Module 4
Chapter 4 Text Book 1
MapReduce, Hive and Pig

Syllabus: MapReduce, Hive and Pig: Introduction, MapReduce Map Tasks, Reduce Tasks and MapReduce Execution, Composing MapReduce for Calculations and Algorithms, Hive, HiveQL, Pig. Text book 1: Chapter 4: 4.1-4.6

Course Outcome 4: Make use of the MapReduce programming model to process the big data along with Hadoop tools

4.1 Introduction

Figure 4.1 shows Big Data architecture design layers: (i) data storage, (ii) data processing and data consumption, (iii) support layer APIs for MapReduce, Hive and Pig running on top of the HDFS Data Store. and (v) application tasks. Pig is a dataflow language, which means that it defines a data stream and a series of transformations.

Hive and Pig are also part of the ecosystem (Figure 4.1). Big Data storage and application support APIs can use Hive and Pig for processing data at HDFS. Processing needs mapping and finding the source file for data. File is in the distributed data store. Requirement is to identify the needed data-block in the cluster. Applications and APIs run at the data nodes stored at the blocks.

The smallest unit of data that can be stored or retrieved from the disk is a block. HDFS deals with the data stored in blocks. The Hadoop application is responsible for distributing the data blocks across multiple nodes. The tasks, therefore, first convert into map and reduce tasks. This requirement arises because the mapping of stored values is very important. The number of map tasks in an application is handled by the number of blocks of input files.

Suppose stored files have key-value pairs. Mapping tells us whether the key is in file or in the value store, a particular cluster and rack. Reduce task uses those values for further processing such as counting, sorting or aggregating

Application sub-task assigned for processing needs only the outputs of reduce tasks. For example, a query needs the required response for a data store. In Example 2.3, a sub-task may just need total daily sales of specific chocolate flavours to compute the analytics and data visualization.

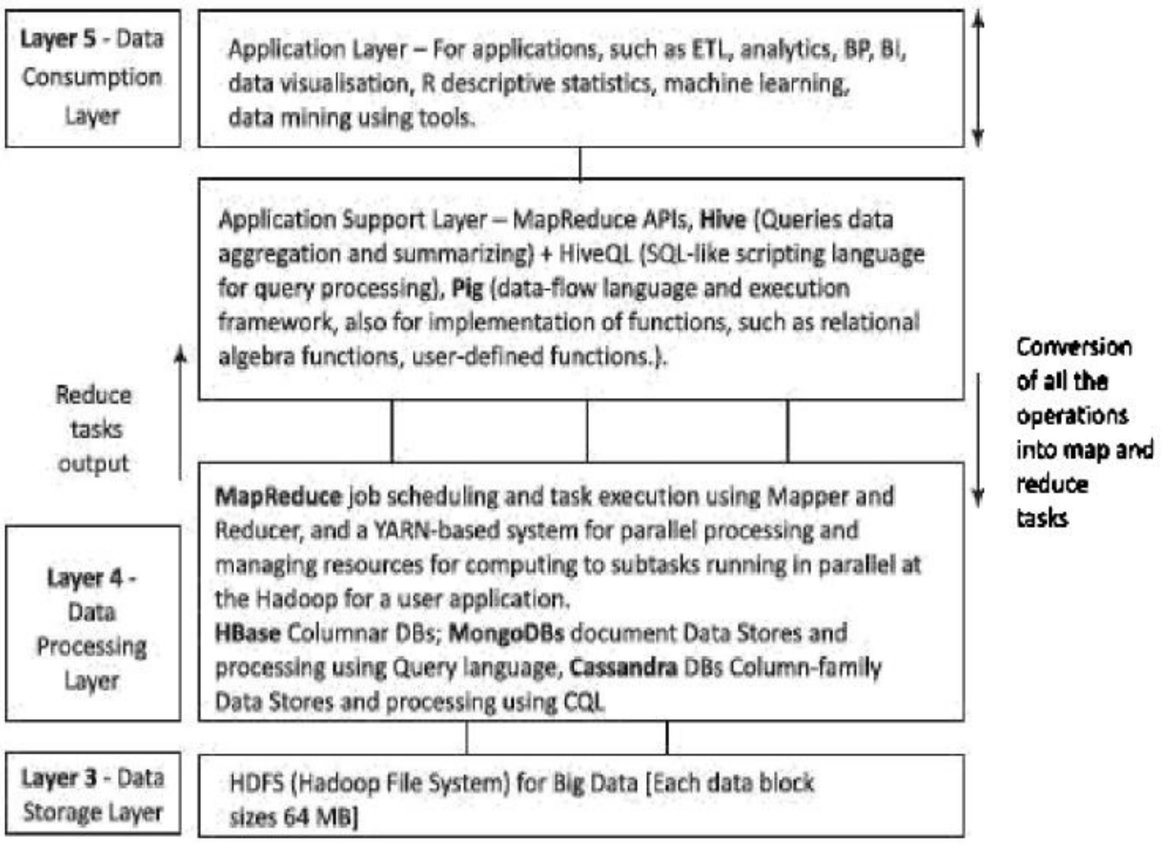

Layer 5 - Data Consumption Layer
Application Layer – For applications, such as ETL, analytics, BP, BI, data visualisation, R descriptive statistics, machine learning, data mining using tools.
Application Support Layer – MapReduce APIs, Hive (Queries data aggregation and summarizing) + HiveQL (SQL-like scripting language for query processing), Pig (data-flow language and execution framework, also for implementation of functions, such as relational algebra functions, user-defined functions.).
Reduce tasks output
Conversion of all the operations into map and reduce tasks
Layer 4 - Data Processing Layer
MapReduce job scheduling and task execution using Mapper and Reducer, and a YARN-based system for parallel processing and managing resources for computing to subtasks running in parallel at the Hadoop for a user application.
HBase Columnar DBs; MongoDBs document Data Stores and processing using Query language, Cassandra DBs Column-family Data Stores and processing using CQL
Layer 3 - Data Storage Layer
HDFS (Hadoop File System) for Big Data [Each data block sizes 64 MB]

Figure 4.1 Big Data architecture design layers

4.2 MapReduce MAP TASKS, REDUCE TASKS AND MapReduce EXECUTION

- Big data processing employs the MapReduce programming model.
- A Job means a MapReduce program. Each job consists of several smaller units, called MapReduce tasks. The Hadoop MapReduce implementation uses Java framework. Figure 4.2 shows MapReduce programming model.

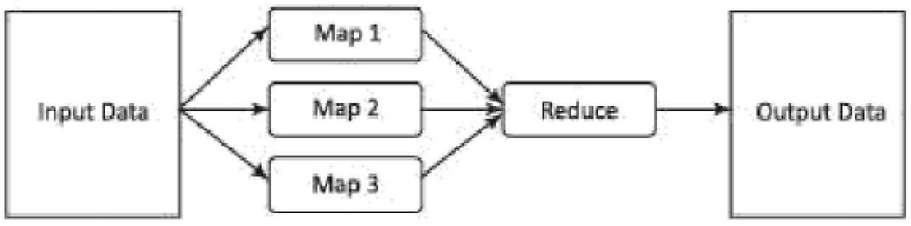
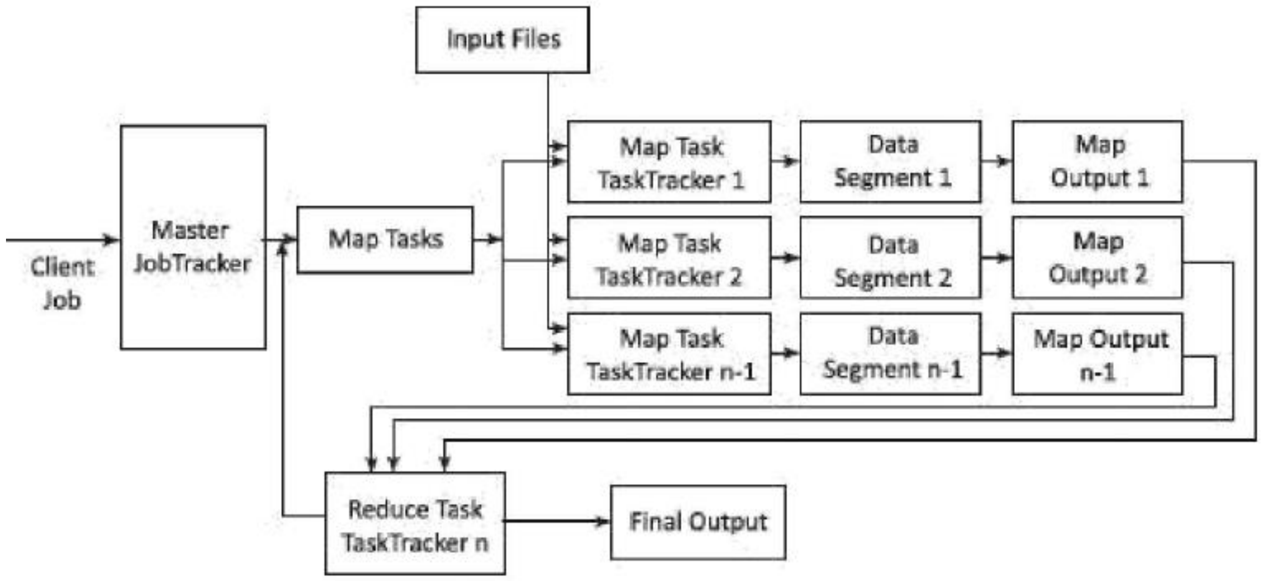


Figure 4.2 MapReduce Programming Model

- The model defines two important tasks, namely Map and Reduce.

- Map takes input data set as pieces of data and maps them on various nodes for parallel processing.
- The reduce task, which takes the output from the maps as an input and combines those data pieces into a smaller set of data. A reduce task always run after the map task (s).
- MapReduce simplifies software development practice.
- It eliminates the need to write and manage parallel codes.
- The YARN resource managing framework takes care of scheduling the tasks, monitoring them and re-executing the failed tasks. Following explains the concept:
- EXAMPLE 4.2
- How does MapReduce enable query processing quickly in Big Data problems?
- MapReduce provides two important functions for query processing. The distribution of task based on user's query to various nodes within the cluster is the first function. The other function is organizing and reducing the results from each node into a cohesive answer to a query.
- The **input data** is in the form of an HDFS file. The **output** of the task also gets stored in the HDFS.
- The **compute nodes and the storage nodes** are the same at a cluster, that is, the MapReduce program and the HDFS are running on the same set of nodes.
- This configuration results in **effectively scheduling** of the sub-tasks on the nodes where the data is already present. This results in high efficiency due to **reduction in network traffic across the cluster**.
- A user application specifies locations of the input/output data and translates into map and reduces functions.
- A job does implementations of appropriate interfaces and/or abstract-classes.
- These, and other job parameters, together 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 by scheduling tasks, monitoring them, and provides status and diagnostic information to the job-client.
- Figure 4.3 shows MapReduce process when a client submits a job, and the succeeding actions by the Job Tracker and TaskTracker.



- **Job Tracker and Task Tracker MapReduce** consists of a single master JobTracker and one slave Task Tracker per cluster node.
- The master is responsible for **scheduling** the component tasks in a job onto the slaves, **monitoring them and re-executing the failed tasks**.
- The **slaves execute** the tasks as directed by the master.
- The data for a MapReduce task is initially at input files. The input files typically reside in the HDFS.
- The files may **be line-based** log files, binary format file, multi-line input records, or something else entirely different.
- These input files are **practically very large**, hundreds of terabytes or even more than it.
- Most importantly, the MapReduce framework operates entirely on key, value-pairs.
- The framework views the input to the task as a set of (key, value) pairs and produces a set of (key, value) pairs as the output of the task, possibly of different types (Section 2.4.2). Example 2.3 explained the process of converting input files into key-values.

4.2.1 Map-Tasks

- Map task means a task that implements a map(), which runs user application codes for each key-value pair (kl, v1). Key k1 is a set of keys. Key k1 maps to a group of data values (Section 3.3.1). Values v1 are a large string which is read from the input file(s).
- The output of map() would be zero (when no values are found) or intermediate key-value pairs (k2, v2).
- The value v2 is the information for the transformation operation at the reduce task using aggregation or other reducing functions.

- Reduce task refers to a task which takes the output v2 from the map as an input and combines those data pieces into a smaller set of data using a combiner.
- The reduce task is always performed after the map task.
- The Mapper performs a function on individual values in a dataset irrespective of the data size of the input.
- That means that the Mapper works on a single das set. Figure 4.4 shows logical view of functioning of map()

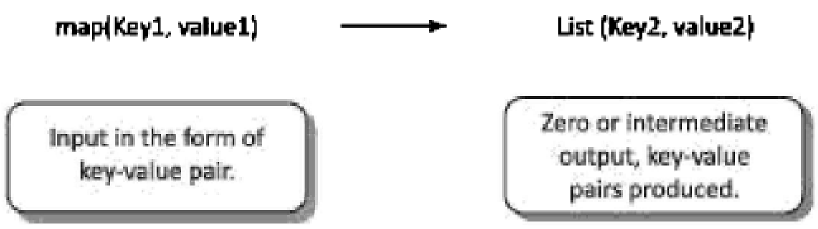

map(Key1, value1)
List (Key2, value2)
Input in the form of key-value pair.
Zero or intermediate output, key-value pairs produced.

Figure 4.4 Logical view of functioning of map()

Hadoop Java API includes Mapper class. An abstract function map() is present in the Mapper class. Any specific Mapper implementation should be a subclass of this class and overrides the abstract function, map().

The Sample Code for Mapper Class

```
public class SampleMapper extends Mapper<k1, Vi, k2, v2>
{
void map (kl key, V1 value, Context context) throws IOException, InterruptedException
{……}
}
```

Individual Mappers do not communicate with each other.

- **Number of Maps** The number of maps depends on the size of the input files, i.e., the total number of blocks of the input files.
- Thus, if the input files are of ITB in size and the block size is 128 MB, there will be 8192 maps.
- The number of map task Nmap can be explicitly set by using setNumMapTasks(int).
- Suggested number is nearly 10-100 maps per node. N can be set even higher.

4.2.2 Key-Value Pair

- Each phase (Map phase and Reduce phase) of MapReduce has key-value pairs as input and output. Data

- Should be converted into key value pair before it is passed to Mapper, as Mapper only understands key-value pairs data.
- Key-value pairs Hadoop MapReduce are generated as follows:
- InputSplit Defines a logical representation of data and presents a split data for processing at individual map().
- RecordReader--- Communicates with the InputSplit and converts the Split into records which are in the form of key-value pars in a format suitable for reading by the Mapper. RecordReader uses TextInputFormat by default for converting data into key-value pairs. RecordReader communicates with the InputSplit until the file is read.
- **Generation of key-value** pair MapReduce depends on the dataset and the required output. Also the functions use the key-value pairs at places: map() input, map() output, reduce() input reduce()

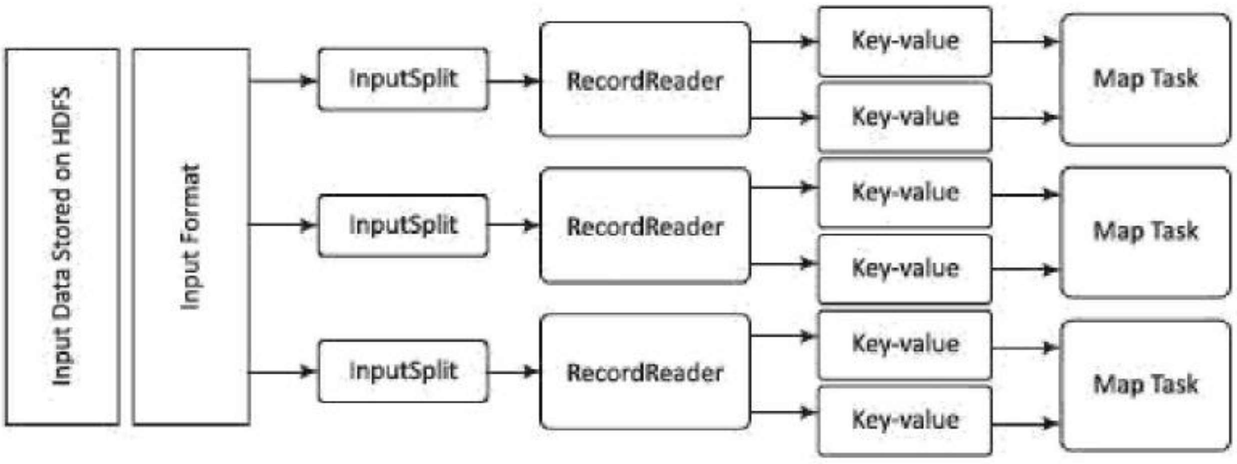


Figure 4.5 Key-value pairing in MapReduce

- **4.2.3 Grouping by Key**
    - When a map task completes, Shuffle process aggregates (combines) all the Mapper outputs by grouping the key-values of the Mapper output, and the value v2 append in a list of values.
    - A "Group By" operation on intermediate keys creates v2.
- **Shuffle and Sorting Phase**
    - Here, all pairs with the same group key (k2) collect and group together, creating one group for each key. So, the Shuffle output format will be a List of <k2, List (v2)>.
    - Thus, a different subset of the intermediate key space assigns to each reduce node. These subsets of the intermediate keys (known as "partitions") are inputs to the reduce tasks.

- Each reduce task is responsible for reducing the values associated with partitions. HDFS sorts the partitions on a single node automatically before they input to the Reducer.

4.2.4 Partitioning

- The Partitioner does the partitioning. The partitions are the semi-mappers in MapReduce.
- Partitioner is an optional class. MapReduce driver class can specify the Partitioner.
- A partition processes the output of map tasks before submitting it to Reducer tasks.
- Partitioner function executes on each machine that performs a map task. Partitioner is an optimization in MapReduce that allows local partitioning before reduce-task phase.
- Functions for Partitioner and sorting functions are at the mapping node. The main function of a Partitioner is to split the map output records with the same key.

4.2.5 Combiners

- Combiners are semi-reducers in MapReduce.
- Combiner is an optional class. MapReduce driver class can specify the combiner.
- The combiner() executes on each machine that performs a map task. Combiners optimize MapReduce task that locally aggregates before the shuffle and sort phase.

Typically, the same codes implement both the combiner and the reduce functions, combiner() on map node and reducer() on reducer node

- The main function of a Combiner is to consolidate the map output records with the same key. The output (key-value collection) of the combiner transfers over the network to the Reducer task as input.
- This limits the volume of data transfer between map and reduce tasks, and thus reduces the cost of data transfer across the network. Combiners use grouping by key for carrying out this function. The combiner works as follows:
- (i) It does not have its own interface and it must implement the interface at reduce(). (ii) It operates on each map output key. It must have the same input and output key-value types as the Reducer class.
- (iii) It can produce summary information from a large dataset because it replaces the original Map output with fewer records or smaller records.

4.2.6 Reduce Tasks

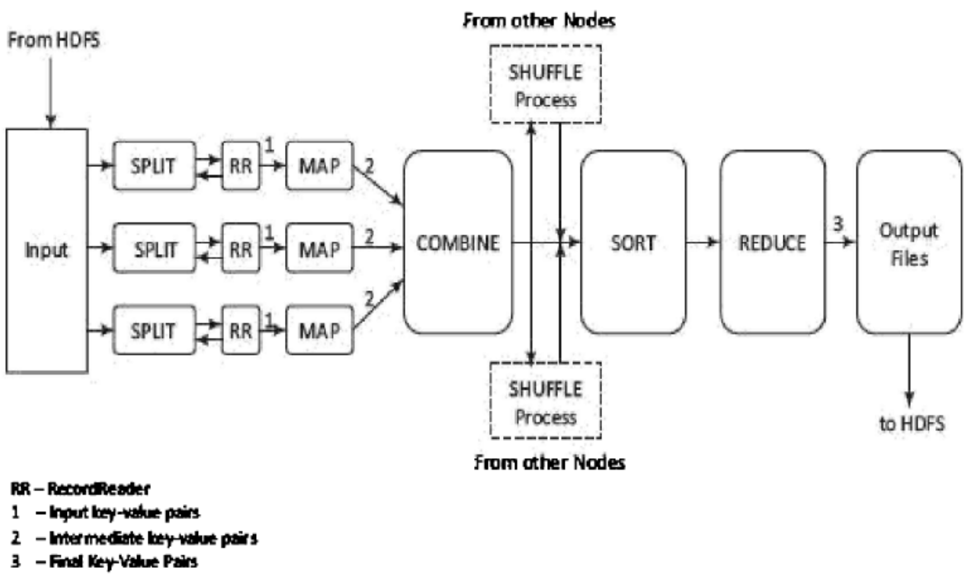
- Java API at Hadoop includes Reducer class. An abstract function, reduce() is in the Reducer. Any specific Reducer implementation should be subclass of this class and override the abstract reduce().
- Reduce task implements reduce() that takes the Mapper output (which shuffles and sorts), which is grouped by key-values (k2, v2) and applies it in parallel to each group.

- Intermediate pairs are at input of each Reducer in order after sorting using the key. Reduce function iterates over the list of values associated with a key and produces outputs such as aggregations and statistics.
- The reduce function sends output zero or another set of key-value pairs (k3, v3) to the final the output file. Reduce: ((k2, list (v2)-> list (k3, v3)}
- **Sample code for reducer class**

```
public class ExampleReducer extends Reducer<k2, v2, k3, v3)

{

void reduce (k2 key, Iterable<v2> values, Context context) throws IOException,
InterruptedException

{...}

}
```

4.2.7 Details MapReduce Processing Steps

- Execution MapReduce job does consider how distributed processing implements. Rather, execution involves formatting (transforming) of data at each
- Figure shows execution steps, data flow, splitting, partitioning sorting on a map node reducer reducer node.
- Example described sales data of the five types chocolates in large number ACVMs (Automatic Chocolate Vending Machines). The example gave answers to following: (i) hourly data log for flavor sales on number ACVMs save using HDFS, (ii) how sample data-collect in a file for 0-1,1-2, 12-13,13-14, 15-16, ... for up 23-24 specific hour sales, (iii) what will be streams map which are input streams to reduce() tasks, and (iv) what will be Reducer outputs.
- Let explore another example, Automotive Components and Predictive Automotive Maintenance Services (ACPAMS). ACPAMS is application (Internet) connected which renders services customers maintenance and servicing (Internet) connected [Chapter Example 1.6(ii)].


From HDFS
Input
SPLIT
RR
MAP
1
2
COMBINE
From other Nodes
SHUFFLE Process
SORT
REDUCE
3
Output Files
to HDFS
RR – RecordReader
1 – Input key-value pairs
2 – Intermediate key-value pairs
3 – Final Key-Value Pairs