MODULE II

CHAPTER 2

Hadoop HDFS and MapReduce

University Prescribed Syllabus w.e.f Academic Year 2022-2023

2.1 Distributed File Systems: Physical Organization of Compute Nodes, Large-Scale File-System Organization.

2.2 MapReduce: The Map Tasks, Grouping by Key, The Reduce Tasks,Combiners, Details of MapReduce Execution, Coping With Node Failures.

2.3 Algorithms Using MapReduce: Matrix-Vector Multiplication by MapReduce,Relational-Algebra Operations, Computing Selections by MapReduce,Computing Projections by MapReduce, Union ,Intersection,and Difference by MapReduce

2.4 Hadoop Limitations

2.1 ARCHITECTURE OF HADOOP HDFS

GQ. Explain Architecture of HDFS in detail. (5 Marks)

- The HDFS is the primary storage system used by Hadoop applications. HDFS is a distributed file system and a framework provided by Hadoop for the analysis and transformation of huge data sets which uses the MapReduce paradigm. The HDFS is based on Google File System (GFS). It provides high-performance access to data across Hadoop clusters (thousands of computers), HDFS has become a key tool for managing pools of big data and supporting big data analytics applications.
- HDFS is usually deployed on commodity hardware of low-cost where the possibility of server failures is common. The file system is designed to be highly fault-tolerant.

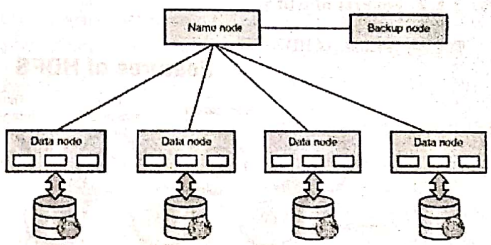

Fig. 2.1.1 : Hadoop Distributed File System

- The HDFS facilitates the rapid transfer of data between different computer nodes and enables Hadoop systems to precede its execution even if one or more nodes gets failed. That decreases the risk of catastrophic failure, even in the event that numerous nodes fail.
- The architecture used by HDFS is known as master/slave architecture.
- NameNode which manages the metadata of file system and DataNode which stores the actual data.
- The HDFS namespace is a hierarchy of files and directories. Inodes are used to represent these file and directories. Inodes are used to record attributes such as permissions, modification and access times etc. The file content is split into large blocks and each block of the file is independently replicated at multiple DataNodes.
- The tree structure of namespace is maintained by the NameNode. It maps the blocks to DataNodes. In a cluster there may be hundreds of DataNodes and thousands of HDFS clients per cluster, as number of application tasks can be executed by each DataNode simultaneously.

2.1.1 Advantages and Disadvantages of HDFS

Advantages of HDFS	Disadvantages of HDFS
1. High scalability	1. Still rough - means software under active development.
2. Low limitation.	2. Programming model is very restrictive.
3. Open source.	3. Cluster management is high.
4. Low cost.	

2.1.2 Features of HDFS

The key features of HDFS are:

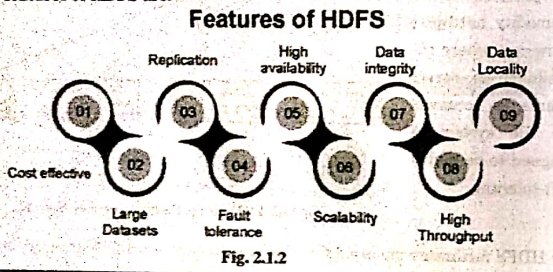

Fig. 2.1.2

1. Cost-effective	2. Large Datasets/ Variety and volume of data
3. Replication	4. Fault Tolerance and reliability
5. High Availability	6. Scalability
7. Data Integrity	8. High Throughput
9. Data Locality	

1. Cost-effective

In HDFS architecture, the DataNodes, which stores the actual data are inexpensive commodity hardware, thus reduces storage costs.

2. Large Datasets/ Variety and volume of data

HDFS can store data of any size (ranging from megabytes to petabytes) and of any formats (structured, unstructured).

3. Replication

- Data Replication is one of the most important and unique features of HDFS. In HDFS replication of data is done to solve the problem of data loss in unfavorable conditions like crashing of a node, hardware failure, and so on.
- The data is replicated across a number of machines in the cluster by creating replicas of blocks. The process of replication is maintained at regular intervals of time by HDFS and HDFS keeps creating replicas of user data on different machines present in the cluster.
- Hence whenever any machine in the cluster gets crashed, the user can access their data from other machines that contain the blocks of that data. Hence there is no possibility of a loss of user data.

4. Fault Tolerance and reliability

- HDFS is highly fault-tolerant and reliable. HDFS creates replicas of file blocks depending on the replication factor and stores them on different machines.
- If any of the machines containing data blocks fail, other DataNodes containing the replicas of that data blocks are available. Thus, ensuring no loss of data and makes the system reliable even in unfavorable conditions.

5. High Availability

- The High availability feature of Hadoop ensures the availability of data even during NameNode or DataNode failure.
- Since HDFS creates replicas of data blocks, if any of the DataNodes goes down, the user can access his data from the other DataNodes containing a copy of the same data block.
- Also, if the active NameNode goes down, the passive node takes the responsibility of the active NameNode. Thus, data will be available and accessible to the user even during a machine crash.

6. Scalability

- As HDFS stores data on multiple nodes in the cluster, when requirements increase, we can scale the cluster.
- There is two scalability mechanism available: Vertical scalability – add more resources (CPU, Memory, Disk) on the existing nodes of the cluster.

- Another way is horizontal scalability – Add more machines in the cluster. The horizontal way is preferred since we can scale the cluster from 10s of nodes to 100s of nodes on the fly without any downtime.

7. Data Integrity

- Data integrity refers to the correctness of data. HDFS ensures data integrity by constantly checking the data against the checksum calculated during the write of the file.
- While file reading, if the checksum does not match with the original checksum, the data is said to be corrupted. The client then opts to retrieve the data block from another DataNode that has a replica of that block. The NameNode discards the corrupted block and creates an additional new replica.

8. High Throughput

Hadoop HDFS stores data in a distributed fashion, which allows data to be processed parallelly on a cluster of nodes. This decreases the processing time and thus provides high throughput.

9. Data Locality

- Data locality means moving computation logic to the data rather than moving data to the computational unit.
- In the traditional system, the data is brought at the application layer and then gets processed.
- But in the present scenario, due to the massive volume of data, bringing data to the application layer degrades the network performance.
- In HDFS, we bring the computation part to the Data Nodes where data resides. Hence, with Hadoop HDFS, we are not moving computation logic to the data, rather than moving data to the computation logic. This feature reduces the bandwidth utilization in a system.

2.2 PHYSICAL ORGANIZATION OF COMPUTER NODES

GQ. What do you mean by Physical Organization of Compute Nodes? **(5 Marks)**
GQ. Write short note on cluster Computing. **(5 Marks)**
GQ. What are the problems faced by parallel computing architecture and state the solution in detail? **(5 Marks)**

- The new parallel-computing architecture, sometimes called cluster computing, is organized as follows. Compute nodes are stored on racks, perhaps 8–64on a rack.
- The nodes on a single rack are connected by a network, typically gigabit Ethernet. There can be many racks of compute nodes, and racks are connected by another level of network or a switch.
- The bandwidth of inter-rack communication is somewhat greater than the intra rack Ethernet, but given the number of pairs of nodes that might need to communicate between racks, this bandwidth may be essential.
- However, there may be many more racks and many more compute nodes per rack. It is a fact of life that components fail, and the more components, such as compute nodes and interconnection networks, a system has, the more frequently something in the system will not be working at any given time. Some important calculations take minutes or even hours on thousands of compute nodes. If we had to abort and restart the computation every time one component failed, then the computation might never complete successfully.
- The solution to this problem takes two forms:
  1. Files must be stored redundantly. If we did not duplicate the file at several compute nodes, then if one node failed, all its files would be unavailable until the node is replaced. If we did not back up the files at all, and the disk crashes, the files would be lost forever.
  2. Computations must be divided into tasks, such that if any one task failsto execute to completion, it can be restarted without affecting other tasks. This strategy is followed by the map-reduce programming system

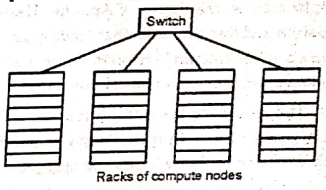

Fig. 2.2.1 : Architecture of a large scale computing system

2.3 LARGE SCALE FILE SYSTEM ORGANIZATION

To exploit cluster computing, files must look and behave somewhat differently from the conventional file systems found on single computers. This new file system, often called a distributed file system or DFS (although this term had other meanings in the past), is typically used as follows :

- Files can be enormous, possibly a terabyte in size. If you have only smallfiles, there is no point using a DFS for them.

- Files are rarely updated. Rather, they are read as data for some calculation, and possibly additional data is appended to files from time to time.
- Files are divided into chunks, which are typically 64 megabytes in size. Chunks are replicated, perhaps three times, at three different compute nodes. Moreover, the nodes holding copies of one chunk should be located on different racks, so we don't lose all copies due to a rack failure. Normally, both the chunk size and the degree of replication can be decided by the user.
- To find the chunks of a file, there is another small file called the master node or name node for that file. The master node is itself replicated, and a directory for the file system as a whole knows where to find its copies. The directory itself can be replicated, and all participants using the DFS know where the directory copies are.

2.4 INTRODUCTION TO MAPREDUCE

UQ. What is MapReduce? Explain the role of Combiner with help of an example. (MU - Dec. 18, 10 Marks)

UQ. Explain Concept of MapReduce using an example. (MU - Dec. 17, 5 Marks)

MapReduce is a programming paradigm that enables massive scalability across hundreds or thousands of servers in a Hadoop cluster. As the processing component, MapReduce is the heart of **Apache Hadoop.** The term "MapReduce" refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

2.4.1 How MapReduce Works

- At a high level, MapReduce breaks input data into fragments and distributes them across different machines.
- The input fragments consist of key-value pairs. Parallel map tasks process the chunked data on machines in a cluster. The mapping output then serves as input for the reduce stage. The reduce task combines the result into a particular key-value pair output and writes the data to HDFS.
- The Hadoop Distributed File System usually runs on the same set of machines as the MapReduce software. When the framework executes a job on the nodes that also store the data, the time to complete the tasks is reduced significantly.

2.4.2 Basic Terminology of Hadoop MapReduce

As we mentioned above, MapReduce is a processing layer in a Hadoop environment. MapReduce works on tasks related to a job. The idea is to tackle one large request by slicing it into smaller units.

JobTracker and TaskTracker

- In the early days of Hadoop (version 1), JobTracker and TaskTracker daemons ran operations in MapReduce. At the time, a Hadoop cluster could only support MapReduce applications.
- A JobTracker controlled the distribution of application requests to the compute resources in a cluster. Since it monitored the execution and the status of MapReduce, it resided on a master node.
- A TaskTracker processed the requests that came from the JobTracker.

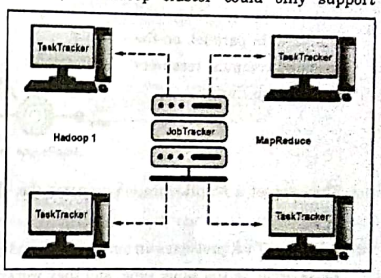

Fig. 2.4.1

- All task trackers were distributed across the slave nodes in a Hadoop cluster.

YARN

Later in Hadoop version 2 and above, YARN became the main resource and scheduling manager. Hence the name Yet Another Resource Manager. Yarn also worked with other frameworks for the distributed processing in a Hadoop cluster.

MapReduce Job

- A MapReduce job is the top unit of work in the MapReduce process. It is an assignment that Map and Reduce processes need to complete. A job is divided into smaller tasks over a cluster of machines for faster execution.
- The tasks should be big enough to justify the task handling time. If you divide a job into unusually small segments, the total time to prepare the splits and create tasks may outweigh the time needed to produce the actual job output.

MapReduce Task

MapReduce jobs have two types of tasks.

(1) A Map Task is a single instance of a MapReduce app. These tasks determine which records to process from a data block. The input data is split and analyzed, in parallel, on the assigned compute resources in a Hadoop cluster.

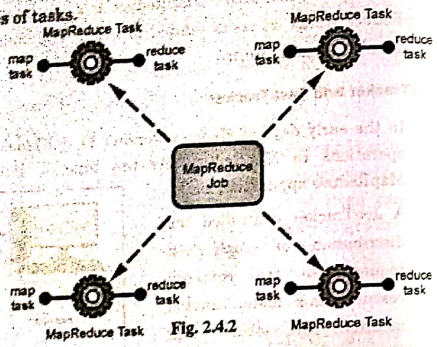

Fig. 2.4.2

This step of a MapReduce job prepares the <key, value> pair output for the reduce step.

(2) A Reduce Task processes an output of a map task. Similar to the map stage, all reduce tasks occur at the same time, and they work independently. The data is aggregated and combined to deliver the desired output. The final result is a reduced set of <key, value> pairs which MapReduce, by default, stores in HDFS.

2.4.3 How Hadoop Map and Reduce Work Together

- As the name suggests, MapReduce works by processing input data in two stages – Map and Reduce. To demonstrate this, we will use a simple example with counting the number of occurrences of words in each document.
- The final output we are looking for is: How many times the words Apache, Hadoop, Class, and Track appear in total in all documents.
- For illustration purposes, the example environment consists of three nodes. The input contains six documents distributed across the cluster. We will keep it simple here, but in real circumstances, there is no limit. You can have thousands of servers and billions of documents.

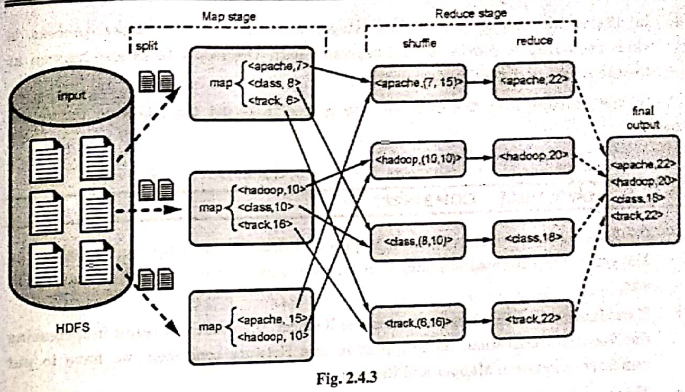

Fig. 2.4.3

1. First, in the map stage, the input data (the six documents) is split and distributed across the cluster (the three servers). In this case, each map task works on a split containing two documents. During mapping, there is no communication between the nodes. They perform independently.
2. Then, map tasks create a <key, value> pair for every word. These pairs show how many times a word occurs. A word is a key, and a value is its count. For example, one document contains three of four words we are looking for: Apache 7 times, Class 8 times, and Track 6 times. The key-value pairs in one map task output look like this:
   - <apache, 7>
   - <class, 8>
   - <track, 6>

   This process is done in parallel tasks on all nodes for all documents and gives a unique output.
3. After input splitting and mapping completes, the outputs of every map task are shuffled. This is the first step of the Reduce stage. Since we are looking for the frequency of occurrence for four words, there are four parallel Reduce tasks. The reduce tasks can run on the same nodes as the map tasks, or they can run on any other node.

   The shuffle step ensures the keys Apache, Hadoop, Class, and Track are sorted for the reduce step. This process groups the values by keys in the form of <key, value-list> pairs.

4. In the reduce step of the Reduce stage, each of the four tasks process a <key, value-list> to provide a final key-value pair. The reduce tasks also happen at the same time and work independently.

In our example from the diagram, the reduce tasks get the following individual results:

- <apache, 22> • <hadoop, 20> • <class, 18> • <track, 22>

2.5 MAPREDUCE – COMBINERS

- Combiner always works in between Mapper and Reducer. The output produced by the Mapper is the intermediate output in terms of key-value pairs which is massive in size.
- If we directly feed this huge output to the Reducer, then that will result in increasing the Network Congestion. So to minimize this Network congestion we have to put combiner in between Mapper and Reducer.
- These combiners are also known as semi-reducer. It is not necessary to add a combiner to your Map-Reduce program, it is optional.
- Combiner is also a class in our java program like Map and Reduce class that is used in between this Map and Reduce classes.
- Combiner helps us to produce abstract details or a summary of very large datasets. When we process or deal with very large datasets using Hadoop Combiner is very much necessary, resulting in the enhancement of overall performance.

2.5.1 How does combiner works

- In the above example, we can see that two Mappers are containing different data. the main text file is divided into two different Mappers. Each mapper is assigned to process a different line of our data. in our above example, we have two lines of data so we have two Mappers to handle each line.
- Mappers are producing the intermediate key-value pairs, where the name of the particular word is key and its count is its value. For example, for the data Geeks For Geeks For the key-value pairs are shown below.

// Key Value pairs generated for data

(Geeks,1)

(For,1)

(Geeks,1)

(For,1)

- The key-value pairs generated by the Mapper are known as the intermediate key-value pairs or intermediate output of the Mapper.
- Now we can minimize the number of these key-value pairs by introducing a combiner for each Mapper in our program. In our case, we have 4 key-value pairs generated by each of the Mapper. since these intermediate key-value pairs are not ready to directly feed to Reducer because that can increase Network congestion so Combiner will combine these intermediate key-value pairs before sending them to Reducer.
- The combiner combines these intermediate key-value pairs as per their key. For the above example for data for the combiner will partially reduce them by merging the same pairs according to their key value and generate new key-value pairs as shown below.

// Partially reduced key-value pairs with combiner

(Geeks,2)

(For,2)

- With the help of Combiner, the Mapper output got partially reduced in terms of size(key-value pairs) which now can be made available to the Reducer for better performance.
- Now the Reducer will again Reduce the output obtained from combiners and produces the final output that is stored on **HDFS(Hadoop Distributed File System)**.

2.5.2 Advantage of combiners

1. Reduces the time taken for transferring the data from Mapper to Reducer.
2. Reduces the size of the intermediate output generated by the Mapper.
3. Improves performance by minimizing Network congestion.

2.5.3 Disadvantage of combiners

- The intermediate key-value pairs generated by Mappers are stored on Local Disk and combiners will run later on to partially reduce the output which results in expensive Disk Input-Output.
- The map-Reduce job cannot depend on the function of the combiner because there is no such guarantee in its execution.

2.6 MATRIX VECTOR MULTIPLICATIONBY MAPREDUCE

UQ. Write pseudo code for Matrix vector Multiplication by MapReduce. Illustrate with an example showing all thesteps. **(MU - May 19, 10 Marks)**

GQ. What happen when vector does not fit in memory in matrix vector multiplication? **(5 Marks)**

UQ. Write MapReduce pseudocode to multiply to matrices. Illustrate the procedure on the following matrices.Clearly show all the steps. **(MU - May 18, 10 Marks)**

- MapReduce is a technique in which a huge program is subdivided into small tasks and run parallelly to make computation faster, save time, and mostly used in distributed systems. It has 2 important parts:
- **Mapper:** It takes raw data input and organizes into key, value pairs. For example, In a dictionary, you search for the word "Data" and its associated meaning is "facts and statistics collected together for reference or analysis". Here the Key is Data and the Value associated with is facts and statistics collected together for reference or analysis.
- **Reducer:** It is responsible for processing data in parallel and produce final output.

Algorithm 1 : The map function

1. **For** each element m_{ij} of M **do**
2. product (key value) pair as ((i, k), (M, j, m_{ij})), for k = 1, 2,3,...up to the number of columns of N
3. for each element n_{jk} of N do
4. product (key value) pair as ((i, k), (N, j, n_{jk})), for i = 1, 2,3,...up to the number of rows of M.
5. **return** set of (key value) pairs that each key, (i, k), has a list with values (M, j, m_{ij}) and (N, j, n_{jk}) for all possible values of j

Algorithm 2 : The reduce function

1. For each key (i, k) **do**
2. sort value begin with M by j in $list_M$
3. sort value begin with N by j in $list_N$
4. multiply m_{ij} and n_{jk} for j_{th} value of each list
5. sum up $m_{ij} * m_{jk}$
6. **return** (i, k), $\sum_{j=1} m_{ij} * n_{jk}$

Let us consider the matrix multiplication example to visualize MapReduce

Consider the following matrix

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \times \begin{bmatrix}5 & 6\\7 & 8\end{bmatrix}$$

Here matrix A is a 2×2 matrix which means the number of rows(i) = 2 and the number of columns(j)=2. Matrix B is also a 2×2 matrix where number of rows(j) = 2 and number of columns(k)=2. Each cell of the matrix is labelled as A_{ij} and B_{jk}. Ex. element 3 in matrix A is called A_{21} i.e. 2nd- row 1stcolumn. Now one step matrix multiplication has 1 mapper and 1 reducer. The Formula is:

Mapper for Matrix A (k, v) = ((i, k), (A, j, A_{ij})) for all k

Mapper for Matrix B (k, v) = ((i, k), (B, j, B_{jk})) for all i

Therefore, ecomputing the mapper for Matrix A:

k, i, j computes the number of times it occurs .

Here all are 2, therefore when k = l, i can have

2 values 1 & 2, each case can have 2 further

values of j = 1 and j = 2. Substituting all values

in formula

k = 1 i = 1 j = 1 ((1, 1), (A, 1, 1))

j = 2 ((1, 1), (A, 2, 2))

i = 2 j = 1 ((2, 1), (A, 1, 3))

j = 2 ((2, 1), (A, 2, 4))

k = 2 i = 1 j = 1 ((1, 2), (A, 1, 1))

j = 2 ((1, 2), (A, 2, 2))

i = 2 j = 1 ((2, 2), (A, 1, 3))

j = 2 ((2, 2), (A, 2, 4))

Matrix-Vector Multiplication by Map-Reduce

Computing the mapper for matrix B

i = 1 j = 1 k = 1 ((1, 1), (B, 1, 5))

k = 2 ((1, 2), (B, 1, 6))

j = 2 k = 1 ((1, 1), (B, 2, 7))

j = 2 ((1, 2), (B, 2, 8))

i = 2 j = 1 k = 1 ((2, 1), (B, 1, 5))

k = 2 ((2, 2), (B, 1, 6))

j = 2 k = 1 ((2, 1), (B, 2, 7))

k = 2 ((2, 2), (B, 2, 8))

- Matrix-Vector Multiplication by Map-Reduce

Reducer (k, v) = (i, k)=>Make sorted A_{list} and B_{list}

(i,k) => Summation (A_{ij} * B_{jk})) for J

Output =>((i k),sum)

Therefore, computing the reducer:

We can observe from Mapper computation

that 4 pairs are common (1, 1), (1, 2),

(2, 1) and (2, 2)

Make a list separate for Matrix A &

B with adjoining values taken from

Mapper step above :

Matrix-Vector Multiplication by Map-Reduce

(1, 1) =>A_{list}={(A, 1, 1), (A, 2, 2)}

B_{list}={(B, 1, 5), (B, 2, 7)}

Now $A_{ij} \times B_{jk}$: [(1 * 5) + (2*7)] = 19 ...(i)

(1, 2) =>A_{list}={(A, 1, 1), (A, 2, 2)}

B_{list}={(B, 1, 6), (B, 2, 8)}

Now $A_{ij} \times B_{jk}$: [(1*6) + (2*8)] :22 ...(ii)

(2, 1) =>A_{list}={(A, 1, 3), (A, 2, 4) }

B_{list}={(B, 1, 5), {B, 2, 7)}

Now $A_{ij} \times B_{jk}$: [(3* 5) + (4*7)] =43 ...(iii)

(2, 2) =>A_{list}={(A, 1, 3), (A, 2, 4)}

B_{list}={(B, 1, 6), (B, 2, 8)}

Now $A_{ij} \times B_{jk}$: [(3*6) + (4*8)] =50 ...(iv)

From (i), (ii), (iii) and (iv) we conclude that

((1, 1), 19)

((1, 2), 22)

((2, 1), 43)

((2, 2), 50)

Therefore, the final matrix is

$$\begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

2.7 RELATIONAL ALGEBRA OPERATIONS

1. Selection.	2. Projection.
3. Union & Intersection	4. Natural Join.
5. Grouping & Aggregation.	

1. Selection

- Apply a condition c to each taple in the relation and produce as output only those tuples that satisfy c.
- The result of this selection is denoted by 6c(R)6c(R)
- Selection really does not need the full power of MapReduce.
- They can be done most conveniently in the map portion alone, although they could also be done in the reduce portion also.
- The pseudo code is as follows :

```
Map (key, valve)
for tuple in valve :
if tuple satisfies C :
emit (tuple, tuple)
Reduce (key, valves)
emit (key, key)
```

2. Projection

- for some subset s of the attribute of the relation, produce from each tuple only the components for the attributes in S.
- The result of this projection is denoted TTs (R)
- Projection is performed similarly to selection.
- As projection may cause the same tuple to appear several times, the reduce function eliminate duplicates.

- The pseudo code for projection is as follows :

```
Map (key, valve)
for tuple in valve :
ts = tuple with only the components forthe attributes in S.
emit (ts, ts)
Reduce (key, values)
emit (key, key)
```

3. Union Using Map Reduce

- Both selection and projection are operations that are applied on a single table, whereas Union, intersection and difference are among the operations that are applied on two or more tables.
- Let's consider that schemas of the two tables are the same, and columns are also ordered in same order.
- **Map Function:** For each row r generate key-value pair (r, r) .
- **Reduce Function:** With each key there can be one or two values (As we don't have duplicate rows), in either case just output first value.

This operations has the **map function of the selection and reduce function** of **projection.** Let's see the working using an example. Here yellow colour represents one table and green colour is used to represent the other one stored at two map workers.

Map worker 1

Table 1	
A	B
1	2
2	3
5	6

Table 2	
A	B
2	3
4	4
6	1

Map worker 2

Table 1	
A	B
6	1
6	3
7	6

Table 2	
A	B
9	8
3	3
0	1

Initial data at map workers

After applying the map function and grouping the keys we will get output as:

Map worker 1

Key	Value
(1,2)	[(1,2)]
(2,3)	[(2,3), (2, 3)]
(5,6)	[(5, 6)]
(4, 4)	[(4, 4)]
(6, 1)	[(6, 1)]

Map worker 2

Key	Value
(6, 1)	[(6, 1)]
(6, 3)	[(6, 3)]
(7, 6)	[(7, 6)]
(9, 8)	[(9, 8)]
(3, 3)	[(3, 3)]
(0, 1)	[(0, 1)]

Map and grouping the keys

The data to be sent to reduce workers will look like:

Map worker 1

RW 1	
Key	value
(1,2)	[(1,2)]
(2,3)	[(2,3), (2, 3)]
(5,6)	[(5, 6)]

RW 2	
Key	value
(4, 4)	[(4, 4)]
(6, 1)	[(6, 1)]

Map worker 2

RW1	
Key	value
(6, 3)	[(6, 3)]
(3, 3)	[(3, 3)]
(0, 1)	[(0, 1)]

RW2	
Key	Value
(6, 1)	[(6, 1)]
(7, 8)	[(7, 8)]
(9, 8)	[(9, 8)]

Files to be sent to reduce workersData at reduce workers after will be:

Reduce worker 1

RW 1	
Key	value
(1,2)	[(1,2)]
(2,3)	[(2,3), (2, 3)]
(5,6)	[(5, 6)]

RW 1	
Key	value
(6, 3)	[(6, 3)]
(3, 3)	[(3, 3)]
(0, 1)	[(0, 1)]

Reduce worker 2

Key	Value
(4, 4)	[(4, 4)]
(6, 1)	[(6, 1)]

Key	Value
(6, 1)	[(6, 1)]
(7, 8)	[(7, 8)]
(9, 8)	[(9, 8)]

Files At reduce workers

At reduce workers aggregation on keys will be done.

Reduce worker 1

Key	Value
(1,2)	[(1,2)]
(2,3)	[(2,3), (2, 3)]
(5,6)	[(5, 6)]
(6, 3)	[(6, 3)]
(3, 3)	[(3, 3)]
(0, 1)	[(0, 1)]

Reduce worker 2

Key	Value
(4, 4)	[(4, 4)]
(6, 1)	[(6, 1), (6, 1)]
(7, 6)	[(7, 6)]
(9, 8)	[(9, 8)]

Aggregated data at reduce workers

The final output after applying the reduce function which takes only the first value and ignores everything else is as follows:

Reduce worker 1

A	B
1	2
2	3
5	6
6	3
3	3
0	1

Reduce worker 2

A	B
4	4
6	1
7	6
9	8

Final table after union

Here we note that in this case same as projection we can this done without moving data around in case we are not interested in removing duplicates. And hence this operation is also efficient it terms of data shuffle across machines.

Intersection Using Map Reduce

For intersection, let's consider the same data we considered for union and just change the map and reduce functions

- **Map Function:** For each row r generate key-value pair (r, r) (Same as union).
- **Reduce Function:** With each key there can be one or two values (As we don't have duplicate rows), in case we have length of list as 2 we output first value else we output nothing.

As the map function is same as union and we are considering the same data lets skip to the part before reduce function is applied.

Reduce worker 1

Key	Value
(1,2)	[(1,2)]
(2,3)	[(2,3), (2, 3)]
(5,6)	[(5, 6)]
(6, 3)	[(6, 3)]
(3, 3)	[(3, 3)]
(0, 1)	[(0, 1)]

Reduce worker 2

Key	Value
(4, 4)	[(4, 4)]
(6, 1)	[(6, 1), (6, 1)]
(7, 6)	[(7, 6)]
(9, 8)	[(9, 8)]

Data at reduce workers

Now we just apply the reduce operation which will output only rows if list has a length of 2.

Reduce worker 1

File 1	
A	B
2	3

Reduce worker 2

File 1	
A	B
6	1

Output of intersection

Difference Using Map Reduce

Let's again consider the same data. The difficulty with difference arises with the fact that we want to output a row only if it exists in the first table but not the second one. So the reduce function needs to keep track on which tuple belongs to which relation. To

visualize that easier, we will keep those rows green which come from 2nd table and yellow for which come from 1st table and **purple** which comes from both tables.

- **Map Function:** For each row r create a key-value pair (r, T1) if row is from table 1 else product key-value pair (r, T2).
- **Reduce Function:** Output the row if and only if the value in the list is T1 , otherwise output nothing.

The data taken initially is the same as it was for union

Map worker 1

Table 1	
A	B
1	2
2	3
5	6

Table 2	
A	B
2	3
4	4
6	1

Map worker 2

Table 1	
A	B
6	1
6	3
7	6

Table 2	
A	B
9	8
3	3
0	1

Initial Data

After applying the map function and grouping the keys the data looks like the following figure

Reduce worker 1

Key	Value
(1, 2)	[T1]
(2, 3)	[T1, T2]
(5, 6)	[T1]
(4, 4)	[T2]
(6, 1)	[T2]

Reduce worker 2

Key	Value
(6, 3)	[T1]
(7, 6)	[T1]
(9, 8)	[T2]
(6, 1)	[T1]
(3, 3)	[T2]
(0, 1)	[T2]

Data after applying map function and grouping keys

After applying map function files for reduce workers will be created based on hashing keys as has been the case so far.

Reduce worker 1

RW1	
Key	**Value**
(1, 2)	[T1]
(2, 3)	[T1, T2]
(5, 6)	[T1]

RW2	
Key	**Value**
(4, 4)	[T2]
(6, 1)	[T2]

Reduce worker 2

RW1	
Key	**Value**
(6, 3)	[T1]
(7, 6)	[T1]
(9, 8)	[T2]

RW2	
Key	**Value**
(6, 1)	[T1]
(3, 3)	[T2]
(0, 1)	[T2]

Files for reduce workers

The data at the reduce workers will look like

Reduce worker 1

RW1	
Key	**Value**
(1, 2)	[T1]
(2, 3)	[T1, T2]
(5, 6)	[T1]

RW1	
Key	**Value**
(6, 3)	[T1]
(7, 6)	[T1]
(9, 8)	[T2]

Reduce worker 2

RW1	
Key	**Value**
(4, 4)	[T2]
(6, 1)	[T2]

RW1	
Key	**Value**
(6, 1)	[T1]
(3, 3)	[T2]
(0, 1)	[T2]

Files at reduce workers

After aggregation of the keys at reduce workers the data looks like:

Reduce worker 1

Key	Value
(1, 2)	[T1]
(2, 3)	[T1, T2]
(5, 6)	[T1]
(6, 3)	[T1]
(9, 8)	[T2]

Key	Value
(4, 4)	[T2]
(6, 1)	[T1, T2]
(3, 3)	[T2]
(0, 1)	[T2]
(7, 6)	[T1]

Data after aggregation of keys at reduce workers

The final output is generated after applying the reduce function over the output.

File 1	
A	B
1	2
5	6
6	3

File 1	
A	B
7	6

Output of difference of the tables

For the difference operation we notice that we cannot get rid of the reduce part and hence have to send data across the workers as the context of from which table the value came is needed. Hence it will be **more expensive** operation as compared to selection, projection, union and intersection.

Grouping and Aggregation Using Map Reduce

Usually understanding grouping and aggregation takes a bit of time when we learn SQL, but not in case when we understand these operations using map reduce. The logic is already there in the working of the map. Map workers implicitly group keys and the reduce function acts upon the aggregated values to generate output.

- **Map Function:** For each row in the table, take the attributes using which grouping is to be done as the key, and value will be the ones on which aggregation is to be performed. For example, If a relation has 4 columns A, B, C, D and we want to group by A, B and do an aggregation on C we will make (A, B) as the key and C as the value.
- **Reduce Function:** Apply the aggregation operation (sum, max, min, avg, ...) on the list of values and output the result.

For our example lets group by (A, B) and apply sum as the aggregation.

Map worker1							
File 1				File 2			
A	B	C	D	A	B	C	D
1	2	3	1	4	2	1	3
2	2	3	2	6	8	4	4
1	2	1	3	3	2	2	4

Map worker 2							
File 1				File 2			
A	B	C	D	A	B	C	D
1	2	5	2	3	2	5	2
2	3	2	4	2	3	9	2
1	3	1	3	3	4	2	1

Initial data at the map workers

The data after application of map function and grouping keys will creates (A, B) as key and C as value and D is discarded as if it doesn't exist.

Reduce worker 1	
Key	Value
(1, 2)	[3, 1]
(2, 2)	[3]
(4, 2)	[1]
(6, 8)	[4]
(3, 2)	[2]

Reduce worker 2	
Key	Value
(1, 2)	[5]
(2, 3)	[2, 9]
(1, 3)	[1]
(3, 2)	[1]
(3, 4)	[2]

Data at map workers

Applying partitioning using hash functions, we get

Files for the reduce workers

Map worker 1			
RW1		RW2	
Key	Value	Key	Value
(1, 2)	[3, 1]	(6, 8)	[4]
(2, 2)	[3]	(3, 2)	[2]
(4, 2)	[1]		

Map worker 2			
RW1		RW2	
Key	Value	Key	Value
(1, 2)	[5]	(3, 2)	[1]
(2, 3)	[2, 9]	(3, 4)	[2]
		(1, 3)	[1]

Files for reduce workers

The data at the reduce workers will look like

Reduce worker 1			
RW 1		RW1	
Key	Value	Key	Value
(1, 2)	(3, 1)	(1, 2)	[5]
(2, 2)	[3]	(2, 3)	[2, 9]
(4, 2)	[1]		

Reduce worker 1			
RW 2		RW 2	
Key	Value	Key	Value
(6, 8)	[4]	(3, 2)	[1]
(3, 2)	[2]	(3, 4)	[2]
		(1, 3)	[1]

Data at reduce workers

The data is aggregated based on keys before applying the aggregation function (sum in this case).

Reduce worker 1	
Key	Value
(1, 2)	[3, 1, 5]
(2, 2)	[3]
(4, 2)	[1]
(2, 3)	[(2, 9)]

Reduce worker 2	
Key	Value
(6, 8)	[4]
(3, 2)	[1, 2]
(3, 4)	[2]
(1, 3)	[1]

Aggregated data based on keys

After applying the sum over the value lists we get the final output

Reduce worker 1		
A	B	Sum
1	2	9
2	2	3
4	2	1
2	3	11

Reduce worker 2		
A	B	Sum
6	8	4
3	2	3
3	4	2
1	3	1

Output of group by (A, B) sum(C)

Here also like difference operation we can't get rid of the reduce stage. The context of tables isn't wanted here but the aggregation function makes it necessary for the values to be in one place for a single key. This operation is also inefficient as compared to selection, projection, union, and intersection. The column that is not in aggregation or grouping clause is ignored and isn't required. So, if the data be stored in a columnar format, we can save cost of loading a lot of data. Usually there are only a few columns involved in grouping and aggregation it does save up a lot of cost both in terms of data that is sent over the network and the data that needs to be loaded to main memory for execution.

2.8 NATURAL JOIN USING MAP REDUCE

The natural join will keep the rows that matches the values in the common column for both tables. To perform natural join, we will have to keep track of from which table the value came from. If the values for the same key are from different tables we need to form pairs of those values along with key to get a single row of the output. Join can explode the number of rows as we have to form each and every possible combination of the values for both tables.

- **Map Function:** For two relations Table 1(A, B) and Table 2(B, C) the map function will create key-value pairs of form b: [(T1, a)] for table 1 where T1 represents the fact that the value a came from table 1, for table 2 key-value pairs will be of the form b: [(T2, c)].
- **Reduce Function:** For a given key b construct all possible combinations for the values where one value is from table T1 and the other value is from table T2. The output will consist of key-value pairs of form b: [(a, c)] which represent one row a, b, c for the output table.

For an example let's consider joining Table 1 and Table 2, where B is the common column.

Map worker 1			
Table 1		Table 2	
A	B	B	C
1	2	2	3
2	3	4	4
5	6	6	1

Map worker 2			
Table 1		Table 2	
A	B	B	C
6	1	9	8
6	3	3	4
7	6	2	1

Initial data at map workers

The data after applying the map function and grouping at the map workers will look like:

Map worker 1	
Key	Value
2	[T1, 1), (T2, 3)]
3	[T(1, 2)]
6	[(T1, 5), (T2, 1)]
4	[(T1, 4)]

Map worker 2	
Key	Value
1	[T1, 6)
3	[T1, 6), (T2, 4)]
6	[(T1, 7)
9	[(T2, 8)]
2	[(T2, 1)]

Data at map workers after applying map function and grouping the keys

As has been the case so far files for reduce workers will be created at the map workers

Map worker 1			
RW 1		RW 2	
Key	Value	Key	Value
2	[(T1, 1), (T2, 3), (T2, 1)]	6	[(T1, 5), (T2, 1)]
3	[(T1, 2)]	4	[(T1, 4)]

Map worker 2			
RW 1		RW 2	
Key	Value	Key	Value
1	[(T1, 6)	6	[(T1, 7)]
3	[(T1, 6), (T2, 4)]	9	[(T2, 8)]

Files constructed for reduce workers

The data at the reduce workers will be:

Reduce worker 1				Reduce worker 1			
RW1		RW1		RW2		RW2	
Key	Value	Key	Value	Key	Value	Key	Value
2	[(T1, 1), (T2, 3), (T2, 1)]	1	[(T1, 6)]	6	[(T1, 5), (T2, 1)]	6	[(T1, 7)]
3	[(T1, 2)]	3	[(T1, 6), (T2, 4)]	4	[(T1, 4)]	9	[(T2, 8)]

Data at reduce workers

Applying aggregation of keys at the reduce workers we get:

Reduce workers 1		Reduce workers 2	
Key	value	Key	value
2	[(T1, 1), (T2, 3), (T2, 1)]	6	[(T1, 5), (T1, 7), (T2, 1)]
3	[(T1, 2), (T1, 6), (T2, 4)]	4	[(T1, 4)
1	[(T1, 6)]	9	[(T2, 8)]

Data after aggregation of keys at the reduce workers

After applying the reduce function which will create a row by taking one value from table T1 and other one from T2. If there are only values from T1 or T2 in the values list that won't constitute a row in output.

Reduce worker 1		
B	A	C
2	1	3
2	1	1
3	2	4
3	6	4

Reduce worker 1		
B	A	C
6	5	1
6	7	1

Output of the join

- As we need to keep context from which table a value came from, we can't get rid of the data that needs to be sent across the workers for application of reduce task, this operation also becomes costly as compared to others we discussed so far.
- The fact that for each list of values we need to create pairs also plays a major factor in the computation cost associated with this operation.

2.9 MAPREDUCE – UNDERSTANDING WITH REAL-LIFE EXAMPLE

MapReduce is a programming model used to perform distributed processing in parallel in a Hadoop cluster, which Makes Hadoop working so fast. When you are dealing with Big Data, serial processing is no more of any use. MapReduce has mainly two tasks which are divided phase-wise:

- Map Task
- Reduce Task

Let us understand it with a real-time example, and the example helps you understand Mapreduce Programming Model in a story manner:

- Suppose the Indian government has assigned you the task to count the population of India. You can demand all the resources you want, but you have to do this task in 4 months. Calculating the population of such a large country is not an easy task for a single person(you). So what will be your approach?.
- One of the ways to solve this problem is to divide the country by states and assign individual in-charge to each state to count the population of that state.
- Task Of Each Individual: Each Individual has to visit every home present in the state and need to keep a record of each house members as:

  State_Name Member_House1

  State_Name Member_House2

  State_Name Member_House3

  State_NameMember_House n

  .For Simplicity, we have taken only three states.

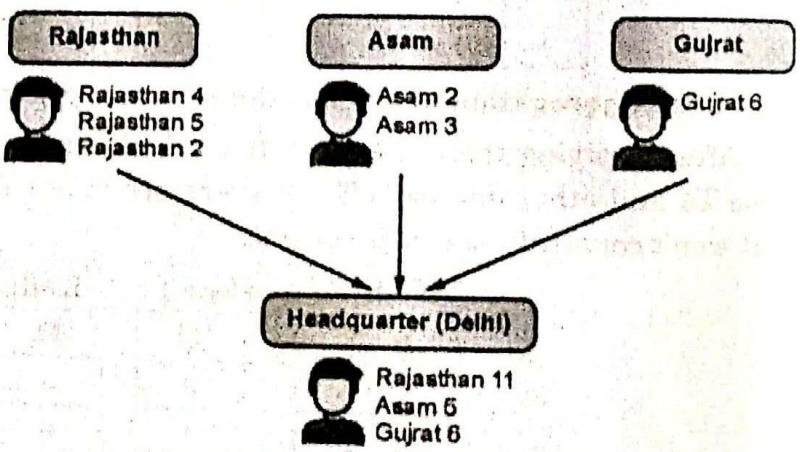

Fig. 2.9.1

- This is a simple Divide and Conquer approach and will be followed by each individualto count people in his/her state.
- Once they have counted each house member in their respective state. Now they need to sum up their results and need to send it to the Head-quarter at New Delhi.
- We have a trained officer at the Head-quarter to receive all the results from each state and aggregate them by each state to get the population of that entire state. and

Now, with this approach, you are easily able to count the population of India by summing up the results obtained at Head-quarter.

- The Indian Govt. is happy with your work and the next year they asked you to do the same job in 2 months instead of 4 months. Again you will be provided with all the resources you want.
- Since the Govt. has provided you with all the resources, you will simply double the number of assigned individual in-charge for each state from one to two. For that divide each state in 2 division and assigned different in-charge for these two divisions as:
  - State_Name_Incharge_division1
  - State_Name_Incharge_division2
- Similarly, each individual in charge of its division will gather the information about members from each house and keep its record.
- We can also do the same thing at the Head-quarters, so let's also divide the Head-quarter in two division as:
  - Head-qurter_Division1
  - Head-qurter_Division2

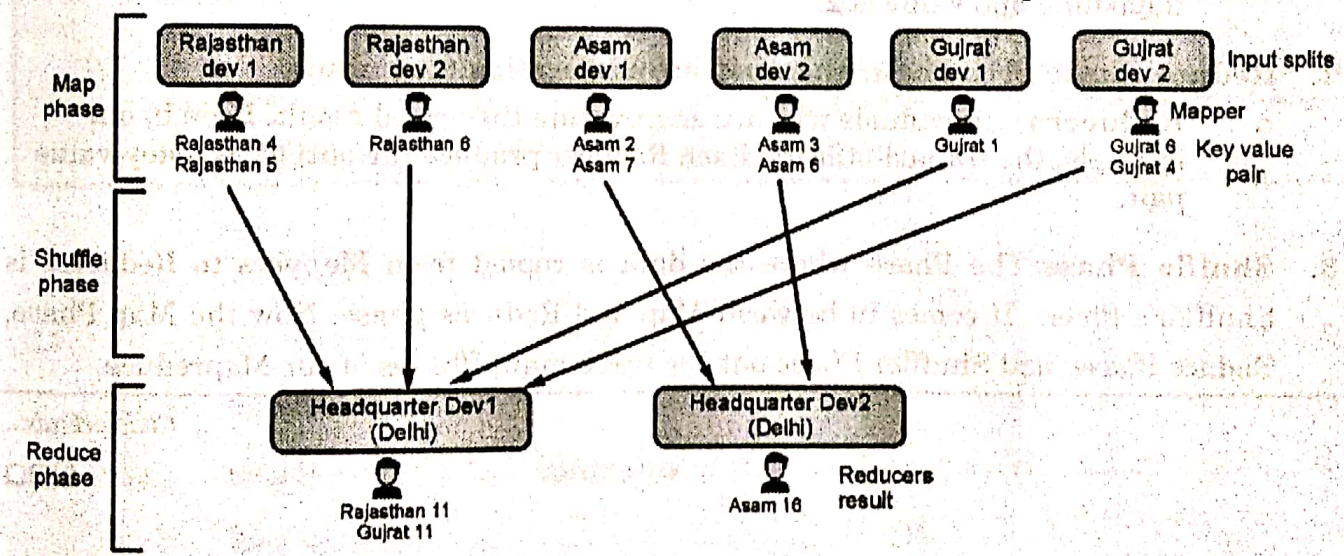

Fig. 2.9.2

- Now with this approach, you can find the population of India in two months. But there is a small problem with this, we never want the divisions of the same state to send their result at different Head-quarters then, in that case, we have the partial population of that state in Head-quarter_Division1 and Head-quarter_Division2 which is inconsistent because we want consolidated population by the state, not the partial counting.
- One easy way to solve is that we can instruct all individuals of a state to either send

there result to Head-quarter_Division1 or Head-quarter_Division2. Similarly, for all the states.

- Our problem has been solved, and you successfully did it in two months.
- Now, if they ask you to do this process in a month, you know how to approach the solution.
- Great, now we have a good scalable model that works so well. The model we have seen in this example is like the MapReduce Programming model. so now you must be aware that MapReduce is a programming model, not a programming language.

Now let's discuss the phases and important things involved in our model.

1. **Map Phase:** The Phase where the individual in-charges are collecting the population of each house in their division is Map Phase.
   - **Mapper :**Involved individual in-charge for calculating population.
   - **Input Splits :** The state or the division of the state.
   - **Key-Value Pair :** Output from each individual Mapper like the key is Rajasthan and value is 2.
2. **Reduce Phase :** The Phase where you are aggregating your result.
   - **Reducers :** Individuals who are aggregating the actual result. Here in our example, the trained-officers. Each Reducer produce the output as a key-value pair.
3. **Shuffle Phase:** The Phase where the data is copied from Mappers to Reducers is Shuffler's Phase. It comes in between Map and Reduces phase. Now the Map Phase, Reduce Phase, and Shuffler Phase out the three main Phases of our Mapreduce.

ChapterEnds...

□□□