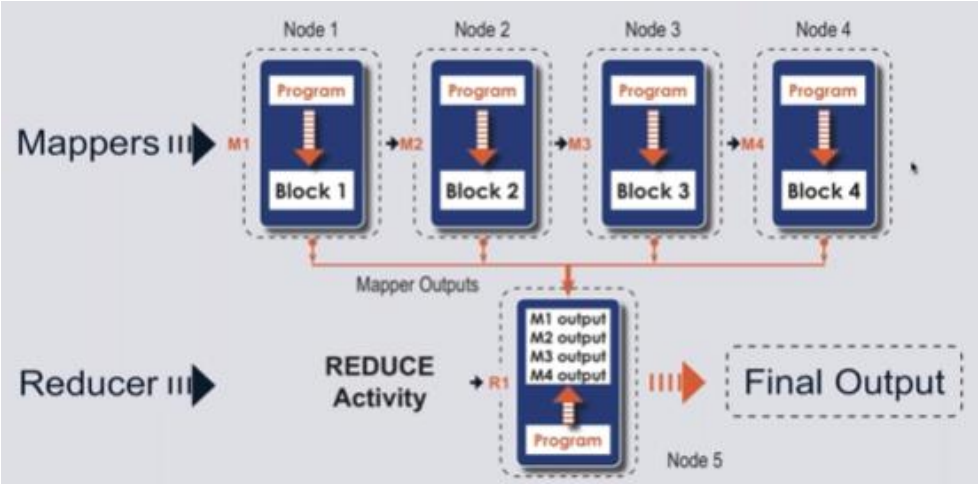
MAP REDUCE

Map Reduce -- Processing of data in a distributed manner.
It is a computing **paradigm** for processing data that resides on many computers.

2 Phases -

- ✓ Map - a piece of code running on block parallelly
- ✓ Reduce - a program required for aggregation to get the final output -- REDUCE ACTIVITY
  - node 5 can be different machine or among 4 original machines
  - a reducer doesn't give much parallelism because only one node is working, we can inc it also.



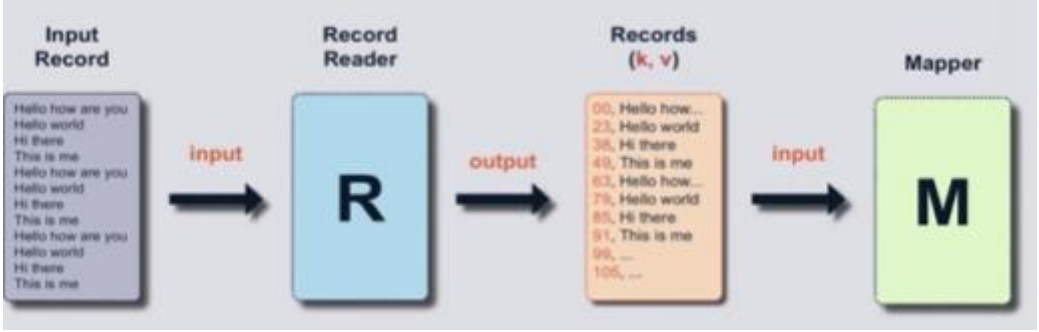
= Both these Map & Reduce work only on Key-Value pairs.
(K,V) -> MAP -> (K,V) -> (K,V) -> Reduce -> (K,V)

= Map Reduce is a programming paradigm.
= Traditional programming models work when data is kept on a single machine.
= and when we have data kept on multiple machines we require new programming model to solve a problem

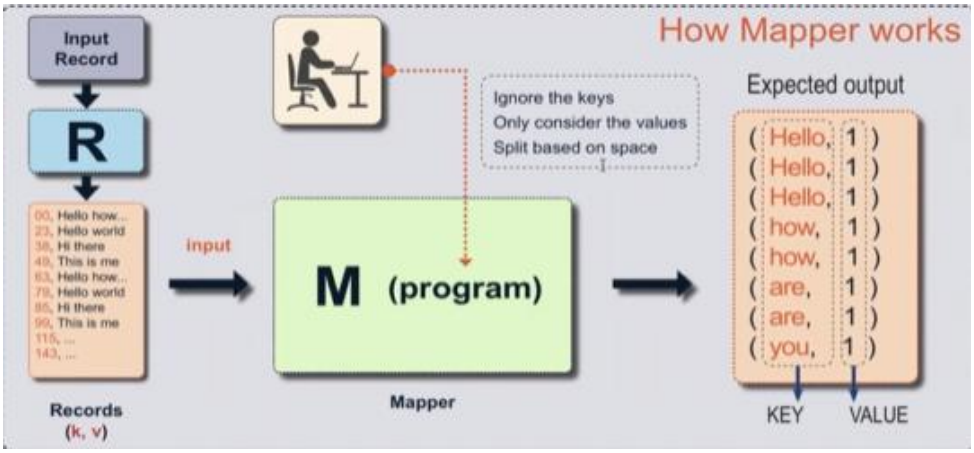
= Hadoop works on **Principle of Data Locality**.
- That the data is processed where it is kept. That is my code is going to the data and data is not coming to the code.

Record Reader - the role of RR is to covert each input line into (key, value) pair suitable for reading by the mapper. We don't write code for RR
Que Ex.

HELLO HOW ARE YOU HELLO. HELLO WORLD HI THERE	(0, HELLO HOW ARE YOU HELLO) (1, HELLO WORLD) (2, HI THERE)



= the logic/program for mapper has to be written by the developer.
= keys are of no relevance for us, we only consider the values and split based on space b/w words
= each of 4 machines is parallelly working trying to do same steps giving these intermediate outputs



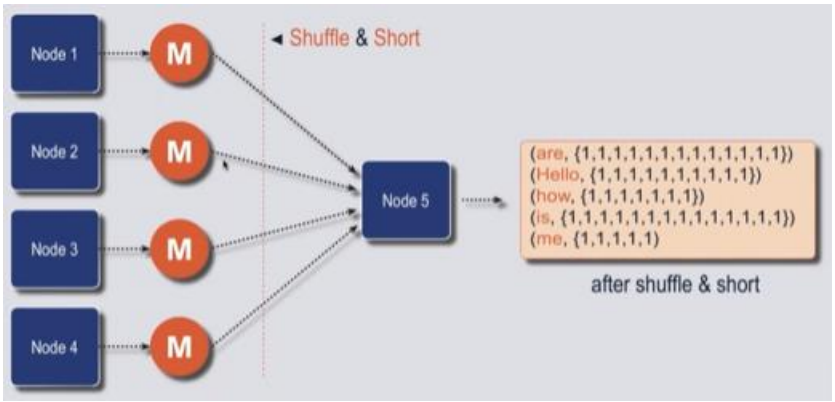
= output of mapper should be

DataNode1 (hello, 1) (how, 1) (are, 1) (you, 1) (hello, 1)
(hello, 1) (world, 1)
(hi, 1), (there, 1)
DataNode2 (from, 1) (is, 1)
DataNode3 (hello, 1) (hi, 1)
DataNode4 (where, 1) (situation, 1)

Shuffling - movement of data from mapper machine to reducer machine.
Sorting - will happen on reducer machine that is Node5 to get same keys together in a group

Shuffle & Sort is taken care by the framework.

Reducer logic should be - we iterate over the list of values and sum it up



Final Output -
(hello, 13)
(how, 10)
(are, 50)
(you, 5)

Map Reduce in Depth

- Map Reduce is used to process huge amount of data that needs to be across the cluster.
- MR, practically is not imp nowadays, but theory is super imp.

= MR gives us parallelism.
= Reduce phase gives us aggregation.

= Try doing more work at mapper end and try doing minimum work at reducer end to get more use of distributed system.
- as a good developer, try to shift your computation from reducer to mapper..

Example: **Linked In profile views**
How to calculate number of views for each profile on Linked In

St. I want to check how many people viewed sumit's profile

ID, From-member, To-Member
1, Manasa, Sumit
2, Deepa, Sumit
3, Sumit, Manasa
4, Mansa, Deepa
5, Deepa, Mansa
6, Shilpy, Mansa

== output of **record reader to mapper**.
(0, [1, Manasa, Sumit])
(1, [2, Deepa, Sumit])
(2, [3, Sumit, Manasa])

== the mapper output will be

MAPPER	Shuffling		Sorting	**I/P TO REDUCER**		**FINAL O/P**
(Sumit, 1)		(Deepa, 1)		(Deepa, {1})		(Deepa, 1)
(Sumit, 1)		(Manasa, 1)		(Manasa, {1, 1, 1})	---->	(Manasa, 3)
(Manasa, 1)	---->	(Manasa, 1)	------>	(Sumit, {1, 1})		(Sumit, 2)
(Deepa, 1)		(Manasa, 1)				
(Manasa, 1)		(Sumit, 1)				
(Manasa, 1)		(Sumit, 1)				

Que. How many mappers are launched ?
Ans. **No. Of mappers = No. Of Blocks ---> but in parallelism no of mappers depends on no. Of nodes in a cluster.**
Ex. 1Gb file -> 128mb ->8 blocks ->8 mappers
But if there is a 3 node cluster, so 8 mappers might not be running parallelly.

Que. How many reducers are launched ?
Ans. By default we have only 1 reducer.. but as a developer we can change the number of reducers.
-- we can make it 0 or we can make it more than 1.

Changing the number of Reducers
==> When should we consider to increase the number of reducers.
5 mappers completed in 2 mins… and the 1 reducer is taking 10 mins…. total 12 mins.
----->Shifting work to mappers----->
5 mappers completed in 3 mins… and the 1 reducer is taking 5 mins…. total 8 mins
-----> Increasing no. Of reducers---->
5 mappers and 2 reducers --> 5 mappers in 3 mins + 2 reducers in 2.5 mins = 5.5 mins

==> when to make the number of reducers to 0.
- there can be few jobs which do not require aggregation, shuffle, sort only filter is required..
- shuffle and sort only happens when we have 1 or more reducers.
- when number of reducer is 0 then your mapper output will be the final output.
Ex. 1 Gb file -> 128Mb -> 8 blocks -> 8 mappers -> 8 output files.

== Concept of **partition** comes into play when we have more than 1 reducer.
This is to tell which key value pair goes to which reducer.
By default there is a **Hash Function**.
- this hash function is **consistent** that means same key should go to same reducer.
(hello, 1) - R1
(hi, 1) - R2
(hello, 1) - R1
We can change or customize with our own custom partitioning logic
== **MAP -->PARTITION, SHUFFLE, SORT -->REDUCER**

Combiner = Local aggregation at the mapper end.

- ✓ Improve the parallelism as more is happening at the mapper side.
- ✓ Reduce the data transfer, as output of mappers becomes less in number after aggregation.
- ✓ Conclusion - combiner plays a very good role at the optimization.

= Use combiner with caution because -->
-- Max, Min, Sum ==> using combiner or without using combiner the result should be same.
-- Average ==> little tricky but need to optimize carefully such that end o/p doesn't change.

MR Code Explanation

3 codes : Map, Reduce and Main in Java Lang.

= What is Context?
- Context is nothing but the place where we need to put the output so that it is considered for further map reduce activities.

= value is of text type.
- text is understood by hadoop. But not by java.
So we need to convert this text type to string type so that our java program can understand it.

Map reduce Practical - JAR (Java Archive) creation
Running MR job on HDFS file

Basically If the input file is present in hdfs instead of local.
- To create a JAR file.
- We will be bounding all our code in jar file and will then run the jar on a input file which reside in hdfs.
- Local location - /home/cloudera/Desktop/inputfolder/inputfile.txt

we want to move the file from local to hdfs.
- create a new dir in hdfs
- hadoop fs -mkdir /user/cloudera/inputfolder
- hadoop fs -put Desktop/inputfolder/inputfile.txt /user/cloudera/inputfolder
- next step is to create a jar file.

Step 1: we moved a input file in hdfs.
Step 2: we have created a jar file.
Step 3: we want to run the jar file on the file that resides in hdfs.

- hadoop jar <jarname> <inputfilepath> <outputfilepath>
- hadoop fs -ls /user/cloudera/outputfolder

= If you want to see all past jobs also.
- localhost:8088

= If we want to browse your hdfs file system from UI.
- localhost:50070 ->utilities/browse the file system

Program Variation

- ➢ **Change number of reducers -**
  Inside Main file, add following property --> job.setNumReduceTasks(2)
  By default system defined hash partitioning function will come into play for dividing the keys acroos the reducers.
  The hash function is consistent. Similar keys go to the same reducer.

- **Customizing the partitioning logic** --> write your own custom partitioner class.
  We need to mention below configuration in main
  Job.setNumReducetasks(2)
  Job.setPartitioerClass(CustomPartitioner.class)

-- logic -- if key length is less than 4 then it should go to reducer ID 0 else it should go to reducer ID 1.
-- In this case we need to write our own custom partitioner.
-- mappers output is input to partitioner.

- **Changing the number of reducers to 0 -->** inside main.java
  Job.setNumReduceTasks(0);
  Only mappers will work, no reducer.
  Number of output files depends on the number of mappers. If your file is 500mb then there will be 4 output files.

- **Introducing a combiner -->** local aggregation on mapper machines.
  You can have your reducer code as your combiner code as well.
  In Main.java add this property ---> job.setCombinerClass(Reduce.class);

-- output of mapper is the input to combiner. --> (Text, IntWritable)
-- output of combiner is the input to reducer. --> (Text, IntWritable)

- **Introducing a custom combiner -->**
  In main class, add this property --> job.setCombinerClass(CustomCombiner.java);