MODULE - 4

MapReduce Hive and Pig

Introduction

The data processing layer is the application support layer, while the application layer is the data consumption layer in Big-Data architecture design. When using HDFS the Big Data processing layer includes the APIs of Programs such as MapReduce and Spark.

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, ∴ 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.

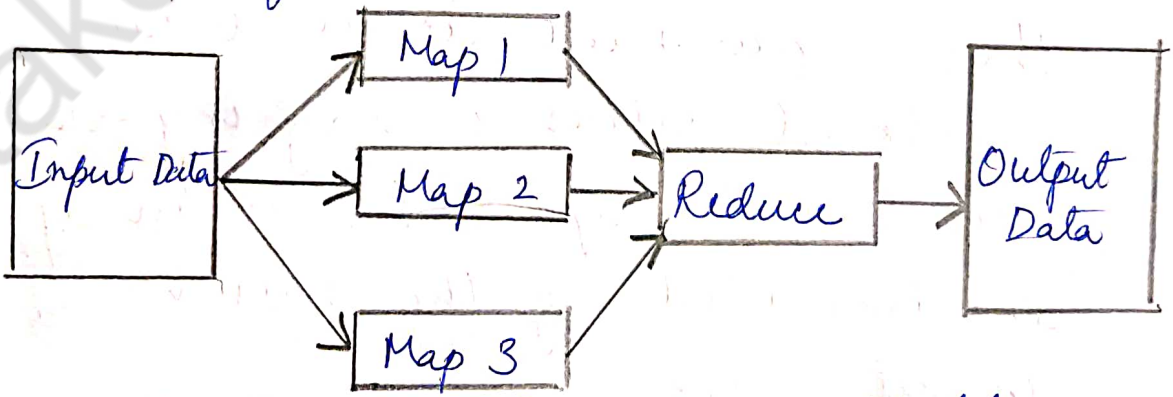
Suppose stored files have key-value pairs. Mapping tells us whether the key

www.takeiteasyengineers.com

is in file or in the value store, in a particular cluster and rack. Reduce task uses those values for further processing such as counting, sorting or aggregating.

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. A software execution framework in MapReduce programming defines the parallel tasks. The tasks give the required result. The Hadoop MapReduce implementation uses Java framework.



MapReduce Programming Model

The model defines two tasks, namely Map and Reduce

www.takeitEasyengineers.com

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. The reduce task always run after the map task.

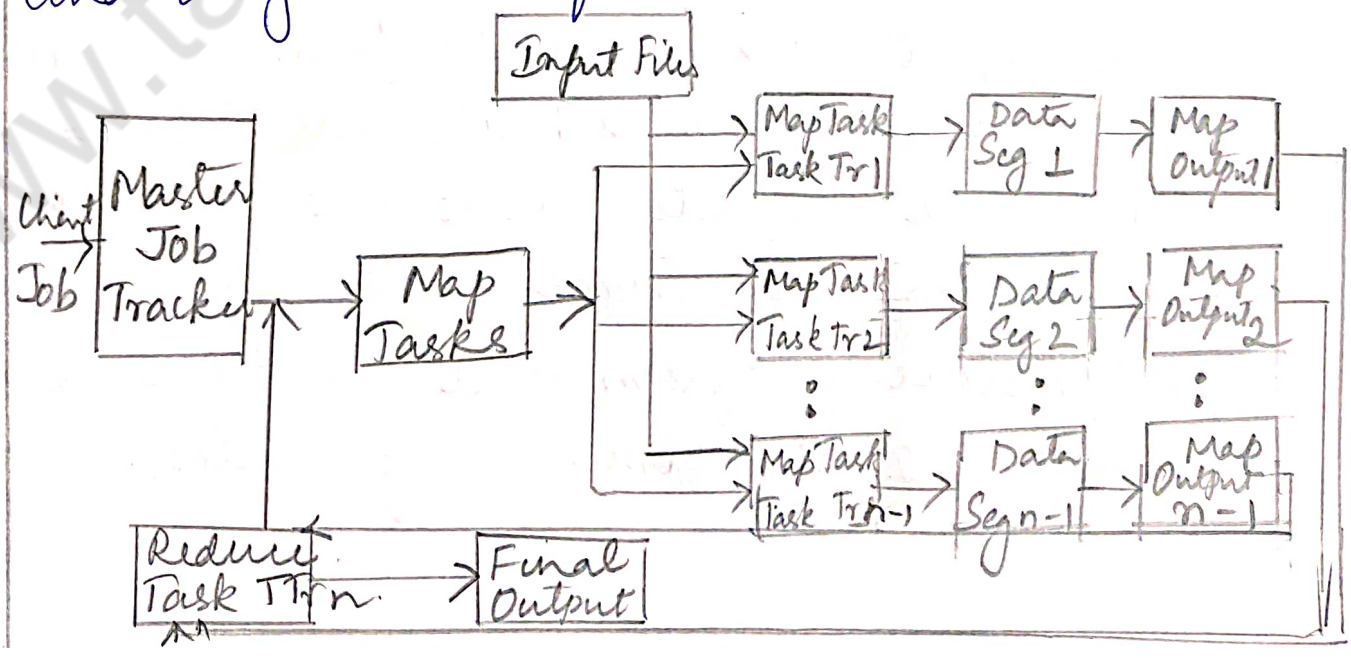
Many real-world situations are expressible using this model. Such Model describes the essence of MapReduce programming where the programs written are automatically parallelize and execute on a large cluster. MapReduce simplifies software development practice. It eliminates the need to write and manage parallel codes. The YARN resource managing framework takes care of scheduling the tasks, monitoring them and re-executing the failed tasks.

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 ie the MapReduce program and the HDFS are running on the same set of nodes.

www.takeitasyengineers.com

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 &/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.



www.takeitasyengineers.com

Figure shows MapReduce process when a client submits a job and the succeeding actions by the JobTracker and TaskTracker.

JobTracker and TaskTracker

MapReduce consists of a single master JobTracker and one slave TaskTracker 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, these input files are practically very large hundreds of terabytes or even more than it.

The MapReduce framework operates entirely on key, value-pairs.

Map-Tasks

Map task means a task that implements a map() which runs user application codes for each key-value pair (K1, V1). Key K1 is a set of keys. Key K1 maps to a group of

www.rahulsaveengineers.com

data values. 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. This means that the Mapper works on a single data set.

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

www.takeeasyengineers.com

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()

Ex:

```
public class SampleMapper extends Mapper <K1, V1, K2, V2>
{
  void map(K1 Key, V1 value, Context context) throws
      IOException, InterruptedException IOException,
  } {..}
```

Individual Mappers do not communicate with eachother

Number of Maps : The number of maps depends on the size of the input files ie the total number of blocks of the input files. Thus, if the input files are of 1TB in size and the block size is 128 MB, there will be 8192 maps. The number of map task N_{map} can be explicitly set by using setNumMapTasks(int) (10-100).

Key - Value Pair

Each phase of MapReduce has Key-value pairs as input and output. Data should be first converted into Key-value pairs

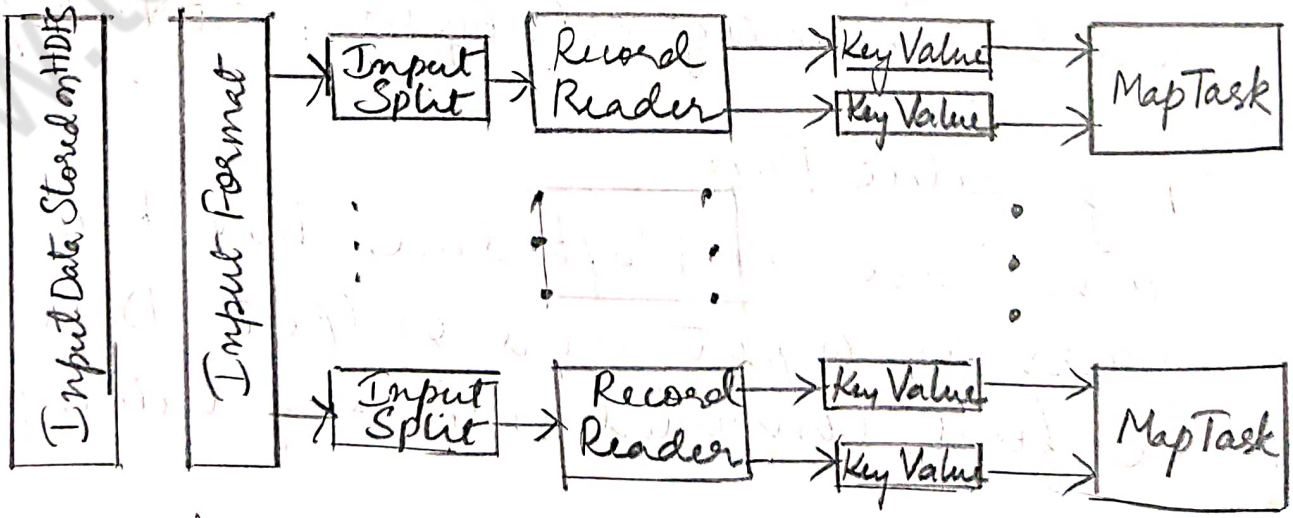
www.takeiteasyengineers.com

as input and output. Data should be first converted into key-value pairs before it is passed to the Mapper, as Mapper understands key-value pairs of data.

Key value pairs in 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 pairs 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.



Key-Value Pairing in MapReduce

www.tutorialsengineering.com

Generation of a key-value pair in MapReduce depends on the dataset and the required output. Also the functions use the key-value pairs at four places: map() input, map() output, reduce() input and reduce() output.

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.

www.takeeasyengineers.com

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. Typically the same codes implement the Partitioner, Combiner as well as reduce() functions. 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.

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

www.takeiteasyengineers.com

Optimize MapReduce task that locally aggregates before the shuffle and sort phase. Combiner() on map node and reducer() on reducer.

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.

i) It does not have its own interface and it must implement the interface at reducer.

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.

www.takeiteasyengineers.com

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 & 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)}

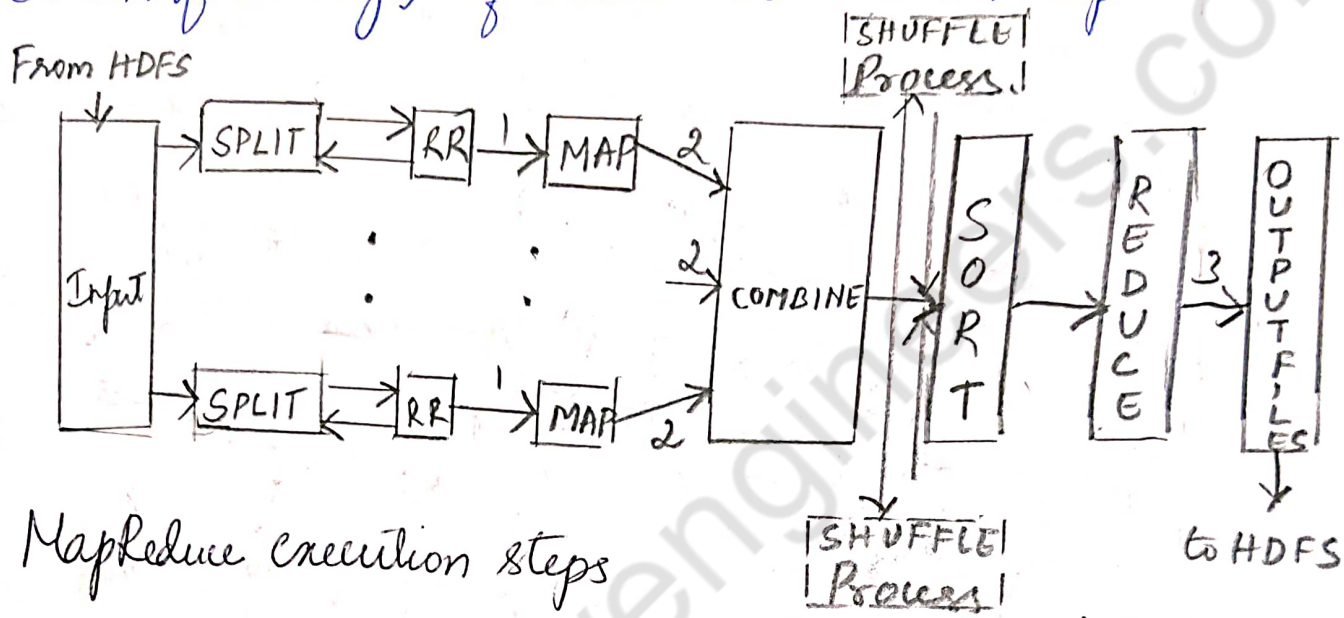
Ex:

```
Public class ExampleReducer extends Reducer<K2,V2,K3,V3>
{
Void reduce(K2 Key, Iterable<V2> values, Context
    context) throws IOException, InterruptedException
    {...}
}
```

www.takeiteasyengineers.com

Details of MapReduce Processing Steps

Execution of MapReduce job does not consider how the distributed processing implements. Rather, the execution involves the formatting (transforming) of data at each step.



MapReduce execution steps

The application submits the inputs. The execution framework handles all other aspects of distributed processing transparently on clusters ranging from a single node to a few thousand nodes. The aspects include scheduling code distribution, synchronization and error and fault handling.

Coping with Node Failures

Node failure occurs when the TaskTracker fails to communicate with the JobTracker for a pre defined period. The JobTracker

www.takeiteasyengineers.com

restarts the TaskTracker for coping with the node failure. By.

The primary way using which Hadoop achieves fault tolerance is through restarting the tasks. Each task nodes (Task Tracker) regularly communicates with the master node, Job Tracker. If a Task Tracker fails to communicate with the JobTracker for a pre defined period (Default: 10min), a task node failure by the Job Tracker is assumed. The JobTracker knows which map and reduce tasks were assigned to each TaskTracker.

If the job is currently in the mapping phase, then another TaskTracker will be assigned to reexecute all map tasks previously run by the failed TaskTracker. All completed map tasks also need to be assigned for reexecution if they belong to incomplete jobs. This is because the intermediate results residing in the failed TaskTracker file system may not be accessible to the reduce task. If the job is in the reducing phase then another TaskTracker will re-execute all reduce tasks that were in progress on the failed TaskTracker.

Once reduce tasks are completed the output writes back to the HDFS. Thus, if a TaskTracker

www.takeiteasyengineers.com

has already completed nine out of ten reduce tasks assigned to it only the tenth task must execute at a different node.

www.takeiteasyengineers.com

4.3 COMPOSING MapReduce FOR CALCULATIONS AND ALGORITHMS

LO 4.2

Composing MapReduce programs for calculations, such as counting and summing; and algorithms for relational algebraic operations, such as projections, unions, intersections, natural joins, grouping, aggregation operations and matrix multiplication

The following subsections describe the use of MapReduce program composition in counting and summing, algorithms for relational algebraic operations, projections, unions, intersections, natural joins, grouping and aggregation, matrix multiplication and other computations.

4.3.1 Composing Map-Reduce for Calculations

The calculations for various operations compose are:

Counting and Summing Assume that the number of alerts or messages generated during a specific maintenance activity of vehicles need counting for a month. Figure 4.8 showed the pseudocode using emit() in the map() of *Mapper* class. *Mapper* emits 1 for each message generated. The reducer goes through the list of 1s and sums them. Counting is used in the data querying application. For example, count of messages generated, word count in a file, number of cars sold, and analysis of the logs, such as number of tweets per month. Application is also in business analytics field.

Sorting Figure 4.6 illustrated MapReduce execution steps, i.e., dataflow, splitting, partitioning and sorting on a map node and reduce on a reducer node. Example 4.3 illustrated the sorting method. Many applications need sorted values in a certain order by some rule or process. *Mappers* just emit all items as values associated with the sorting keys which assemble as a function of items. *Reducers* combine all emitted parts into a final list.

Finding Distinct Values (Counting unique values) Applications such as web log analysis need counting of unique users. Evaluation is performed for the total number of unique values in each field for each set of records that belongs to the same group. Two solutions are possible:

(i) The *Mapper* emits the dummy counters for each pair of field and groupId, and the *Reducer* calculates the total number of occurrences for each such pair.

(ii) The *Mapper* emits the values and groupId, and the *Reducer* excludes the duplicates from the list of groups for each value and increments the counter for each group. The final step is to sum all the counters emitted at the *Reducer*. This requires only one MapReduce job but the process is not scalable, and hence has limited applicability in large data sets.

Collating Collating is a way to collect all items which have the same value of function in one document or file, or a way to process items with the same value of the function together. Examples of applications are producing inverted indexes and extract, transform and load operations.

Mapper computes a given function for each item, produces value of the function as a key, and the item itself as a value. *Reducer* then obtains all item values using group-by function, processes or saves them into a list and outputs to the application task or saves them.

Filtering or Parsing Filtering or parsing collects only those items which satisfy some condition or transform each item into some other representation. Filtering/parsing include tasks such as text parsing, value extraction and conversion from one format to another. Examples of applications of filtering are found in data validation, log analysis and querying of datasets.

www.knowledgeengineer.com

Mapper takes items one by one and accepts only those items which satisfy the conditions and emit the accepted items or their transformed versions. *Reducer* obtains all the emitted items, saves them into a list and outputs to the application.

Distributed Tasks Execution Large computations divide into multiple partitions and combine the results from all partitions for the final result. Examples of distributed running of tasks are physical and engineering simulations, numerical analysis and performance testing.

Mapper takes a specification as input data, performs corresponding computations and emits results. *Reducer* combines all emitted parts into the final result.

Graph Processing using Iterative Message Passing Graph is a network of entities and relationships between them. A node corresponds to an entity. An edge joining two nodes corresponds to a relationship. Path traversal method processes a graph. Traversal from one node to the next generates a result which passes as a message to the next traversal between the two nodes. Cyclic path traversal uses iterative message passing.

Web indexing also uses iterative message passing. Graph processing or web indexing requires calculation of the state of each entity. Calculated state is based on characteristics of the other entities in its neighborhood in a given network. (State means present value. For example, assume an entity is a course of study. The course may be Java or Python. Java is a state of the entity and Python is another state.)

A set of nodes stores the data and codes at a network. Each node contains a list of neighbouring node IDs. MapReduce jobs execute iteratively. Each node in an iteration sends messages to its neighbors. Each neighbor updates its state based on the received messages. Iterations terminate on some conditions, such as completion of fixed maximal number of iterations or specified time to live or negligible changes in states between two consecutive iterations.

Mapper emits the messages for each node using the ID of the adjacent node as a key. All messages thus group by the incoming node. *Reducer* computes the state again and rewrites a node new state.

Cross Correlation Cross-correlation involves calculation using number of tuples where the items co-occur in a set of tuples of items. If the total number of items is N, then the total number of values = N × N. Cross correlation is used in text analytics. (Assume that items are words and tuples are sentences). Another application is in market-analysis (for example, to enumerate, the customers who buy item x tend to also buy y). If N × N is a small number, such that the matrix can fit in the memory of a single machine, then implementation is straightforward.

Two solutions for finding cross correlations are:

(i) The *Mapper* emits all pairs and dummy counters, and the *Reducer* sums these counters. The benefit from using combiners is little, as it is likely that all pairs are distinct. The accumulation does not use in-memory computations as N is very large.

(ii) The *Mapper* groups the data by the first item in each pair and maintains an associative array ("stripe") where counters for all adjacent items accumulate. The *Reducer* receives all stripes for the leading item, merges them and emits the same result as in the pairs approach.

[Stripe means a set of arrays associated with a dataset or a set of rows that belong to a common key with each row having a number of columns.]

The grouping:

- Generates fewer intermediate keys. Hence, the framework has less sorting to do.
- Greatly benefits from the use of combiners.
- In-memory accumulation possible.

www.rajeevsinghcs.com

- Enables complex implementations.
- Results in general, faster computations using stripes than "pairs".

4.3.2 Matrix–Vector Multiplication by MapReduce

Numbers of applications need multiplication of n × n matrix **A** with vector **B** of dimension **n**. Each element of the product is the element of vector **C** of dimension **n**. The elements of **C** calculate by relation, $c_i = \sum_{j=1}^{n} a_{ij} b_j$. An example of calculations is given below.

$$\text{Assume } \mathbf{A} = \begin{vmatrix} 1 & 5 & 4 \\ 2 & 1 & 3 \\ 4 & 2 & 1 \end{vmatrix} \text{ and } \mathbf{B} = \begin{vmatrix} 4 \\ 1 \\ 3 \end{vmatrix}. \quad \ldots (4.1)$$

$$\text{Multiplication } \mathbf{C} = \mathbf{A} \times \mathbf{B} = \begin{bmatrix} 1\times4 + 5\times1 + 4\times3 \\ 2\times4 + 1\times1 + 3\times3 \\ 4\times4 + 2\times1 + 1\times3 \end{bmatrix}$$

$$\text{Hence, } \mathbf{C} = \begin{bmatrix} 21 \\ 18 \\ 21 \end{bmatrix} \quad \ldots (4.2)$$

Algorithm for using MapReduce: The Mapper operates on **A** and emits row-wise multiplication of each matrix element and vector element ($a_{ij} \times b_j \; \forall$ i). The Reducer executes sum() for summing all values associated with each i and emits the element c_i. Application of the algorithm is found in linear transformation.

4.3.3 Relational–Algebra Operations

Explained ahead are the some approaches of algorithms for using MapReduce for relational algebraic operations on large datasets.

4.3.3.1 Selection

Example of Selection in relational algebra is as follows: Consider the attribute names (ACVM_ID, Date, chocolate_flavour, daily_sales). Consider relation

R = {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72), (525, 12122017, KitKat, 82), (525, 12122017, Oreo, 72), (526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)}.

Selection $\sigma_{ACVM\_ID <= 525}(R)$ selects the subset *R*= {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72), (525, 12122017, KitKat, 82), (525, 12122017, Oreo, 72)}.

Selection $\sigma_{chocolate\_flavour = Oreo}$ selects the subset {(524, 12122017, Oreo, 72), (525, 12122017, Oreo, 72), (526, 12122017, Oreo, 72)}.

The test() tests the attribute values used for a selection after the binary operation of an attribute with the value(s) or value in an attribute name with value in another attribute name and the binary operation by which each tuple selects. Selection may also return *false* or *unknown*. The test condition then does not select any.

The *Mapper* calls test() for each tuple in a row. When test satisfies the selection criterion then emits the tuple. The *Reducer* transfers the received input tuple as the output.

www.tekteachengineers.com

4.3.3.2 Projection

Example of *Projection* in relational algebra is as follows:

Consider attribute names (ACVM_ID, Date, chocolate_flavour, daily_sales).

Consider relation R = {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72)}.

Projection $\Pi_{ACVM\_ID}(R)$ selects the subset {(524)}.

Projection, $\Pi_{chocolate\_flavour,\ 0.5*\ daily\_sales}$ selects the subset {(KitKat, 0.5 × 82), (Oreo, 0.5 × 72)}.

The test() tests the presence of attribute (s) used for projection and the factor by an attribute needs projection.

The *Mapper* calls test() for each tuple in a row. When the test satisfies, the predicate then emits the tuple (same as in selection). The *Reducer* transfers the received input tuples after eliminating the possible duplicates. Such operations are used in analytics.

4.3.3.3 Union

Example of Union in relations is as follows: Consider,

R1 = {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72)}

R2 = {(525, 12122017, KitKat, 82), (525, 12122017, Oreo, 72)}

and R3 = {(526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)}

Result of Union operation between R1 and R3 is:

R1 **U** R3 = {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72), (526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)}

The *Mapper* executes all tuples of two sets for union and emits all the resultant tuples. The *Reducer* class object transfers the received input tuples after eliminating the possible duplicates.

4.3.3.4 Intersection and Difference

Intersection Example of Interaction in relations is as follows: Consider,

R1 = {(524, 12122017, Oreo, 72)}

R2 = {(525, 12122017, KitKat, 82)}

and R3 = {(526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)}

Result of Intersection operation between R1 and R3 are

R1 ∩ R3 = {(12122017, Oreo)}

The *Mapper* executes all tuples of two sets for intersection and emits all the resultant tuples. The *Reducer* transfers only tuples that occurred twice. This is possible only when tuple includes primary key and can occur once in a set. Thus, both the sets contain this tuple.

Difference Consider:

R1 = {(12122017, KitKat, 82), (12122017, Oreo, 72)}

and R3 = {(12122017, KitKat, 82), (12122017, Oreo, 25)}

Difference means the tuple elements are not present in the second relation. Therefore, difference set_1 is

R1 – R3 = (12122017, Oreo, 72) and set_2 is R3 – R1 = (12122017, Oreo, 25).

The *Mapper* emits all the tuples and tag. A tag is the name of the set (say, set_1 or set_2 to which a tuple belongs to). The *Reducer* transfers only tuples that belong to set_1.

www.takeengineers.com

Symmetric Difference Symmetric difference (notation is $A \Delta B$ (or $A \ominus B$)] is another relational entity. It means the set of elements in exactly one of the two relations A or B. R3 $\ominus$ R1 = (12122017, Oreo, 25).

The *Mapper* emits all the tuples and tag. A tag is the name of the set (say, set_1 or set_2 this tuple belongs to). The *Reducer* transfers only tuples that belong to neither set_1 or set_2.

4.3.3.5 Natural Join

Consider two relations R1 and R2 for tuples a, b and c. Natural Join computes for R1 (a, b) with R2 (b, c). Natural Join is R (a, b, c). Tuples b joins as one in a Natural Join. The *Mapper* emits the key-value pair (b, (R1, a)) for each tuple (a, b) of R1, similarly emits (b, (R2, c)) for each tuple (b, c) of R2.

The *Mapper* is mapping both with Key for b. The *Reducer* transfers all pairs consisting of one with first component R1 and the other with first component R2, say (R1, a) and (R2, c). The output from the key and value list is a sequence of key-value pairs. The key is of no use and is irrelevant. Each value is one of the triples (a, b, c) such that (R1, a) and (R2, c) are present in the input list of values.

The following example explains the concept of join, how the data stores use the INNER Join and NATURAL Join of two tables, and how the Join compute quickly.

EXAMPLE 4.6

An SQL statement "`Transactions INNER JOIN KitKatStock ON Transactions.ACVM_ID = KitKatStock.ACVM_ID`"; selects the records that have matching values in two tables for transactions of KitKat sales at a particular ACVM. One table is KitKatStock with columns (KitKat_Quantity, ACVM_ID) and second table is Transactions with columns (ACVM_ID, Sales_Date and KitKat_SalesData).

1. What will be INNER Join of two tables KitKatStock and Transactions?
2. What will be the NATURAL Join?

SOLUTION

1. The INNER JOIN gives all the columns from the two tables (thus the common columns appear twice). The INNER JOIN of two tables will return a table with five column: (i) KitKatStock. Quantity, (ii) KitKatStock. KitKat_ACVM_ID, (iii) Transactions.ACVM_ID, (iv)Transactions. KitKat_SalesDate, and (v) Transactions.KitKat_SalesData.
2. The NATURAL JOIN gives all the unique columns from the two tables. The NATURAL JOIN of two tables will return a table with four columns: (i) KitKatStock.Quantity, (ii) KitKatStock. ACVM_ID, (iii) Transactions.KitKat_SalesDate, and (iv) Transactions.KitKat_SalesData.

Values accessible by key in the first table KitKatStock merges with Transactions table accessible by the common key ACVM_ID.

NATURAL JOIN gives the common column once in the output of a query, while INNER JOIN gives common columns of both tables.

Join enables fast computations of the aggregate of the number of chocolates of specific flavour sold.

4.3.3.6 Grouping and Aggregation by MapReduce

Grouping means operation on the tuples by the value of some of their attributes after applying the aggregate function independently to each attribute. A Grouping operation denotes by <grouping attributes> $\Im$ <function-list> (R). Aggregate functions are count(), sum(), avg(), min() and max().

www.tbtsasyenginers.com