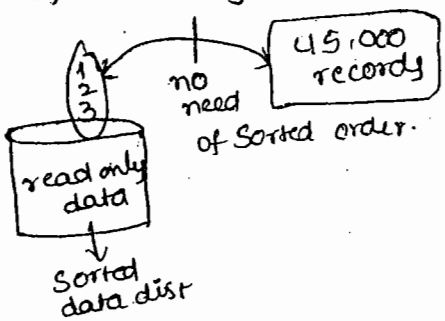


Reduce-side join :-

Reduce side join is a technique for merging data from different sources based on a specific key. There are no memory restrictions.

no need of sorting [overridden is not here]

Ex :-



Distributed Cache :-

- Distributed cache distributes application specific, large, read only files efficiently.
- Distributed cache is a facility provided by the map/Reduce framework to cache files (text, archives, jars and so on) needed by applications.

The framework will copy the necessary files to the slave node before any tasks for the job are executed on that node.

It is efficiency stems from the fact that files are only copied once per job and the ability to cache archives which are un-archived on the slaves. It can also be used as a rudimentary s/w distribution mechanism for use in the map and/or reduce tasks. It can be used to distribute both jars and native libraries and they can be put on the classpath or native library path for the mapreduce tasks.

- The distributed cache is designed to distribute a small no. of medium size articrafts, ranging from a few MBs to few tens of MB's

- One drawback of the current implementation of the distributed cache is that there is no way to specify map or reduce specific artifacts.

Counters :-

Counters are the useful channel for gathering statistics about the job.

- for the quality control OR application level statistics
- For problem diagnosis.

Built-in counters :-

- Hadoop maintains some built-in counters for every job, which report various metrics for our job.

Ex:- expected amount of INPUT consumed.
expected amount of OUTPUT produced.

Some-built in counters:

① map Input Records :- number of input records consumed by all the maps in the JOB. Incremented every time a record is read from InputSplit (through RecordReader) before passing to map() method of mapper.

② map output Records :- number of output records produced by all the maps in the JOB. Incremented every time a collect() method is called on context object, like wise,

③ Reduce Input Records

④ Reduce output records.

- Counters are maintained by the task with which they are associated, and periodically sent to the task tracker and then to JobTracker. so they all can be globally aggregated.

- The built-in job counters are actually maintained by the JobTracker, so they do not need to be set across the N/w unlike the all other counters, including the user defined ones.

Compression - MapReduce

Compression In MapReduce :-

- Compression reduces no. of bytes written to/ read from HDFS.
- Compression effectively improves the efficiency of network bandwidth and disk space.
- This saves the amount of data being transfored between MAP nodes to Reduce nodes.

LZO :-

- LZO is a compression/decompression library.

compression format : LZO
Hadoop CompressionCodec :
Com.hadoop.compression.lzo.lzopcodec

note :- "codec" is the implementation of a compression-decompression algorithm. In Hadoop, a "codec" is represented by an implementation of the "compressioncodec" interface.

Why using Compression :-

1. Reduce storage requirements
2. speed up data transfers (across the N/w or from disks)

LZO Key charackistics:-

- very fast decompression.
- Requires an additional buffer during the compression (size of 8 Kb or 64 Kb depends on the compression level).
- It does not requires the additional buffer during the decompression other than the source and destination. buffers. . thats why fast decompresson is possible with LZO.
- Allows the user to adjust the balance between compression ration and compression speed, without affecting the speed of decompression.

Hadoop provides the below compression codecs:-

- com.hadoop.compression.DefaultCodec
- com.hadoop.compression.lzoCodec.
- com.hadoop.compression.SnappyCodec.

Configurations Required for LZO compression - In my "mapred-site.xml":

By default compression is not enabled in the MapReduce (i.e value is false). To achieve the compression we have to edit the below property of mapred-site.xml

```
<property>
    <name> mapred.output.compress </name>
    <value> false </value>

</property>
```

note :- to enable the compression, value should be "true".

which compression codec to be used while compressing job output. By default "DefaultCodec" will be used. Inorder to use other than default (Lzo or Snappy), we have to replace their corresponding codecs;

Like Below:

```
<property>
    <name> mapred.output.compression.codec </name>
    <value> org.apache.hadoop.io.compress.DefaultCodec </value>
</property>
```

note: 1. in place of DefaultCodec, give LzoCodec, SnappyCodec

2. we can also specify which compression codec to be used while compressing the map outputs.

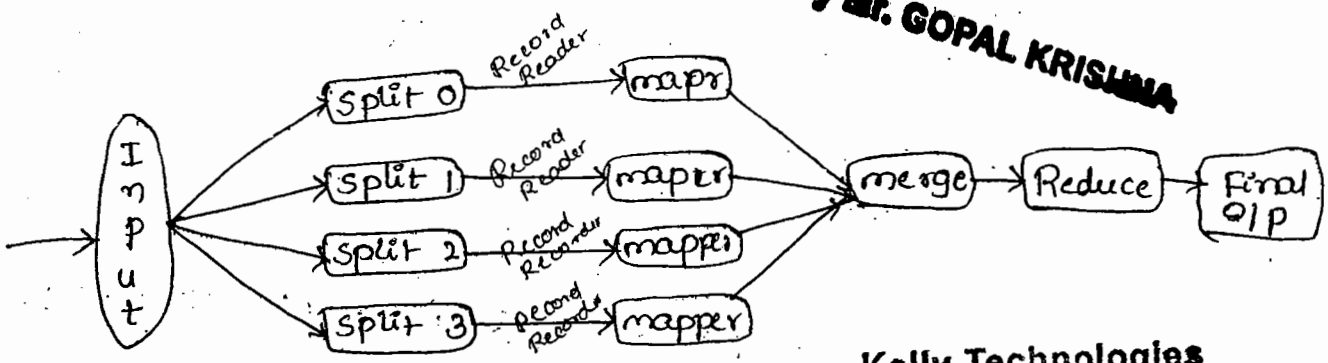
Snappy Compression :-

Snappy is also compression/decompression library. It does not aim for maximum compression, or compatability with other compression library.

Snappy aims for very high speeds and reasonable compression.

Input Splits :-

- if mapReduce jobs can process the entire input in single shot there will not be any concept of parallelism with the same reason hadoop divides the input to mapReduce jobs into number of fixed size chunks known as Input splits (or) splits.



mapred.splits.max-size
mapred.split.min-size

part-m-00000 → typical output of mapper
part-r-00000 → typical output of Reducer

- mapReduce will not take the input as it is which the client gives instead, it divides the input data into multiple chunks which is known Inputsplits (or) splits.
- Split size should always be equal to or greater than blocksize.

Note:- General practise would be blocksize should be equal to splitsize.

By the means of split concept mapReduce achieves the parallelism in Hadoop.

- if our splitsize is less than blocksize we will have so many smallersize splits and thereafter multiple mappers will be created on each and every split which will resultant into a poor performance.

- mapReduce job is a unit of work which client is expective. This mapReduce jobs can be driven by two daemons like

① JobTracker [which will co-ordinate the task & Scheduling & Rescheduling the task]

② TaskTracker [which is exactly responsible for execution of the task on the datanodes]

Creating Input and output formats in mapReduce jobs :-

① Input Formats ② output formats.

Input Formats :-

- specification for reading data.
- Defines how to break the inputs.
- provides RecordReader objects to read the files.
  - Read data and converts to <Key, value> pairs.
- It is possible to write Custom Input Formats.

File Input Format :- It is base implementation class for all the file formats.
- Base implementation for all the classes.

) Text Input Format :-

- default format.

Here Key → System Byte Offset values
value → each and every line treated as value.

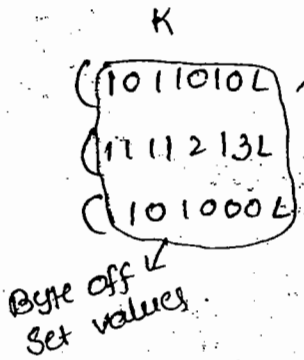
- It is the default file format of mapreduce which will be used when the incoming data is in the form of "Text".

- Each and every line of the code is the record here and each & every record with be separated by a newline character '\n'.

- Text Input file format, Generally

  Key → Byte offset values

  Value → The entire text of the record.



- Plain Text Input

  Split → Single HDFS block (Can be configured)

  Record → Single line of text; line feed or carriage return used to locate end of line.

  Key → LongWritable - position in the file

  Value → Text - line of text.

=

② **Key-value Text Input Format:-** Each line as key-value (tab delimited)

- Key value Text Input Format will be used in mapReduce programming whenever we are getting the input data in the form of (K, V).
- By default (K, V) Text Input Format is '\t' is the default delimiter (However we can change the same in code).
- As we have the specific key in this input formats the Byteoffset values will not get be generated.

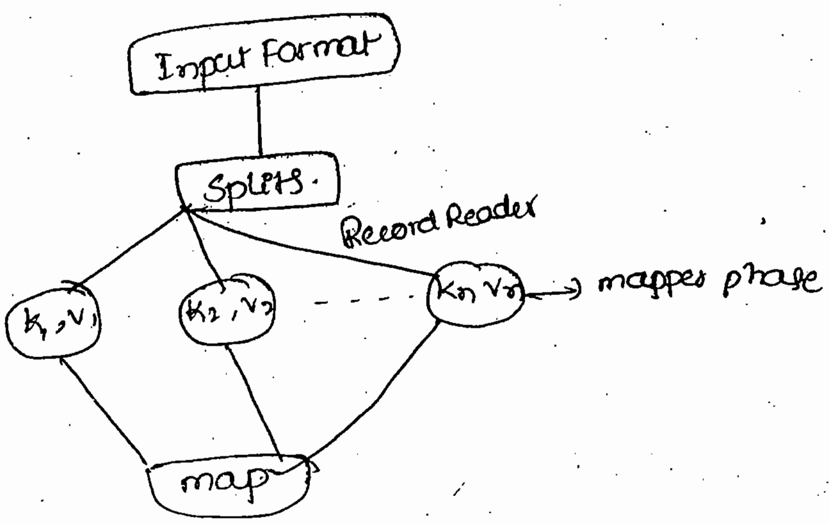
Ex :-

K		V
hadoop	\t	bigdata.
Bigdata	\t	Analysis.
Android	\t	mobileApps.

③ **N-line Input Format :-**

- Same as Text Input Format but splits equal to configured N-lines.

N-lines → Number of the maximum records in each and every split.



- Record Reader is a class which is primarly responsible for the (K, V) pairs given in the Input Splits.

- whenever we are providing the data multiple splits into mapper functions, if we have variable no. of records and each and every split then we will not have proper control on when exactly a particular split will be completed.
- with the same reason if we want to put a fixed no. of records each & every split then we can go head with NLineInputFormat.

Split → N lines: configured via mapred.line.input.format or NLineInputFormat.setNumLinesPerSplit(Job, 100);

Record → Single line of text

Key → LongWritable - position in the file

value → Text - line of text.

④ Table Input format :-

- converts data in HTable to format Consumable to MapReduce.
- mapper must accept proper key/values.

Split → Rows in on HBase Region (provided scan may narrow down in the result)

Record → Row, returned columns are controlled by a provided scan.

Key → Immutable Bytes writable.

value → Result (HBase class)

Sequence File InputFormat:-

- Hadoop specific binary representation.
- Special type of file to store key-value pairs.
- Store keys and values as byte arrays.
- uses length encoded bytes as format.
- often used as input or output format for MR jobs.
- Has built in compression on values.

Output formats:- Specification for writing data.

- How the result <key, value> pairs are written in to files.
- possible to write custom output Formats.
- Hadoop default output formats.
  - Text output Format
    <key TAB value>
  - Sequence File Output Format.
    - Hadoop specific binary representation.
- validates output specification for that job.
  - you may have seen annoying messages that output directory already exists.
- Creates implementation of Record writer.
  - Responsible for actually writing data.
- Creates Implementation of output committer.
  - Setup and clean-up job's and task's artifacts (ex:- directories)
  - commit or discard tasks output.

① Text output format :-

- outputs plain text.
- saves key-value pairs separated by tab.
- configured via mapreduce.output.textoutputformat.separa property
- Set output path

TextOutputFormat.setOutputPath(Job, newPath(mypath));

② Table output Format :-

- saves data into HTable
- Reducer output key is ignored.
- Reducer output value must be put or delete objects.

Data localization in mapReduce :-

- Every mapper will be stored in localfile.
- once the mapper phase has been completed by TaskTracker, the mapper o/p will be stored in the localfile System (but not on HDFS). The reason might be all the mapper output from respective TaskTracker put on the N/w in order to consume by the Reducer phase.
- In this whole process due to some N/w Errors or between related errors, if we loss the mapper o/p, we have to redo the entire process of mapper which is a time consuming process and performance overhead.
- To avoid the same, The mapper o/p will be stored in the localfile System. This is called data localization.

Note :- Data localization is only for the mapper but not for reducer, The reason might be the o/p whatever we get from reducer is the final output.

=

Combiner :-

- To optimize the N/w & bandwidth limitations combiner concept will be used in mapReduce programming.
- Combiner will act as local reducer (or) mini Reducer as whatever the data consumed by Reducer phase, the same copy will reside in the combiner.
- Hadoop does not provide any guarantee on combiner's execution.
- Hadoop may call combiner function zero, one or many times for a particular map output record.

Note :-

- While calling reduce method, Hadoop does not provide any guarantee on values in stored order corresponding to a particular key, to achieve this we used the Secondary sorting.
- The combiner function does not replace the reduce function.
- Combiner is specify the combiner function.

partitioners :-

- partitioner allows you to distribute how outputs from the map storage are sent to the reducers. Basically it partitions the keyspace.
- partitioner controls the partitioning the keys of the intermediate map-outputs. The key [subset of the key] is used to derive the partition.

The total no of partitions is same as the no of reduce tasks for the job.

- partitioner run on the same machine after mapper had completed it's execution, by consuming mapper o/p. so entire mapper output (record) is sent to partitioner and partitioner forms r (no. of reduce task) groups from the mapper output.

- By default hadoop framework Hash based partitioner. This particular partitions the keyspace by using the hashcode.
- The following is logic Hashpartitioner executes to determine a reducer for a particular 'key'.
- (key.hashcode() & Integer.MAX-VALUE) % num Reduce Tasks.

How to write Custom partitioner :-

- TO have Hadoop use a Custom partitioner you will have to do minimum the following three.

① Create a new class that extends partitioner class.

② Override method get partition.

③ In the wrapper that runs the mapReduce, either

* add the Custom partitioner to the job programatically using method set partitioner class or

* add the Custom partitioner to the job as a config file (if your wrapper reads from config file or oozie).

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCound{

    public static class TokenizerMapper extends Mapper
    <Object, Text, Text, IntWritable>{

private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

public void map(Object key, Text value, Context context)
    throws IOException, InterruptedException{

StringTokenizer itr = new StringTokenizer(value.toString());
while(itr.hasMoreTokens()){
    word.set(itr.nextToken());
context.write(word, one);
}
}
}
}
```

```
public static class IntSumReducer extends Reducer
        <Text, IntWritable, Text, IntWritable>{
private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable> values,
                   Context context) throws IOException,
        InterruptedException{
    int sum = 0;
    for(IntWritable val : values){
        sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
}

public static void main(String[] args) throws Exception{
    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordcount");
    job.setJarByClass(WordCountNew.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```


Everything comes through object

operator can be used to create object

Configuration is a Base class from Hadoop (mapred-site.xml)

pre defined class

Configuration conf = new Configuration();

Job job = new Job(conf, "wordcount");
↳ Job name.

mapper & reducer class

job.setJarByClass(WordCountNew.class); (class name is same)
job.setMapperClass(TokenizerMapper.class); → mapper class name
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class); → Reducer class name
job.setOutputKeyClass(Text.class); → (K, V) → Reducer output (or) final output
job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);

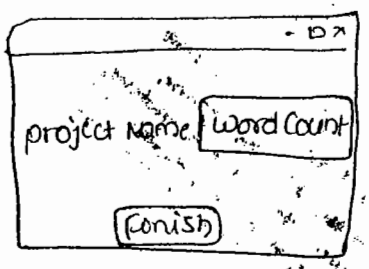
→ Job is a particular class. it derived from base class is a Configuration.

→ we can create the job compulsory Conf object will be created.

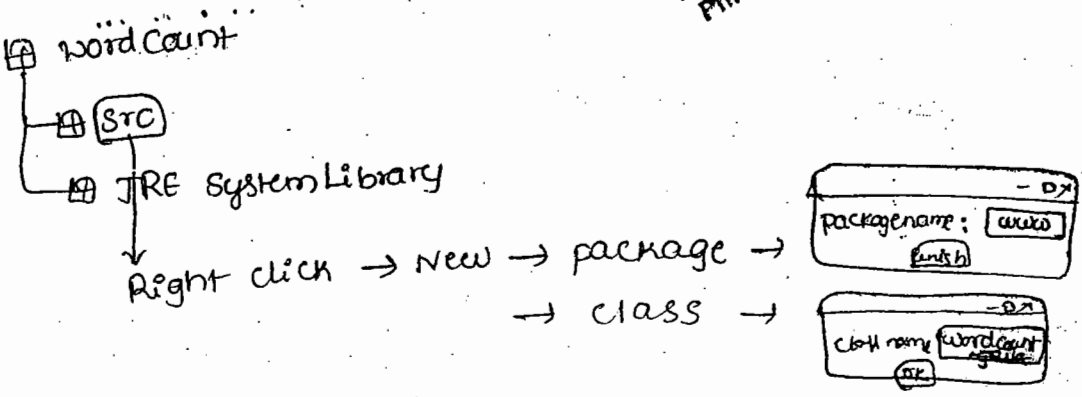
How to create program in NetBeans (or) MyEclipse IDE :-

Step1 :-

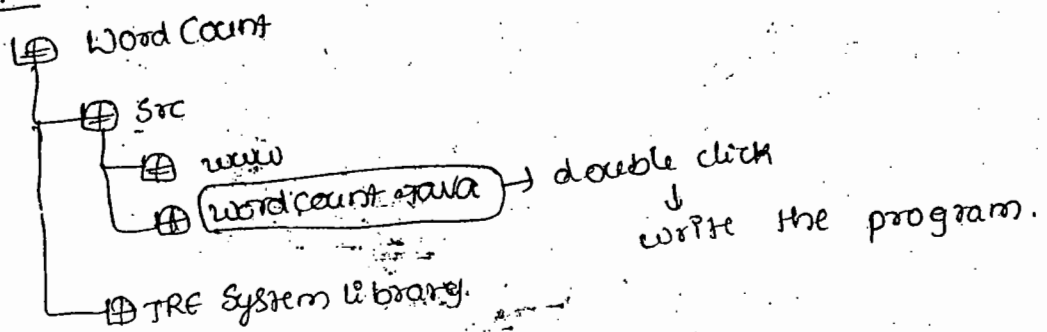
File → New → Java project → click OK.



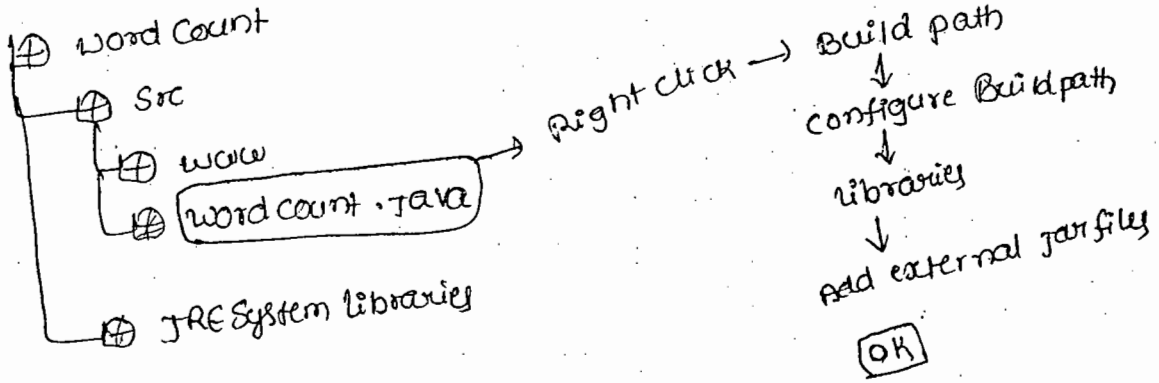
Step 2 :-



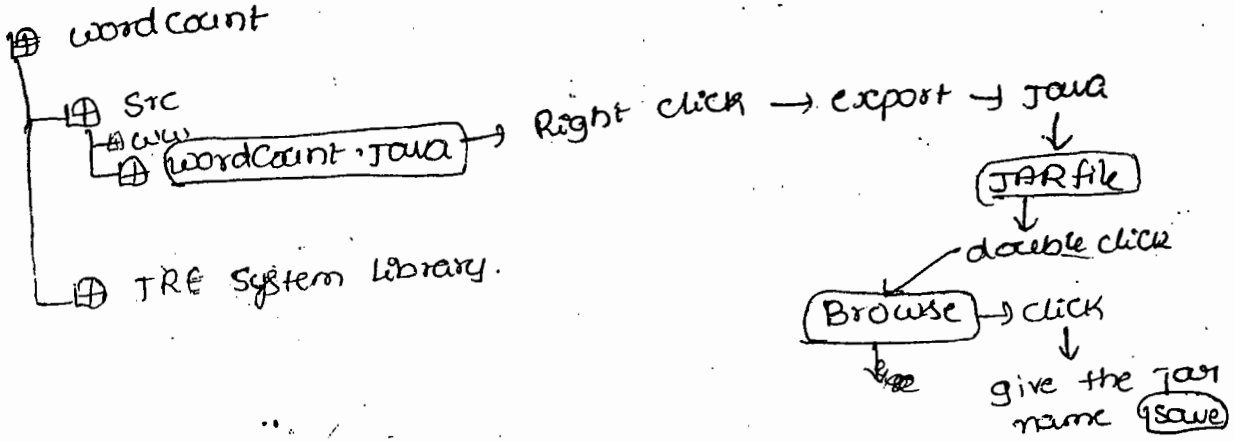
Step3 :-



Step4 :- How to create add jar file.



Steps :- How to export file jarfile to linux.



Step6 :- Open the vmware.

if config ↵

192.168.235.131

Step7 :- Start → Rightclick → open windows explorer

↓

//192.168.235.131 ↵

Downloads.

↓

open → In this Downloads copy paste the jar file.

Step8 :-

Downloads # ls ↵

wordcount.jar

Step9 :- # nano Input.txt ↵

Step10 :- # hadoop fs -mkdir anitha ↵

hadoop fs -put Input.txt anitha ↵

Step11 :- # hadoop jar <Runnable jar name> classname <without extension>
<HDFS Inputpath with file> <HDFS output path file>

hadoop jar wordcount.jar wordcount anitha/Input.txt anitha/output ↵

hadoop fs -ls anitha/output ↵ hadoop fs -cat anitha/output

	Traditional RDBMS	MapReduce
Data Size	GB - TBs	TBs - PBs
Access	Interactive & Batch	Batch
Updates	Read/write many times	Write once, read many times
Structure	Static schema	Dynamic Schema
Integrity	high	Low
Scaling	Non Linear	Linear

MapReduce optimizations :-

- overlapping maps, shuffle and sort
- mapper locality.
  * schedule mappers close to the data.
- combiner
  * mappers may generate duplicate keys
  * side effect free reducer can be run on mapper node.
  * minimizes data size before transfer
  * Reducer still is run

Speculative execution to help with load-balancing.

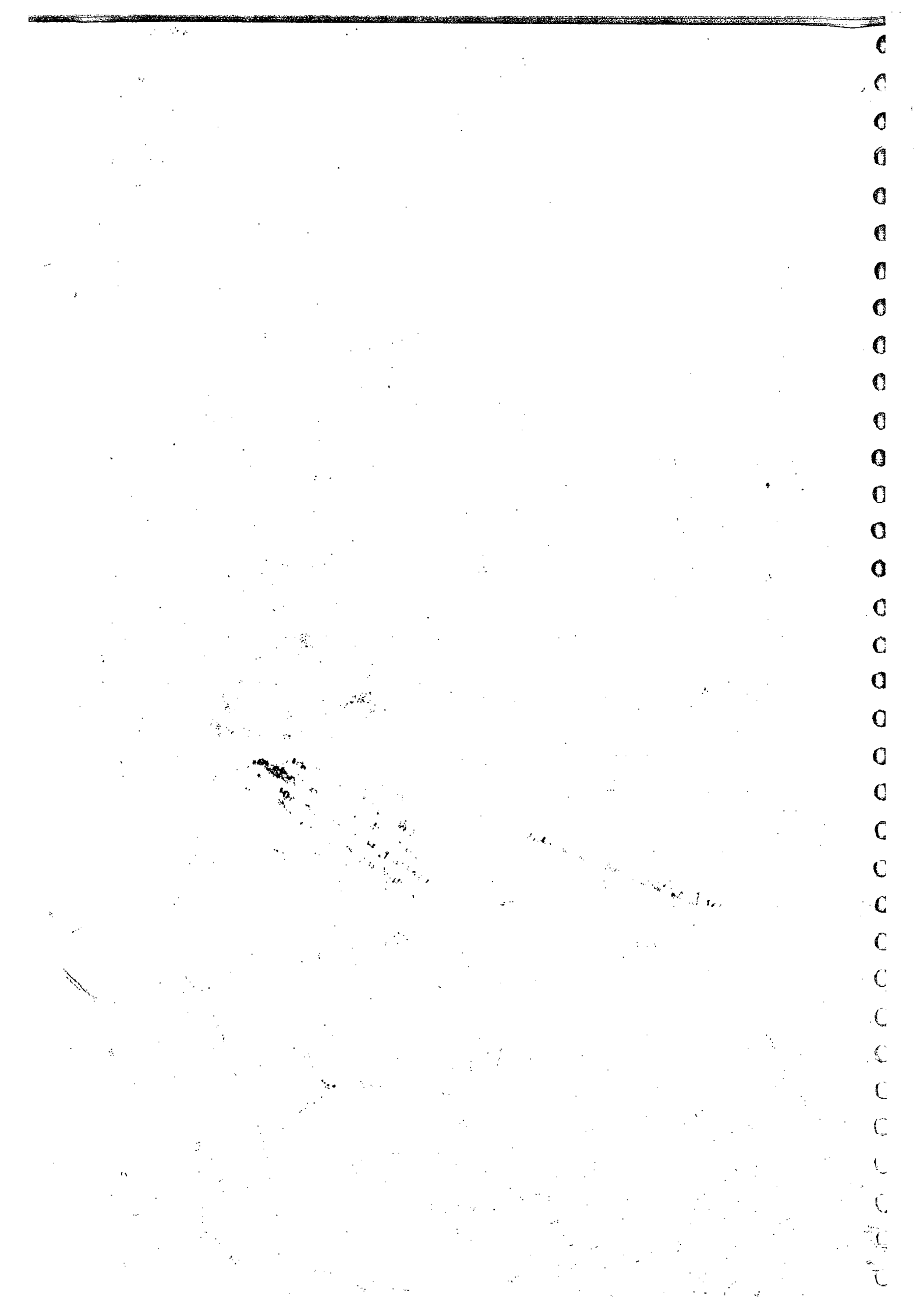
* Some nodes may be slower.
* Run Duplicate task on another node, take first answer as correct and abandon other duplicate tasks.
* only done as we start getting toward the end of the tasks.

Serialization :-

- Transforms structured objects into a byte stream.
  - For transmission over the network. Hadoop uses RPC
  - For persistent storage on disks.

- Hadoop uses its own serialization format, Writable
  - Comparison of types is crucial (Shuffle and sort phase) Hadoop provides a Custom Raw Comparator, which avoids deserialization.
  - Custom Writable for having full control on the binary representation of data.
  - Also external frameworks are allowed : enter Avro.

- Fixed length or variable-length encoding ?
  - Fixed length : when the distribution of values is uniform.
  - Variable length : when the distribution of values is not uniform.



Pig

- Apache Pig is one of the Components of Hadoop which is an abstract layer (high level procedural language) on top of Hadoop and mapreduce platform.
- Pig is also known as Data flow language (or) Transformation language.
- pig was initially introduced at yahoo laboratories at very first time inorder to address their adhoc request.
- Pig language was officially opted by Apache in the year 2006 from there onwards changed to pig language called Apache Pig.
- The language to express Pig language transformation is known as "PigLatin".
- As we are dealing with multiple transformation and Dataflow will be through the transformations. we can call pig is a transformation language (or) Dataflow language.
- pig is a simple language that extensively executes statements.
- A statement is an operator that takes input (such as bag represents a) and emits another bag as it's output.
- pig latin are not case-sensitive. UDF names are case sensitive.

Ex: COUNT → COUNT

=

Ex: COUNT → count

=

pig VS SQL

pig	SQL
① pig is procedural (HOW)	① SQL is declarative.
② Nested relational data model	② Flat relational data model
③ Schema is optional	③ schema is required
④ Scan-centric analytic workloads	④ OLTP + OLAP workloads.
⑤ Limited query optimization	⑤ significant opportunity for query optimization

Advantages of using Pig:-

① pig can be treated as a higher level language.
- increases programmer productivity
- decreases duplication of effort.
- opens the M/R programming system to more users

② pig insulates against hadoop complexity.
- Hadoop version upgrades
- Job configuration tunning.

Pig Features:-

- Data-flow Language:- User specifies a sequence of steps where each step specifies only a single high-level data transformation.

The step-by-step method of creating a program in Pig is much cleaner and simpler to use than the single block method of SQL. It is easier to keep track of what your variables are, and where you are in the process of analyzing your data.

- User Defined function (UDF)
- Debugging Environment
- Nested data model

What is Pig :-

- Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs.
- Pig generates and compiles a map/Reduce programs on the internally.

Why Pig :-

- Ease of programming :- It is trivial to achieve parallel execution of simple, "embrassingly parallel" data analysis tasks.

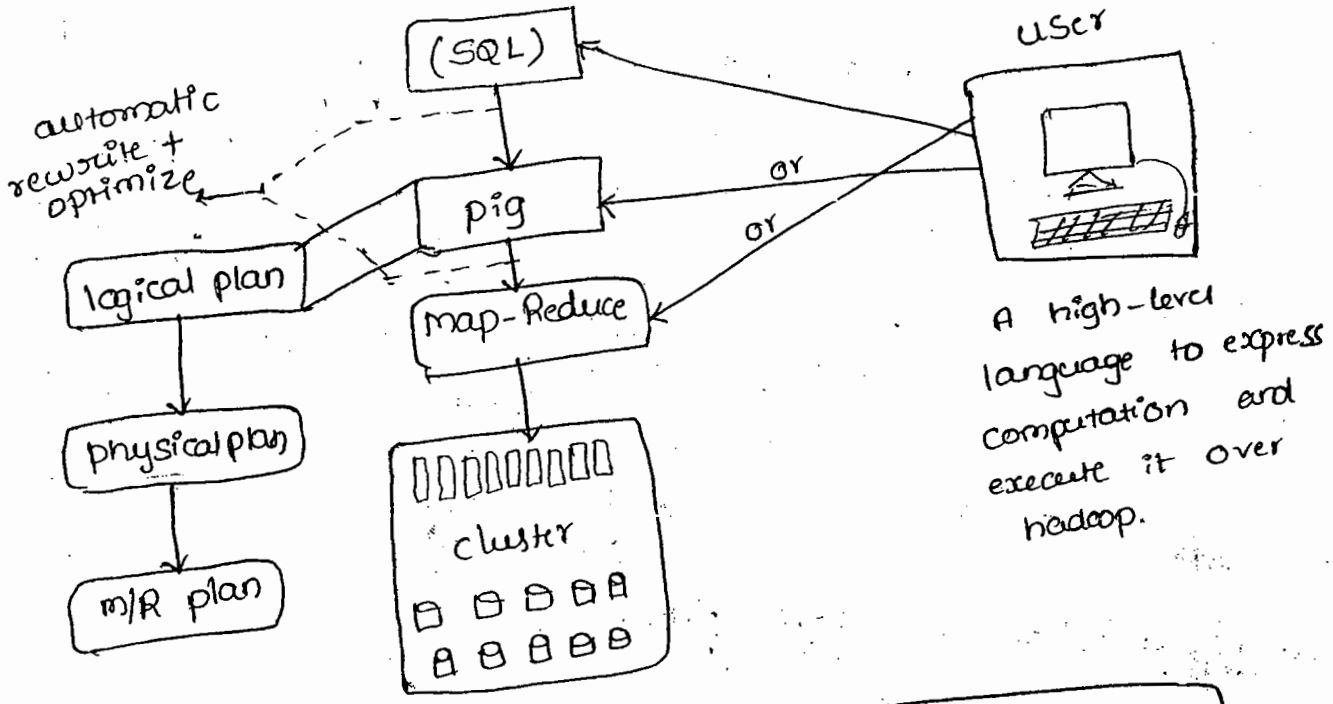
Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand and maintain.

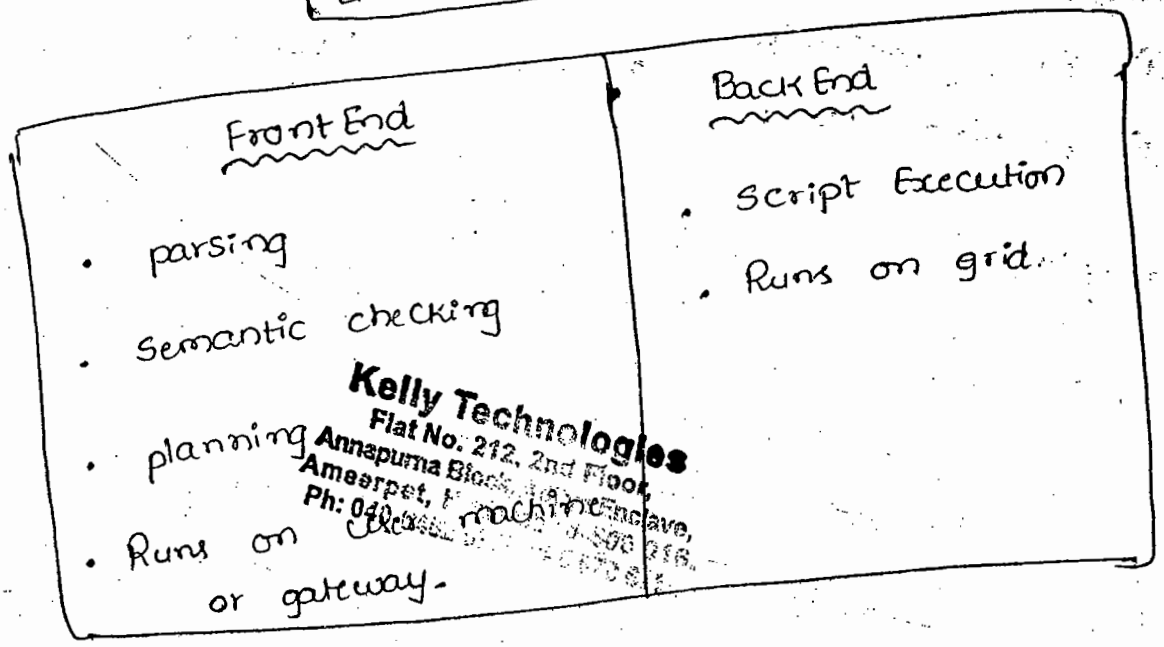
Pig takes Care Of ....

- Schema and type checking.
- Translating into efficient physical Dataflow.
  - (i.e Sequence of one or more MapReduce Jobs)
- Exploiting data reduction opportunities
  - (eg. early partial aggregation via a combiner)

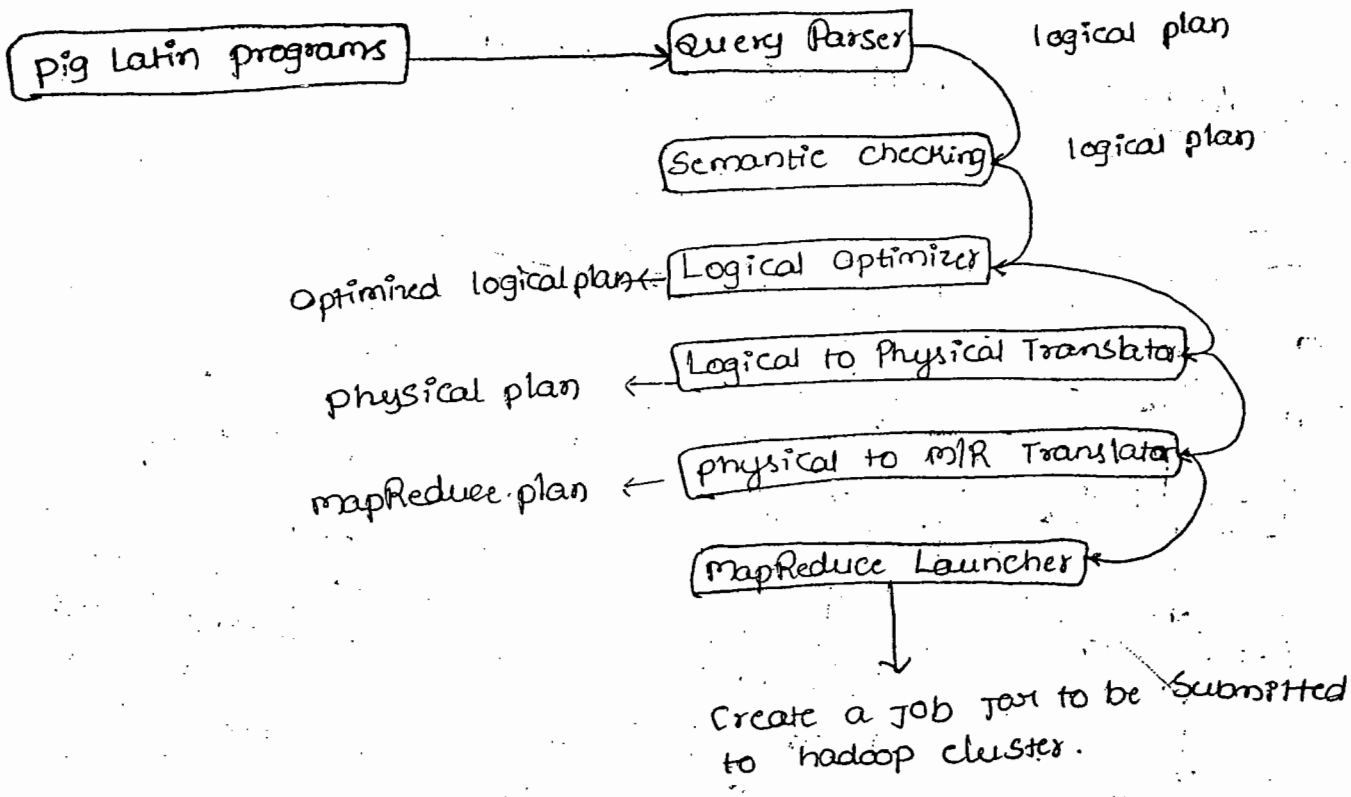
- Executing the System-level dataflow.
  (i.e running the mapReduce jobs)
- Tracking Progress, errors, etc.

Pig Architecture (mapReduce as Backend)





Front End Architecture :-



During optimization of Logical plan

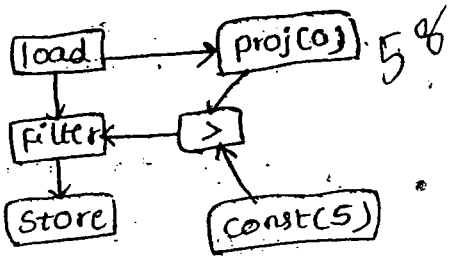
1. Early projections
2. pushing limit
3. Streaming Load and Store.

Logical plan :-

- Directed Acyclic Graph
  - logical operators as Node
  - Data-flow as edges.
- Logical operators-
  - one per pig statement
  - Type checking with Schema

Ex:-

```
a = load 'my file';
b = filter a by $0>5;
store b into 'myfiledir';
```



physical plan :-

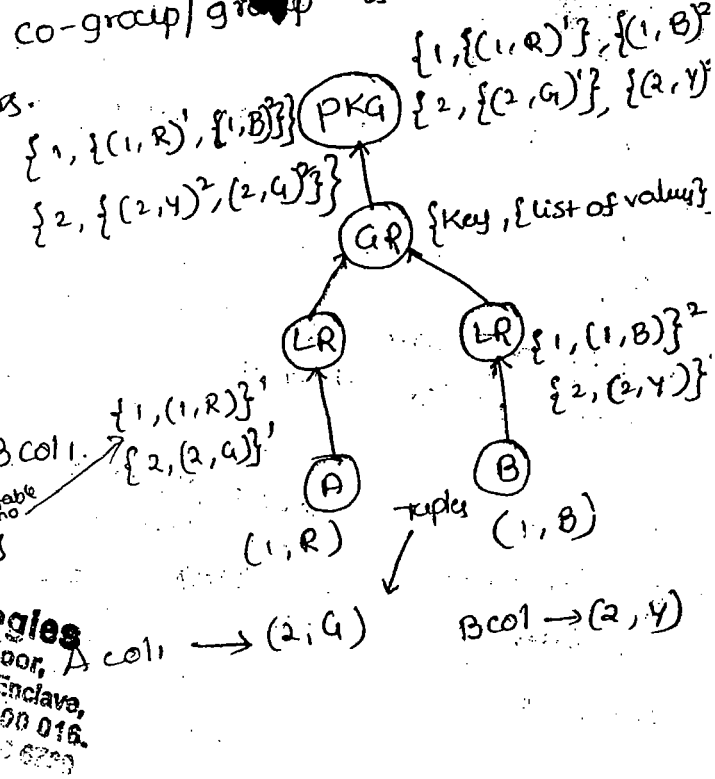
- Pig supports two back-ends.
  - Local
  - Hadoop MapReduce

logical to physical Translation

- 1:1 correspondence with most logical operators.
- Except Cross, Distinct, Group, & Co-group and Order.

Ex:- logical to physical plan for group operator :-

- Logical operator for co-group/group is converted to 3 physical operators.
  - Local Rearrange (LR)
  - Global Rearrange (GR)
  - Package (PKG)



- Ex:-
  - group A by Acol1, B by Bcol1.

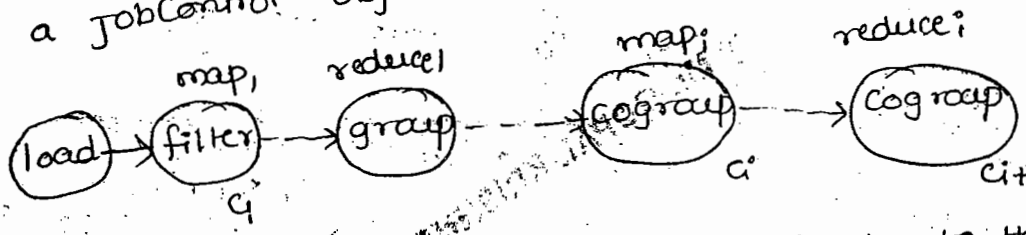
MapReduce plan :-

- physical to MapReduce (M/R) plan conversion happens through the MRCompiler.
  - converts a physical plan into a DAG of M/R operations.

- Boundaries for M/R include cogroup/group, distinct, cross, order by, limit (in some cases).

- push all subsequent operators between cogroup to next cogroup into reduce.
- order by is implemented as 2 M/R jobs.

- JobControl compiler then uses the M/R plan to construct a JobControl object.



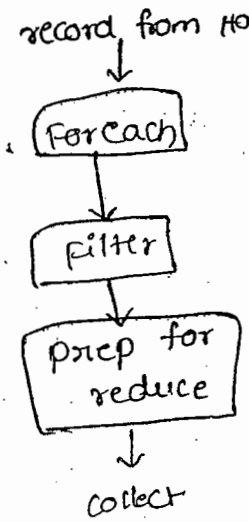
- job.jar is created and submitted to Hadoop JobControl.

Back End Architecture :-

- pig operators are placed in a pipeline and each record is then pulled through.

```
A = load ...
B = foreach...
C = filter...
D = group...
```

map phase will have foreach & filter →



Lazy Execution :-

- Nothing really executes, untill you request output.
- only when the Store, dump, explain, describe, illustrate command is encountered does the M/R execution takes place either on your cluster or front end.

Advantages :-

- In-memory pipelining.
- Filter re-ordering across multiple commands.

parallelism :-

- Split wise parallelism on map-side operators.
- By default, 1 reducer
- PARALLEL Keyword.
- group, cogroup, cross, join, distinct, order

Running modes of execution of pig :-

① Local mode

② mapReduce mode

→ **Local mode :-** To run pig in local mode, you need access to single machine. (All files are installed and run using your localhost & filesyste

In the local mode pig will expect the input from local mode & output from local mode

Syntax :-

```
pig -x
```

- Local host and local file system is used.
- useful for prototyping & debugging.

② Hadoop (MapReduce) Mode :- To run pig in Hadoop (mapreduce) mode, you need access to a Hadoop cluster and HDFS installation. mapreduce mode is the default mode.

In mapreduce mode, pig will expect the input from HDFS and produce the output from on top of HDFS.

Syntax :- pig (or) pig -x mapred

- Take a advantage of a Hadoop cluster and HDFS.

Running Pig :-

① Interactive mode (Grunt shell)

② Batch mode (script mode)

③ Java mode (embed pig statements in java)
- Keep pig.jar in the class path

① Grunt Shell :- Grunt shell is an interactive based shell. which means where exactly we will get the output than and their itself. whether it is success (or) fail.

② Script mode :- In script mode all the commands [with transformations] will be included in a single file called .pig file.

Ex :- sample.pig ↵

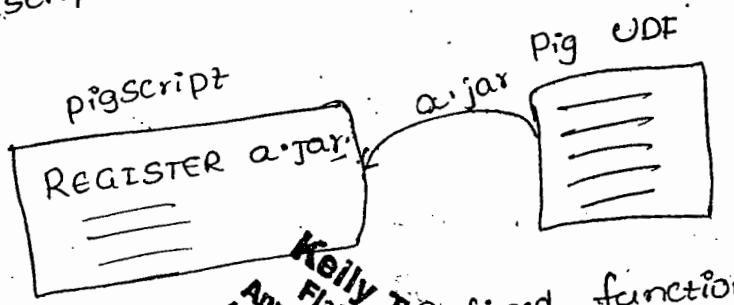
- And this executive in .pig file, the whole fill will be executed in a single shot and we will get the result.

local mode	pig -x local sample.pig
mapreduce mode	pig sample.pig

③ Embeded mode :-

- if certain things are directly possible through the transformati we can also provision of writing user defined functions (Pig Customizations) and we will the some UDF code with in our pigscript.

- In Embeded mode of pig executions with in the our pigscript we will call the Javamode.



Incorporate macros and User Defined functions :-

REGISTER → Registers JAR with the pig runtime

DEFINE → Creates an alias for a macro, UDF, Streamin script (or) command specification.

IMPORT → import macros defined in separate file into a script.

Data types of pig :- pig data types can be divided into two types.

① scalar types → contain a single value

② complex types → contain other types.

① scalar types :-

Normal language	pig
int	int
String	chararray
float	float
double	double
long	long
boolean	boolean
byte	bytearray
tuple	tuple
bag	bag
map<object, object>	map

② Complex types :-

Custom data types in pig :-

① atom :- atom is the smallest unit in apache pig. which we can denote with a single character.

(or)

Each and every character is a atom.

Ex:-

empid	ename	esal
100	xyz	10000

② Tuple :- Collection of all these fields (or) ordered set of fields.

- Field can be simple (or) Complex type
- Nested relational model.

Ex :-

empid	empname	esal
100	xyz	10000

Field :- field is nothing but collection of atoms.

Ex :-

empid	empname	esal
100	xyz	10000

③ Bag :- Collection of all these tuples is known as Bag.

Ex :-

empid	empname	esal
100	xyz	10000
101	abc	4000
102	cde	8000

④ map :- Set of (key, value) pairs.

- key → must be unique; set to be chararray
- value → can be any scalar (or) complex datatype
- maps must be enclosed by square bracket.
- Each key-value pair must be separated by #.
- key must be a chararray

Type

① Data Atom
an atomic value

② tuple
a record

③ Data Bag
Collection supporting scan

④ Data map
Collection with key lookup

Example

'alice'

('alice', 'lakers')

{ ('alice', 'lakers')
('alice', 'kings') }

['likes' # { ('lakers'), ('kings') }
'age' # 22]

PigStorage :-

- Loads or stores relations using field-delimited text format.
- Each line is broken into fields using a configurable field delimiter (defaults to a tab character) to be stored in the tuple's fields. It is the default storage when none is specified.

BinStorage :-

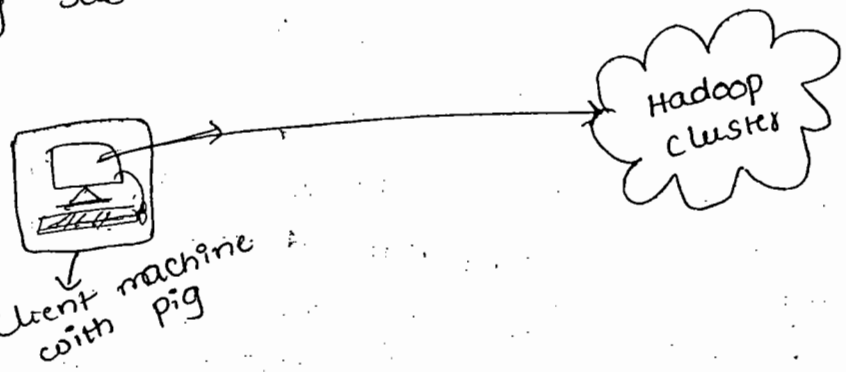
- loads or stores relations from or to binary files. An internal pig format is used that uses hadoop writable objects.
- It stores arbitrarily nested data.
- used for loading intermediate results that were previously stored using it.

TextLoader :-

- loads relations from a plain-text format. Each line corresponds to a tuple whose single field is the line of text.

where does pig live?

- pig is installed on user machine.
- No need to install anything on the Hadoop cluster.
  - pig and Hadoop versions must be compatible.
- pig submits and executes jobs to the Hadoop cluster.



what is pig used for?

ICM UW → discover knowledge from large collections of documents (academic papers)

LinkedIn :- Ex: discovering 'people you may know"

Mendeley :- Ex: finding trending keyword over time.

Nokia (Ovi Maps) :- exploring unstructured data sets from logs, database dumps and data feeds.

Real web :- computing statistics on banner views, clicks and user behaviour on target websites after click.

- weblog processing
- Data processing for web search platforms.
- Adhoc queries across large data sets.
- Rapid prototyping of algorithms for processing large data sets
- ETL
- machine learning
- Text processing (Semi, Structured)
- Data Analytics
- processing many data sources

→ what is the difference between mapReduce & pig.

mapreduce	pig
① mapReduce expects the programming language skills for writing the Business logic.	① pig there is no much of programming skills. As we are writing whole logic will making use of pig transformations (or operators.
② if we can do any change in the MR program, we need to certain problems we can change the process entire. ① compiling the program ② executing the program ③ packing up the " ④ deploying the same (cluster environment)	② In the pig, we can completely dealing with simple scripting we can avoid other transaction process. → 5% of the MRcode → 5% of the MR development time → Readable & reusable. → Increases programmer productivity → 25% of the MR execution time.
③ As a General saying of Hadoop MR program 200 lines of MR code.	③ In pig we can that type of MR program, we can write 10 lines of code.
④ mapReduce requires multiple stages, leading to long development life cycles. Rapid prototyping. Increased productivity.	④ . pig provides the log analysis . adhoc queries across various large data sets. . analytics.

Note :- Eventhough we can do processing, Internally every statement convert to a MR job.

Pig Transformations (or) Pig operators :-

Statement	Description
① load	→ read data from the file system
② Store	→ write data to the file system
③ Dump	→ write output to stdout
④ Foreach	→ Apply expression to each record and generate one or more records.
⑤ Filter	→ Apply predicate to each record and remove records where false.
⑥ Group/Cogroup	→ collect records with the same key from one more inputs.
⑦ Join	→ Joins two or more inputs based on a ke
⑧ Order	→ Sort records based on a key.
⑨ Distinct	→ Remove duplicate records.
⑩ union	→ merge datasets.
⑪ limit	→ Limit the of records.
⑫ Split	→ Split data into more sets, based on filter conditions.
⑬ cross	→ Creates the cross product of two (or) more relations.

Pig Diagnostic operators :-

Statement	Description
① Describe	→ Returns the schema of the relation
② Dump	→ results to the screen
③ Explain	→ Displays plans
④ Illustrate	→ Displays a step-step execution of a sequence of statements.

Relational operation in pig

* Filter, Foreach, Group, orderBy, Sample, Distinct, Limit, Join(inner), Join(outer).

Built-in functions in pig :-

1. EVAL
2. MATH
3. String
4. Date Time
5. Tuple, Bag, Map Specific functions

Evaluation (EVAL) functions in Pig :-

1. AVG :- Calculates the average value of entries in a bag.
2. CONCAT :- Concatenates two byte arrays or two character arrays together.
3. COUNT :- Calculates the no. of entries in a bag.
4. COUNT_STAR :- calculates the no. of elements in a bag.
5. DIFF :- compares two fields in a tuple.
6. IsEmpty :- check if a bag or map is empty.
7. MAX :- computes the Max function to maximum of the numeric values or chararrays in a single-column bag.
8. MIN :- MIN function to compute the minimum of a set of numeric values or char-arrays in a single column bag.
9. SIZE :- computes the no. of elements based on any Pig data type.
10. SUM :- compute the sum of a set of numeric values in a single-column bag.
11. TOKENIZE :- TOKENIZE function to split a string of words (all words in a single tuple) into bag of words (each word in a single tuple)

Math functions in Pig :-

① ABS :- returns the absolute value of an expression.
note :- expression result is int, long, float or double.

② ACOS :- Returns the arc cosine of an expression.
expression whose result is type double.
Acos (expression);

③ ASIN :- Returns the arc sine of an expression.

④ ATAN :- Returns the arc tangent of an expression.

⑤ CBRT :- Returns the cube root of an expression.

⑤ CEIL :- Returns the value of an expression rounded up to the nearest Integer.

⑥ COS :- Returns the trigonometric cosine of an expression.

⑦ COSH :- Returns the hyperbolic cosine of expression.

⑧ EXP :- Returns Eulers number a raised to power of x.

⑨ FLOOR :- Returns the value of an expression rounded down to the nearest Integer.

⑩ LOG :- Returns the natural algorithm (base e) of an expression.

⑪ LOG10 :- Returns the base 10 algorithm of an expression.

⑫ RANDOM :- Returns a pseudo number.

⑬ ROUND :- Returns the value of expression rounded to an integer.

⑭ SIN :- Returns the sine of an expression

(15) SINH :- Returns the hyperbolic sine of an expression.

(16) SQRT :- Returns the positive square root of an expression.

(17) TAN :- Returns the trignometric tangent of an angle

(18) TANH :- Returns the hyperbolic tangent of an expression.

Pig commands :-

(1) Load :- read the data from file system.

```
grunt > A = load 'input.txt' using PigStorage(',');
```

(2) dump :- display the output itself.

```
dump A;
```

(3) store :- write data to the file system.

```
store B into 'output';
```

(4) Foreach :- display specific column.

```
B = foreach A generate id;
```

(5) order :- display the records ascending order.

```
> B = order A by name;
```

(6) Desc :- display the records decending order.

```
B = order A by name desc;
```

⑦ Filter :- Apply predicate to each record and remove records where false.

```
B = filter A by $1 == 'Gopal' (or) name == 'Gopal';
```

⑧ Limit :- display the records from begining.

```
B = limit A 2;
```

⑨ Distinct :- Removing all duplicate values

```
B = Distinct A;
```

⑩ Group By | coGroup By :- Group the data in single Relation is called the Group By.

Group the data in two (or) more relations is called the coGroupBy.

```
B = group A by id;
C = cogroup A by id;
```


⑪ split :- splits a relation into two (or) more relati

```
split A into B if id == 100,
                C if name == 'Anitha';
```

⑫ Join :- Joins two or more relations.

```
C = JOin A by id, B by id;
```

intersection

(13) Left outer join :- Joins two (or) more inputs based on a key is LEFT.

```
C = join A by id LEFT, B by id;
```

(14) Right outer join :- Joins two (or) more inputs based on a key is Right.

```
C = join A by id RIGHT, B by id;
```

(15) Cross :- Creates the cross product of two (or) more relations.

```
C = cross A, B;
```

(16) Describe :- prints relation's schema.

```
Describe A;
```

(17) Illustrate :- Displays a step-by-step execution of a sequence of statements.

```
illustrate A;
```

(18) Explain :- prints the logical and physical plans.

```
Explain A;
```

(19) union :- Combine two or more relations into one.

```
Q = union A, B;
```

(20) FLATTEN :- unpack (or) unnest all the things.

```
B = group A by id;
C = foreach B generate group, FLATTEN(A);
```

(21) TOKENIZE :- Tokenize function Split a string of words

B = foreach A generate TOKENIZE(record);

Aggregate functions :-
↓ To make the aggregate functions compulsory — make the group.

① COUNT :- calculate the no. of entries in a bag.

B = order A by id;
C = group B by age;
D = foreach C generate group, COUNT(B);

② Sum :- calculate the sum of a set of numeric value in a single-column bag.

E = foreach C generate group, SUM(B.sal);

③ MAX :- calculate the MAX function to maximum of the numeric values or char arrays in a single column bag.

F = foreach C generate group, MAX(B.sal);

④ MIN :- compute the minimum of a set of numeric value or char-arrays in a single column bag.

G = foreach C generate group, MIN(B.sal);

⑤ AVG :- Calculates the average value of entries in a bag.

H = foreach C generate group, AVG(B.age);

⑥ CONCAT :- Concatenates two byte arrays (or) two character arrays together.

```
I = foreach C generate group, CONCAT(B.name)
```

UDF → User Defined functions.

① UDFs are required to define Custom processing.

② UDFs can be written in the Java, python, JavaScript and ruby and permit pig to support Custom processing.

③ Support for writing UDFs in python, Javascript and Ruby still evolving.

④ UDFs provide an opportunity to extend pig into your application domain.

How to write Java UDF :-

- UDF's can be developed by extending EvalFunc class and overriding exec method.

Embedded mode of pig :-

⓪ → putting one thing to another thing is called Embedded mode.

Q:- UDF String Convert to Upper case Letters ?

```
Package myudfs;
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.util.WrappedIOException;
public class UPPER Extends EvalFunc<String>
{
public String exec(Tuple input) throws IOException{
    if (input==null || input.size()==0)
     return null;
    try
     {
      String str = (String) input.get(0);
      return str.toUpperCase();
     }
    catch(Exception e){
    throw WrappedIOException.wrap("caught Exception process
    input row", e);
}
}
```


execution:- ifconfig ↵

192.168.144.129

→ pig embedScript.py

paan loads → myudfs.jar

REGISTER myudfs.jar; ← jar name.

A = load 'input.txt' using PigStorage('\t') as (id:int, name:chararray, marks:int);

B = foreach A generate myudfs.UPPER(name);
- myudfs → package name
- UPPER → class name.

dump B;

Run

pig -x local embedscript.pig

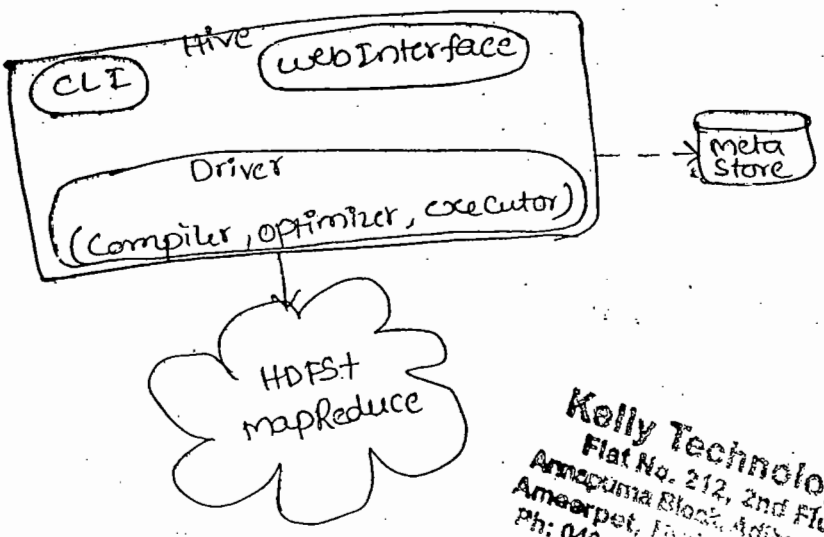
Hive

① Hive is a one of the component of the Hadoop. Hive was Initially introduced at facebook inorder to fullfill requirements with respect to ETL tools jobs.

Extract Transformation load.

- Hive is a datawarehouse system for hadoop that facilitates easy data summarization, adhoc queries, and the analysis of large datasets stored in Hadoop compatible file systems.

- Hive provides a mechanism to project structure onto this data and query the mechanism data using a SQL like language called HiveQL. At the same time this language also allows traditional map/reduce programers to plug in their custom mappers and reducers when it is inconvinient or inefficient to express this logic in HiveQL.



- Hive is an abstraction on top of mapReduce it allows users to query data in the Hadoop cluster without knowing java or mapReduce. It uses HiveQL language, very similar to SQL.

About Hive :-

- Hive was originally developed at facebook.
- provides a very SQL - like language.
- Can be used by people who know SQL.
- Enabling Hive requires almost no extra work by the system administrator.
- Hive "layers" table definitions on top of data in HDF
- Hive tables are stored in Hive's "warehouse" directory i- HDFS, By default /user/hive/warehouse.
- Tables are stored in subdirectories of the ware house directory.
- Actual data is stored in flat files - control character - de delimited text, or sequence files.

Hive is Data warehousing tool on top of Hadoop. It same as SQL :-

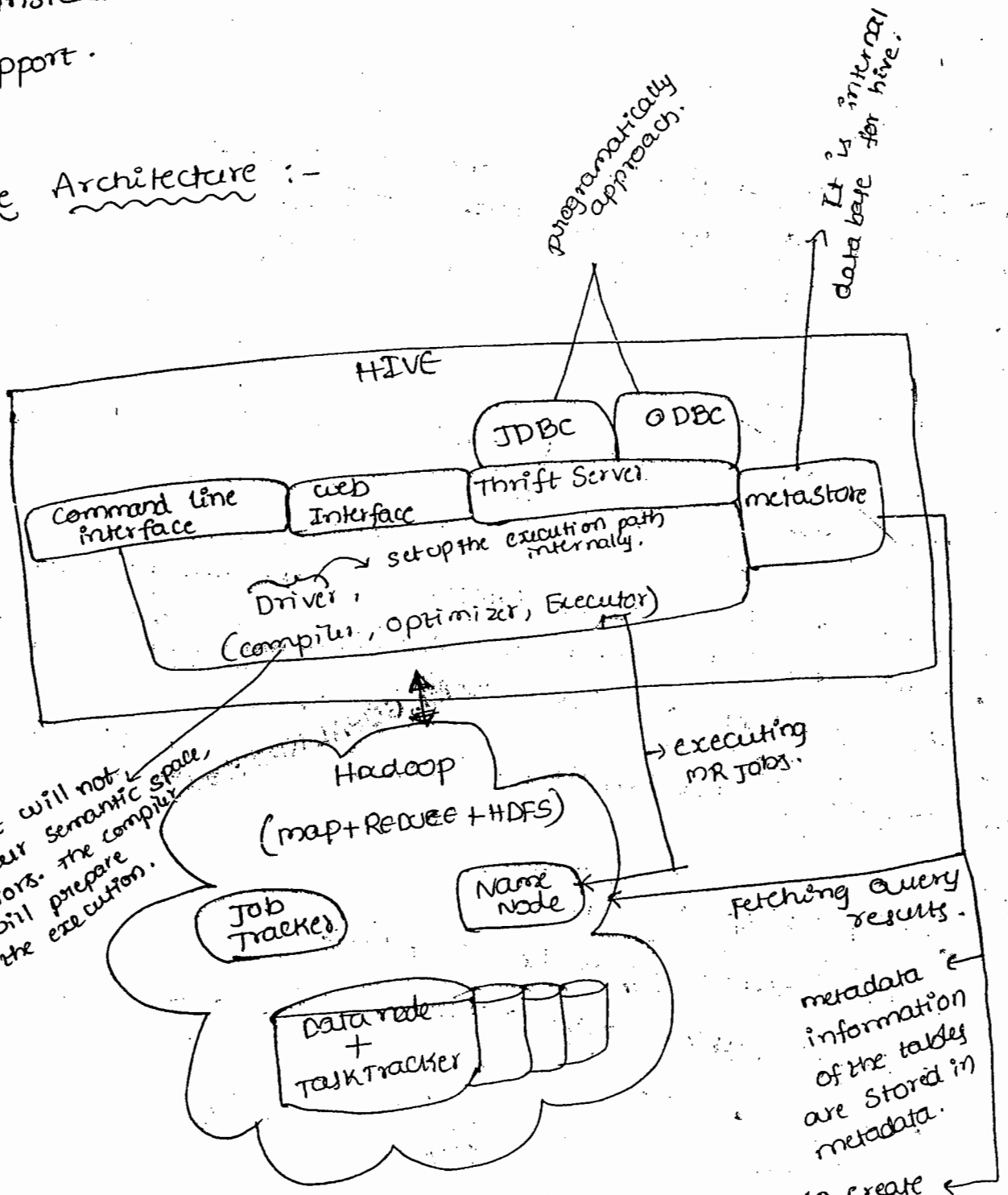
- SQL like queries
- Show tables, Describe, Drop table
- Create table, alter table
- Select, insert

Hive Limitations :-

- Not all "standard" SQL is supported.
- NO support for UPDATE or DELETE.
- NO support for INSERTing single rows

- Relatively limited no. of built-in functions.
- No datatypes for date or time - use the STRING datatype instead. In new version date or time datatype will support.

Hive Architecture :-



→ Every query will converted MR job internally.

- metastore :- Stores system catalog.
- Driver :- manages life cycle of HiveQL query as it moves thru 'HIVE'; also manages session handle and session statistics.
- query compiler :- compiles HiveQL into a directed acyclic graph of map/reduce tasks.
- Execution engines :- the component executes the tasks in proper dependency order; interacts with hadoop.
- Hive server :- provides thrift interface and JDBC/ODBC for integrating other applications.
- client components :- CLI, webInterface, Jdbc/odbc interfa
- Extensibility interface include SerDe, user Defined funct and user Defined Aggregate function.

Hive CLI

- DDL (Data Definition Language)
  - create table / drop table / rename table
  - alter table add column.
- Browsing
  - show tables
  - describe table
  - cat table
- Loading data
- Queries

web UI for Hive :-

- metastore UI :
  - Browse and navigate all tables in the System.
  - Comment on each table and each column.
  - Also captures data dependencies.

Why hive ?

- Need a multi petabyte warehouse.
- Files are insufficient data abstractions.
  - Need tables, schemas, partitions, indices
- SQL is highly popular.
- Need for an open data format
  - RDBMS have a closed data format
  - flexible schema.
- Hive is a Hadoop subproject !

Hive used for types of applications :-

- Summarization
  - Eg :- Daily/weekly aggregations of impression/click counts.
  - Complex measure of user engagement.
- adhoc analysis
  Eg :- how many group admins broken down by State/Country.
- Data mining (Assembling Training Data)
  Ex :- User Engagement as a function of user attributes.

- Spam Detection.
  - anomalous patters for site Integrity.
  - Application API usage patterns.

- Ad optimizations
- Document indexing
- Customer facing business intelligence (eg: Google analytics)
- predictive modeling, hypothesis testing.

Hive Components :-

- Shell :- allows interactive queries like mysql shell connected to database - also supports web and JDBC clients.
- Driver :- Session handles, fetch, execute.
- Compiler :- parse, plan, optimize
  - when Driver invokes compiler with HiveQL, the compiler converts string into a plan.

- plan can be
  - metadata operation for DDL statement.
  - HDFS operation for LOAD statement.
  - For insert / queries consists of DAG (Directed Acyclic Graph) of mapReduce (mr) jobs.

- parser transform query into a parse tree representation
- Semantic analyzer transform parse tree to a block based internal query representation — retrieve information of the input table from metastore verifies the column names, expand SELECT * and does type checking including implicit type conversions.

- physical plan Generator converts logical plan into physical plan consisting of DAG of map-reduce jobs.

- Logical plan Generator Converts internal query representation to a logical plan consists of a tree of logical operators.

- optimizer perform multiple passes over logical plan and rewrites in several ways.
  - Combine multiple joins which share the join key into a single multi-way JOIN → a single mapReduce job.
  - Prune columns early and pushes predicates closer to the table scan operator to minimize data transfer.
  - Prunes unneeded partitions by query.
  - For sampling query — prunes unneeded bucket.

- Execution Engine :- DAG of stages (MR, HDFS or metadata).

- MetaStore :- meta.Store is the Central repository of Hive metadata.

  The metaStore is divided into two parts.

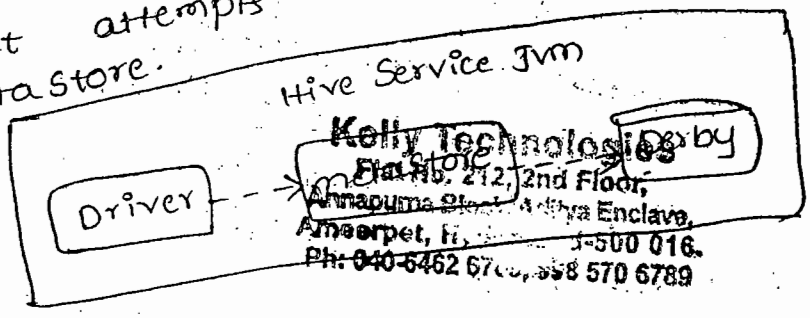
  ① Service

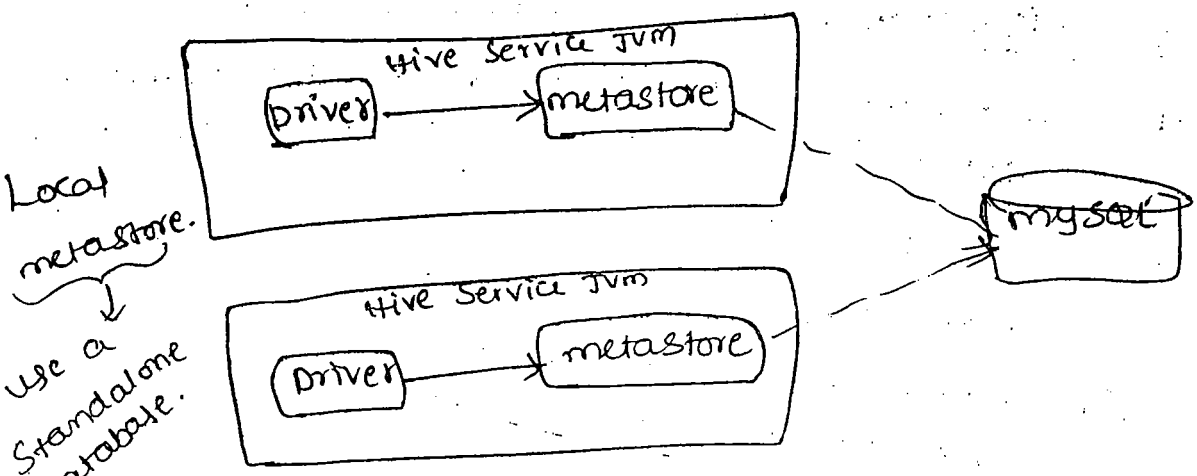
  ② backing store for the data.

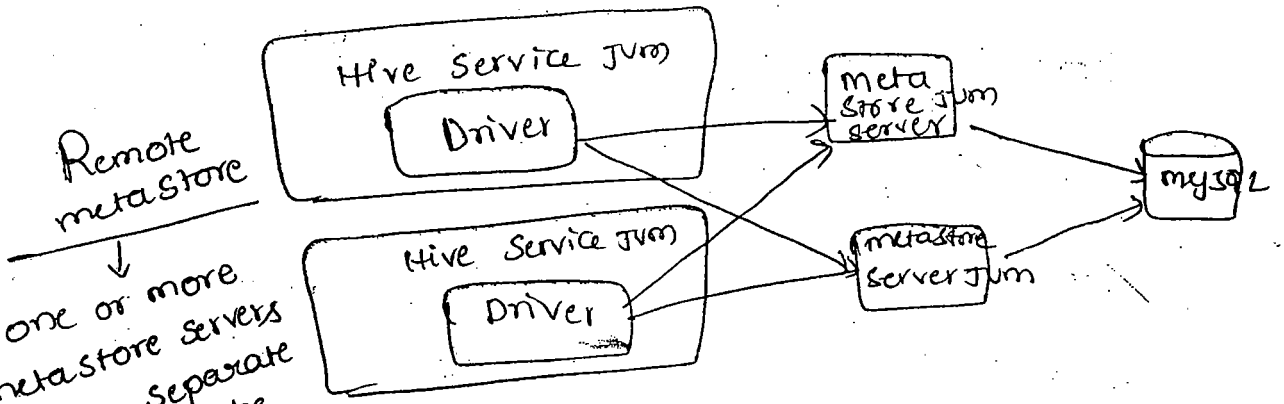
By default, metastore Service runs in the same JVM as the Hive Service and it contains an embedded Derby database instance backed by the localdisk. This is called the Embeded metaStore Configuration.

metastore configuration :-

- all the metadata information of the hive table will recide in internal database of hive so called metastore.
- metastore will provide below information.
  1. schema information of the table
  2. column information of the table
  3. table locations
  4. partition keys -- etc.
- metastore is a keyrole in hive architecture. metasto is not running, hive process the data is not possi
- without existings of metastore. concepts it is not possible process the data in hive.
- By default hive comes with derby database as it metastore. however we can very well configure other database(mysql, thrift server -- etc) as the metas of hive.
- the metastore configuration will be done in the hive configuration file "hive-site.xml"
- Failed to start database 'metastore_db' :-
- when it attempts to open a connection to the metastore.







Hive Database :-

- Data model
  - Tables :-
    - analogous to tables in relational database
    - Each table has a corresponding HDFS Dir
    - Data is serialized and stored in files within Dir
    - Support external tables on data stored in HDFS, NFS or local directory.
    - Typed columns (int, float, string, date, boolean) - also list : map (for JSON-like data).
  - ~~partitions~~

Partitions :-

- analogous to dense indexes on partition columns
- Nested Subdirectories in HDFS for each combination of partition column values.
- allows users to efficiently retrieve rows.
- @ table can have one or more partitions (1-level) which determine the distribution of data within Subdirectories of table directory.

Ex :- Table T under /wh/T and is partitioned column on ds + ctry.

For ds = 20090101
ctry = US.

Then data is stored within dir.
/wh/T/ds = 20090101/ctry = US.

Buckets :-

- split data based on hash a column - mainly for parallelism.
- Data in each partition may in turn be divided into Buckets based on the value of a hash function of some column of a table.
- Example :- ... into 32 buckets.
  - Bucket ... with 0
  - HDFS file for user ... /wh/pvs/ds = 20120410/cntr = US/part-00000
  - HDFS file for user hash bucket 20 /wh/pvs/ds = 20120410/cntr = US/part-00021

Hive physical layout :-

- ware house directory in HDFS

  Ex :- /home/hive/warehouse.

- Table stored in subdirectories of warehouse - Partitions, buckets from subdirectories of tables.

- Actual data stored in flat files - control char - delimited text, or sequence files - with custom SerDe, can use arbitrary format.

Hive datatype :-

⊙ Support primitive column types.

- Integers : TINYINT, SMALLINT, INT, BIGINT
- Boolean : BOOLEAN
- Floating point numbers : FLOAT, DOUBLE
- String : STRING.
- Date : ~~Date~~ String

⊙ Nested collections :- (complex types)

- ~~Structs : {a INT; b INT}~~
- Maps : M['group']
- Arrays : ['a', 'b', 'c'], A[1] returns 'b'.

⊙ User Defined data types :-

- user can also Define their own type programmatically.
- Structures with attributes of any type.

  Ex :- Structs : {a INT; b INT}.

Structured data :-

⊙ Type System

- primitive types
- Recursively build up using composition/maps/lists.

⊙ ObjectInspector interface for user-defined data types:- → 4

- To recursively list schema
- To recursively access fields within a row object.

⊙ Generic (De) Serialization Interface SerDe :- (Serialization/Deserial

⊙ Serialization families implement interfaces:-

- Thrift DDL base SerDe
- Delimited text based SerDe
- you can write your own SerDe (JSON, XML ---)

→ Serialization / DeSerialization :-

- Generic (De) Serialization Interface SerDe.
- used LazySerDe
- Flexible Interface to translate unstructured data into structured data.
- Designed to read data separated by different delimited characters
- That SerDes are located 'hive-contrib-jar';

Hive File Formats :-

- Hive lets users store different file formats.
- Helps in performance improvements.

SQL Example :-

Create table dept (key INT, value STRING) STORED AS

INPUTFORMAT 'org.apache.hadoop.mapred.SequenceFileInputFormat

OUTPUTFORMAT 'org.apache.hadoop.mapred.SequenceFileOutput

HiveQL :-

- Support SQL-like query language called HiveQL for select, join, aggregate, union all and sub query in the from Cause.
- Support DDL stmt such as CREATE table with serialization format, partitioning and bucketing columns.
- Command to load data from external sources and INSERT into HIVE tables.
- Do not support UPDATE and DELETE.
- Support multitable INSERT
- Support user-Defined column transformation (UDF) and aggregation (UDAF) function written in Java.

Hive VS pig

Hive	pig
① Language is SQL-like	① Language is pig latin.
② Schema: Table definitions that are stored in a metastore	② A schema is optionally defined at runtime.
③ Hive programatical access is JDBC, ODBC.	③ pig access is pigServer.
④ In the hive have partitions.	④ There is no partitions
⑤ Server is optional (Thrift)	⑤ no Server.
⑥ UDFS (Java)	⑥ UDFS (Java)
⑦ Custom Serializer/Deserializer	⑦ Custom Serializer/Deserializer

⑧ DFS direct access at run time

⑨ Join/order/set is possible

⑩ Shell command Interface is possible

⑪ Streaming is supported

⑫ webInterface is possible

⑧ DFS direct access at default.

⑨ Join/order/set is possi

⑩ Shell command inter have.

⑪ Streaming is Support

⑫ There is no web Interface.

=

→ The entire data of hive will be reciding in two types of tables.

① managed tables

② External tables

① managed tables :- managed tables are the hive default tables i.e these tables will be completely manage by hive warehouse only.

∴ All the managed tables will recide under the subfolder of the default hive directory is known as

/user/hive/warehouse

② External tables :- As the name suggested, these tables are hive external warehouse. i.e wherever the external tables that we are creating will recide in an external path will hive warehouse directory.

Syn:- create external table <table-name>>
(------ keyword it is
------ treated as external table.
------ otherwise if it is not
used that keyword it is
treated as Managed ta
) Location <<HDFS path>

→ what is the difference between Managed Tables & External tables.

Managed Table

① point to existing data directories in default hive warehouse.

② Can not create partitions.

③ Data is not assumed to be in Hive Compatiable format.

④ Dropping managed table, drops the both metadata and data.

⑤ managed tables are stored in /user/hive/warehouse.
hive default path.

Ex:- Create table emp(id int, name string

hive>
create table emp(empid int,
ename string,
esal double)
rowformat delimited fields
terminated by '\t' stored
as textfile ↵

external table

① point to existing data directories in HDFS

② Can create table and partitions

③ Data is assumed to be in Hive compatiable format.

④ Dropping external table, drops the only metadata.

metadata → no of columns,
types of columns,
terminators.

⑤ external tables are stored in external path of the hive warehouse directory.

Ex:- hive > ~~create~~
Create External table emp
(empid int,
ename String,
esal double) location
'/Extritablelocation';
↓
It will create by default by hive.

Hive Commands :-

To launch the Hive shell, start a terminal and run $hive

Note :- example is the table name for all query.

hive >

Hive : Creating Tables

⊙ Create table example(id INT, name STRING) ROW form delimited fields terminated by '\t' stored as textfile

→ Describe example;

→ show tables;

Hive : loading Data into Hive :-

Data is loaded into Hive with LOAD DATA INPATH statement - Assumes that the data is already in HDFS.

hive > Load data inpath 'file-txtdata.txt' into table example;

if the data is on the localfile system, use load data local inp automatically loads it into HDFS.

hive > load data local inpath "file-txtdata.txt" into table exam

Hive : Select queries :

Hive supports most fam... syntax.

hive > Select * from example LIMIT 10;

hive > Select * from example where id > 100 ORDER BY name ASC Limit 10;

Joining Tables :-

SELECT e.name, e.dep, s.id FROM example e JOIN sample s ON (e.dep = s.dep) where e.id >= 20;

Creating user-defined functions :-

Insert overwrite table u_data_new

Select Transform(userid, movieid, rating, unixtime) Using 'python weekday_mapper.py' AS (userid, movieid, rating, weekday) From u_data;

Join Query : sample

Select A * from example A Join(Select id, max(start-time) as start-time from example B.

where start-time <25 group by id) MAXSP

ON A.id = MAXSP.id and A.start-time = MAXsp.start-time;

using NOT IN/IN hive query :-

Select * FROM example where not array_contains(array(7,6,5,4,2,1), id).

Alter tables in Hive :-

hive > alter table example RENAME to new example;

hive > alter table newexample ADD COLUMNS(doj string, emplike int);

partitions :-

ive > create table partitab(logid int, logerror string) PARTITIONED BY (logdate string, columns string) row format delimited fields terminated by '\t' lines terminated by '\n' stored as textfile.

→ hive supports 3 fileformats.

① Textfile
② Sequence file
③ Record character file...

→ CTAS :- (Create table as select)

hive > create table CTASTab as select id, name, sal from new managed tab;

→ JOIN :- Create table merge as select n.id, e.name, e.sal from newmanagedtab n JOIN empmanaged e ON (n.id = e.id);

→ LEFT OUTER JOIN :-

> create table merge1 as select n.id, e.name, e.sal from newmanagedtab n LEFT OUTER JOIN empnewmanagedtab e on (n.id = e.id).

2) → partitions :- schema information is same.

these columns not at all be inputfile.

partition columns information provided at run time or

Distribute by :-

① Distribute by controls how map output will be divided among Reducers.

② In the below example DISTRIBUTE BY ensures that records of ename will go to the same reducer and then to SORT the same data the way we want (ascending order of empid)

③ Distribute by works similar to GroupBy in the sense that how it controls the reducers to receives the rows for processing.

Note :- Hive requires Distribute by clause comes Before the sort by if we are using both in a query.

hive > select * from disttab.

> Select empid, ename, esal from disttab DISTRIBUTE BY ename SORT BY ename asc, esal asc;

CLUSTER BY :-

① CLUSTER BY clause is a short-hand way of expressing the DISTRIBUTE BY with SORT BY.

Ex:- hive > select empid, ename, esal from disttab
CLUSTER BY ename;

ORDER BY :-

① The ORDER BY clause is familar from othe SQL dialects.

② It performs the 'total ordering of the query result set' i.e 'all the data will be passed to a single reducer' which will take longer time if we have larger datasets.

③ Hive requires the clause for ORDER BY as it will take longer time (if hives property hive.mapred.mode is set to 'strict' which is by default 'nonstrict'.

SORT BY :-

① where as SORT BY ("Hive's alternative to ORDER BY), will orders the data only within each reducer and performs the 'local ordering of the data' where each REDUCER'S output will be SORTTED.

② Better performance is traded for total ordering.

Ex:- Select empid, ename, esal from testdf order by empid asc, esal desc;

Ex:- Select empid, ename, esal from testdf Sort by empid asc, esal desc;

Indexes In HIVE:-

① > Create index partitab_Index on table partitab(logid) as 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' with DEFERRED REBUILD;

② Create table tt(i int, j int);

Create Index X ON Table tt(j) As 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' with DEFERRED REBUILD;

→ UDF

```
package com.hive.gopal;
import org.apache.hadoop.hive.ql.exec.UDAF;
import org.apache.hadoop.hive.ql.exec.UDAFEvaluator;
import org.apache.hadoop.io.IntWritable;
public class Maximum extends UDAF{

public static class MaximumIntUDAFEvaluator implement
        UDAFEvaluator{

    private IntWritable result;
    public void init(){
        result = null;
    }
    public boolean iterate(IntWritable value){
        if (value == null)
            return true;
    }
```

```
if (result == null) {
    result = new IntWritable (value.get());
}
else {
    result.set (Math.max (result.get(), value.get()));
```

Execution :-

hive > add jar maximum.jar

> create temporary function maximum as 'com.hive.gopal.Maximum';

Sqoop create Hive-table :-

sqoop create emptable --connect jdbc:mysql://localhost/Gopal --table student ~~--emp-table~~ --hive-table GopalStudent;

sqoop hive-import :-

sqoop import --connect jdbc:mysql://localhost/Gopal --table student --hive-import;

HBase

What is HBase ?

HBase is a database: the Hadoop database. It is indexed by rowKey, columnKey and timeStamp. HBase stores Structured and SemiStructured data naturally so you can load it with tweets and parsed log files and a catalog of all your products right along with their customer reviews. It can store unstructured data too, as long as it's not too large.

HBase is designed to run on a cluster of computers instead of single computer. The cluster can be built using commodity hardware. HBase scales horizontally as you add more machines to the cluster. Each node in the cluster provides a bit of storage, a bit of cache, and a bit of computation as well. This makes HBase incredibly and forgiving. No node is unique, so if one of those machines breaks down, you simply replace it with another. This adds upto a powerful, Scalable approach to data that, until now, has not been commonly available to more mortals.

How can HBase help ?

1. Replicate dimension tables from transactional databases with low latency and without sharding.
   (Fact data can stay in Hive since it is append only)
2. only move changed rows
   "Full scrape" is too slow does not scale as data keeps growing.

- Hive by itself is not good at row-level operations.

③ Integrate into Hive's map/reduce query execution plans for parallel distributed processing.

④ multiversioning for snapshot consistency?

Why HBase?

① we own infrastructure, no usage limits.

② Data model
- Semi structured data in HBase (easily handles multiple types in same table)
- Time-Series ordered.
- Scaling is built-in (just add more servers)
- But extra indexing is DIY.

③ very active developer community.

④ Established, mature project (in relative terms!)

⑤ matches our own toolset (Java/linux based)

→ HBase: (HDFS + random read/write)

① clone of Google's Bigtable
- Distributed (automatic partitioning)
- column-oriented.
- Semi-Structured (columns can be added just by inserting)
- Built-in versioning.

② NOT an RDBMS
- NO JOINs
- NO SQL
- Data usually not normalized
- Transactions & built in Secondary indexes available (as contrib) but immature.

⊙ need to think differently how about how you structure data.

- Denormalize your data where necessary.
- Structure data & row keys around common access.

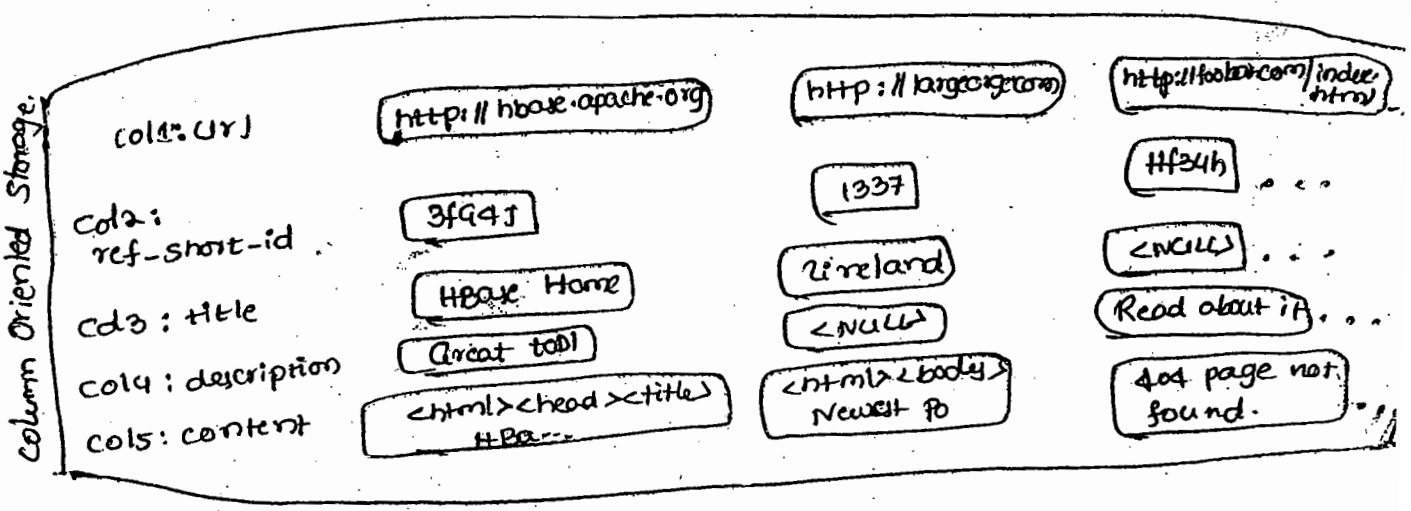
HBase Data Storage :-

* HBase is column-oriented data storage.

* column-oriented means

- The reason to store values on a per column basis instead is based on the assumption.
- that for specific queries, not all of the values are needed.

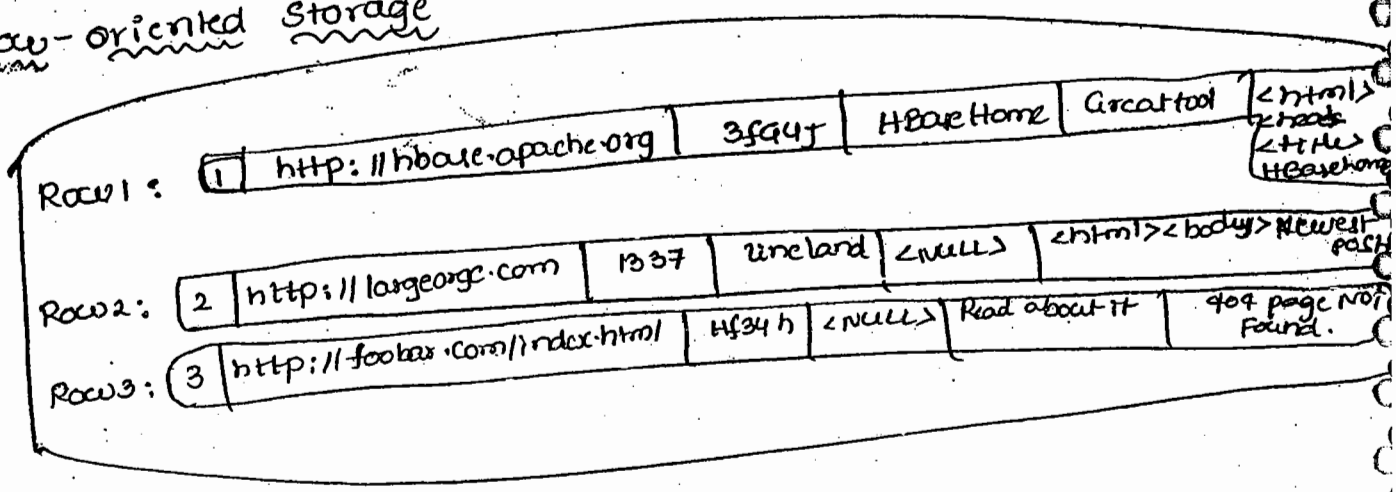
: Reduced I/O



- Column oriented databases save their data grouped by columns. Subsequent column values are stored contigously on disk. The differs from the usual row-oriented approach of traditional databases, which store entire rows contigously

- The reason to store values on a per column basis instead is based on the assumption that, for specific queries, not all of the values are needed. This is often the case in analytical databases in particular, and therefore they are good candidates for this different storage schema.
- Reduced I/O is one of the primary reasons for this new layout, but it offers additional advantages playing into the same category: since the values of one column are often very similar in nature of even vary only slightly between logical rows they are often much better suited for compression than the heterogeneous values of a row-oriented record structure; most compression algorithms only look at a finite window. Specialized algorithms - for example, delta and/or prefix compression - selected based.

Row-oriented storage



HBase Data model :-

- Tables are sorted by Row
- Table schema only define it's column families
  - Each family consists of any no. of columns.
  - Each column consists of any no. of versions.
  - columns only exist when inserted, NULL are free.
  - Columns with in a family are sorted and sorted together.
- Everything except table names are byte[]
- (Row, Family: Column, Timestamp) → value.

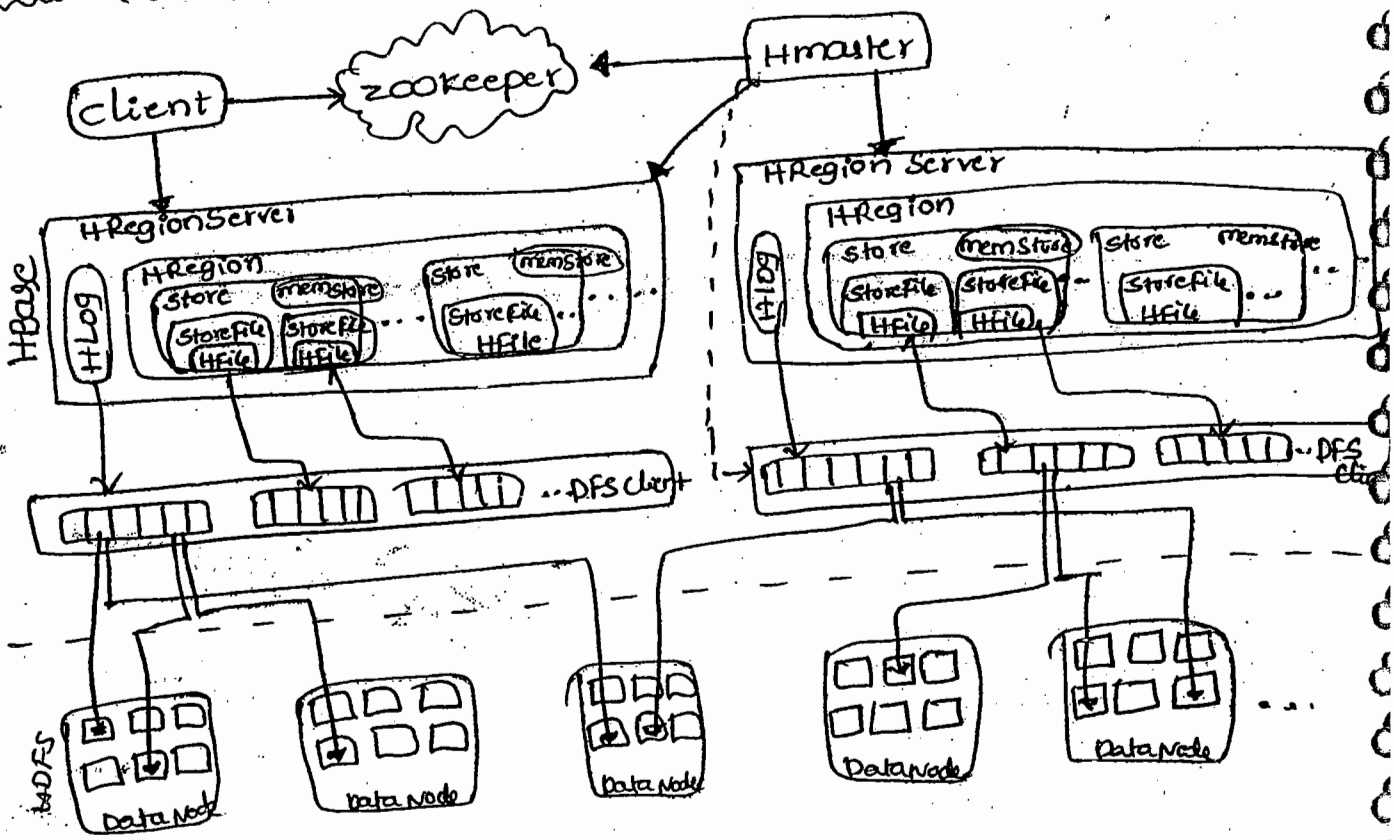
RowKey	ColumnKey	Timestamp	CCI
Com.cnn.www	anchor:cnnsi.com	t9	CNN
Com.cnn.www	Anchor:my.look.ca	t8	CNN.com
Com.cnn.www	Contents:	t6	<html>...
Com.cnn.www	Contents:	t5	<html>...
Com.cnn.www	Contents:	t3	<html>..

Column Families

- Different sets of columns may have different properties and access patterns.
- Configurable by column family.
  - Compression (none, gzip, LZO)
  - Version retention policies
  - Cache priority.
- CF's stored separately on disk. access one without costing IO on the other.

- Row's columns are grouped into families.
- column family members identified by a common 'printable' prefix.
- column family should be predefined.
  - but column family members can be added dynamically
  - member name can be bytes.

HBase Architecture :-



Components

→ HBase master

→ HRegion Server

→ client

① Hmaster :-

- master server is responsible for monitoring all RegionServer instances in the cluster, and is the interface for all metadata changes, it runs on the server which hosts namenode.
- master controls critical functions such as RegionServer failover and completing region splits. so while the cluster can still run for a time without the master, the master should be restarted as soon as possible.
- master does not handle write any request. (not a DB master)
- Does not handle location finding requests.
- not involved in the read/write path.
- Even master(s) is (are) down, cluster can response to write/read request.
- Generally does very little most the time.

master is stateless
- all the data & state into stored in HDFS & zookeeper.
- Master is not SPOF!

② HRegionServer :-

Table : HBase Table

Region : Regions for the table.

Store : Store per column family for each region for the table.

Memstore : Memstore for each store for region for the table.

StoreFile : Store files for each state for each Region for the table.

Block : Blocks within a Store File within a Store for each region for the table.

Hlog used for recovering.

→ Send heartbeat (loadinfo) to master
→ write requests handle
→ read request handle.
→ flush
→ compaction
→ Region splits. (merge)

Regions :-

① horizontal partitioning
② Every region has a subset of the table's rows.
③ Region identified as
[- table -> first row(+), last row(-)]
④ Table starts on a single region
⑤ Split into two equal sized regions as the orginal region grows bigger and so on...

zookeeper :-

- master election and Server availability.
- cluster Management
  - assignment transaction state Management.
- client contacts zookeeper to bootstrap connection to the HBase cluster.
- Region key ranges, region server address.
- Guarantees consistency of data across clients.

HBase client

- The HBase client is responsible for finding RegionServers that are serving the particular row range of interest.
- It does this by querying the .META. and -ROOT- catalog tables in ZooKeeper.
- After locating the required region(s), the client directly contacts the RegionServer serving that region.

HBase Tables and Regions:-

- HBase table is made up of roughly equal sized regions.
- Each region may live on different node and is made up of several HDFS files and blocks, each of which is replicated by Hadoop.
- Region is specified by its startkey and end key.

Accessing HBase

- Java API (thick client)
- REST/HTTP.
- Apache Thrift (any language)
- Hive/pig for analytics

HBase API

- get(row)
- put(row, Map<column, value>)
- scan(keyrange, filter)

- increment (row, columns)
- ... (checkAndput, delete - etc)
- MapReduce / Hive

HBase vs RDBMS

RDBMS	HBase
① row-oriented.	① column-family oriented
② multi-row ACID	② single row only.
③ SQL	③ get / put / Scan / etc *
④ Authentication / Authorization on arbitrary columns	④ work in progress.
⑤ TB's	⑤ Row-key only.
⑥ 1000s queries / second	⑥ ~1 PB.
	⑦ Millions of queries / second

HBase vs NOSQL

- Favors Consistency over availability (but availability is good in practice).
- Great Hadoop integration (very efficient bulk loads, MapReduce analysis)
- Ordered range partitions (not hash)
- Automatically shards / scales (just run on more servers)
- Sparse column storage (not key-value)

HBase vs HDFS

HDFS/~~XXR~~	HBASE
append-only	Random write, bulk incremental.
Full table scan, partition table scan	Random read, small range scan, or table scan.
~~Hive~~ performance very good in Hive	4-5 x slower.
structured storage (Sequence file)	Sparse - column - family data model.
max data size (30+PB)	~1PB.

Client side Buffering in HBase :-

HBase:

HBase is an open source, non-relational, distributed database modeled after google's. It runs on top of Hadoop distributed file system, providing Big table like capabilities for Hadoop. Hbase provides a way to store large amounts of data in a fault-tolerant way. Hbase can be also termed as the hadoop database.

we can perform operations like insertion into HBase using put method. put operation - is like an Rpc that transfers data from client to server and back. when the no. of insertions are less, it is feasible but when we need to have like 1000 insertions per second, it's better that we go for buffering.

Client Side Buffering:-

HBase API has a built in client side buffer that can buffer all the puts and sent it in a single RPC call. The following methods are used for this purpose.

```
void setAutoFlush(boolean autoFlush)
boolean isAutoFlush()
```

we need to set the 'autoFlush' to 'false' to enable the client side buffer as by default it's not enabled.

```
table.setAutoFlush(false)
```

once this has been done, we can check the state of the 'autoFlush' flag using the isAutoFlush() method. Now, all the put instances we created are stored in the memory in the client process. To force the data out of the buffer, we need to call another function provided by API


```
void flushCommits() throw IOException
```

This method flushes the data from buffer and ships it to the servers. client batches the puts and send them to the appropriate region servers in a single RPC call. you also have the option of setting the bufferSize (client-side write buffer). In this case, API will track the size of data buffered and the required heap size for each instance. So it tracks the total overhead of the data. when it goes beyond a certain limit, it will implicitly flush the data. we can configure the max client side buffer size using the following.

void setWriteBufferSize(long writeBufferSize) throws IOException

- you can retrieve the client side buffer size using getWriteBufferSize() method in the API. the default size is 2MB.

- the data is hence flushed in 2 cases:

(a) Implicit Flush:-

It's triggered whenever put() or setWriteBufferSize() is called. This will cause the API to compare the current buffersize with the configured maximum buffersize. if it exceeds the limit, then it triggers the flush. if the client buffer is disabled then each invocation of the put() will cause a differ buffer flush. close() also triggers buffer flush.

(b) Explicit Flush:-

The flush commits() is called explicitly to clear the buffer and sent the data to the server.

Here's an example of client side buffering,

```
HTable table = new HTable(config, "example-table");
table.setAutoFlush(false); //setting autoflush to false to
                           enable buffering.
Put put1 = new put(Bytes.toBytes("row-1"));
put1.add(Bytes.toBytes("column-fam1"), Byte.toBytes("column-
    equal"),
Bytes.toBytes("value-1"));
table.put(put1);
```

```
put put2 = new put(Bytes.toBytes("row-2 put1.add(Bytes.toBytes(v
Bytes.toBytes("column-qual2"),                                  colum
Bytes.toBytes("value-2"));                                      fam
  table.put(put2);

put put3 = new put(Bytes.toBytes("row-3"));
  put1.add(Bytes.toBytes("column-fam3"),
  Bytes.toBytes("column-qual3"),
  Bytes.toBytes("value-3"));
  table.put(put3);

Get get = new Get(Bytes.toBytes("row-1"));
Result res1 = table.get(get);
  table.flushCommits(); // an explicit flush is performed here.
```


→ In this example, we are creating puts one by one and inserting into the table.

→ When you need to access the write buffer content, you can use

```
ArrayList<put> getWriteBuffer().
```

→ This will return the internal list of the put instances that have been buffered so far by invoking table.

→ In the previous example, the put instances were being added one by one. we can also create a list of put

```
void put(List<put> puts) throws IOException
```

HBase shell mechanism :-

```
hbase > create 'test', 'cf'

hbase > put 'test', 'row1', 'cf:a', 'value1'

hbase > scan 'test'
```

Row	Column + Cell
row1	column = cf:a, timestamp = 1332311427315, value = value1.

Java mechanism :-

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.util.Bytes;

public class HBaseClient {

    public static void main(String[] args) throws IOException{

Configuration config = HBaseConfiguration.create();
config.set("hbase.zookeeper.quorum", "hadoop1");
HTable testTable = new HTable(config, "test");
for(int i=0; i<100; i++){
byte[] family = Bytes.toBytes("cf");
 byte[] qual = Bytes.toBytes("a");
Scan scan = new Scan();
  scan.addColumn(family, qual);
ResultScanner rs = testTable.getScanner(Scan);
for( Result r = rs.next(); r != null; r = rs.next()){
 byte [] valueObj = r.getValue(family, qual);
  String value = new String(valueObj);
  System.out.println(value);
} } testTable.close(); } }
```

Dependent JAR files :-

→ commons-configuration-1.6.jar
→ commons-lang-2.5.jar
→ commons-logging-1.1.1.jar
→ hadoop-core-1.0.0.jar
→ hbase-0.92.1.jar
→ log4j-1.2.16.jar
→ slf4j-api-1.5.8.jar
→ slf4j-log4j12-1.5.8.jar
→ zookeeper-3.4.3.jar.

→ HBase shell commands are mainly categorized into 6 parts.

① General HBase shell commands
② Tables Management commands
③ Data Manipulation commands
④ HBase Surgery tools
⑤ Cluster replication tools
⑥ Security tools.

① General HBase shell commands :-

Status :- Show cluster status. Can be 'summary', 'simple', or 'detailed'. The default is 'summary'.

hbase > status
> status 'simple'
> status 'summary'
> status 'detailed'.

Version :- output this HBase version.

hbase > version.

Who am i :- Show the Current hbase user.

```
hbase > whoami
```

② Tables management Commands :-

```
mrinmoy@ubuntu:~$ su root ↵
password: ' ↵
root@ubuntu:/home/Gopal # JPS ↵
    1100 DataNode
    1745 QuorumPeerMain
    1702 TaskTracker
    1443 NameNode
    1311 JobTracker
    2251 JPS
    1590 Secondary NameNode.
```


How to start HBase :-

```
# Kill 1745
# cd /usr/bin/hbase/bin/ ↵
root@ubuntu:/usr/lib/hbase/bin # ./start-hbase.sh ↵
starting master, logging to /usr/lib/hbase/bin/../logs/hbase-
root-master-ubuntu.out ↵
root@ubuntu:/usr/lib/hbase/bin # JPS ↵
    1100 DataNode
    2353 Master
    1702 TaskTracker
    1443 NameNode
    2407 JPS
    1311 JobTracker
    1590 Secondary NameNode
```


HBase

HBase Shell Commands :-

root@ Ubuntu : /usr/lib/hbase/bin # hbase shell ↵

hbase > version ↵
hbase > whoami ↵
hbase > status;
~~Hbase >~~

To see the list of tables in HBase :-

hbase > list ↵

How to create a table in HBase :-

hbase > create 'Kelly', 'Hadoop'
(Kelly → table name; Hadoop → column family name.)

hbase > create 'HBaseTab', 'KellyHadoop', 'GopalHadoop', 'SteHadoop'

Describe Command :-

① hbase > describe 'Kelly'
② hbase > describe 'HbaseTab'.
③ hbase > Slect * from table --> equivalent.
④ hbase > scan 'Kelly'
⑤ How to enable and disable the tables

hbase > enable 'Kelly' ↵
hbase > disable 'Kelly' ↵

Altering tables :-

hbase > alter 'Kelly', NAME => 'NEWHADOOP', VERSIONS = 6

hbase > alter 'Kelly', NAME = 'NEWHADOOP', VERSIONS = 6

hbase > describe 'Kelly' ↵

How should we know whether a table is enabled or disabled ?

```
hbase > is_enabled 'Kelly'
      > is_enabled 'HBaseTab'
```

Is table exists or not ?

```
hbase > exists 'kelly'
      > exisH 'HBaseTab'
      > exisH 'HBaseTable.
```



```
hbase > create 'table1', 'colFam1'
      > put 'table1', 'row1', 'colFam1:colName1', 'FirstValue'
      > put 'table1', 'row1', 'colFam1:colName2', 'FirstValue'
      >put 'table1', 'row2', 'colFam1:colName3', 'FirstValueFirst'
      >put 'table1', 'row3', 'colFam1:colName4', 'SecondValueFirst'

hbase > scan 'table1';
      > scan 'table1'
```


How to Insert (Load) the data in table having multiple Column Families (CFs) :-

```
hbase > create 'sampletable', 'colFam1', 'colFam2', 'colFam3'.
hbase > put 'sampletable', 'rowKey1', 'colFam1:colName1', 'column-
        value'
hbase > put 'sampletable', 'rowKey2', 'colFam2:colName1', '
        Column-1 value in ColumnFamily2'
```

hbase > scan 'sampletable' ↵

How to get the values from the table :-

hbase > get 'sampletable', 'rowkey3'

Count :-

hbase > count 'sampletable' ↵

Delete :-

hbase > delete 'sampletable', 'rocokey2', 'colFam2:colNam1

that have a copy of that block and the datanodes are sorted according to their proximity to the client

c) The DFS returns a FSDataInputStream (an input stream that supports file seeks) to the client for it to read data from.

d) The client then calls read() on the stream

e) DFSStream connects to the first (closest) datanode for the first block in the file.

f) Data is streamed from the datanode back to the client, which calls read() repeatedly on the stream

g) When the end of the block is reached, DFSInputStream will close the connection to the datanode, then find the best datanode for the next block

h) When the client has finished reading, it calls close() on the FSDataInputStream

2) Writing Data Flow:

a) The client creates the file by calling create() on DFS

b) DFS makes an RPC call to the namenode to create a new file in the filesystem's namespace, with no blocks associated with it

c) The namenode performs various checks to make sure the file doesn't already exist, and then The DFS returns aFSDataOutputStream for the client to start writing data to

d) As the client writes data, DFSOutputStream splits it into packets, which it writes to an internal queue, called the data queue.

e) The data queue is consumed by the DataStreamer, whose responsibility it is to ask the namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas. The list of datanodes forms a pipeline

f) The DataStreamer streams the packets to the first datanode in the pipeline, which stores the packet and forwards it to the second datanode in the pipeline. Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipeline

DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowledged by datanodes, called the ack queue.

A packet is removed from the ackqueue only when it has been acknowledged by all the datanodes in the pipeline

What if a datanode fails while data is being written to it.

a) The pipeline is closed

b) Any packets in the ackqueue are added to the front of the data queue so that datanodes that are downstream from the failed node will not miss any packets.

c) The partial block on the failed datanode will be deleted if the failed datanode recovers later on thefailed datanode is removed from the pipeline and the remainder ofthe block's data is written to the two good datanodes in the pipeline.

d) The namenode notices that the block is under-replicated, and it arranges for a further replica to be created on another node

HDFS Administration

As a administrator you must know all the basic configurations, admin commands, upgrading and health check related commands.

Namenode directory structure:

After formatting namenode creates the following directory structure:

```
kelly1@kelly1-LIFEBOOK-LH530:~/dfs/name$ ll
total 20
drwxr-xr-x 5 kelly1 kelly1 4096 2012-09-19 20:23 ./
drwxr-xr-x 5 kelly1 kelly1 4096 2012-09-13 00:00 ../
drwxr-xr-x 2 kelly1 kelly1 4096 2012-09-19 20:23 current/
drwxr-xr-x 2 kelly1 kelly1 4096 2012-09-12 23:58 image/
-rw-r--r-- 1 kelly1 kelly1    0 2012-09-19 20:23 in_use.lock
drwxr-xr-x 2 kelly1 kelly1 4096 2012-09-18 23:19 previous.checkpoint/
kelly1@kelly1-LIFEBOOK-LH530:~/dfs/name$ cd current/
kelly1@kelly1-LIFEBOOK-LH530:~/dfs/name/current$ ll
total 24
drwxr-xr-x 2 kelly1 kelly1 4096 2012-09-19 20:28 ./
drwxr-xr-x 5 kelly1 kelly1 4096 2012-09-19 20:23 ../
-rw-r--r-- 1 kelly1 kelly1    4 2012-09-19 20:28 edits
-rw-r--r-- 1 kelly1 kelly1 3198 2012-09-19 20:28 fsimage
-rw-r--r-- 1 kelly1 kelly1    8 2012-09-19 20:28 fstime
-rw-r--r-- 1 kelly1 kelly1  100 2012-09-19 20:28 VERSION
```

edilog--->filesystem metada(for every fs operation)

fsimage(periodically updated)

When namenode goes down, the latest state of its metadata can be reconstructed by loading the fsimage from disk into memory, then applying each of the operations in the edit log.

This is what happens when namenode restarts.....called as SafeMode

Safe Mode:

When the namenode starts, the first thing it does is load its image file(fsimage) into memory and apply the edits from the edit log(edits). Once it has reconstructed a consistent in-memory image of the filesystem metadata, it creates a new fsimage file and an empty edit log. Only at this point does the namenode start listening for RPC and HTTP requests. However, the namenode is running in safe mode, which means that it offers only a read-only view of the filesystem to clients.

The below properties are related to hadoopsafemode:

1) dfs.replication.min
2) dfs.safemode.threshold.pct
3) dfs.safemode.extension

Commands for entering and leaving from safemode:

1) To check whether namenode is in safemode or not:

% hadoopdfsadmin -safemode get

Safe mode is ON

The front page of the HDFS web UI provides another indication of whether the namenode is in safe mode.

2) When you want to wait for the namenode to exit safe mode before carrying out a command, particularly in scripts.

% hadoopdfsadmin -safemode wait
3) To enter safe mode, use the following command:
% hadoopdfsadmin -safemode enter
Safe mode is ON
[This is useful while upgrading the cluster]
4) To make the namenode leave safe mode:
% hadoopdfsadmin -safemode leave
Safe mode is OFF
fsimage contains a serialized form of all the directory and file inodes in the filesystem. Each inode is internal represenation of a file or directory's metadat, and contains such information as the file's replication level, modification and access times, access permissions, block size, and blocks a file made up of. For directories, the modification time, permissions, and quota metadata is stored.
The fsimage file does not record the datanodes on which the blocks are stored.Instead the namenode keeps this mapping in memory, which it constructs by asking the datanode for their block lists when they join cluster and periodically afterward to ensure the namenode's block mapping is up-to-date.
The edits file would grow without bound. Though this state of affairs would have no impact on the system while the namenode is running, if the namenode were restarted, it would take a long time to apply each of the operations in its edit log. During this time the filesystem would be offline, which is generally undesirable.

The solution is to run the secondary namenode, whose purpose is to produce check points of the primary's in memory filesystem metadata(This is applicable onle the versions <= 0.21.0). The checkpointing process proceeds as follows.
i) The secondary asks the primary to roll its edits file, so new edits go to a new file.
ii) The secondary retrieves fsimage and edits from the primary(using HTTP GET).
iii) The secondary loads fsimage into memory, applies each operation from edits, then creates a new consolidated fsimage file.
iv) The secondary sends the new fsimage back to the primary(using HTTP POST).
v) The primary replaces the old fsimage with the new one from the secondary, and the old edits file with the new one it started in step 1. It also updates the fstime file to record the time that the checkpoint was taken.
At the end of the process, the primary has an up-to-date fsimage file, and a shorter edits file (it is not necessarily empty, as it may have received some edits while the checkpoint was being taken). It is possible for an administrator to run this process manually while the namenode is in safe mode, using the "hadoopdfsadmin -saveNamespace" command.
The secondary has similar memory requirements to the primary(since it loads the fsimage into memory), which is the reason that the secondary needs a dedicated machine on large clusters.
The schedule for checkpointing is controlled by two configuration parameters. The secondary namenode checkpoints every hour(fs.checkpoint.period in seconds) or sooner if the edit log has reached 64MB(fs.checkpoint.size in bytes), which it checks every five minutes.

Secondary namenode directiry structure:

```
kelly1@kelly1-LIFEBOOK-LH530:~/dfs/namesecondary$ ls
current  in_use.lock
kelly1@kelly1-LIFEBOOK-LH530:~/dfs/namesecondary$ cd current/
kelly1@kelly1-LIFEBOOK-LH530:~/dfs/namesecondary/current$ ls
edits  fsimage  fstime  VERSION
kelly1@kelly1-LIFEBOOK-LH530:~/dfs/namesecondary/current$ ll
total 24
drwxr-xr-x 2 kelly1 kelly1 4096 2012-09-18 23:19 ./
drwxr-xr-x 4 kelly1 kelly1 4096 2012-09-19 20:23 ../
-rw-r--r-- 1 kelly1 kelly1    4 2012-09-18 23:19 edits
-rw-r--r-- 1 kelly1 kelly1 2864 2012-09-18 23:19 fsimage
-rw-r--r-- 1 kelly1 kelly1    8 2012-09-18 23:19 fstime
-rw-r--r-- 1 kelly1 kelly1  100 2012-09-18 23:19 VERSION
```

The layout of secondary's current directory is identical to the namenode's. This is by design, since in the event of total namenode failure, it allows recovery from a secondary namenode.This can be achieved either by copying the relevant storage directory to a new namenode, or, if the secondary is taking over as the new primary namenode, by using the -importCheckpoint option when starting the namenode daemon. The -importCheckpoint option will load the namenode metadata from the latest checkpoint in the directory defined by the fs.checkpoint.dir property, but if there is no metadat in the dfs.name.dir directory, so there is no risk of overwriting previous metadata.

Datanode directory structure:

As datanodes creates their storage directories automatically on startup, so no need to do explicitely formatted.

```
drwxr-xr-x 2 kelly1 kelly1 4096 2012-09-19 20:24 current/
drwxr-xr-x 2 kelly1 kelly1 4096 2012-09-12 23:59 detach/
-rw-r--r-- 1 kelly1 kelly1    0 2012-09-19 20:23 in_use.lock
-rw-r--r-- 1 kelly1 kelly1  157 2012-09-12 23:59 storage
drwxr-xr-x 2 kelly1 kelly1 4096 2012-09-19 20:23 tmp/
kelly1@kelly1-LIFEBOOK-LH530:~/dfs/data$ cd current/
```

```
kelly1@kelly1-LIFEBOOK-LH530:~/dfs/data$ cd current/
kelly1@kelly1-LIFEBOOK-LH530:~/dfs/data/current$ ll
total 210072
drwxr-xr-x 2 kelly1 kelly1     4096 2012-09-19 20:24 ./
drwx------ 4 kelly1 kelly1     4096 2012-09-19 20:22 ../
-rw-r--r-- 1 kelly1 kelly1    25591 2012-09-15 18:32 blk_1360557715013600028_1006.meta
-rw-r--r-- 1 kelly1 kelly1      456 2012-09-15 18:29 blk_-1627393122322682795
-rw-r--r-- 1 kelly1 kelly1       11 2012-09-15 18:29 blk_-1627393122322682795_1005.meta
-rw-r--r-- 1 kelly1 kelly1    20413 2012-09-15 18:34 blk_1961868600997551736
-rw-r--r-- 1 kelly1 kelly1      167 2012-09-15 18:34 blk_1961868600997551736_1012.meta
-rw-r--r-- 1 kelly1 kelly1 67108864 2012-09-14 06:22 blk_-2806769607607470537
-rw-r--r-- 1 kelly1 kelly1   524295 2012-09-14 06:22 blk_-2806769607607470537_1003.meta
-rw-r--r-- 1 kelly1 kelly1    16680 2012-09-15 18:40 blk_-2947544845638000018
-rw-r--r-- 1 kelly1 kelly1      139 2012-09-15 18:40 blk_-2947544845638000018_1024.meta
-rw-r--r-- 1 kelly1 kelly1 67108864 2012-09-15 18:32 blk_3451198358097522603
-rw-r--r-- 1 kelly1 kelly1   524295 2012-09-15 18:32 blk_3451198358097522603_1006.meta
-rw-r--r-- 1 kelly1 kelly1    20735 2012-09-15 18:39 blk_4015418824033689825
-rw-r--r-- 1 kelly1 kelly1      171 2012-09-15 18:39 blk_4015418824033689825_1021.meta
-rw-r--r-- 1 kelly1 kelly1 67108864 2012-09-15 18:32 blk_-417367468448558263
-rw-r--r-- 1 kelly1 kelly1   524295 2012-09-15 18:32 blk_-417367468448558263_1006.meta
-rw-r--r-- 1 kelly1 kelly1 14786560 2012-09-14 06:22 blk_-5461212830652165877
-rw-r--r-- 1 kelly1 kelly1   115527 2012-09-14 06:22 blk_-5461212830652165877_1003.meta
-rw-r--r-- 1 kelly1 kelly1    10028 2012-09-15 18:39 blk_6147703354942118379
-rw-r--r-- 1 kelly1 kelly1       87 2012-09-15 18:39 blk_6147703354942118379_1023.meta
-rw-r--r-- 1 kelly1 kelly1    16718 2012-09-15 18:34 blk_-8102873251210258478
-rw-r--r-- 1 kelly1 kelly1      139 2012-09-15 18:34 blk_-8102873251210258478_1015.meta
-rw-r--r-- 1 kelly1 kelly1      688 2012-09-15 18:34 blk_-8252010316073749959
-rw-r--r-- 1 kelly1 kelly1       15 2012-09-15 18:34 blk_-8252010316073749959_1014.meta
-rw-r--r-- 1 kelly1 kelly1       36 2012-09-16 11:45 blk_8787259867977429757
-rw-r--r-- 1 kelly1 kelly1       11 2012-09-16 11:45 blk_8787259867977429757_1026.meta
-rw-r--r-- 1 kelly1 kelly1        4 2012-09-19 20:24 blk_-8860307820344948466
-rw-r--r-- 1 kelly1 kelly1       11 2012-09-19 20:24 blk_-8860307820344948466_1028.meta
-rw-r--r-- 1 kelly1 kelly1     2796 2012-09-19 20:23 dncp_block_verification.log.curr
-rw-r--r-- 1 kelly1 kelly1      153 2012-09-19 20:23 VERSION
```

In VERSION, the namespaceID, cTime, and layoutVersion are all the same as the nameNode VERSION values. The namespaceID is retrieved from the namenode when the datanode first connects). The storageID is unique to the datanode(it is the same across all storage directories) and is used by the namenode to uniquely identify the datanode. The storageType identifies this directory as a datanode storage directory.

The files in the datanode's current storage directory are two types:

1) blk_<id_i>: It just consists of the raw bytes of a portion of the file being stored.

2) blk_<id_i>.metadata: It is made up of a header with version and type information, followed by a series of checksums for sections of the block.

Datanode creates a new subdirectory every time the number of blocks in a directory reaches 64(It can be configured using dfs.datanode.numblocks). By taking this measure, it ensures that there is a manageable number of files per directory, which avoids the problems that most operating systems encounter when there are a large number of files in a single directory.

Some important tools for administartor:

1)hadoopdfsadmin: To find information about the state of HDFS and to perform administartion operations on HDFS

```
[-safemode enter | leave | get | wait]
[-saveNamespace]
[-refreshNodes]
[-finalizeUpgrade]
[-upgradeProgress status | details | force]
[-metasave filename]
[-refreshServiceAcl]
[-refreshUserToGroupsMappings]
[-refreshSuperUserGroupsConfiguration]
[-setQuota <quota> <dirname>...<dirname>]
[-clrQuota <dirname>...<dirname>]
[-setSpaceQuota <quota> <dirname>...<dirname>]
[-clrSpaceQuota <dirname>...<dirname>]
[-setBalancerBandwidth <bandwidth in bytes per second>]
[-help [cmd]]
```

2) hadoopfsck: To provide an fsck utility for checking the health of the files in HDFS.

```
kelly1@kelly1-LIFEBOOK-LH530:~/Work/hadoop/bin$ ./hadoop fsck
Usage: DFSck <path> [-move | -delete | -openforwrite] [-files [-blocks [-locations | -racks]]]
      <path>  start checking from this path
      -move   move corrupted files to /lost+found
      -delete delete corrupted files
      -files  print out files being checked
      -openforwrite   print out files opened for write
      -blocks print out block report
      -locations      print out locations for every block
      -racks  print out network topology for data-node locations
```

Datanode block scanner:

All the datanodes runs the block scanner, which periodically verifies all the blocks stored on the datanode. This allows bad blocks to be detected and fixed before they are read by clients.

It maintains

1) a list of blocks to verify
2) it scans them one by one for checksum errors.
3) block scanner report can be verify by visiting http://datanode:50075/blockScannerReport.

Upgarding the Cluster:

Steps for upgarding the cluster:

1) Make sure that any previous upgrade is finalized before proceeding with another upgrade.
2) Shut down MapReduce and kill any orphaned task processes on the tasktrackers.
3) Shut down HDFS and backup the namenode directories.
4) Install new versions of Hadoop HDFS and Mapreduce on the cluster and on clients.
5) Start HDFS with the -upgrade option.
6) Wait until the upgrade is complete.

7) Perform some sanity checks on HDFS.
8) Start MapReduce.
9) Roll back or finalize the upgrade

We can define two environment variables as HADOOP_OLD and HADOOP_NEW.
Start the upgrade:
% $HADOOP_NEW/bin/start-dfs.sh -upgrade
[This causes the namenode to upgrade its metadata,placing the previous version in a new directory called previous:
Wait until the upgrade is complete:
% $HADOOP_NEW/bin/hadoopdfsadmin -upgradeProgress status
Do sanity check using fsck
If there is any problem in new version just roollback:
% $HADOOP_NEW/bin/stop-dfs.sh
% $HADOOP_OLD/bin/start-dfs.sh -rollback
Finalyze the upgrade:
% $HADOOP_NEW/bin/hadoopdfsadmin -finalizeUpgrade
% $HADOOP_NEW/bin/hadoopdfsadmin -upgradeProgress status

How to kill the running mapreduce job?

```
kelly1@kelly1-LIFEBOOK-LH530:~/work/hadoop/bin$ ./hadoop job
Usage: JobClient <command> <args>
        [-submit <job-file>]
        [-status <job-id>]
        [-counter <job-id> <group-name> <counter-name>]
        [-kill <job-id>]
        [-set-priority <job-id> <priority>]. Valid values for priorities are: VE
RY_HIGH HIGH NORMAL LOW VERY_LOW
        [-events <job-id> <from-event-#> <#-of-events>]
        [-history <jobOutputDir>]
        [-list [all]]
        [-list-active-trackers]
        [-list-blacklisted-trackers]
        [-list-attempt-ids <job-id> <task-type> <task-state>]

        [-kill-task <task-id>]
        [-fail-task <task-id>]
```

```
kelly1@kelly1-LIFEBOOK-LH530:~/work/hadoop/bin$ ./hadoop job
Usage: JobClient <command> <args>
        [-submit <job-file>]
        [-status <job-id>]
        [-counter <job-id> <group-name> <counter-name>]
        [-kill <job-id>]
        [-set-priority <job-id> <priority>]. Valid values for priorities are: VERY_HIGH HIGH NORMAL LOW VERY_LOW
        [-events <job-id> <from-event-#> <#-of-events>]
        [-history <jobOutputDir>]
        [-list [all]]
        [-list-active-trackers]
        [-list-blacklisted-trackers]
        [-list-attempt-ids <job-id> <task-type> <task-state>]

        [-kill-task <task-id>]
        [-fail-task <task-id>]
```

MapReduce

1. MapReduce Introduction
2. Combiners
3. Partitioners
4. Writables
5. Writable Comparables
6. Distributed Cache
7. Input Formatters
8. Out Formatters
9. Schedulers
   a. FIFO Scheduler
   b. FAIR Scheduler
   c. Capacity Scheduler
10. Compression
    a. LZO
    b. Snappy
11. Hadoop cluster up
    a. Prerequisites
    b. Pseudo Cluster Mode
    c. Cluster Mode

MapReudce Introduction

MapReduce is a programming model for processing BigData. It processes data in parallel. To take advantage of the parallel processing that Hadoop provides, express your business logic as a MapReduce Job.

MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output. Input key-value pair of map phase is determined by the type of Input Formatter being used, where as rest of the key-value pair types may be chosen by the programmer. The programmer also specifies two functions : the map function and the reduce function.

The map function is represented by extending the Mapper Abstract class. Mapper class is a generic type, with four formal type of parameters that specify the input key, input value, output key and output value types of map function.

The reduce function is similarly defined by extending the Reducer Abstract class. Reducer class is also a generic type, with four formal type of parameters that specify the input key, input value, output key and output value types of map function.

Reducer aggregates the map outputs. So one needs to remember that map output key type, map output value type should match with reduce input key type and reduce input value type.

Let us write a MapReduce application for counting words in a text file.
Example-1 shows the implementation of map function for WordCount example.

```
publicstaticclass WordCountMapper
extends Mapper<LongWritable, Text, Text, IntWritable>{

privatefinalstatic IntWritable one = new IntWritable(1);
private Text word = new Text();

publicvoid map (LongWritable key, Text value, Context context
        ) throws IOException, InterruptedException{
   StringTokenizer itr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) {
word.set(itr.nextToken());
    context.write(word, one);
  }
 }
}
```

Example-1

WordCountMapper extends Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> with specific types likes LongWritable, Text, Text, IntWritable.

Hadoop uses its own data types rather than using Java data types. These data types are optimized for network serialization. These data types are found in the org.apache.hadoop.io package. Here LongWritable and Text corresponds to long and String respectively in Java.

For the above example, input key is line offset in the file and value is the entire line. The third parameter is a context object, which allows the input and output from the task. It is only supplied to the Mapper and Reducer.

For the above example, as we are counting number of times each word occurred, take the value, which is the entire line and convert it into String using toString() method in Text class. Split the line into words using StringTokenizer class. Iterate over all the words in the line and output word and its count 1 (the word has already occurred one time, thats why 1). For

outputting word and its count, use the context object, as discussed above.
Context has write method to write the map output.

Map outputs are stored in the Local FileSystem, where mapper has run. Also Map outputs are stored in the sorted order, sorted on Key. *If the Job is Map only*
Job, Map outputs are not sorted and directly stored in the configured File System in core-site.xml (In most cases it is HDFS).

Let us consider the following input which has 4 Lines for the WordCount example

Hadoop is used for BigData Analysis.
Hadoop has two components namely HDFS and MapReduce.
HDFS is a Distributed Data Storage Framework.
MapReduce is a Distributed Computing Framework.
input-1

The output of the Example-1 is
a 1
a 1
components 1
<complete this>

Example-2 shows the implementation of reduce function for WordCount example.

```
publicstaticclass WordCountReducer
extends Reducer<Text,IntWritable,Text,IntWritable> {
private IntWritable result = new IntWritable();

publicvoid reduce(Text key, Iterable<IntWritable> values,
                  Context context
                 ) throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
    sum += val.get();
  }
result.set(sum);
  context.write(key, result);
 }
}
```

Example-2
WordCountReducer extends Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> with specific types likes Text, IntWritable, Text, IntWritable.

Note that WordCountReducer input key type is Text and value type is IntWritable, matches with the output key type and value type of WordCountMapper.

Reducer first copies all the map outputs and merges them. This process is called Shuffle and Sorting. During this phase, all the values corresponding to each key will be aggregated. reduce method is called for each key with all the values aggregated for that key.

For the above map output, Reducer first copies map output and as there is only one map output, merge is trivial. reduce method is called for each key and its aggregated values. So reduce method in WordCountReducer get the following input

A 1,1
Distributed 1,1
<Complete this>

In the reduce method, iterate over all the values and counting how many them.
Finally writing the reduce method output, which is word and its count using write method provided by Context class.

Finally we need to provide a driver class, which submits the Job to the HADOOP cluster with our own mapper and reduce implementation.

```
publicclass WordCount {

publicstaticvoid main(String[] args) throws Exception {
  Configuration conf = new Configuration();

if (otherArgs.length != 2) {
   System.err.println("Usage: WordCount <in><out>");
   System.exit(2);
  }
  Job job = new Job(conf, "kelly word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(WordCountMapper.class);
  job.setReducerClass(WordCountReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
  FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Example-3

In Example-3, created a Job object with configuration object and name of the job being the arguments.

Configuration is an object through which Hadoop reads all the configuration properties mentioned in core-site.xml, hdfs-site.xml and mapred-site.xml,
By passing the conf object while job object creation, job object avails all the properties from the afore mentioned files during its life cycle (like map or reduce phase). **Also a user can define their own property with a value (value type could be any data type) using setter methods provided in the Configuration object like conf.setInt (use the appropriate API based on the value type) method.Hence Configuration is very useful if you need to pass a small piece of metadata to your tasks. To retrieve the values in the task, using context object and get configuration object using context.getConfiguration() and then get the required metadata.**

Job object has been specified with Mapper, Reducer classes, output key type,output value type. Input path is mentioned using the static method of FileInputFormat class's addInputPath having job instance and input directory as arguments. Similarly output directory has been mentioned using static method provided by FileOutputFormat class.Finally submit the job to cluster using waitForCompletion method and wait for job's completion. Once job completes successfully, find the output of the job in the output directory.

How to submit the job to cluster

1. *Go to HADOOP_HOME directory*
2. *Run the following command to submit the job*
   bin/hadoop jar <path of jar file><package-name>.WordCount /input /output.

Suppose name of the jar file is jobs.jar and let us say this jar file is in /home/hadoop directory and package-name is kelly.training.hadoop then above command becomes

bin/hadoop jar /home/hadoop/jobs.jar kelly.training.hadoop.WordCount /input /output.

Exercises:

1. Write a MapReduce Job to display all the lines which has given word. (Simulating UNIX 'grep' functionality).

Ex: Suppose given word is : *BigData.*
For the input-1, MapReduce Job output should contain only the following line, since this line contains BigData word.

Output : Hadoop is used for BigData Analysis.

2. Write a MapReduce Job to display all the lines by replacing the given word and another input word. (Simulating UNIX 'sed' functionality).

Ex: Suppose given word is : *BigData* and another word is : *LargeData.*
For the input-1, MapReduce Job output should contain only the following line, since this line contains BigData word.However BigData should be replace with LargeData, so the output should be

Output : Hadoop is used for LargeData Analysis.

3. Write a MapReduce Job to find the average of frequency of each word.

The format of the Text file is as follows

hadoop 10
hbase 15
hive 30
hadoop 47
solr 33
katta 68
hbase 26
hive 22

input-2

For input-2, the output should be
hadoop 28.5
hbase 20.5
hive 26
solr 33
katta 68.

Combiners

Combiners are used to reduce the amount of the data being transfered over the network. This is to optimize the usage of the network bandwidth, which is limited in most of the cases.

Combiners are treated as local reducers. They run by consuming the mapper output and run on the same machine where mapper had run earlier.*Hadoop does not provide any guarantee on combiner's execution. Hadoop may call*
combiner function zero, one or many times for a particular map outputrecord.

Imagine the first mapper produced the output

hadoop 1

hadoop 1
hadoop 1
hadoop 1
hadoop 1

and second mapper produced

hadoop 1
hadoop 1
hadoop 1

We could use a combiner function that, just like reduce function, counts the word frequency. The reduce would then be called with:

hadoop <5,3>

Note: While calling reduce method, Hadoop does not provide any guarantee on values in sorted order corresponding to a particular key. To achieve this we used Secondary Sorting.

The combiner function does not replace the reduce function. Combiner is also implemented by extending the Reduce Abstract class and overriding the reduce method.

Specifying a combiner function

For the WordCount example WordCountReducer can also be used as Combiner and the same can be set using setCombinerClass API from Job class as highlighted below.

```
publicclass WordCount {

publicstaticvoid main(String[] args) throws Exception {
  Configuration conf = new Configuration();

if (otherArgs.length != 2) {
  System.err.println("Usage: wordcount <in><out>");
  System.exit(2);
  }
  Job job = new Job(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(WordCountMapper.class);
job.setCombinerClass(WordCountReducer.class);
  job.setReducerClass(WordCountReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
```

```
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Exercises

4. Can we use a combiner class for problem 3? Provide your explanation.
5. Write an MR job to find the highest repeated words in a file.
For the input-1, "is" repeated 3 times so the output should be
is 3
If there are more than one such word, print all those words.

Partitioners

Partitioner allows you to distribute how outputs from the map stage are sent to the reducers. Basically it partitions the key space.

Partitioner controls the partitioning the keys of the intermediate map-outputs. The key (or subset of the key) is used to derive the partition.

The total number of partitions is the same as the number of reduce tasks for the job. Hence Partitioner controls which of the r reduce tasks the intermediate key (and hence record) is sent for reduction.

Partitioner run on the same machine after Mapper had completed its execution, by consuming Mapper output. So entire Mapper output (record) is sent to Partitioner and Partitioner forms r (number of reduce tasks) groups from the mapper output.

By default Hadoop framework Hash based partitioner. This Partitioner partitions the key space by using the hashcode.

The following is logic HashPartitioner executes to determine a reducer for a particular Key 'key'.

```
(key.hashCode() & Integer.MAX_VALUE) % numReduceTasks
```

How to Write Custom Partitioner

Hadoop provides Partitioner abstract class with a single method which can be extended to write a custom partitioner.

```
publicabstractclass Partitioner<KEY, VALUE> {
publicabstractint getPartition(KEY key, VALUE value, int numPartitions);
```

}

As discussed earlier KEY and VALUE type must match with MapOutput key and MapOutput value.

Example

Let us take the WordCount Example and suppose say requirement is all the words which starts with "a" go to one reducer and all the words which starts with "b" go to another reducer etc...

So the number of reducer required is 26.

Let us write a Partitioner for that

```
public class WordCountPartitioner extends Partitioner<Text, IntWritable> {

  public int getPartition(Text key, IntWritable value, int numPartitions) {
    return (key.toString().charAt(0) - 'a') % numPartitions;

}
```

WordCountMappe's output key is Text and value type is IntWritable, so WordCountPartitioner Key type also Text and Value type is IntWritable.

In the getPartition method, take the first character of the word and finding its position relative to 'a'.

Suppose say firstChar is 'c'. So the ascii code of c is 99 and ascii code of a is 97.

so 'c' - 'a' gives 2, which is relative to 'a'.

Hence all words which starts with 'c' goes to 2nd reducer.

Specifying a Partitioner function

Partitioner class can be set using the setPartitionerClass method from the job class. The same has been highlighted below. Also specify number of reduce taks as 26.

```
publicclass WordCount {

publicstaticvoid main(String[] args) throws Exception {
  Configuration conf = new Configuration();
```

```
if (otherArgs.length != 2) {
    System.err.println("Usage: wordcount <in><out>");
    System.exit(2);
   }
   Job job = new Job(conf, "word count");
   job.setJarByClass(WordCount.class);
   job.setMapperClass(WordCountMapper.class);
job.setCombinerClass(WordCountReducer.class);
   job.setPartitionerClass(WordCountPartitioner.class);
job.setNumReduceTasks(26);
   job.setReducerClass(WordCountReducer.class);
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);
   FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
   FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

By default number of reduce task be 1.

Caution: It is always recommended to provide implementation of hashcode when you write your own composite key.

In the WordCountPartitioner, observe that getPartition() method has been implemented without considering hashcode of the key. But it may not possible to write getPartition() method without considering hashcode of the key always.

Writable

Writable is an interface, which Hadoop uses for serialization and de-serialization purpose. Any key or value type in Hadoop Map-Reduce framework must implement this interface.

This interface is a simple, efficient, serialization protocol, based on DataInput and DataOutput.

To implement your own Writable class, you must implement the following methods.

void write(DataOutput out) **throws** IOException;

Serialize of the fields of this object (state of the object) to out (DataOutput binary stream) using write implementation.

void readFields(DataInput in) **throws** IOException;

De-serialize the fields of this object (state of the object)from in (DataInput binary stream) using readFields implementation.

WritableComparable

WritableComparable is also an interface which extends both Writable interface and java.lang.Comparable interface.

Any Key in Hadoop Map-Reduce framework must implement this interface. During the sorting phase keys are compared with one another using implementation provided for the following method

publicint compareTo(T o);

The Following are some of mostly used WritableComparable classes provided by Hadoop framework

BooleanWritable -- boolean
ByteWritable -- byte
IntWritable -- int
FloadWritable -- float
LongWritable -- long
DoubleWritable -- double
Text -- String

Please go through the org.apache.hadoop.io API documentation for more Writable and WritableComparable classes.

Distributed Cache

Distributed cache is a mechanism for copying large and read only files and archives to the task nodes in time for the tasks use them when they run. To save network bandwidth, files or archives are normally copied to any particular node once per job.

How to specify Distributed Cache entries?

Hadoop provides 2 options to achieve the same

a) API :

Job class have provided various APIs to add the files/archives into Distributed Cache.

The following are APIs from Job class

addCacheArchive(URI uri) : Add any archive to be cached using URI object.
addCacheFile(URI uri) : Add any read only file (like lookup) to be cached using URI object.
addFileToClassPath(Path file) : If a file is required to be in Child's classpath, use this API. It also adds the file to cache as well.
Ex: Suppose say your job depends on third party jars, you can use this API to place all those jar in child's classpath.
addArchiveToClassPath(Path archive) : If an archive is required to be in Child's classpath, use this API. It also adds the archive to cache as well.

To get the cache entries in either map/ reduce use the following APIs.

Use the context object in map/reduce/setup method in Mapper/Reducer class.

getCacheFiles() : To get the cache files.
getCacheArchives() : To get the cache archives.

b) Using CLI.

Rather than using the API, while running the job, we can use "-libjars" and "-files" options to add the entries to Distributed Cache.

-libjars : comma separated jar files to include in the classpath.
-files : comma separated files to be copied to the map reduce cluster
-archives : comma separated archives to be unarchived on the compute machines.
Go through the following example which illustrates how to use Distributed Cache .
PhoneRetrievalJob is to retrieve the phone numbers of the users. Input to the job contains list of users. A lookup file contains user and phone number. If the input user is present in the lookup file, we emit his phone number otherwise we just ignore the user.

```
package kelly.hadoop.training.jobs.dc;

import java.net.URI;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

```
public class PhoneRetrievalJob implements Tool {

    private Configuration conf;

    @Override
    public int run(String[] args) throws Exception {
        Job phoneJob = new Job(getConf());
        phoneJob.setJobName("Kelly Phone JOb");
        phoneJob.setJarByClass(this.getClass());

        phoneJob.setNumReduceTasks(0);

        phoneJob.setMapperClass(PhoneRetrievalMapper.class);
        phoneJob.setOutputKeyClass(Text.class);
        phoneJob.setOutputValueClass(LongWritable.class);

        phoneJob.setInputFormatClass(KeyValueTextInputFormat.class);

        Path lookupPath = new Path(args[2]);
        URI[] uris = new URI[1];
        uris[0] = lookupPath.toUri();

        DistributedCache.setCacheFiles(uris, phoneJob.getConfiguration());

        FileInputFormat.setInputPaths(phoneJob, new Path(args[0]));
        FileOutputFormat.setOutputPath(phoneJob, new Path(args[1]));

        return phoneJob.waitForCompletion(true) == true ? 0 : -1;
    }

    @Override
    public Configuration getConf() {
        return conf;
    }

    @Override
    public void setConf(Configuration conf) {
        this.conf = conf;
    }

    public static void main(String[] args) throws Exception {
        ToolRunner.run(new Configuration(), new PhoneRetrievalJob(), args);
    }

}
```

To run the above job, user need to mention the location of the lookup file in the File System. If you are run this job in local mode, the lookup file should present in Local File System. If you are running this job in Cluster mode, copy the lookup file into HDFS, and then give the path of HDFS while running the job.

The same procedure should be followed for all the jobs, when your job uses files in the Distributed cache.

Highlighted text contains, how to set the file in the Distributed cache to job instance.

In the mapper, we get the file from the Distributed Cache, open the lookup file and if the input record (user) present in the lookup file we emit user and his phone number, otherwise we just ignore the record. Usage of the Distributed Cache in **PhoneRetrievalMapper** has been highlighted in bold.

```
package kelly.hadoop.training.jobs.dc;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;

import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class PhoneRetrievalMapper extends Mapper<Text, Text, Text, Text> {

    private BufferedReader reader;
    private Path lookUp;
    private Text phone = new Text();

    @Override
    protected void setup(Context context) throws java.io.IOException,
            InterruptedException {
        Path[] cacheFiles = DistributedCache.getLocalCacheFiles(context
                .getConfiguration());
        lookUp = cacheFiles[0];
    }

    @Override
    protected void map(Text key, Text value, Context context)
            throws java.io.IOException, InterruptedException {
        File file = new File(lookUp.toString());
        reader = new BufferedReader(new InputStreamReader(new FileInputStream(
                file)));
        String line;
```

```
        while ((line = reader.readLine()) != null) {
            if (line.contains(key.toString())) {
                String phoneNum = line.split("\t")[1];
                phone.set(phoneNum);
                context.write(key, phone);
            }
        }
        reader.close();
    };

}
```

InputFormatters

Inputformatters describe the input specification of the job. The responsibility of the Input Formatters

1) Validate the input specification of the job
2) Split up the input file(s) into logical Input Split. Each input split is then assigned to an individual Mapper.
3) Provides a Record Reader to glean input record from logical Input Split for processing by Mapper.

Input split is computed using the following formulae

Max (MinSplitSize,Min(MaxSplitSize, HDFS Blk)

Where split size max is configured using **mapred.max.split.size** property and min split size is configured using **mapred.min.split.size** property.

Each record is splitted into Key-Value pairs. These key-value pairs will act as Map input key and Map input value. The type of input formatter to use is depends on the nature of the data.

There are various types of input formatters provided by the framework.

TextInputFormatter

This IF works on the text files. TIF recognizes each line (by default delimited by \n)in the file as one record. For TIF, key will be line offset in the file (LongWritable) and value will be entire line (Text). By default Hadoop used TIF.

Ex:

I am new to Hadoop.

Hadoop is great software.

As there are two lines in the files, so there are two records.

For the first record: key: 0(line offset) value: I am new to Hadoop

For the second record: key: 21(line offset) value: Hadoop is great software.

KeyValueTextInputFormatter

This IF works on the text files; however, it works on the text files which are in the key value format. KVTIF recognizes each line (by default delimited by \n)in the file as one record. Each line is divided into key and value parts by a separator byte. By default key and values in the file are delimited by \t (tab). But it can be overridden. If no such a byte exists, the key will be the entire line and value will be empty.

Ex:

Andhra Pradesh Hyderabad
Madhya Pradesh Bhopal
Karnataka Bangalore.

From the above input file, it is obvious that there are 3 records (as there are 3 lines in the file). Also observe that above file is in Key Value format (Keys and Values are delimited by tab).

For the first record: key: Andhra Pradesh value: Hyderabad
For the second record: key: Madhya Pradesh value: Bhopal
So here both key and value types are Text.
Let us illustrate the same using an example

```
package kelly.hadoop.training.jobs.inputformatter;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordPairCountJob implements Tool {

    private Configuration conf;

    @Override
    public int run(String[] args) throws Exception {
        Job wordPairCountJob = new Job(getConf());
        wordPairCountJob.setJobName("Kelly WordPair Count");
        wordPairCountJob.setJarByClass(this.getClass());

        wordPairCountJob.setMapperClass(WordPairCountMapper.class);
        wordPairCountJob.setMapOutputKeyClass(WordPairCountKey.class);
        wordPairCountJob.setMapOutputValueClass(LongWritable.class);

        wordPairCountJob.setReducerClass(WordPairCountReducer.class);
        wordPairCountJob.setOutputKeyClass(WordPairCountKey.class);
        wordPairCountJob.setOutputValueClass(LongWritable.class);
```

```
        wordPairCountJob.setInputFormatClass(KeyValueTextInputFormat.class);

        FileInputFormat.setInputPaths(wordPairCountJob, new Path(args[0]));
        FileOutputFormat.setOutputPath(wordPairCountJob, new Path(args[1]));

        return wordPairCountJob.waitForCompletion(true) == true ? 0 : -1;
    }

    @Override
    public Configuration getConf() {
        return conf;
    }

    @Override
    public void setConf(Configuration conf) {
        this.conf = conf;
    }

    public static void main(String[] args) throws Exception {
        ToolRunner.run(new Configuration(), new WordPairCountJob(), args);
    }

}

package kelly.hadoop.training.jobs.inputformatter;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.WritableComparable;

public class WordPairCountKey implements WritableComparable<WordPairCountKey> {

    private String word1;
    private String word2;

    public String getWord1() {
        return word1;
    }

    public void setWord1(String word1) {
        this.word1 = word1;
    }
```

```
    public String getWord2() {
        return word2;
    }

    public void setWord2(String word2) {
        this.word2 = word2;
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        word1 = in.readUTF();
        word2 = in.readUTF();
    }

    @Override
    public void write(DataOutput out) throws IOException {
        out.writeUTF(word1);
        out.writeUTF(word2);
    }

    @Override
    public int compareTo(WordPairCountKey o) {
        int diff = word1.compareTo(o.word1);
        if (diff == 0) {
                diff = word2.compareTo(o.word2);
        }
        return diff;
    }

    @Override
    public int hashCode() {
        return word1.hashCode() + 31 * word2.hashCode();
    }

    @Override
    public String toString() {
        return "[word1=" + word1 + ", word2=" + word2 + "]";
    }

}
```

```
package kelly.hadoop.training.jobs.inputformatter;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Mapper;

public class WordPairCountMapper extends
        Mapper<Text, Text, WordPairCountKey, LongWritable> {

    private final static LongWritable one = new LongWritable(1);

    @Override
    protected void map(Text key, Text value, Context context)
            throws java.io.IOException, InterruptedException {
        WordPairCountKey wpKey = new WordPairCountKey();
        wpKey.setWord1(key.toString());
        wpKey.setWord2(value.toString());
        context.write(wpKey, one);
    };

}
```

```
package kelly.hadoop.training.jobs.inputformatter;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class WordPairCountReducer extends
        Reducer<WordPairCountKey, LongWritable, WordPairCountKey, LongWritable> {

    @Override
    protected void reduce(WordPairCountKey key,
            java.lang.Iterable<LongWritable> values, Context context)
            throws java.io.IOException, InterruptedException {
        long sum = 0;
        while (values.iterator().hasNext()) {
            sum += values.iterator().next().get();
        }
        context.write(key, new LongWritable(sum));
    }
}
```


Sequence File Input Formatter

This IF works only on special binary files called Sequence file. Sequence files are specific to Hadoop. Sequence files are also in key value format, however, unlike KVTIF, keys and values can be any type.

Sequence files can be generated as the output of other MapReduce tasks and are an efficient intermediate representation for data that is passing from one MapReduce job to another.

This IF is used to process binary data like images, video and audio files. Also it can also be used to process small files. While processing small files, we can use file name as key and entire file as value.

We can apply compression to values while storing them in Sequence Files.

```
package kelly.hadoop.training.jobs.inputformatter;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.SequenceFile.CompressionType;
import org.apache.hadoop.io.SequenceFile.Writer;

public class SequenceFileGenerator {

    private static final int BUFFER_SIZE = 2 * 1024 * 1024;

    public static void main(String[] args) throws IOException {
        Writer writer = null;
        Configuration conf = new Configuration();
        Path inputDir = new Path(args[0]);
        Path seqFile = new Path(args[1]);

        try {
            writer = SequenceFile.createWriter(seqFile.getFileSystem(conf),
                    conf, seqFile, Text.class, BytesWritable.class,
                    CompressionType.BLOCK);
            FileSystem fs = inputDir.getFileSystem(conf);
            BytesWritable content = new BytesWritable();
            Text fileName = new Text();
            byte[] data = new byte[BUFFER_SIZE];
            for (FileStatus file : fs.listStatus(inputDir)) {
                FSDataInputStream stream = fs.open(file.getPath());
                stream.read(data);
                content.set(data, 0, (int) file.getLen());
                fileName.set(file.getPath().getName());
                writer.append(fileName, content);
                stream.close();
```

```
            }
            writer.close();
        } finally {
            if (writer != null) {
                writer.close();
            }
        }
    }
}
```

```
package kelly.hadoop.training.jobs.inputformatter;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCountPerFileJob implements Tool {

    private Configuration conf;

    @Override
    public int run(String[] args) throws Exception {
        Job wordCountPerFileJob = new Job(getConf());
        wordCountPerFileJob.setJobName("Kelly Word Count Per File");
        wordCountPerFileJob.setJarByClass(this.getClass());

        wordCountPerFileJob.setMapperClass(WordCountPerFileMapper.class);
        wordCountPerFileJob.setMapOutputKeyClass(WordCountPerFileKey.class);
        wordCountPerFileJob.setMapOutputValueClass(IntWritable.class);

        wordCountPerFileJob.setReducerClass(WordCountPerFileReducer.class);
        wordCountPerFileJob.setOutputKeyClass(WordCountPerFileKey.class);
        wordCountPerFileJob.setOutputValueClass(IntWritable.class);


        wordCountPerFileJob.setInputFormatClass(SequenceFileInputFormat.class);

        FileInputFormat.setInputPaths(wordCountPerFileJob, new Path(args[0]));
        FileOutputFormat.setOutputPath(wordCountPerFileJob, new Path(args[1]));
```

```
        return wordCountPerFileJob.waitForCompletion(true) == true ? 0 : -1;
    }

    @Override
    public Configuration getConf() {
        return conf;
    }

    @Override
    public void setConf(Configuration conf) {
        this.conf = conf;
    }

    public static void main(String[] args) throws Exception {
        ToolRunner.run(new Configuration(), new WordCountPerFileJob(), args);
    }

}
```

```
package kelly.hadoop.training.jobs.inputformatter;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.WritableComparable;

public class WordCountPerFileKey implements
        WritableComparable<WordCountPerFileKey> {

    private String fileName;
    private String word;

    public String getFileName() {
        return fileName;
    }

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public String getWord() {
        return word;
    }
```

```
    public void setWord(String word) {
        this.word = word;
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        fileName = in.readUTF();
        word = in.readUTF();
    }

    @Override
    public void write(DataOutput out) throws IOException {
        out.writeUTF(fileName);
        out.writeUTF(word);
    }

    @Override
    public int compareTo(WordCountPerFileKey o) {
        int diff = fileName.compareTo(o.fileName);
        if (diff == 0) {
                diff = word.compareTo(o.word);
        }
        return diff;
    }

    @Override
    public String toString() {
        return "[fileName=" + fileName + ", word=" + word + "]";
    }

}

package kelly.hadoop.training.jobs.inputformatter;

import java.util.StringTokenizer;

import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountPerFileMapper extends
        Mapper<Text, BytesWritable, WordCountPerFileKey, IntWritable> {
```

```
    private final static IntWritable one = new IntWritable(1);

    @Override
    protected void map(Text key, BytesWritable value, Context context)
            throws java.io.IOException, InterruptedException {
        String fileName = key.toString();
        String content = new String(value.getBytes());
        WordCountPerFileKey wcKey = new WordCountPerFileKey();
        wcKey.setFileName(fileName);
        StringTokenizer strTock = new StringTokenizer(content, " ");
        while (strTock.hasMoreTokens()) {
            wcKey.setWord(strTock.nextToken());
            context.write(wcKey, one);
        }
    };

}
```

```
package kelly.hadoop.training.jobs.inputformatter;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountPerFileReducer
        extends
        Reducer<WordCountPerFileKey, IntWritable, WordCountPerFileKey, IntWritable> {

    @Override
    protected void reduce(WordCountPerFileKey key,
            java.lang.Iterable<IntWritable> values, Context context)
            throws java.io.IOException, InterruptedException {
        int count = 0;
        for (IntWritable intW : values) {
            count = count + intW.get();
        }
        context.write(key, new IntWritable(count));
    }
}
```


OutFormatters

The responsibility of the output formatters is to write the output of jobs to underlying storage(either HDFS or LocalFS or DataBase etc...). The output of the job is written using

help of Record Writers. So it means final output is a set of records. The type of output depends on the type of OF being used.

Also Outputformatters verify the whether the output directory already exists or not.

Mapreduce frame work provides the following Output Formatters

TextOutputFormatter

If we don't specify any output formatter, Hadoop by default considers this output formatter. TOF writes the output to plain text files.

SequenceFileOutputFormatter

This OF writes output to Sequence Files

Schedulers

The functionality of the Scheduler is to schedule the tasks(either Map Task or Reduce Task) to TaskTrackers. There are 3 Schedulers available in Hadoop.

1) Fifo Scheduler
2) Fair Scheduler
3) Capacity Task Scheduler

1) Fifo Scheduler

This is the default scheduler, Hadoop uses to schedule the tasks. This Scheduler simply schedules the tasks of a job in order of submission. Each job uses the whole cluster, so jobs have to wait their turn. Users can assign priority to a job while submission to the Cluster through mapred.job.priority property. Priorties could be one among VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW. Default priority of a job is NORMAL.

When the job scheduler is choosing the next job to run, it selects one with the highest priority. However, with the FIFO scheduler,priorities do not support preemption, so a high-priority job can still be blocked by a long running low priority job that started before the high-priority job scheduled.

2) Fair Scheduler

Fair scheduling is a method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time.

When there is a single job running, that job uses the entire cluster. When other jobs are submitted, tasks slots that free up are assigned to the new jobs, so that each job gets roughly the same amount of CPU time.Unlike Fifo scheduler, this lets short jobs finish in reasonable time while not starving long jobs.It is also an easy way to share a cluster between multiple of users.

Fair sharing can also work with job priorities - the priorities are used as weights to determine the fraction of total compute time that each job gets.

The fair scheduler organizes jobs into pools, and divides resources fairly between these pools. By default, there is a separate pool for each user, so that each user gets an equal share of the cluster.

Within each pool, jobs can be scheduled using either fair sharing or first-in-first-out (FIFO) scheduling.

In addition to providing fair sharing, the Fair Scheduler allows assigning guaranteed *minimum shares* to pools, which is useful for ensuring that certain users, groups or production applications always get sufficient resources. When a pool contains jobs, it gets at least its minimum share, but when the pool does not need its full guaranteed share, the excess is split between other pools.

If a pool's minimum share is not met for some period of time, the scheduler optionally supports *preemption* of jobs in other pools. The pool will be allowed to kill tasks from other pools to make room to run. A pool can also be allowed to preempt tasks if it is below half of its fair share for a configurable timeout. When choosing tasks to kill, the fair scheduler picks the most-recently-launched tasks from over-allocated jobs, to minimize wasted computation. Preemption does not cause the preempted jobs to fail, because Hadoop jobs tolerate losing tasks; it only makes them take longer to finish.

The Fair Scheduler can limit the number of concurrent running jobs per user and per pool.This can be useful when a user must submit hundreds of jobs at once, or for ensuring that intermediate data does not fill up disk space on a cluster when too many concurrent jobs are running.Setting job limits causes jobs submitted beyond the limit to wait until some of the user/pool's earlier jobs finish. Jobs to run from each user/pool are chosen in order of priority and then submit time.

The Fair Scheduler can limit the number of concurrent running tasks per pool. This can be useful when jobs have a dependency on an external service like a database or web service that could be overloaded if too many map or reduce tasks are run at once.

Installation

set the following property in the Hadoop config file *HADOOP_CONF_DIR/mapred-site.xml* to have Hadoop use the fair scheduler:

```
<property>
<name>mapred.jobtracker.taskScheduler</name>
<value>org.apache.hadoop.mapred.FairScheduler</value>
</property>
```

Once you restart the cluster, you can check that the fair scheduler is running by going to *http://<jobtracker URL>/scheduler* on the JobTracker's web UI.

Configuration

The Fair Scheduler contains configuration in two places -- algorithm parameters are set in *HADOOP_CONF_DIR/mapred-site.xml*, while a separate XML file called the *allocation file*, located by default in *HADOOP_CONF_DIR/fair-scheduler.xml*, is used to configure pools, minimum shares, running job limits and preemption timeouts. The allocation file is reloaded periodically at runtime, allowing you to change pool settings without restarting your Hadoop cluster.

Important settingfor mapred-site.xml

mapred.fairscheduler.preemption : Boolean property for enabling preemption. Default: false.
mapred.fairscheduler.pool : Specify the pool that a job belongs in.
mapred.fairscheduler.sizebasedweight : Take into account job sizes in calculating their weights for fair sharing. By default, weights are only based on job priorities. Setting this flag to true will make them based on the size of the job (number of tasks needed) as well,though not linearly.This lets larger jobs get larger fair shares while still providing enough of a share to small jobs to let them finish fast. Boolean value, **default: false.**

Important settings for fair-scheduler.xml

The allocation file configures minimum shares, running job limits, weights and preemption timeouts for each pool.

- *pool* elements, which configure each pool. These may contain the following sub-elements:
  - *minMaps* and *minReduces*, to set the pool's minimum share of task slots.
  - *maxMaps* and *maxReduces*, to set the pool's maximum concurrent task slots.
  - *schedulingMode*, the pool's internal scheduling mode, which can be *fair* for fair sharing or *fifo* for first-in-first-out.
  - *maxRunningJobs*, to limit the number of jobs from the pool to run at once (defaults to infinite).
  - *weight*, to share the cluster non-proportionally with other pools. For example, a pool with weight 2.0 will get a 2x higher share than other pools. The default weight is 1.0.
  - *minSharePreemptionTimeout*, the number of seconds the pool will wait before killing other pools' tasks if it is below its minimum share (defaults to infinite).
- *minSharePreemptionTimeout*, the number of seconds the pool will wait before killing other pools' tasks if it is below its minimum share (defaults to infinite).
- *user* elements, which may contain a *maxRunningJobs* element to limit jobs.
- *poolMaxJobsDefault*, which sets the default running job limit for any pools whose limit is not specified.
- *userMaxJobsDefault*, which sets the default running job limit for any users whose limit is not specified.
- *fairSharePreemptionTimeout*, which sets the preemption timeout used when jobs are below half their fair share.

An example allocation file is given below :

```
<?xml version="1.0"?>
<allocations>
<pool name="kelly_pool">
<minMaps>5</minMaps>
<minReduces>5</minReduces>
<maxMaps>25</maxMaps>
<maxReduces>25</maxReduces>
<minSharePreemptionTimeout>300</minSharePreemptionTimeout>
</pool>
<user name="kelly_user">
```

```
<maxRunningJobs>6</maxRunningJobs>
</user>
<userMaxJobsDefault>3</userMaxJobsDefault>
<fairSharePreemptionTimeout>600</fairSharePreemptionTimeout>
</allocations>
```

This example creates a pool kelly_pool with a guarantee of 5 map slots and 5 reduce slots. The pool also has a minimum share preemption timeout of 300 seconds (5 minutes), meaning that if it does not get its guaranteed share within this time, it is allowed to kill tasks from other pools to achieve its share. The pool has a cap of 25 map and 25 reduce slots, which means that once 25 tasks are running, no more will be scheduled even if the pool's fair share is higher. The example also limits the number of running jobs per user to 3, except for kelly_user, who can run 6 jobs concurrently. Finally, the example sets a fair share preemption timeout of 600 seconds (10 minutes). If a job is below half its fair share for 10 minutes, it will be allowed to kill tasks from other jobs to achieve its share.

Administration

The fair scheduler provides support for administration at runtime through two mechanisms:

1. It is possible to modify minimum shares, limits, weights, preemption timeouts and pool scheduling modes at runtime by editing the allocation file. The scheduler will reload this file 10-15 seconds after it sees that it was modified.
2. Current jobs, pools, and fair shares can be examined through the JobTracker's web interface, at *http://<JobTracker URL>/scheduler*. On this interface, it is also possible to modify jobs' priorities or move jobs from one pool to another and see the effects on the fair shares (this requires JavaScript).

The following fields can be seen for each job on the web interface:

- *Submitted* - Date and time job was submitted.
- *JobID, User, Name* - Job identifiers as on the standard web UI.
- *Pool* - Current pool of job. Select another value to move job to another pool.
- *Priority* - Current priority. Select another value to change the job's priority
- *Maps/Reduces Finished*: Number of tasks finished / total tasks.
- *Maps/Reduces Running*: Tasks currently running.
- *Map/Reduce Fair Share*: The average number of task slots that this job should have at any given time according to fair sharing. The actual number of tasks will go up and down depending on how much compute time the job has had, but on average it will get its fair share amount.

In addition, it is possible to view an "advanced" version of the web UI by going to *http://<JobTracker URL>/scheduler?advanced*. This view shows two more columns:

- *Maps/Reduce Weight*: Weight of the job in the fair sharing calculations. This depends on priority and potentially also on job size and job age if the *sizebasedweight*and *NewJobWeightBooster* are enabled.

3. CapacityScheduler

The CapacityScheduler is designed to run Hadoop Map-Reduce as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster while running Map-Reduce applications.

The CapacityScheduler is designed to allow sharing a large cluster while giving each organization a minimum capacity guarantee. The central idea is that the available resources in the Hadoop Map-Reduce cluster are partitioned among multiple organizations who collectively fund the cluster based on computing needs. There is an added benefit that an organization can access any excess capacity no being used by others. This provides elasticity for the organizations in a cost-effective manner.

Sharing clusters across organizations necessitates strong support for multi-tenancy since each organization must be guaranteed capacity and safe-guards to ensure the shared cluster is impervious to single rouge job or user. The CapacityScheduler provides a stringent set of limits to ensure that a single job or user or queue cannot consume dispropotionate amount of resources in the cluster. Also, the JobTracker of the cluster, in particular, is a precious resource and the CapacityScheduler provides limits on initialized/pending tasks and jobs from a single user and queue to ensure fairness and stability of the cluster.

The primary abstraction provided by the CapacityScheduler is the concept of *queues*. These queues are typically setup by administrators to reflect the economics of the shared cluster.

Features

The CapacityScheduler supports the following features:

- Capacity Guarantees - Support for multiple queues, where a job is submitted to a queue.Queues are allocated a fraction of the capacity of the grid in the sense that a certain capacity of resources will be at their disposal. All jobs submitted to a queue will have access to the capacity allocated to the queue. Adminstrators can configure soft limits and optional hard limits on the capacity allocated to each queue.
- Security - Each queue has strict ACLs which controls which users can submit jobs to individual queues. Also, there are safe-guards to ensure that users cannot view and/or modify jobs from other users if so desired. Also, per-queue and system administrator roles are supported.
- Elasticity - Free resources can be allocated to any queue beyond it's capacity. When there is demand for these resources from queues running below capacity at a future point in time, as tasks scheduled on these resources complete, they will be assigned to jobs on queues running below the capacity. This ensures that resources are available in a

predictable and elastic manner to queues, thus preventing artifical silos of resources in the cluster which helps utilization.

- Multi-tenancy - Comprehensive set of limits are provided to prevent a single job, user and queue from monpolizing resources of the queue or the cluster as a whole to ensure that the system, particularly the JobTracker, isn't overwhelmed by too many tasks or jobs.
- Operability - The queue definitions and properties can be changed, at runtime, by administrators in a secure manner to minimize disruption to users. Also, a console is provided for users and administrators to view current allocation of resources to various queues in the system.
- Resource-based Scheduling - Support for resource-intensive jobs, wherein a job can optionally specify higher resource-requirements than the default, there-by accomodating applications with differing resource requirements. Currently, memory is the the resource requirement supported.
- Job Priorities - Queues optionally support job priorities (disabled by default). Within a queue, jobs with higher priority will have access to the queue's resources before jobs with lower priority. However, once a job is running, it will not be preempted for a higher priority job, *premption* is on the roadmap is currently not supported.

Installation

set the following property in the Hadoop config file *HADOOP_CONF_DIR/mapred-site.xml* to have Hadoop use the fair scheduler:

```
<property>
<name>mapred.jobtracker.taskScheduler</name>
<value>org.apache.hadoop.mapred.CapacityTaskScheduler</value>
</property>
```

Once you restart the cluster, you can check that the fair scheduler is running by going to *http://<jobtracker URL>/scheduler* on the JobTracker's web UI.

Setting up queues

You can define multiple queues to which users can submit jobs with the CapacityScheduler. To define multiple queues, you should use the *mapred.queue.names* property mapred-site.xml.

The CapacityScheduler can be configured with several properties for each queue that control the behavior of the Scheduler. This configuration is in the *capacity-scheduler.xml*.

Important settings for capacity-scheduler.xml

mapred.capacity-scheduler.queue.<queue-name>.capacity : Percentage of the number of slots in the cluster that are made to be available for jobs in this queue. The sum of capacities for all queues should be less than or equal 100.

mapred.capacity-scheduler.queue.<queue-name>.maximum-capacity : maximum-capacity defines a limit beyond which a queue cannot use the capacity of the cluster.This provides a means to limit how much excess capacity a queue can use. By default, there is no limit. The

maximum-capacity of a queue can only be greater than or equal to its minimum capacity. Default value of -1 implies a queue can use complete capacity of the cluster. This property could be to curtail certain jobs which are long running in nature from occupying more than a certain percentage of the cluster, which in the absence of pre-emption, could lead to capacity guarantees of other queues being affected. One important thing to note is that maximum-capacity is a percentage , so based on the cluster's capacity it would change. So if large no of nodes or racks get added to the cluster , maximum Capacity in absolute terms would increase accordingly.

mapred.capacity-scheduler.queue.<queue-name>.minimum-user-limit-percent : Each queue enforces a limit on the percentage of resources allocated to a user at any given time, if there is competition for them. This user limit can vary between a minimum and maximum value. The former depends on the number of users who have submitted jobs, and the latter is set to this property value. For example, suppose the value of this property is 25. If two users have submitted jobs to a queue, no single user can use more than 50% of the queue resources. If a third user submits a job, no single user can use more than 33% of the queue resources. With 4 or more users, no user can use more than 25% of the queue's resources. A value of 100 implies no user limits are imposed.

mapred.capacity-scheduler.queue.<queue-name>.supports-priority : If true, priorities of jobs will be taken into account in scheduling decisions.

Capacity scheduler lazily initializes the jobs before they are scheduled, for reducing the memory footprint on jobtracker. Following are the parameters, by which you can control the initialization of jobs per-queue.

mapred.capacity-scheduler.maximum-system-jobs : Maximum number of jobs in the system which can be initialized, concurrently, by the CapacityScheduler. Individual queue limits on initialized jobs are directly proportional to their queue capacities.

mapred.capacity-scheduler.queue.<queue-name>.maximum-initialized-active-tasks : The maximum number of tasks, across all jobs in the queue, which can be initialized concurrently. Once the queue's jobs exceed this limit they will be queued on disk.

mapred.capacity-scheduler.queue.<queue-name>.maximum-initialized-active-tasks-per-user : The maximum number of tasks per-user, across all the of the user's jobs in the queue, which can be initialized concurrently. Once the user's jobs exceed this limit they will be queued on disk.

mapred.capacity-scheduler.queue.<queue-name>.init-accept-jobs-factor : The multipe of (maximum-system-jobs * queue-capacity) used to determine the number of jobs which are accepted by the scheduler. The default value is 10. If number of jobs submitted to the queue exceeds this limit, job submission are rejected.

Example

```
<?xml version="1.0"?>

    <configuration>

    <!-- system limit, across all queues -->

    <property>
     <name>mapred.capacity-scheduler.maximum-system-jobs</name>
     <value>3000</value>
    <description>Maximum number of jobs in the system which can be initialized, concurrently,
    by the CapacityScheduler.
    </description>
    </property>

    <!-- queue: kellyA -->
    <property>
     <name>mapred.capacity-scheduler.queue.kellyA.capacity</name>
     <value>80</value>
    </property>
    <property>
     <name>mapred.capacity-scheduler.queue.kellyA.supports-priority</name>
     <value>false</value>
    </property>
    <property>
     <name>mapred.capacity-scheduler.queue.kellyA.minimum-user-limit-percent</name>
     <value>20</value>
    </property>

    <!-- queue: kellyB -->
    <property>
     <name>mapred.capacity-scheduler.queue.kellyB.capacity</name>
     <value>20</value>
    </property>
    <property>
     <name>mapred.capacity-scheduler.queue.kellyB.supports-priority</name>
     <value>false</value>
    </property>
    <property>
     <name>mapred.capacity-scheduler.queue.kellyB.minimum-user-limit-percent</name>
     <value>20</value>
    </property>
    <property>
     <name>mapred.capacity-scheduler.queue.kellyB.user-limit-factor</name>
     <value>1</value>
    </property>
```

```
<property>
 <name>mapred.capacity-scheduler.queue.kellyB.maximum-initialized-active-
tasks</name>
 <value>200000</value>
</property>
<property>
 <name>mapred.capacity-scheduler.queue.kellyB.maximum-initialized-active-tasks-per-
user</name>
 <value>100000</value>
</property>
```

Compression

Compression reduces the number of bytes written to/read from HDFS.

Compression effectively improves the efficiency of network bandwidth and disk space.This saves the amount of data being transfored between map nodes to reduce nodes.

LZO

LZO is a compression/decompression library.It has following characterstics

- Very fast decompression
- Requires an additional buffer during compression (size of 8KB or 64KB, depends on compression level)
- Requires no additional memory for decompression other than source and destination buffers.
- Allows the user to adjust the balance between compression ratio and compression speed, without affecting the speed of decompression.

Snappy

Snappy is a compression/decompression library. It does not aim for maximum compression, or compatibility with any other compression library; instead, it aims for very high speeds and reasonable compression.

For instance, compared to the fastest mode of zlib, Snappy is an order of magnitude faster for most inputs, but the resulting compressed files are anywhere from 20% to 100% bigger.

Typical compression ratios are about 1.5-1.7x for plain text, about 2-4x for HTML, and of course 1.0x for JPEGs, PNGs and other already-compressed data. Similar numbers for zlib in its fastest mode are 2.6-2.8x, 3-7x and 1.0x, respectively.

CompressionCodecs are implementation compression algorithms. **To use either LZO or Snappy compression codecs, LZO or snappy libraries must be installed on all the nodes in the cluster.**

Hadoop provides Compression Codecs for Gzip, LZO and Snappy and are
org.apache.hadoop.io.compress.DefaultCodec
org.apache.hadoop.io.compress.LzoCodec
org.apache.hadoop.io.compress.SnappyCodec
By Default Compression is not enabled in Mapreduce.Edit the following properties in mapred-site.xml to achieve compression

If job outputs are to be compressed set value as true.

```
<property>
<name>mapred.output.compress</name>
<value>false</value>
<description>Should the job outputs be compressed?
</description>
</property>
```

Which compression codec to be used while compressing job outputs. By default DefaultCodec uses. To use lzo or snappy replace with their corresponding codecs.

```
<property>
<name>mapred.output.compression.codec</name>
<value>org.apache.hadoop.io.compress.DefaultCodec</value>
<description>If the job outputs are compressed, how should they be compressed?
</description>
</property>
```

If map outputs are to be compressed set the value as true

```
<property>
<name>mapred.compress.map.output</name>
<value>false</value>
<description>Should the outputs of the maps be compressed before being
        sent across the network. Uses SequenceFile compression.
</description>
</property>
```

Specify which compression codec to be used while compressing map outptus

```
<property>
<name>mapred.map.output.compression.codec</name>
<value>org.apache.hadoop.io.compress.DefaultCodec</value>
<description>If the map outputs are compressed, how should they be
        compressed?
</description>
</property>
```

By Default Hadoop comes with the following CompressionCodecs. If you want to use any other codec apart from these, mention the same in **core-site.xml**

```
<property>
<name>io.compression.codecs</name>
<value>org.apache.hadoop.io.compress.DefaultCodec,
    org.apache.hadoop.io.compress.GzipCodec,
    org.apache.hadoop.io.compress.BZip2Codec
</value>
<description>A list of the compression codec classes that can be used for
    compression/decompression.</description>
</property>
```

Setting Up a Hadoop Cluster

The following are prerequisites for installing Hadoop.
Note step1 is mandatory, but recommended.

1. Create a user 'hadoop' in your machine. (If not present)
To test whether 'hadoop' user present in your machine, enter the following command at the console
$ su hadoop
If there is no 'hadoop' user in your machine, it gives
Unknown id: hadoop
If it is already present just ignore the rest of the point. Otherwise carry on
Enter the following commands at console to create hadoop user.

```
$ sudo adduser hadoop
Adding user `hadoop' ...
Adding new group `hadoop' (1001) ...
Adding new user `hadoop' (1001) with group `hadoop' ...
Creating home directory `/home/hadoop' ...
Copying files from `/etc/skel' ...
Enter new UNIX password: <give hadoop as password>
Retype new UNIX password: <just enter hadoop again>
passwd: password updated successfully
Changing the user information for hadoop
Enter the new value, or press ENTER for the default
    Full Name []: <Just Press enter>
    Room Number []: <Just Press enter>
    Work Phone []: <Just Press enter>
    Home Phone []: <Just Press enter>
    Other []: <Just Press enter>
Is the information correct? [Y/n] Y
```

2. Install Java (Java 6 or later is required to run Hadoop.)
Check whether java is already installed or not with the following command
$java
If console comes back with
Usage: java [-options] class [args...]
(to execute a class)
It means java is installed in your machine.
Otherwise install java with the following command (considering the UNIX flavor is Ubuntu)

$ sudo apt-get install openjdk-6-jdk

The latest stable Sun JDK is preferred option.

If you install java with above command JAVA_HOME would be **"/usr/lib/jvm/java-6-openjdk".**

3. Install openssh-server and openssh-client with the following commands
$ sudo apt-get install openssh-server
$ sudo apt-get install openssh-client

4. Go to hadoop user with the following command
$ su **hadoop**
Password:hadoop <we have given this earlier>

5. Download Hadoop from the Apache Hadoop releases page (http://hadoop.apache.org/core/releases.html) and unpack the contents of the distribution in a directory such as "/home/hadoop/".

All the following commands has to be executed while you are at hadoop user.

6. Untar the hadoop distribution in /home/hadoop with the following command
$ sudo tar xzf <Path of downloaded directory>/hadoop-1.0.3.tar.gz
Finally you will be able find /home/hadoop/hadoop-1.0.3 directory. Here onwards we refer this directory as **HADOOP_HOME.**

7. Change the owner of the Hadoop files to be the hadoop user and group.
$sudo chown -R hadoop:hadoop hadoop-1.0.3

8. SSH configuration (For passwordless)
$ ssh-keygen -t rsa
Just press on **enter,** you are prompted to enter any thing. Finally you find the following

Generating public/private rsa key pair.

```
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa):
Created directory '/home/hadoop/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop/.ssh/id_rsa.
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub.
The key fingerprint is:
43:02:31:17:b8:59:a9:a2:38:d0:79:3f:12:76:29:5a hadoop@kelly-desktop
The key's randomart image is:
+--[ RSA 2048]----+
|   +ooo          |
|   .+o           |
|  . . =...       |
|  . + E oo       |
|  o. * = S       |
|  +. . o .       |
|  . . ..         |
|                 |
|                 |
+-----------------+
```

9. Go to .ssh directory. You will find 2 files in that directory namely
id_rsa id_rsa.pub
10. Enter the following command to place the public key in authorized_keys
$ cat id_rsa.pub >> authorized_keys

Pseudo Distributed Mode

11. Go to HADOOP_HOME/conf directory.
12. Find the host name of your machine with the following command
$ hostname
hadoop-desktop
In my machine "hadoop-desktop" is my host name.

13. Edit the core-site.xml with the following properties.

```
<configuration>
   <property>
   <name>fs.default.name</name>
   <value><hostname of your machine>:9000</value>
   </property>
</configuration>
```

Description

fs.default.name : The property fs.default.name is a HDFS filesystem URI, whose host is the namenode's hostname or IP address, and port is the port that the namenode will listen on for RPCs. If no port is specified, the default of 8020 is used.

14. Edit the hdfs-site.xml with the following properties.

```
<configuration>
    <property>
    <name>dfs.replication</name>
    <value>1</value>
    </property>
    <property>
    <name>dfs.data.dir</name>
    <value>/home/hadoop/dfs/name</value>
    </property>
    <property>
    <name>dfs.name.dir</name>
    <value>/home/hadoop/dfs/data</value>
    </property>
</configuration>
```

Description

dfs.replication : Each block of the file is stored in number of servers specified for this property.

If the value of this property is 3, then each block of the file is stored in 3 DataNodes.

dfs.data.dir : This property specifies a list of the directories for a datanode to store its blocks. A datanode round robins writes between its storage directories.

dfs.name.dir: This property specifies a list of the directories where the namenode stores persistent filesystem metadata (the edit log, and filesystem image). A copy of each of the metadata file is stored in each directory for redundancy.

15. Edit the mapred-site.xml with the following properties.

```
<configuration>
    <property>
    <name>mapred.job.tracker</name>
    <value><hostname of your machine>:9001</value>
    </property>
    <property>
    <name>mapred.local.dir</name>
    <value>/home/hadoop/mapred/local</value>
    </property>
    <property>
    <name>mapred.system.dir</name>
    <value>/hadoop/mapred/system</value>
```

```
    </property>
</configuration>
```

Description

mapred.job.tracker : Specify the hostname or IP address and port that the job tracker will listen on. Default port is 8021.

mapred.local.dir : This property specifies the list of directories separated by commas. During MapReduce job, intermediate data and working files are written to temporary local files. Since this data includes the potentially very large output of map tasks, you need to ensure the mapred.local.dir property which controls the location of local temporary storage.

mapred.system.dir : MapReduce uses a distributed file system to share files (such as the job JAR file) with the tasktrackers that run the MapReduce tasks.
This property is used to specify a directory where these files can be stored.

16. Edit the masters file by replacing localhost with host name of your machine.

17. Edit the slaves file by replacing localhost with host name of your machine.

18. Edit the hadoop-env.sh by uncommenting the below line and replacing the JAVE_HOME of your machine (# is comment in shell script)
export JAVA_HOME=/usr/lib/jvm/java-6-openjdk
19. Go to HADOOP_HOME/bin directory
20. Format the Namenode with the following command.
    ./hadoop namenode -format
Caution: Format the namenode only when are setting up the cluster first time.
If you format the namenode, when there is any data in HDFS, you will loose all the data in HDFS. So don't format the namenode. If you stop the Hadoop cluster and while restarting the cluster you will find all the data in HDFS.

21. Start the Hadoop cluster with the following command.
    ./start-all.sh
The above command starts Namenode, DataNode, Secondary Namenode, JobTracker and TaskTracker.

22. To common the above, just type the following command in the CLI.
$ jps
The above command lists all the java processes started by the user.

23. There are separate scripts to start/stop HDFS, MapReduce and are
    start-dfs.sh -- To start HDFS only
    stop-dfs.sh -- To stop HDFS only
    start-mapred.sh -- To start MapReduce only

stop-mapred.sh -- To stop MapReduce only
stop-all.sh -- To stop entire Hadoop Cluster.

Cluster Mode

24. Perform the steps 1-9 on every machine, which we call node here onwards.
25. Designate one machine as master and others as slaves.
26. Assuming hostname of the master be master and slaves be slave1, slave2, slave3 etc.
27. Copy the id_rsa.pub (public key) to all the slave nodes. Execute the following command on master to copy the id_rsa.pub of the master machine.

$ ssh-copy-id hadoop@slave1
$ ssh-copy-id hadoop@slave2
$ ssh-copy-id hadoop@slave3

(Assuming 3 node cluster)
The above command copies the id_rsa.pub to authorized_keys of slave1, slave2, slave3.
The following steps has to be performed on all machines (both master and slaves)
28. Edit the masters file by replacing localhost with master in all the machines (both master and slaves)
29. Edit the slaves file by listing all slaves in the cluster (mention hostname of all the slaves)

Example: slaves file entries would be
slave1
slave2
slave3

30. Perform steps 11, 13, 14 and 15. Replace 1 by 3 in *dfs.replication* property in hdfs-site.xml.
31. Perform steps from 18 to 22.

References
Hadoop: The Definitive Guide by Tom White, First Edition
http://code.google.com/p/snappy/
http://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Oberhumer
http://hadoop.apache.org/docs/r1.0.3/fair_scheduler.html
http://hadoop.apache.org/docs/r1.0.3/capacity_scheduler.html

MRV2 Source code set-up and debugging

1) To build mrv2 we need to install protbuff and maven.

Once you install both of them set the following
root@kelly1:/usr/local/apache-maven# export M2_HOME=/usr/local/apache-maven/apache-maven-3.0.3
root@kelly1:/usr/local/apache-maven# export M2=$M2_HOME/bin
root@kelly1:/usr/local/apache-maven# export MAVEN_OPTS="-Xms256m -Xmx512m"
root@kelly1:/usr/local/apache-maven# export PATH=$M2:$PATH

```
root@kelly1:/usr/local/apache-maven# export JAVA_HOME=/usr/lib/jvm/java-1.6.0-openjdk
root@kelly1:/usr/local/apache-maven# export PATH=$JAVA_HOME/bin:$PATH
root@kelly1:/usr/local/apache-maven# mvn --version
Apache Maven 3.0.3 (r1075438; 2011-02-28 23:01:09+0530)
Maven home: /usr/local/apache-maven/apache-maven-3.0.3
Java version: 1.6.0_23, vendor: Sun Microsystems Inc.
Java home: /usr/lib/jvm/java-6-openjdk/jre
Default locale: en_IN, platform encoding: UTF-8
OS name: "linux", version: "3.0.0-12-generic", arch: "amd64", family: "unix"
```

export LD_LIBRARY_PATH=/usr/local/lib

For protbuff installation you can also reffer following links:
http://svn.apache.org/repos/asf/hadoop/common/trunk/hadoop-mapreduce-project/hadoop-yarn/README
http://svn.apache.org/repos/asf/hadoop/common/trunk/hadoop-mapreduce-project/INSTALL

2) clone the hadoop
mkdir mr2
cd mr2
svn checkout http://svn.apache.org/repos/asf/hadoop/common/trunk/ hadoop-trunk

3) After cloning the trunk: mvn clean install -DskipTestsmvneclipse:eclipsemvn clean package -Pdist -Dtar -DskipTests

4) General settings for MR2

```
export JAVA_HOME=/usr/lib/jvm/java-1.6.0-openjdk
export PATH=$JAVA_HOME/bin:$PATH

export HADOOP_DEV_HOME=`pwd`
export HADOOP_MAPRED_HOME=${HADOOP_DEV_HOME}
export HADOOP_COMMON_HOME=${HADOOP_DEV_HOME}
export HADOOP_HDFS_HOME=${HADOOP_DEV_HOME}
export YARN_HOME=${HADOOP_DEV_HOME}
export HADOOP_CONF_DIR=${HADOOP_DEV_HOME}/conf/
export YARN_CONF_DIR=~${HADOOP_DEV_HOME}/conf/
export LD_LIBRARY_PATH=/usr/local/lib
```

5) You can use the attched configurations for reference

6) If everything is proper then now you should be able start hadoop
hadoop-0.24.0-SNAPSHOT$ bin/hadoopnamenode -format

And start the HDFS services:
hadoop-0.24.0-SNAPSHOT$ sbin/hadoop-daemon.sh start namenode
hadoop-0.24.0-SNAPSHOT$ sbin/hadoop-daemon.sh start datanode

And the new MR2 services
hadoop-0.24.0-SNAPSHOT$ bin/yarn-daemon.sh start resourcemanager
hadoop-0.24.0-SNAPSHOT$ bin/yarn-daemon.sh start nodemanager
hadoop-0.24.0-SNAPSHOT$ bin/yarn-daemon.sh start historyserver

Make sure all needed services are running:
hadoop-0.24.0-SNAPSHOT$ jps
Jps
ResourceManager
JobHistoryServer
NodeManager
DataNode
NameNode

you can also reffer http://www.cloudera.com/blog/2011/11/building-and-deploying-mr2/

7) For contribution:
Please refer http://wiki.apache.org/hadoop/HowToContribute

For debugging:
As you are already built the mapreduce project, you can add debug conf's in yarn-en.sh and put debug points in the code and start analyzing it.

1) Copy yarn-env.sh(hadoop-dist/target/hadoop-0.24.0-SNAPSHOT/etc/hadoop/yarn-env.sh) into YARN_CONF_DIR

2) Add following debug conf's

YARN_RESOURCEMANAGER_OPTS="-Xdebug
Xrunjdwp:transport=dt_socket,server=y,address=5000"
YARN_NODEMANAGER_OPTS="-Xdebug
Xrunjdwp:transport=dt_socket,server=y,address=5001"
YARN_JOB_HISTORYSERVER_OPTS="-Xdebug –
Xrunjdwp:transport=dt_socket,server=y,address=5002"

3) In eclipse, go to corresponding class file for example ResourceManager.java -> debug as -> Remote Java Apllication ----> new ---> set corresponding port

Now it's ready to debug.

For testing your patch you can set-up separate testbed.

For contribution: http://wiki.apache.org/hadoop/HowToContribute

References

Hadoop definite guide

HBase definite Guide

The Hadoop Distributed FileSystem paper(http://dl.acm.org/citation.cfm?id=1914427)

Data-Intensive Text Processing with MapReduce(http://www.morganclaypool.com/doi/abs/10.2200/S00274ED1V01Y201006HLT007)

Scheduling for real-time mobile MapReducesystems(http://dl.acm.org/citation.cfm?id=2002305&dl=ACM&coll=DL&CFID=113190881&CFTOKEN=92325790)

Apache Pig

Introduction

→ Pig is a high-level procedural language for querying large semi-structured data sets using Hadoop and MapReduce platform.

➔ Pig is a simple language that executes statements.

➔ A statement is an operation that takes input (such as bag, which represents a set of tuples) and emits another bag as its output.

➔ Pig follows the following format while executing a script

➔ Read the data from the file system.

➔ Perform a number of operations on the data.

➔ Store the final result back to file system.

Grep Example

Pig Code

```
lines = LOAD '/input/sample.txt';
hadoopLines = FILTER lines BY $0 MATCHES '.*hadoop+.*';
STORE hadoopLines INTO '/output/cleanedLines';
```

Explanation

➔ Read the '/input/sample.txt' into a bag 'sample' that represents a collection of tuples.

➔ Filter each tuple (represented as $0) with a regular expression, looking for the character sequence hadoop.

➔ Store the hadoopLines bag, which contains all of those tuples from '/input/sample.txt' that contain 'hadoop' into a new file called '/output/cleanedLines'

Installing Pig

➔ Download a recent stable release from Apache download Mirrors.

➔ Unpack the downloaded Pig installation.

➔ Export HADOOP_HOME as Hadoop Home Directory.

➔ export HADOOP_HOME=<path-to-hadoop>

➔ Go to Pig directory.

➔ Test the Pig installation with the simple command

➔ $<Pig_HOME>/bin/pig -help.

Running Pig

Pig has two execution modes or exec types:

- **Local Mode – To run Pig in local mode, you need access to a single machine; all files are installed and run using your local host and file system. Specify local mode using -x flag**

$ pig -x local

- **Mapreduce Mode – To run Pig in mapreduce mode, you need access to a Hadoop cluster and HDFS installation. Mapreduce mode is the default mode.**
- Pig commands can be executed using the following modes
- Interactive Mode
- Batch Mode

Interactive Mode & Batch Mode Execution

- **To execute the pig statements in interactive mode, invoke the Grunt shell by typing the "pig" command.**
- **Enter the pig statements interactively at the grunt prompt.**
- **Running Grep Example in interactive mode**

```
grunt> lines = LOAD '/input/sample.txt';
grunt> hadoopLines = FILTER lines BY $0 MATCHES '.*hadoop+.*';
2012-09-26 22:19:05,548 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO
_CHARARRAY 1 time(s).
grunt> STORE hadoopLines INTO '/output/cleanedLines';
2012-09-26 22:19:30,128 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO
_CHARARRAY 1 time(s).
```

- Save the Pig latin statements in a file run it as below.

$ bin/pig <file-name>.pig

Kelly Technologies

Pig Operators

- LOAD – Loads data from the file system or other storage into a relation
- STORE – Saves a relation to the file system or other storage
- DUMP – Prints a relation to the console
- FILTER – Removes unwanted rows from a relation
- DISTINCT – Removes duplicate rows from a relation
- FOREACH...GENERATE – Adds or removes fields from a relation
- STREAM – Transforms a relation using an external program
- JOIN – Joins two or more relations
- COGROUP – Groups the data in two or more relations
- GROUP – Groups the data in a single relation
- CROSS – Creates the cross product of two or more relations
- ORDER – Sorts a relation by one or more fields
- LIMIT – Limits the size of a relation to a maximum number of tuples
- UNION – Combines two or more relations into one
- SPLIT – Splits a relation into two or more relations.
-

built in Functions

- AVG – Calculates the average value of entries in a bag.
- CONCAT – Concatenates two byte arrays or two character arrays together.
- COUNT – Calculates the number of entries in a bag.
- MAX – Calculates the maximum value of entries in a bag.
- MIN – Calculates the minimum value of entries in a bag.

- SUM – Calculates the sum of the values of entries in a bag.
- TOKENIZE – Tokenizes a character array into a bag of its constituent words.
- PigStorage – Loads or stores relations using a field-delimited text format. Each line is broken into fields using a configurable field delimiter (defaults to a tab character) to be stored in the tuple's fields. It is the default storage when none is specified.
- BinStorage – Loads or stores relations from or to binary files. An internal Pig format is used that uses Hadoop Writable objects.
- TextLoader – Loads relations from a plain-text format. Each line corresponds to a tuple whose single field is the line of text.
-

Example using PigStorage & foreach

$ pig -x local

grunt> passwdLines = load '/etc/passwd' using PigStorage(':') As (user:chararray,a:chararray,b:chararray,c:chararray,d:chararray,e:chararray,shell:chararray);

grunt> dump passwdLines;

grunt> userShell = foreach passwdLines generate user,shell;

grunt> dump userShell;

Example using GROUP BY, ORDER BY & AVG

Example to find the avg click for each URL.

$pig

grunt> urls = LOAD '/input/urlcount.txt' AS (url:chararray, count:int);

grunt> urlCount = GROUP urls BY url;

grunt> urlAvg = foreach urlCount generate group, AVG(urls.count) as urlC;

grunt> d = ORDER urlAvg BY urlC DESC;

grunt> STORE d INTO '/output/urlAvg' using PigStorage('#');

GROUP BY groups all the same urls together.

group is a reserved word in the pig to identify each group.

ORDER BY orders the urls by their average in the descending order. If we don't mention DESC, it will be ordered in the ascending order.

PARALLEL

- PARALLEL clause increases the parallelism of a job
- PARALLEL sets the number of reduce tasks for the MapReduce jobs generated by Pig. The default value is 1.
- PARALLEL only affects the number of reduce tasks. Map parallelism is determined by the input file, one map for each Input Split.
- If you don't specify PARALLEL, you still get the same map parallelism but only one reduce task.
- Specify the PARALLEL clause with any operator that starts a reduce phase, which includes COGROUP, CROSS, DISTINCT, GROUP, JOIN (inner), JOIN (outer), and ORDER.

PARALLEL – An Example

Let us consider the avg click for each URL pig script, with PARALLEL clause.

$ pig

grunt> urls = LOAD '/input/urlcount.txt' AS (url:chararray, count:int);

grunt> urlCount = GROUP urls BY url PARALLEL 3; grunt> urlAvg = foreach urlCount generate group, AVG(urls.count) as urlC;

grunt> d = ORDER urlAvg BY urlC DESC;

grunt> STORE d INTO '/output/urlAvg' using PigStorage('#');

For the above example 3 reducers will be spawned.

UDFs

➔ UDFs are required to define custom processing.

➔ UDFs can be written in the Java, Python, JavaScript and Ruby and permit Pig to support custom processing.

➔ Support for writing UDFs in Python, JavaScript and Ruby still evolving.

➔ UDFs provide an oppertunity to extend Pig into your application domain.

➔

How to Write Java UDF

UDFs can be developed by extending EvalFunc class and overriding exec method.

Example : This UDF replaces a given string with another string.

```
package kelly.training.pig.udf;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.pig.EvalFunc;

import org.apache.pig.data.Tuple;

import org.apache.pig.impl.util.UDFContext;

public class Transform extends EvalFunc<String> {

    public String exec(Tuple input) throws IOException {

        if (input == null || input.size() == 0) {

            return null;

        }

        Configuration conf = UDFContext.getUDFContext().getJobConf();

        String from = conf.get("replace.string");
```

```
        if (from == null) {
                throw new IOException("replace.string should not be null");
        }
        String to = conf.get("replace.by.string");
        if (to == null) {
                throw new IOException("replace.by.string should not be null");
        }
        try {
                String str = (String) input.get(0);
                return str.replace(from, to);
        } catch (Exception e) {
                throw new IOException("Caught exception processing input row", e);
        }
    }
}
```

Example using UDF

```
-- sed.pig
-- pig-training.jar contains custom udfs developed by a user. It has to be registered
-- before running the pig script.
REGISTER pig-training.jar;
A = LOAD '/input/sample.txt' AS (line: chararray);
B = FOREACH A GENERATE kelly.training.pig.udf.Transform(line);
STORE B INTO '/output/sed';
```

pig-training.jar contains the Transform UDF. These UDFs jar has to be registered.

While using custom UDF, mention UDF with package name.

How to Run the Sed Example using UDF

$ pig -x local -P user.properties sed.pig

Or

$ pig -P user.properties sed.pig

user.properties is propeties file, which contains the following parameters

replace.string=

replace.by.string=

While running the sed.pig, mention the path of the properties file using -P option.

sed.pig:

```
-- sed.pig
-- pig-training.jar contains custom udfs developed by a user. It has to be registered
-- before running the pig script.
REGISTER /home/kelly1/Work/Hadoop\ training/pig_training/pig-training.jar;
--  A = LOAD '/input/sample.txt' AS (line: chararray);
A = LOAD '/home/kelly1/Work/Hadoop\ training/pig_training/sample.txt' AS (line:
chararray);
B = FOREACH A GENERATE kelly.training.pig.udf.Transform(line);
-- STORE B INTO '/output/sed';
STORE B INTO '/home/kelly1/Work/sed';
```

user.properties:

```
replace.string=hadoop
replace.by.string=Hadoop
```

Transform.java:

```
package kelly.training.pig.udf;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.util.UDFContext;

public class Transform extends EvalFunc<String> {
  public String exec(Tuple input) throws IOException {
    if (input == null || input.size() == 0) {
      return null;
    }
    Configuration conf = UDFContext.getUDFContext().getJobConf();
    String from = conf.get("replace.string");
    if (from == null) {
      throw new IOException("replace.string should not be null");
    }
    String to = conf.get("replace.by.string");
    if (to == null) {
      throw new IOException("replace.by.string should not be null");
    }
    try {
      String str = (String) input.get(0);
      return str.replace(from, to);
    } catch (Exception e) {
      throw new IOException("Caught exception processing input row", e);
    }
  }
}
```

sample.txt

```
hadoop is a framework to process big data.
hadoop has two components namely hdfs and mapreduce.
hdfs is a distributed data storage framework where as mapreduce is distributed data
processing framework.
```

Apache Pig

Introduction

➔ Pig is a high-level procedural language for querying large semi-structured data sets using Hadoop and MapReduce platform.

➔ Pig is a simple language that executes statements.

➔ A statement is an operation that takes input (such as bag, which represents a set of tuples) and emits another bag as its output.

➔ Pig follows the following format while executing a script

➔ Read the data from the file system.

➔ Perform a number of operations on the data.

➔ Store the final result back to file system.

Grep Example

Pig Code

```
lines = LOAD '/input/sample.txt';
hadoopLines = FILTER lines BY $0 MATCHES '.*hadoop+.*';
STORE hadoopLines INTO '/output/cleanedLines';
```

Explanation

➔ Read the '/input/sample.txt' into a bag 'sample' that represents a collection of tuples.

➔ Filter each tuple (represented as $0) with a regular expression, looking for the character sequence hadoop.

➔ Store the hadoopLines bag, which contains all of those tuples from '/input/sample.txt' that contain 'hadoop' into a new file called '/output/cleanedLines'

Installing Pig

➔ Download a recent stable release from Apache download Mirrors.

➔ Unpack the downloaded Pig installation.

➔ Export HADOOP_HOME as Hadoop Home Directory.

➔ export HADOOP_HOME=<path-to-hadoop>

➔ Go to Pig directory.

➔ Test the Pig installation with the simple command

➔ $<Pig_HOME>/bin/pig -help.

Running Pig

Pig has two execution modes or exec types:

- Local Mode – To run Pig in local mode, you need access to a single machine; all files are installed and run using your local host and file system. Specify local mode using -x flag

$ pig -x local

- Mapreduce Mode – To run Pig in mapreduce mode, you need access to a Hadoop cluster and HDFS installation. Mapreduce mode is the default mode.
- Pig commands can be executed using the following modes
- Interactive Mode
- Batch Mode

Interactive Mode & Batch Mode Execution

- To execute the pig statements in interactive mode, invoke the Grunt shell by typing the "pig" command.
- Enter the pig statements interactively at the grunt prompt.
- Running Grep Example in interactive mode

```
grunt> lines = LOAD '/input/sample.txt';
grunt> hadoopLines = FILTER lines BY $0 MATCHES '.*hadoop+.*';
2012-09-25 22:19:05,548 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO
_CHARARRAY 1 time(s).
grunt> STORE hadoopLines INTO '/output/cleanedLines';
2012-09-25 22:19:30,128 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO
_CHARARRAY 1 time(s).
```

- Save the Pig latin statements in a file run it as below.

$ bin/pig <file-name>.pig

Pig Operations

- LOAD – Loads data from the file system or other storage into a relation
- STORE – Saves a relation to the file system or other storage
- DUMP – Prints a relation to the console
- FILTER – Removes unwanted rows from a relation
- DISTINCT – Removes duplicate rows from a relation
- FOREACH...GENERATE – Adds or removes fields from a relation
- STREAM – Transforms a relation using an external program
- JOIN – Joins two or more relations
- COGROUP – Groups the data in two or more relations
- GROUP – Groups the data in a single relation
- CROSS – Creates the cross product of two or more relations
- ORDER – Sorts a relation by one or more fields
- LIMIT -- Limits the size of a relation to a maximum number of tuples
- UNION – Combines two or more relations into one
- SPLIT – Splits a relation into two or more relations.
-

Built-in Functions

- AVG – Calculates the average value of entries in a bag.
- CONCAT – Concatenates two byte arrays or two character arrays together.
- COUNT – Calculates the number of entries in a bag.
- MAX – Calculates the maximum value of entries in a bag.

- MIN – Calculates the minimum value of entries in a bag.
- SUM – Calculates the sum of the values of entries in a bag.
- TOKENIZE – Tokenizes a character array into a bag of its constituent words.
- PigStorage – Loads or stores relations using a field-delimited text format. Each line is broken into fields using a configurable field delimiter (defaults to a tab character) to be stored in the tuple's fields. It is the default storage when none is specified.
- BinStorage – Loads or stores relations from or to binary files. An internal Pig format is used that uses Hadoop Writable objects.
- TextLoader – Loads relations from a plain-text format. Each line corresponds to a tuple whose single field is the line of text.
-

Example using PigStorage & foreach

$ pig -x local

grunt> passwdLines = load '/etc/passwd' using PigStorage(':') As (user:chararray,a:chararray,b:chararray,c:chararray,d:chararray,e:chararray,shell:chararray);

grunt> dump passwdLines;

grunt> userShell = foreach passwdLines generate user,shell;

grunt> dump userShell;

Example using GROUP BY, ORDER BY & AVG

Example to find the avg click for each URL.

$ pig

grunt> urls = LOAD '/input/urlcount.txt' AS (url:chararray, count:int);

grunt> urlCount = GROUP urls BY url;

grunt> urlAvg = foreach urlCount generate group, AVG(urls.count) as urlC;

grunt> d = ORDER urlAvg BY urlC DESC;

grunt> STORE d INTO '/output/urlAvg' using PigStorage('#');

GROUP BY groups all the same urls together.

group is a reserved word in the pig to identify each group.

ORDER BY orders the urls by their average in the descending order. If we don't mention DESC it will be ordered in the ascending order.

PARALLEL

- PARALLEL clause increases the parallelism of a job
- PARALLEL sets the number of reduce tasks for the Map Reduce jobs generated by Pig. The default value is 1.
- PARALLEL only affects the number of reduce tasks. Map parallelism is determined by the input file, one map for each Input Split.
- If you don't specify PARALLEL, you still get the same map parallelism but only one reduce task.
- Specify the PARALLEL clause with any operator that starts a reduce phase, which includes COGROUP, CROSS, DISTINCT, GROUP, JOIN (inner), JOIN (outer), and ORDER.

PARALLEL – An Example

Let us consider the avg click for each URL pig script, with PARALLEL clause.

$ pig

grunt> urls = LOAD '/input/urlcount.txt' AS (url:chararray, count:int);

grunt> urlCount = GROUP urls BY url PARALLEL 3;

grunt> urlAvg = foreach urlCount generate group, AVG(urls.count) as urlC;

grunt> d = ORDER urlAvg BY urlC DESC;

grunt> STORE d INTO '/output/urlAvg' using PigStorage('#');

For the above example 3 reducers will be spawned.

UDF

➔ UDFs are required to define custom processing.

➔ UDFs can be written in the Java, Python, JavaScript and Ruby and permit Pig to support custom processing.

➔ Support for writing UDFs in Python, JavaScript and Ruby still evolving.

➔ UDFs provide an oppertunity to extend Pig into your application domain.

➔

How to Write Java UDF

UDFs can be developed by extending EvalFunc class and overriding exec method.

Example : This UDF replaces a given string with another string.

```
package kelly.training.pig.udf;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.util.UDFContext;
public class Transform extends EvalFunc<String> {
    public String exec(Tuple input) throws IOException {
        if(input == null || input.size() == 0) {
            return null;
        }
        Configuration conf = UDFContext.getUDFContext().getJobConf();
```

```
        String from = conf.get("replace.string");
        if (from == null) {
                throw new IOException("replace.string should not be null");
        }
        String to = conf.get("replace.by.string");
        if (to == null) {
                throw new IOException("replace.by.string should not be null");
        }
        try {
                String str = (String) input.get(0);
                return str.replace(from, to);
        } catch (Exception e) {
                throw new IOException("Caught exception processing input row", e);
        }
    }
}
```

Pig Example using UDF

```
-- sed.pig
-- pig-training.jar contains custom udfs developed by a user. It has to be registered
-- before running the pig script.
REGISTER pig-training.jar;
A = LOAD '/input/sample.txt' AS (line: chararray);
B = FOREACH A GENERATE kelly.training.pig.udf.Transform(line);
```