Module -4 : UNDERSTANDING MAP REDUCE FUNDAMENTALS

MapReduce

1. Traditional Enterprise Systems normally have a centralized server to store and process data.
2. The following illustration depicts a schematic view of a traditional enterprise system. Traditionalmodel is certainly not suitable to process huge volumes of scalable data and cannot be accommodated by standard database servers.
3. Moreover, the centralized system creates too much of a bottleneck while processing multiple files simultaneously.

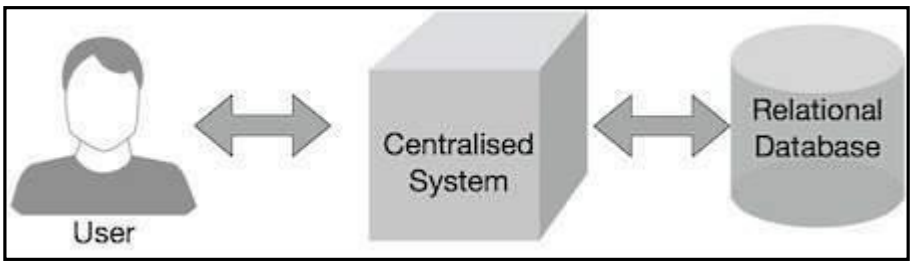

Figure 4.1: MapReduce

4. Google solved this bottleneck issue using an algorithm called MapReduce. MapReduce divides atask into small parts and assigns them to many computers.
5. Later, the results are collected at one place and integrated to form the result dataset.

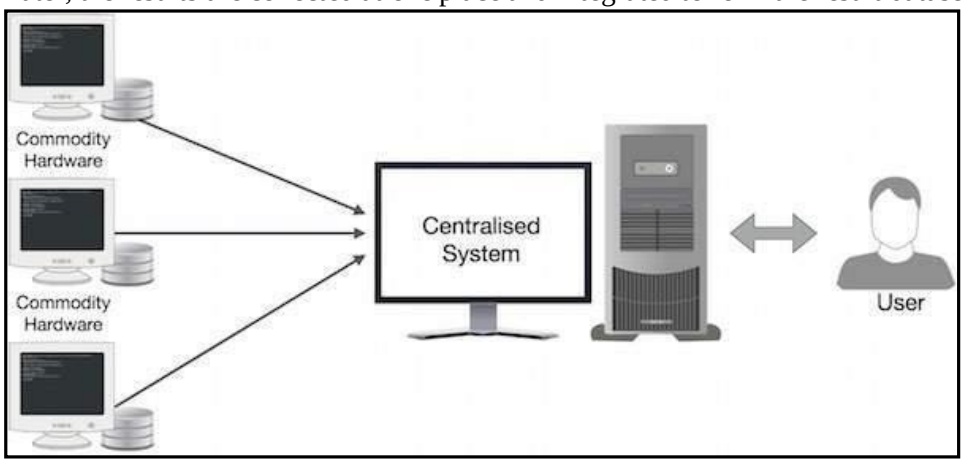

Figure 4.2: Physical structure

6. A MapReduce computation executes as follows:

- Some number of Map tasks each are given one or more chunks from a distributed file system. TheseMap tasks turn the chunk into a sequence of key-value pairs. The way key-value pairs are producedfrom the input data is determined by the code written by the user for the Map function.

The key-value pairs from each Map task are collected by a master controller and sorted by key. Thekeys are divided among all the Reduce tasks, so all key-value pairs with the same key wind up at thesame Reduce task

- The Reduce tasks work on one key at a time, and combine all the values associated with that key insome way. The manner of combination of values is determined by the code written by the user for the Reduce function.

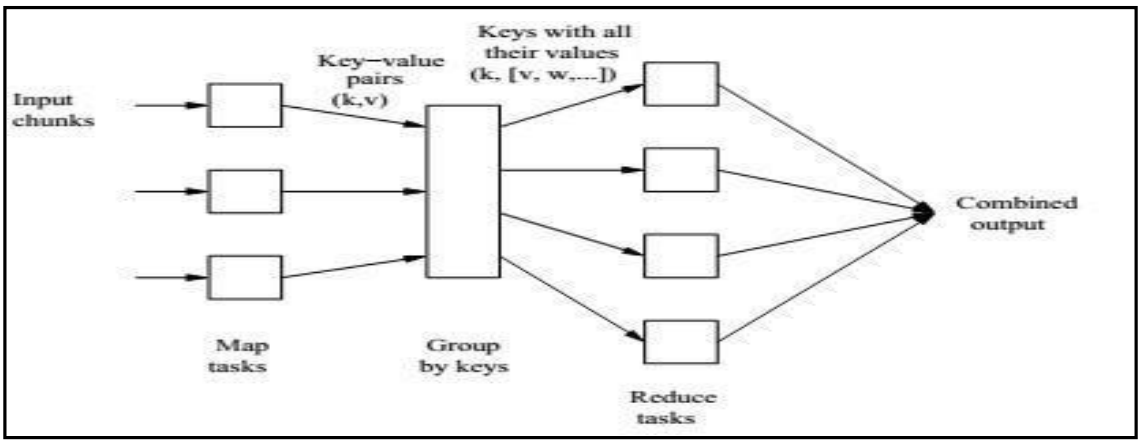

Figure 4.3: Schematic MapReduce Computation

A. The Map Task

i. We view input files for a Map task as consisting of elements, which can be any type: a tuple or a document, for example.

ii. A chunk is a collection of elements, and no element is stored across two chunks.

iii. Technically, all inputs to Map tasks and outputs from Reduce tasks are of the key-value-pair form,but normally the keys of input elements are not relevant and we shall tend to ignore them.

iv. Insisting on this form for inputs and outputs is motivated by the desire to allow composition of several MapReduce processes.

v. The Map function takes an input element as its argument and produces zero or more key-valuepairs.

vi. The types of keys and values are each arbitrary. vii. Further, keys are not "keys" in the usualsense; they do not have to be unique.

vii. Rather a Map task can produce several key-value pairs with the same key, even from the same element.

Example 1: A MapReduce computation with what has become the standard example application: counting the number of occurrences for each word in a collection of documents. In this example, theinput file is a repository of documents, and each document is an element. The Map function for this example uses keys that are of type String (the words) and values that are integers. The Map task readsa document and breaks it into its sequence of words $w_1, w_2, \ldots, w_n$. It then emits a sequence of key-value pairs where the value is always 1. That is, the output of the Map task for this document is the sequence of key-value pairs: $(w_1, 1), (w_2, 1), \ldots, (w_n, 1)$

A single Map task will typically process many documents – all the documents in one or more chunks.Thus, its output will be more than the sequence for the one document suggested above. If a word *w* appears *m* times among all the documents assigned to that process, then there will be m key-value pairs (w, 1) among its output. An option, is to combine these m pairs into a single pair (w, m), but

we can only do that because, the Reduce tasks apply an associative and commutative operation, addition, to the values.

B. Grouping by Key

i. As the Map tasks have all completed successfully, the key-value pairs are grouped by key, and thevalues associated with each key are formed into a list of values.

ii. The grouping is performed by the system, regardless of what the Map and Reduce tasks do.

iii. The master controller process knows how many Reduce tasks there will be, say r such tasks.

iv. The user typically tells the MapReduce system what r should be.

v. Then the master controller picks a hash function that applies to keys and produces a bucket number from 0 to r − 1.

vi. Each key that is output by a Map task is hashed and its key-value pair is put in one of r local files. Each file is destined for one of the Reduce tasks.1.

vii. To perform the grouping by key and distribution to the Reduce tasks, the master controller merges the files from each Map task that are destined for a particular Reduce task and feeds the merged file to that process as a sequence of key-list-of-value pairs.

viii. That is, for each key k, the input to the Reduce task that handles key k is a pair of the form (k, [v1, v2, . . . , vn]), where (k, v1), (k, v2), . . . , (k, vn) are all the key-value pairs with keyk coming from all the Map tasks.

C. The Reduce Task

i. The Reduce function's argument is a pair consisting of a key and its list of associated values.

ii. The output of the Reduce function is a sequence of zero or more key-value pairs.

iii. These key-value pairs can be of a type different from those sent from Map tasks to Reduce tasks, but often they are the same type.

iv. We shall refer to the application of the Reduce function to a single key and its associated list ofvalues as a reducer. A Reduce task receives one or more keys and their associated value lists.

v. That is, a Reduce task executes one or more reducers. The outputs from all the Reduce tasks are merged into a single file.

vi. Reducers may be partitioned among a smaller number of Reduce tasks is by hashing the keys and associating each

vii. Reduce task with one of the buckets of the hash function.

The Reduce function simply adds up all the values. The output of a reducer consists of the word and the sum. Thus, the output of all the Reduce tasks is a sequence of (w, m) pairs, where w is a word that appears at least once among all the input documents and m is the total number of occurrences ofw among all those documents.

D. Combiners

i. A Reduce function is associative and commutative. That is, the values to be combined can be combined in any order, with the same result.

ii. The addition performed in Example 1 is an example of an associative and commutative operation. It doesn't matter how we group a list of numbers $v_1, v_2, \ldots, v_n$; the sum will be the same. iii.When the Reduce function is associative and commutative, we can push some of what the reducers do to the Map tasks

iv. These key-value pairs would thus be replaced by one pair with key w and value equal to the sumof all the 1's in all those pairs.

v. That is, the pairs with key w generated by a single Map task would be replaced by a pair (w, m), where m is the number of times that w appears among the documents handled by this Map task.

E. Details of MapReduce task

The MapReduce algorithm contains two important tasks, namely Map and Reduce.

i. The Map task takes a set of data and converts it into another set of data, where individual elementsare broken down into tuples (key-value pairs).

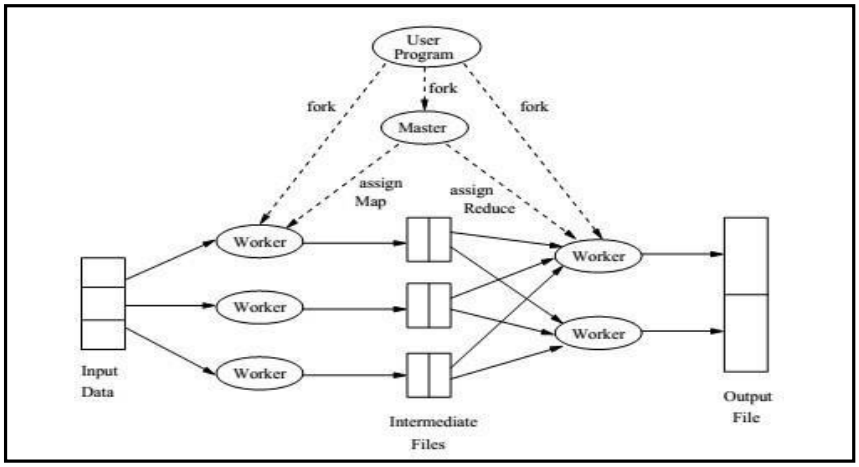

Figure 4.4: Overview of the execution of a MapReduce program

ii. The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.

iii. The reduce task is always performed after the map job.

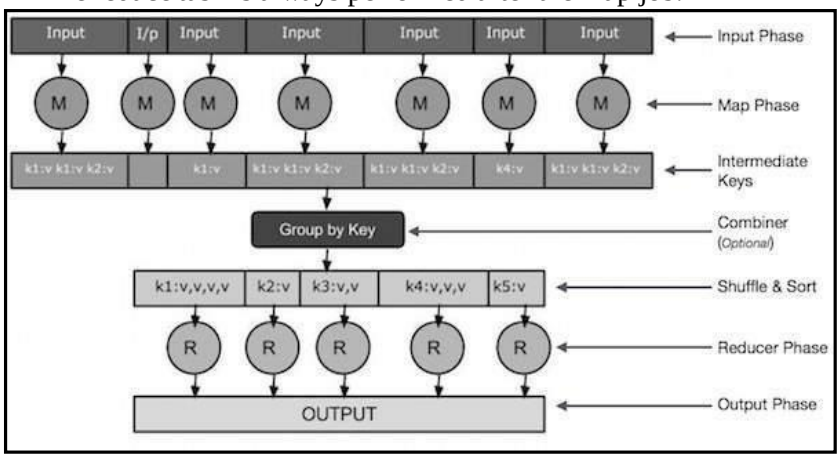

Figure 4.5: Reduce job

- **Input Phase** − Here we have a Record Reader that translates each record in an input file and sendsthe parsed data to the mapper in the form of key-value pairs.
- **Map** − Map is a user-defined function, which takes a series of key-value pairs and processes eachone of them to generate zero or more key-value pairs.
- **Intermediate Keys** − they key-value pairs generated by the mapper are known as intermediate keys.
- **Combiner** − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.
- **Shuffle and Sort** − The Reducer task starts with the Shuffle and Sort step. It downloads the groupedkey-value pairs onto the local machine, where the Reducer is running. The individual key-value pairsare sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.
- **Reducer** − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key- value pairs to the final step.

➢ **Output Phase** − In the output phase, we have an output formatter that translates the final key-valuepairs from the Reducer function and writes them onto a file using a record writer.

iv. The MapReduce phase

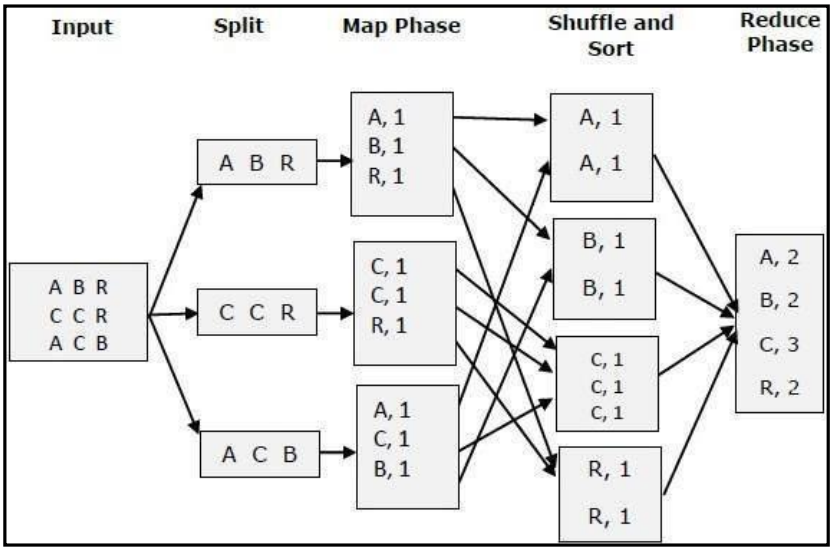

Figure 46 The MapReduce Phase

F. MapReduce-Example

Twitter receives around 500 million tweets per day, which is nearly 3000 tweets per second. The following illustration shows how Tweeter manages its tweets with the help of MapReduce.

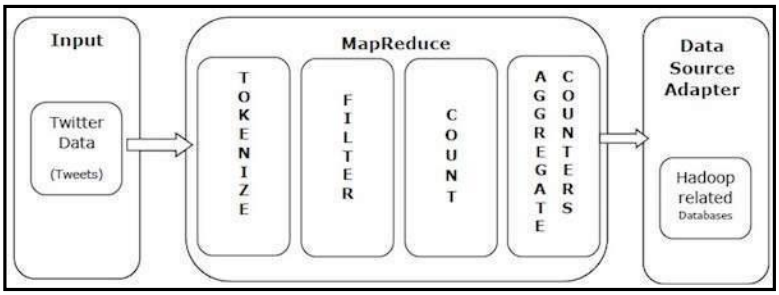

Figure4.7: Example

i. **Tokenize** − Tokenizes the tweets into maps of tokens and writes them as key-value pairs.

ii. **Filter** − Filters unwanted words from the maps of tokens and writes the filtered maps as key-value pairs.

iii. **Count** − Generates a token counter per word.

iv. **Aggregate Counters** − Prepares an aggregate of similar counter values into small manageableunits.

G. MapReduce – Algorithm

The MapReduce algorithm contains two important tasks, namely Map and Reduce.

i. The map task is done by means of Mapper Class

 Mapper class takes the input, tokenizes it, maps and sorts it. The output of Mapper class is used asinput by Reducer class, which in turn searches matching pairs and reduces them.

ii. The reduce task is done by means of Reducer Class.

 MapReduce implements various mathematical algorithms to divide a task into small parts and assign

them to multiple systems. In technical terms, MapReduce algorithm helps in sending the Map & Reduce tasks to appropriate servers in a cluster.

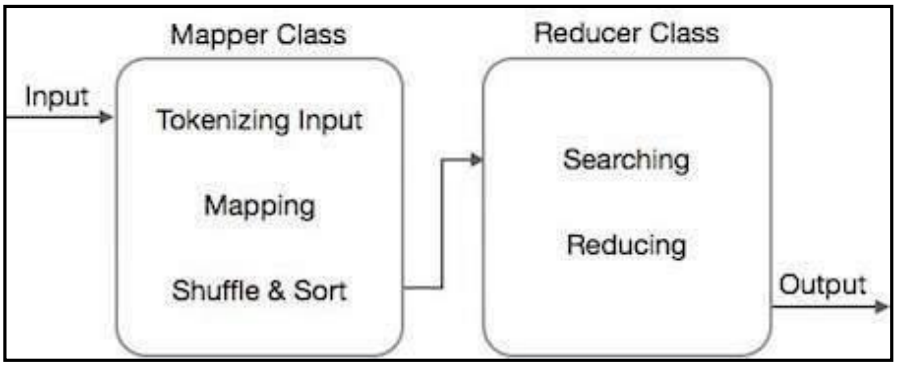

Figure 4.8: The MapReduce Class

H. Coping With Node Failures

i. The worst thing that can happen is that the compute node at which the Master is executing fails. In this case, the entire MapReduce job must be restarted.
ii. But only this one node can bring the entire process down; other failures will be managed by the Master, and the MapReduce job will complete eventually.
iii. Suppose the compute node at which a Map worker resides fails. This failure will be detected by the Master, because it periodically pings the Worker processes.
iv. All the Map tasks that were assigned to this Worker will have to be redone, even if they had completed. The reason for redoing completed Map asks is that their output destined for the Reduce tasks resides at that compute node, and is now unavailable to the Reduce tasks.
v. The Master sets the status of each of these Map tasks to idle and will schedule them on a Worker when one becomes available.
vi. The Master must also inform each Reduce task that the location of its input from that Map task has changed. Dealing with a failure at the node of a Reduce worker is simpler.
vii. The Master simply sets the status of its currently executing Reduce tasks to idle. These will be rescheduled on another reduce worker later.