MODULE 4

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.*

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 results.
- The Hadoop MapReduce implementation uses Java framework
- The MapReduce programming model defines two important tasks, Map and Reduce.

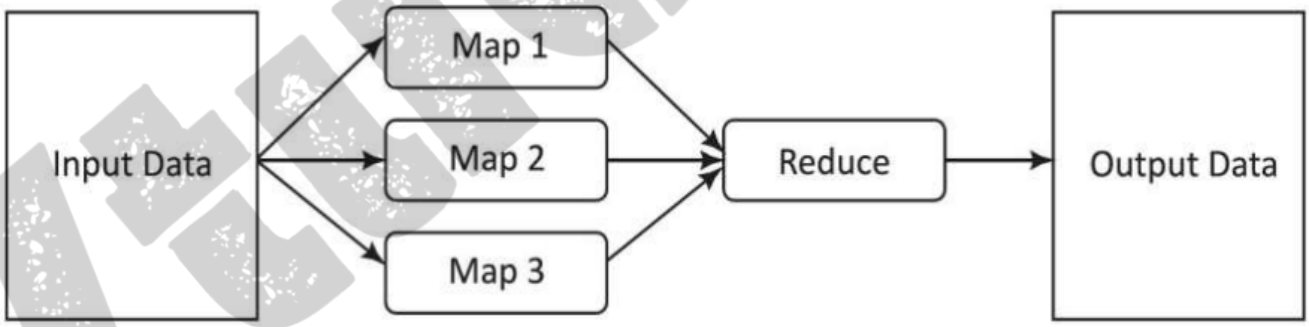


Figure: MapReduce programming model

- 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).
- The input data is in the form of HDFS file. The output of the task also gets stored in the

HDFS.

- The compute nides and the storage nods 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.
- The following figure shows MapReduce process when a client submits a job, and the succeeding actions by the JobTracker and TaskTracker.

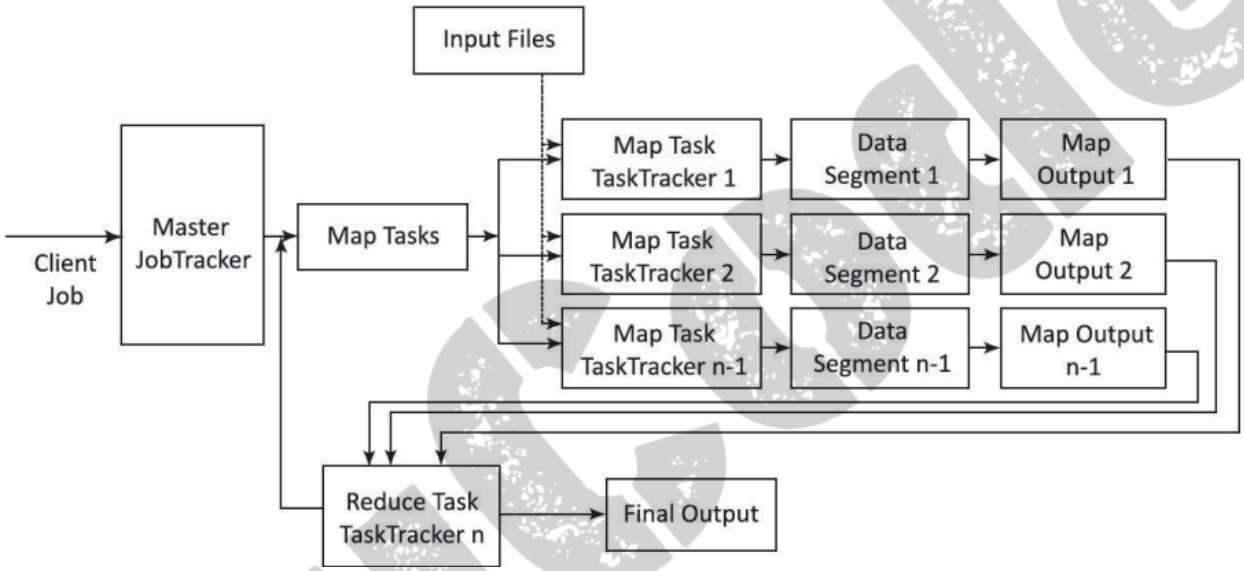
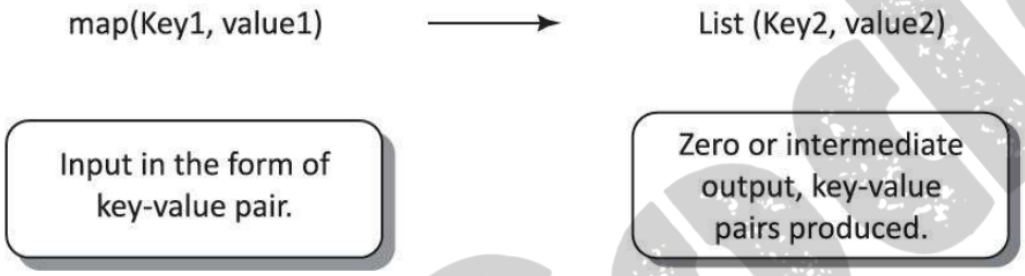


Figure: MapReduce process on client submitting a job

- 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, or something else entirely different

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 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. That means that the Mapper works on a single data set.
- 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 following figure shows the logical view of functioning of map()

The sample code for Mapper class:

```
public class SampleMapper extends Mapper<kl , v2,  k2, v2>
{
    void map(kl key,vl value,Contextcontext}throwsI0Exception, InterruptedException
   { … }
}
```

Key-Value Pair:

- Each phase (Map phase and Reduce phase) of MapReduce has key-value pairs as input and output.
- Data should be first converted into key-value pairs before it is passed to the mapper, as the mapper only 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().

RecordRender:

- 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.

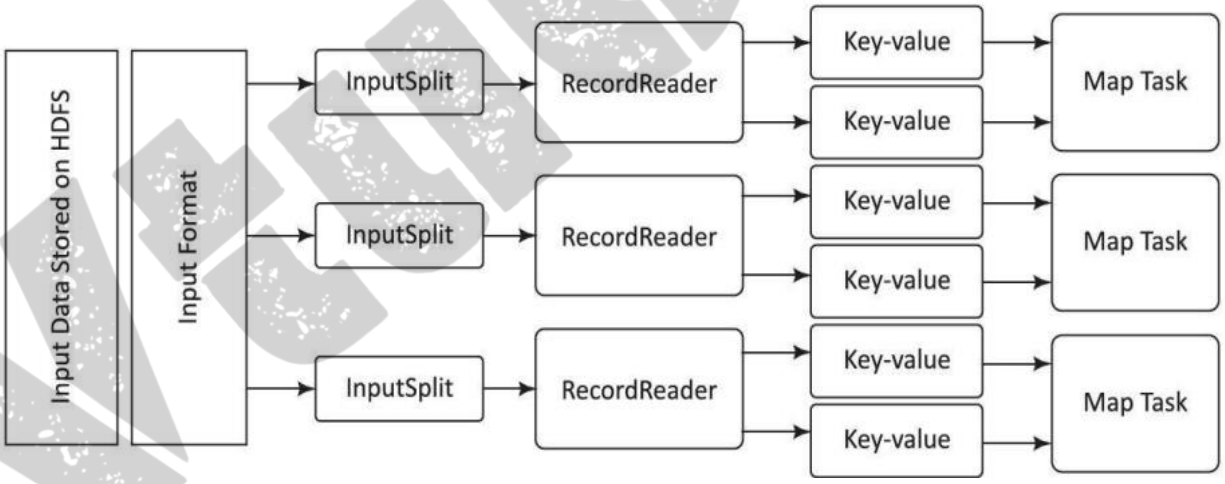


Figure: Key-value pairing in MapReduce

Above figure shows the steps in MapReduce Key-value pairing. Generation of a key-value pair in MapReduce depends on the dataset and the required output. Functions use the key-value pairs at four places:

- map() input
- map() output

- reduce() input
- 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 the 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.

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

Combiners:

- Combiners are semi-reducers in MapReduce and 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.
- 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 record

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
    {…}
}
```

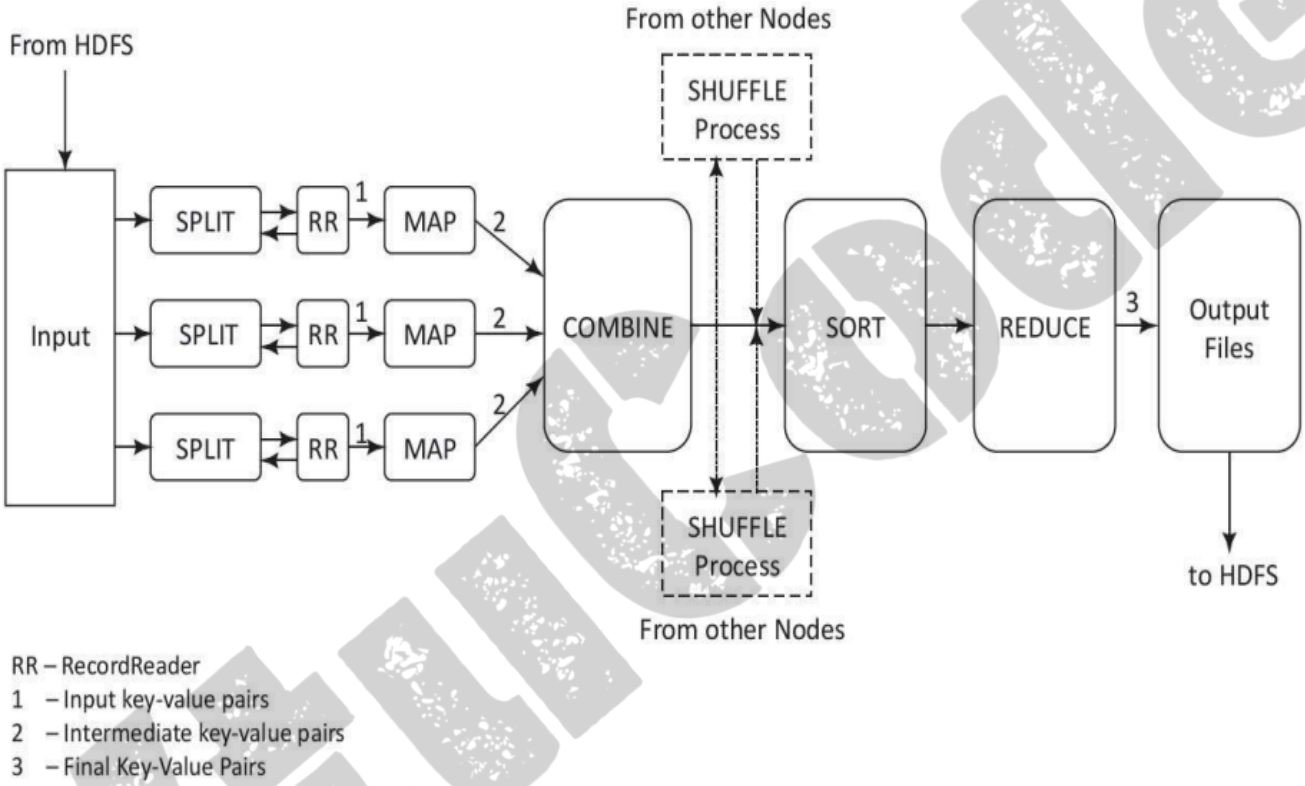


Figure: MapReduce Execution Steps

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.
- The below figure shows the execution steps, data flow, splitting, partitioning and sorting on a map node and reducer on reducer node.

EXAMPLE:

Describe the MapReduce processing steps of a task of ACPAMS.

SOLUTION:

The below figure shows processing steps of an ACPAMS task in MapReduce. Steps are inputs, mapping, combining, shuffling and reducing for the output to application task.

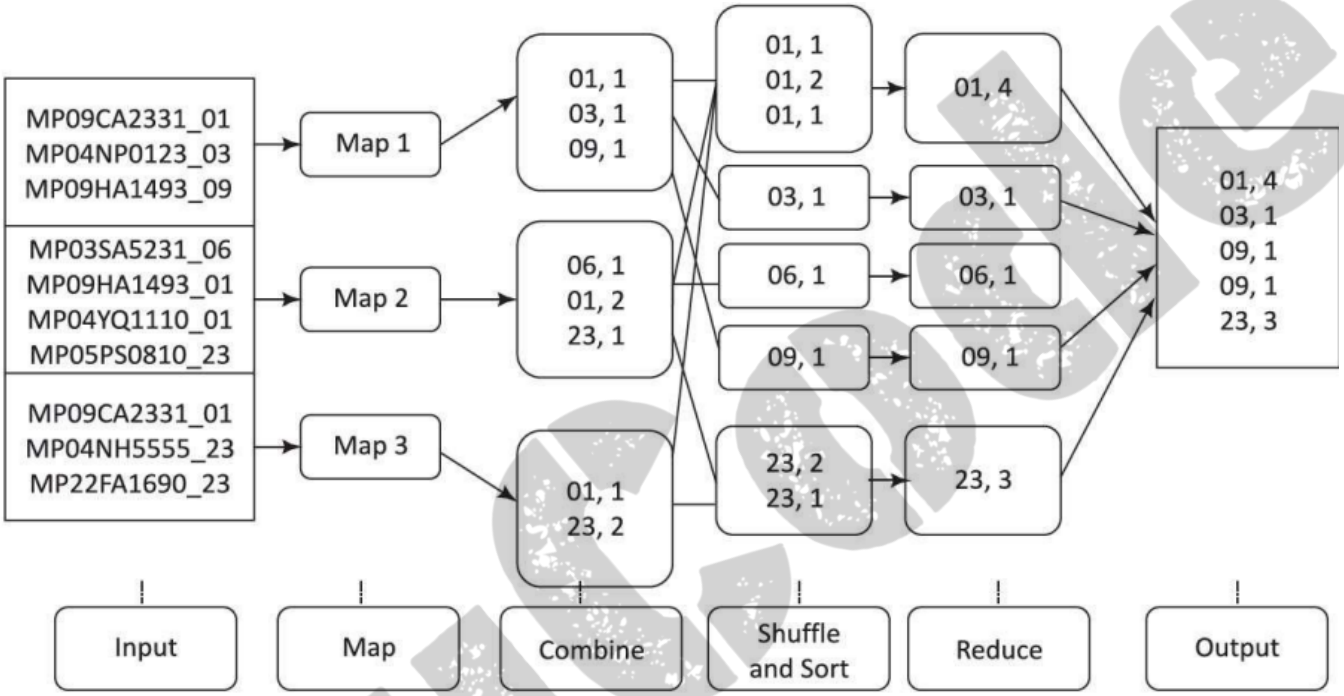


Figure: MapReduce processing steps in ACPAMS application

- 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:

- The primary way using which Hadoop achieves fault tolerance is through restarting the tasks. Each task nodes (TaskTracker) regularly communicates with the master node, JobTracker.
- If a TaskTracker fails to communicate with the JobTracker for a pre-defined period (by default, it is set to 10 minutes), a task node failure by the JobTracker 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 re-execute all map tasks previously run by the failed TaskTracker.
- All completed map tasks also need to be assigned for re-execution if they belong to incomplete jobs
- 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 has already completed nine out of ten reduce tasks assigned to it, only the tenth task must execute at a different node.
- Map tasks are slightly more complicated. A node may have completed ten map tasks but the Reducers may not have copied all their inputs from the output of those map tasks. Now, if a node fails, then its Mapper outputs are inaccessible.
- Thus, any complete map tasks must also be re-executed to make their results available to the remaining reducing nodes. Hadoop handles all of this automatically.
- MapReduce does not use any task identities to communicate between nodes or which re-establishes the communication with other task node.
- Each task focuses on only its own direct inputs and knows only its own outputs. The failure and restart processes are clean and reliable
- The failure of JobTracker (if only one master node) can bring the entire process down; Master handles other failures, and the MapReduce job eventually completes.
- When the Master compute-node at which the JobTracker is executing fails, then the entire MapReduce job must restart

Following points summarize the coping mechanism with distinct Node Failures:

(i) Map TaskTracker failure:

- Map tasks completed or in-progress at TaskTracker, are reset to idle on failure

-Reduce TaskTracker gets a notice when a task is rescheduled on another TaskTracker

(ii) Reduce TaskTracker failure: -Only in-progress tasks are reset to idle

(iii) Master JobTracker failure: -Map-Reduce task aborts and notifies the client (in case of one master node).

Composing Map-Reduce for Calculations

Counting an summing

- Assume that the number of alerts or messages generated during a specific maintenance activity of vehicles needs counting for a month.

Sorting

- The figure 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 Mappe remits 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.

Collating

- Collating is a way to collect all items which have the same value of the 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.

Filtering or Parsing

- Filtering or parsing collects only those items which satisfy some condition or transform each item into some other representation.
- Filtering/parsing includes 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.

Distributed Tasks Execution

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

Graph Processing using Iterative Message Passing

- A 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. The path traversal method processes a graph.
- Traversal from one node to the next generates a result that passes as a message to the next traversal between the two nodes.
- Cyclic path traversal uses iterative message passing.

Cross-Correlation

- Cross-correlation involves calculation using a 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 = NxN. 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).

Matrix-Vector Multiplication by MapReduce

Numbers of applications need multiplication of n x 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,

$C_i = \sum a_{ij} b_j$ .An example of calculations is given below.

Assume $A = \begin{vmatrix} 1 & 5 & 4 \\ 2 & 1 & 3 \\ 4 & 2 & 1 \end{vmatrix}$ and $B = \begin{vmatrix} 1 \\ \\ 3 \end{vmatrix}$

Multiplication $C = A \times B = \begin{bmatrix} 1 \times 4 + 5 \times 1 + 4 \times 3 \\ 2 \times 4 + 1 \times 1 + 3 \times 3 \\ 4 \times 4 + 2 \times 1 + 1 \times 3 \end{bmatrix}$

Hence, C= $\begin{bmatrix} 21 \\ 18 \\ 2 \end{bmatrix}$

Algorithm for using MapReduce:

- The Mapper operates on A and emits row-wise multiplication of each matrix element and vector element ($a_{ij}bji$).
- The Reducer executes sum() for summing all values associated with each I and emits the element I. The application of the algorithm is found in linear transformation.

Relational-Algebra Operations

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 $_{Acvu\_iD<=525}$ (R)selects the subset R= {(524, 12122017, KitKat, 82), (524,12122017,Oreo,72),(525,12122017,KitKat,82),(525,12122017,Oreo,72)}.

Selection 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.

Projection

An 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 IIACVM_ID(R) selects the subset{(524)}.
- Projection, IIchocolate_flavour,0.5*daily_sales*R*selects the subset{(KitKat, 0.5x82), (Oreo, 0.5x72)}.
- The test() tests the presence of attribute (s) used for projection and the factor by an attribute needs projection.

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)}

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 Rl and R3 are

R1∩R3 = {(l2122017, Oreo)}

Difference Consider:

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

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

The difference means the tuple elements are not present in the second relation.

Therefore, difference set_l 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_l or set_2 to which a tuple belongs to). The Reducer transfers only tuples that belong toset_l.

NaturalJoin

Consider two relations R1 and R2 for tuples a,b and c. Natural Join computes

for Rl(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) ofR2.

- 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.

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>J<function-list>

(R). Aggregate functions are count(), sum(), avg(), min() and max().

Matrix Multiplication:

Consider matrices named A (i rows and j columns) and B (j rows and k columns) to produce the matrix C;(i rows and k columns).Consider the elements of matrices A, B and as follows:

a11 a12 . . a1jb11 b12 . . b1kc21 c22 . . c2k

A = a21 a22 . . a2jB = b21 b22 . . b2k C = c21 c22 . . c2k

. . . .

. . . .

ai1 ai2 . . aijbj1 bi2 . . bjkci1 ci2 . . cik

AB = C:Each element evaluates as follow:

Cik = Sum (aij x bjk) j=1to j.va = aij and vb= bj

The first row of C.

C first column element=($a_{11}b_{11}$+$a_{12}b_{21}$+....+a1j.bj_1).

Second column element=(a11$b1_2$+a12b_{22}+…..+ a ij.bj_2).

The k^{th} column element=($a_{11}b_{1k}$+$a_{12}b_{2k}$+....+a1jbj_k).

The second row of C

C first column element=($a_{21}b_{11}$+$a_{22}b_{21}$+....+a2j.bj_1).

Second column element= (a_{21}b12+$a_{22}b_{22}$+….+a2j.bj_2).

The k^{th} Column element=(a21b_{1k}+$a_{22}b_{2k}$+….+a2j.bj_k).

The ith row of C

C first column element=($a_{i1}b_{11}$+ai2b_{21}+….+aij.bj_1).

Second column element = ($a_{i1}b_{12}$+ai2b_{22}+....+aij.bj_2).

Thek^{th}columnelement= ($a_{i1}b_{1k}$+$a_{i2}b_{2k}$+….+aij.bj_k).

Consider two solutions of matrix multiplication.

Matrix Multiplication with Cascadingof Two MapReduce Steps

The table gives the names, attributes, relations R_A, R_B, and R_C, tuples in A, B, C, natural Join of R_A and R_B, keys and values, and seven steps for multiplication of A and B.

Table Seven steps formultiplication of A and B for cascading of two MapReduce Steps:-

Step	Step descriptio n	Matrix A	Matrix B	Matrix C = A.B
1 .	Name	A	B	C
2 .	Specify attribute of (Key,Value pairs of each element[row number, column number,value]	(I,J,va)	(J,K,vb)	(I,K,vc)
3 .	Specify relations	RA=A(J,K,vb)	RB=A(J,K,vb)	RC=A(I,K,vc)
4 .	Consider tuples of A,B and C	(I,j,aij)	(j,k,bjk)	(I,k,cik)
5 .	Find natural Join of RA and RB=Matrix elements (aij,bjk)[j is common in both]	-	-	tuples(I,j,k,va,vb)
6 .	Get tuples for finding Product C			Four-component tuple (I,j,k,va x vb)
7 .	Grouping and aggregation of tuples with attributes I and K			<I,K>J SUM (va x vb)

The product AB = Natural join of tuples in the relations R_A and R_B followed by grouping and aggregation. Natural Join of A (I, J, v_a) and B (J, K, v_b), having only attribute J in common = Tuples (i,j,k,v_a,v_b),from each tuple (i, j,v_a) in A and tuples (j, k, v_b) in B.

1. MapReduce tasks for Steps *5* and 6: Five-component tuple represents the pair of matrix element (aij,bj_k).Requirement is product of these elements. That means four-component tuple (i, j, k, $v_{ax}v_b$),from equation for elements Cik=Sum(aij.bjk)j=1to j.

(a) Mapper Function: (i) Mapper emits the key-value pairs (j, (A, i, a_{ij}))for each matrix element a_{ij},and (ii) Mapper emits the key-value pair (j,

(B,k,bj_k))for each matrix element aij

(b) Reduce Function: Consider the tuples of A=(A, i, aij) for each key j, consider

tuples of B=(B, k, bjk) for each key j. Produce a key-value pair with key equal to(i, k) and value=$a_{ijx}bj_k$.A and B are just the names, which may be represented by0101and 1010.

2. Next MapReduce Steps7: Perform <I, K>JSUM (va x vb).That means do grouping and aggregation, with I and K as the grouping attributes and the sum Ofva x vb as the aggregation.

(c) The Mapper emits the key-value pairs (i, k, v_C) for each matrix element of C inputs with key i and k, and v_C from the earlier task of the reducer va x vb.

(d) Reducer groups (i, k, v_C) in C using [C, i, k, sum (v_C)] from aggregated values of v_C from sum (v_C). Aggregation uses the same memory locations as used by elements v_C. C is just the name, which may be represented by1111.

Matrix Multiplication with One MapReduce Step

MapReduce tasks for Steps 5 to 7 in a single step.

(e) Map Function:

- For each element aij of A,the Mapper emits all the key-value pairs[(i,k),(A,j,aij)]for k=1,2,…, up to the number of columns of B.
- Similarly, emits all the key-value pairs [(i, k), (B, j, bj_k)] for i =1,2,…,up to the number of rows of A. for each element bjk of B.

(f) Reduce Function:

- Consider the tuples of A = (A, i, a_{ij}) for each key j.
- Consider the tuples of B = (B, k, bjk)for each key j.
- Emits the key-value pairs with key equal to (i,k) and value = sum of (aij x bjk) for all values j.

HIVE:

- Hive was created by Facebook.
- Hive is a data warehousing tool and is also a data store on the top of Hadoop.
- An enterprise uses a data warehouse as large data repositories that are designed to enable the tracking, managing, and analyzing the data.
- Hive processes structured data and integrates data from multiple heterogeneous sources. Additionally, also manages the constantly growing volumes of data.

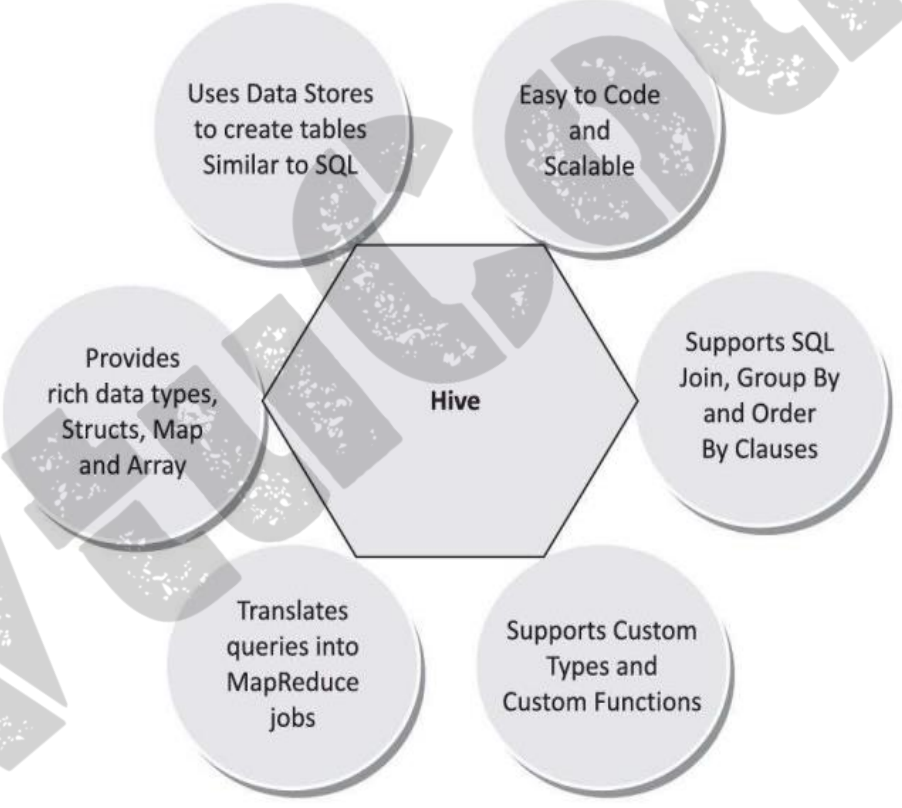


Figure: Main features of Hive

Hive Characteristics:

- Has the capability to translate queries into MapReduce jobs. This makes Hive scalable, able to handle data warehouse applications, and therefore, suitable for the analysis of static data of an extremely large size, where the fast response-time is not a criterion.
- Supports web interfaces as well. Application APIs as well as web-browser clients, can access the Hive DB server.
- Provides an SQL dialect (Hive Query Language, abbreviated HiveQL or HQL).

Limitation:

1. Not a full database. Main disadvantage is that Hive does not provide update, alter and deletion of records in the database.
2. Not developed for unstructured data.
3. Not designed for real-time queries.
4. Performs the partition always from the last column.

Hive Architecture:

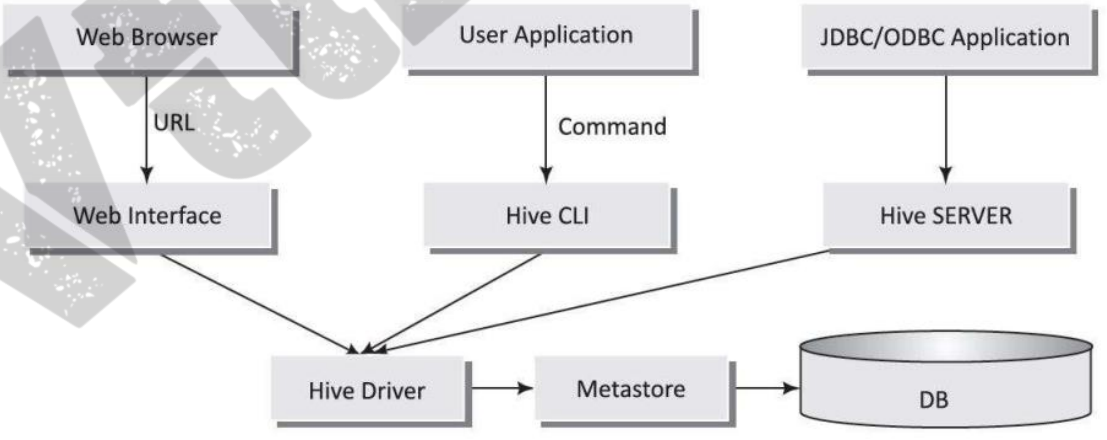


Figure: Hive Architecture

Components of Hive architecture are:

- **Hive Server (Thrift)** - An optional service that allows a remote client to submit requests to Hive and retrieve results. Requests can use a variety of programming languages. Thrift server exposes a very simple client API to execute HiveQL statement.
- **Hive CLI (Command Line Interface) -** Popular interface to interact with Hive. Hive runs in local mode that uses local storage when running the CLI on a Hadoop cluster instead of HDFS.
- **WebInterface**- Hive can be accessed using a web browser as well. This requires a HWI Server running on some designated code. The URL *http:// hadoop:<port no.> / hwi* command can be used to access Hive through the web.
- **Metastore**-It is the system catalog. All other components of Hive interact with the Metastore. It stores the schema or metadata of tables, databases and columns in a table, their data types and HDFS mapping.
- **Hive Driver** - It manages the life cycle of a HiveQL statement during compilation, optimization and execution.

Hive Installation:

Hive can be installed on Windows 10, Ubuntu 16.04 and MySQL.

It requires three software packages:

- Java Development kit for Java compiler (Javac) and interpreter.
- Hadoop
- Compatible version of Hive with Java-Hive1.2 onward supports Java

1.7 or newer.

Steps for installation of Hive in a Linux based OS are as follows:

1. Install Javac and Java from Oracle Java download site. Download jdk 7 or a later version from http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html, and extract the compressed file.
   All users can access Java by Make java available to all users. The user has to move it to the location "/usr/local/" using the required commands.
2. Set the path by the commands for jdk1.7.0_71, export JAVA_HOME=/usr/local/jdk1.7.0_71, export PATH=$PATH: $JAVA_HOME/bin

(Can use alternative install /usr/bin/java usr/local/java/bin/java 2)

3. Install Hadoop http://apache.c1az.org/hadoop/common/hadoop-2.4.1/
4. Make shared HADOOP, MAPRED, COMMON, HDFS and all related files, configure HADOOP and set property such as replication parameter.
5. Name the yarn.nodemanager.aux-services. Assign value to mapreduce_shuffle. Set namenode and datanode paths.
6. Download http://apache.petsads.us/hive/hive-0.14.0/. Use ls command to verify the files $ tar zxvf apache-hive-0.14.0-bin.tar.gz, $ ls

OR

Hive archive also extracts by the command apache-hive-0.14.0-bin apache-hive-0.14.0-bin.tar.gz. ,$ cd $HIVE_HOME/conf, $ cp hive-env.sh.template hive-env.sh, export HADOOP_HOME=/usr/local/hadoop

7. Use an external database server. Configure metastore for the server.

Comparison with RDBMS (Traditional Database):

Hive is a DB system which defines databases and tables. Hive analyzes structured data in DB. Hive has certain differences with RDBMS. Below Table gives a comparison of Hive database characteristics with RDBMS.

Characteristics	**HIVE**	**RDBMS**
Record level queries	No Update and Delete	Insert, Update and Delete
Transaction support	No	Yes
Latency	Minutes or more	In fractions of a second
Data size	Petabytes	Terabytes
Data per query	Petabytes	Gigabytes
Query language	HiveQL	SQL
Support JDBC/ODBC	Limited	Full

Table: comparison of Hive database characteristics with RDBMS

Hive Data Types and File Formats:

- Hive defines various primitive, complex, string, date/time, collection data types and file formats for handling and storing different data formats.

Below Table gives primitive, string, date/time and complex Hive data types and their descriptions.

Data Type Name	Description
TINYINT	1 byte signed integer. Postfix letter is Y.
SMALLINT	2 byte signed integer. Postfix letter is S.
INT	4 byte signed integer.
BIGINT	8 byte signed integer. Postfix letter is L.
FLOAT	4 byte single-precision floating-point number
DOUBLE	8 byte double-precision floating-point number
BOOLEAN	True or False
TIMESTAMP	UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format "YYYY-MM-DD HH:MM:SS.fffffffff"
DATE	YYYY-MM-DD format
VARCHAR	1 to 65355 bytes. Use single quotes (' ') or double quotes (" ")
CHAR	255 bytes
DECIMAL	Used for representing immutable arbitrary precision. DECIMAL (precision, scale) format
UNION	Collection of heterogeneous data types. Create union.
NULL	Missing value representation.

Table: Hive data type and their descriptions

Hive 3 collection data types and their descriptions:

Name	Description
STRUCT	Similar to 'C' struc, a collection of fields of different data types. An access to field uses dot notation. For example, struct ('a', 'b')
MAC	A collection of key-value pairs. Fields access using [ ] notation. For example, map ('key l', 'a', 'key2', 'b')
ARRAY	Ordered sequence of same types. Accesses to fields using array index. For example, array ('a', 'b')

File format and their description:

File Format	Description
Text file	The default file format and a line represent a record. The delimiting characters separate the lines. Text file examples are CSV, TSV, JSON and XML (Section3.3.2).
Sequential file	Flat file which stores binary key-value pairs, and supports compression.
RC File	Record Columnar file (Section 3.3.3.3).

ORCFILE	ORC stands for Optimized Row Columnar which means it can store data in an optimized way than in the other file formats (Section 3.3.3.4).

Record columnar file means one that can be partitioned in rows and then partitioned with columns. Partitioning in this way enables serialization.

Hive Data Model:

Below table gives 3 components of Hive data model and their descriptions

Name	Description
Database	Namespace for tables
Tables	Similar to tables in RDBMS Support filter, projection, join and union operations The table data stores in a directory in HDFS
Partitions	Table can have one or more partition keys that tell how the data stores
Buckets	Data in each partition further divides into buckets based on hash of a column in the table. Stored as a file in the partition directory.

Hive Integration and Workflow Steps:

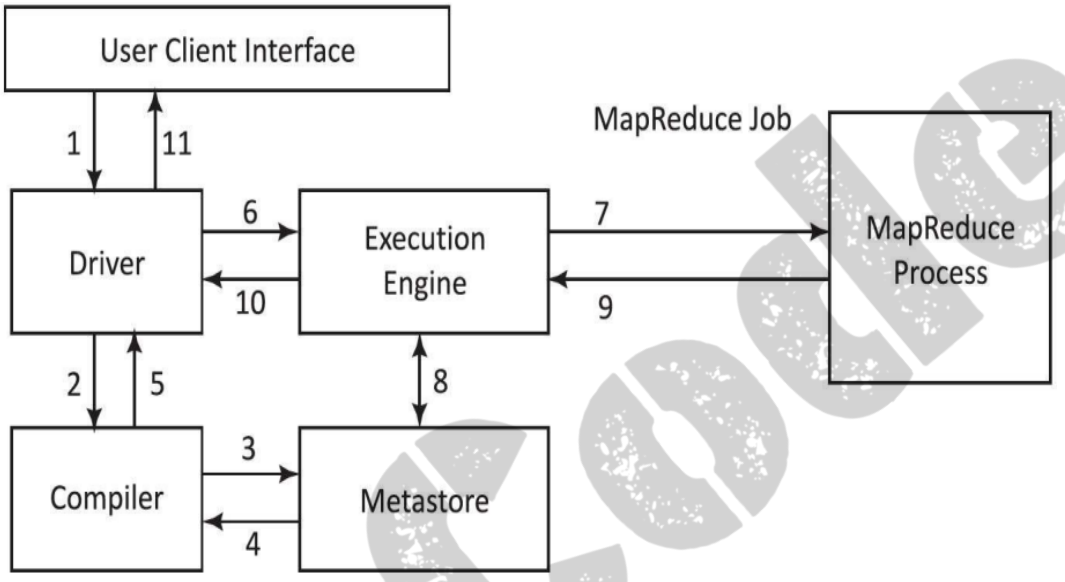


Figure: Dataflow sequences and workflow steps

Steps from 1 to 11 are as follows:

STEP No	OPERATION
1	**Execute Query:** Hive interface (CLI or Web Interface) sends a query to Database Driver to execute the query.
2	**Get Plan:** Driver sends the query to query compiler that parses the query to check the syntax and query plan or the requirement of the query.
3	**Get Metadata:** Compiler sends metadata request to Meta store (of any database, such as MySQL).
4	**Send Metadata:** Meta store sends metadata as a response to compiler.

5	**Send Plan:** Compiler checks the requirement and resends the plan to driver. The parsing and compiling of the query is complete at this place.
6	**Execute Plan:** Driver sends the execute plan to execution engine.
7	**Execute Job**: Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Then, the query executes the job.
8	**Metadata Operations:** Meanwhile the execution engine can execute the metadata operations with Metastore.
9	**Fetch Result:** Execution engine receives the results from Data nodes.
10	**Send Results:** Execution engine sends the result to Driver.
11	**Send Results:** Driver sends the results to Hive Interfaces.

Hive Built-in Functions:

Return Type	Syntax	Description
BIGINT	round(double a)	Returns the rounded BIGINT (8 Byte integer) value of the 8 Byte double-precision floating point number a
BIGINT	floor(double a)	Returns the maximum BIGINT value that is equal to or less than the double.
BIGINT	ceil(double a)	Returns the minimum BIGINT value that is equal to or greater than the double.

double	rand(), rand(int seed)	Returns a random number (double) that distributes uniformly from 0 to 1 and that changes in each row. Integer seed ensured that random number sequence is deterministic.
string	concate(string strl, string str2, ...)	Returns the string resulting from concatenating str1 with str2,
string	substr(string str, int start)	Returns the substring of str starting from a start position till the end of string str.
string	substr(string str, int start, int length)	Returns the substring of str starting from the start position with the given length.
string	upper(string str), ucase (string str)	Returns the string resulting from converting all characters of str to upper case.
string	lower(string str), lcase(string str)	Returns the string resulting from converting all characters of str to lower case.
string	trim(string str)	Returns the string resulting from trimming spaces from both ends. trim (‘12A34 56’) returns ‘12A3456’
string	ltrim(string str); rtrim(string str)	Returns the string resulting from trimming spaces (only one end, left or right hand side or right-hand side spaces trimmed). ltrim(‘12A34 56’) returns 12A3456’ and rtrim(‘ 12A34 56 ’) returns ‘12A3456’.
String	rtrim(string str)	Returns the string resulting from trimming spaces from the end (right hand side) of str.

int	year(string date)	Returns the year part of a date or a timestamp string.
int	month(string date)	Returns the month part of a date or a timestamp string.
int	day(string date)	Returns the day part of a date or a timestamp string.

Table: Return types, syntax and descriptions of the functions

Following are the examples of the returned output:

SELECT floor(10.5) from marks; Output=10.0

SELECT ceil(10.5)from marks; Output=11.0

HiveQL

Hive Query Language (abbreviated HiveQL) is for querying the large datasets which reside in the HDFS environment.

HiveQL script commands enable data definition, data manipulation, and query processing

HiveQL Data Definition Language (DDL)

- HiveQL database commands for data definition for DBs and Tables are CREATE DATABASE, SHOW DATABASE (List of all DBs), CREATE SCHEMA, CREATE TABLE. Following are the HiveQL commands which create a table:

  CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [<database name>.] <table name>

  [(<column name><data type> [COMMENT <column comment>], ...)] [COMMENT <table comment>]

  [ROW FORMAT <row format>]

  [STORED AS <file format>]

- HiveQL database commands for data definition for the DBs and Tables areCREATE DATABASE, SHOW DATABASE (list of all DBs), CREATE SCHEMA, CREATETABLE.
- The following example uses HiveQL commands to create a databasetoys_companyDB

  $HIVE_HOME/binhive- service cli

  hive>set hive.cli.print.current.db=true

  hive>CREATE DATABASE toys_companyDB

  hive>USEtoys_companyDB

  hive (toys_companyDB)> CREATE TABLE toys_tbl (

  >puzzle_code STRING,

  >pieces SMALLINT

  >cost FLOAT);

  hive (toys_company)> quit;

  &ls/home/binadmin/Hive/warehouse/toys_companyDB.db

- The following example uses the command CREATE DATABASE to create a table toy_products
- How do you create a table toy_products with the following fields?

CREATE TABLE IF NOTEXISTStoy_products (ProductCategory String, Productld int, ProductName String, ProductPrice float)

COMMENT 'Toy details'

ROWFORMATDELIMITED

FIELDSTERMINATEDBY'\t'

LINES TERMINATED BY '\n' STORED AS TEXTFILE

Field	Data type
ProductCategory	String
ProductID	Int
ProductName	String
ProductPrice	Float

The option IF NOT EXISTS, Hive ignores the statement in case the table already exists.

HiveQL Data Manipulation Language (DML)

- HiveQL commands for data manipulation are USE <database name>, DROP DATABASE, DROP SCHEMA, ALTER TABLE, DROP TABLE, and LOAD DATA.
- The following is a command for inserting (loading) data into the Hive DBs:

LOAD DATA [LOCAL] INPATH '<file path>' [OVERWRITE] INTO

TABLE <table name> [PARTITION (partcoll=vall, partcol2=val2 ...)]

HiveQL for Querying the Data

- Partitioning and storing are the requirements. A data warehouse should have a large number of partitions where the tables, files, and databases store.
- Querying then requires sorting, aggregating, and joining functions.
- Querying the data is to SELECT a specific entity satisfying a condition, havingthe presence of an entity, or selecting a specific entity using GroupBy.

  SELECT [ALL | DISTINCT] <select expression>,<selectexpression>,...
  FROM <table name>
  [WHERE <where condition>] [GROUP BY <column List>] [HAVING <having condition>]
  [CLUSTER BY <column List> | [DISTRIBUTE BY <columList>]
  [SORT BY<column List>]]

  [LIMIT number];

Partitioning

- Hive organizes tables into partitions. Table partitioning refers to dividing the table data into some parts based on the values of a particular set of columns.

- Partition makes querying easy and fast.
- This is because SELECT is then from the smaller number of column fields.
- The following example explains the concept of partitioning, columnar, and file records formats.

Consider a table T with an eight-column and four-row table. Partition the table, convert in RC columnar format and serialize.

SOLUTION

Firstly, divide the table into four parts t_{r1}, t_{r2}, t_{r3}, and t_{r4} horizontally row-wise. Each sub-table has one row and eight columns. Now, convert each sub-table t_{r1}, t_{r2}, t_{r3}, and t_{r4} into the columnar format, or RC File records.

Each sub-table has eight rows and one column. Each column can serially send data one value at an instance. A column has eight key-value pairs with the same key for all eight.

Create a table with Partition using the command

CREATE [EXTERNAL] TABLE <table name> (<column name l>

<data type 1>, …)

PARTITIONED BY (<column name n><data type n> [COMMENT <columncomment>], …);

Rename a table with Partition using command

ALTER TABLE <table name>PARTITIONpartition_specRENAME TO PARTITION partition_spec;

Adding a Partition in the existing Table using the following command

ALTER TABLE <table name> ADD [IF NOT EXISTS] PARTITION

partition_spec [LOCATION ‘ location1’] partition_spec[LOCATION ’location2’]…;

partition_spec: (p_column=p_col_value, p_column= p_col_value, …)

Drop a Partition in the existing Table using the following command

ALTERTABLE<tablename> DROP[IFEXISTS]PARTITION partition_spec, PARTITION partition_spec;

Advantages of Partition

i. Distributes execution load horizontally.

ii. Query response time becomes faster when processing a small part of the data instead of searching the entire dataset.

Limitations of Partition

i. Creating a large number of partitions in a table leads to a large number of files and directories in HDFS, which is an overhead to Name Node, since it must keep all metadata for the file system in memory only.

ii. Partitions may optimize some queries based on where clauses, but they may be less responsive for other important queries on grouping clauses.

iii. A large number of partitions will lead to a large number of tasks (which will run in separate JVM) in each MapReduce job, thus creating a lot of overhead in maintaining JVM start up and tear down (A separate task will be used for each file).

Bucketing

- A partition itself may have a large number of columns when tables are very large. Tables or partitions can be sub-divided into buckets.
- Division is based on the hash of a column in the table.
- Consider bucketed column C_{bucket_i}. First, define a hash_functionH() according to the type of the bucketed column.
- Let the total number of buckets = $N_{buckets}$
- Let C_{bucket_i} denote i^{th} bucketed column

The hash value h_i = hashing function $H(C_{bucket\_i})$ mod $(N_{buckets\_i})$

Buckets provide an extra structure to the data that can lead to more efficient query processing when compared to undivided tables or partitions.
Buckets store as a file in the partition directory.

Views

- A program uses functions or objects.
- Constructing an object instance enables layered design and encapsulating the complexity due to methods and fields.
- Similarly, Views provide ease of programming. Complex queries simplify using reusable Views.
- A HiveQL View is a logical construct.
- A View provisions the following:
  - Saves the query and reduces the query complexity
  - Use a View like a table but a View does not store data like a table

- Hides the complexity by dividing the query into smaller, more manageable pieces.
- Hive query statement when uses references to a view, the Hive executes the View and then the planner combines the information in View definition with the remaining actions on the query (Hive has a query planner, which plans how a query breaks into sub-queries for obtaining the right answer)

Sub-Queries (Using Views)

Consider the following query with a nested sub-query:
A table *toy_ tbl* contains many values for categories of toys. Assume a table for Toy_Airplane of product code 10725. Consider a nested query:

FROM (

SELECT * toy_tbl_Join people JOIN Toy_Airplane

ON (Toy_Airplane.ProductId= productId.id) WHERE productId=10725) toys_catalog SELECT productMfgDate WHERE productMfgDate =‘2017-10-23’;

Create a View for using that in a nested query.

SOLUTION

Create a view named toy_tbl_MiniJoin
CREATE VIEW toy_tbl_MiniJoin AS
SELECT * toy_tbl_Join people JOIN Toy_Airplane
ON (Toy_Airplane.ProductId=productId.id) WHERE productId=10725
) toys_catalog SELECT productMfgDate WHERE productMfgDate = ‘2017-10-23’

Aggregation:

- Hive supports the following built-in aggregation functions.
- The usage of these functions is same as the SQL aggregate functions.
- The below table lists the functions, their syntax and descriptions.

Table 4.10 Aggregate functions, their return type, syntax and descriptions

ReturnType	Syntax	Description
BIGINT	count(*), count(expr)	Returns the total number of retrieved rows.
DOUBLE	sum(co1), sum(DISTINCT col)	Returns the sum of the elements in the group or the sum of the distinct values of the column in the group.
DOUBLE	avg (col), avg (DISTINCT co1)	Returns the average of the elements in the group or the average of the distinct values of the column in the group.
DOUBLE	min (col)	Returns the minimum value of the column in the group.
DOUBLE	max(co1)	Returns the maximum value of the column in the group.

Usage examples are:

- Example: SELECT Product Category, count(*) FROM toy_tbl GROUP BY Product Category;
- Example: SELECT Product Category, sum(Product Price) FROM toy_tbl GROUP BY Product Category;

Join:

- A JOIN clause combines columns of two or more tables, based on a relation between them.
- HiveQL Join is more or less similar to SQL JOINS.
- Following uses of two tables show the Join operations.
- The below table gives an example of a table named toy_tbl of Product categories, Productld and Product name.

Table: Table of Product categories, Product Id and Product name

ProductCategory	ProductId	ProductName
Toy_Airplane	10725	Lost temple
Toy_Airplane	31047	Propeller plane
Toy_Airplane	31049	Twin spin helicopter
Toy_Airplane	31054	Blue express
Toy_Airplane	10254	Winter holiday Toy_Train

The below table gives an example of a table named price of ID or Product ID and Product Cost.

Table: Table of ID and Product Cost

Id	ProductPrice
10725	100.0
31047	200.0

31049	300.0
31054	450.0
10254	200.0

Different types of joins are follows:

- JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

JOIN:

- Join clause combines and retrieves the records from multiple tables.
- Join is the same as OUTERJOIN in SQL.
- AJOIN condition uses primarykeys and foreign keys of the tables.

```
SELECT t.ProductId, t.ProductName, p.ProductPrice
FROM toy_tbltJOIN price p
ON(t.ProductId =p.Id);
```

LEFTOUTER JOIN:

- ALEFTJOIN returnsallthevaluesfromthelefttable,plusthe matched values from the right table or NULL in case of no matching JOIN predicate.

```
SELECT t.ProductId, t.ProductName, p.ProductPrice
FROM toy_tbl t LEFT OUTER JOIN price p
ON (t.ProductId =p.Id);
```

RIGHT OUTER JOIN:

A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate.

SELECT t.ProductId, t.ProductName, p.ProductPrice

FROM toy_tbl t RIGHT OUTER JOIN price p

ON (t.ProductId = p.Id);

FULLOUTERJOIN:

Hive QLFULLOUTER JOIN combines the records of both the left and the right outer tables that fulfill the JOIN condition.

The joined table contains either all the records from both the tables or fills in NULL values for missing matches on either side.

SELECT t.ProductId, t.ProductName, p.ProductPrice

FROM toy_tbl t FULL OUTER JOIN price p

ON (t.ProductId = p.Id);

Group By Clause:

GROUP BY, HAVING, ORDER BY, DISTRIBUTE BY, CLUSTER BY are HiveQL clauses.

An example of using the clauses is given below:

How do SELECT statement uses GROUP BY, HAVING, DISTRIBUTE BY, CLUSTER BY?

SOLUTION:

(i) Use of SELECT statement with WHERE clause as follows:

SELECT[ALL|DISTINCT]<select expression>,<select expression>,... FROM <table name>

[WHERE <where condition>] [GROUP BY <column List>] [HAVING <having condition >]

[CLUSTER BY <column List> | [DISTRIBUTE BY <column List>] [SORT BY <column List>]]

[LIMIT number];

PIG

Apache developed Pig, which:

- Is an abstraction over MapReduce
- Is an execution framework for parallel processing
- Reduces the complexities of writing a MapReduce program
- Is a high-level dataflow language. Dataflow language means that a Pig operation node takes the inputs and generates the output for the next node.
- Is mostly used in HDFS environment
- Performs data manipulation operations at files at data nodes in Hadoop.

Applications of Apache Pig

1) Analyzing large datasets
2) Executing tasks involving adhoc processing
3) Processing large data sources such as web logs and streaming online data
4) Data processing for search platforms. Pig processes different types of data.
5) Processing time-sensitive data loads; data extracts and analyzes quickly. For example, analysis of data from Twitter to find patterns for user behavior and recommendations.

Features:

(i) Apache PIG helps programmers write complex data transformations using scripts (without using Java). Pig Latin language is very similar to SQL and possess a rich set of built-in operators, such as group, join, filter, limit, order by, parallel, sort and split. It provides an interactive shell known as Grunt to write Pig Latin scripts.

(ii) Creates user defined functions (UDFs) to write custom functions which are not available in Pig.

(iii) Process any kind of data, structured, semi-structured or unstructured data, coming from various sources.

(iv) Reduces the length of codes using multi-query approach. Pig code of 10 linesisequaltoMapReducecodeof200lines.

(v) Handles inconsistent schema in case of unstructured data as well.

(vi) Extracts the data, performs operations on that data and dumps the data in the required format in HDFS.

(vii) Performs automatic optimization of tasks before execution.

(viii) Programmers and developers can concentrate on the whole operation without a need to create mapper and reducer tasks separately.

(ix) Reads the input data files from HDFS or the data files from other sources such as local file system, stores the intermediate data and writes back the output in HDFS.

(x) Pig characteristics are data reading, processing, programming the UDFs in multiple languages and programming multiple queries by fewer codes.

(xi) Pig derives guidance from four philosophies, live anywhere, take anything, domestic and run as if flying.

Differences between Pig and MapReduce:

Pig	MapReduce
A dataflow language	A data processing paradigm
High-level language and flexible	Low-level language and rigid
Performing Join, filter, sorting or ordering operations are quite simple	Relatively difficult to perform Join, filter, sorting or ordering operations between datasets
Programmer with a basic knowledge of SQL can work conveniently	Complex Java implementations require exposure to Java language
Uses multi-query approach, thereby reducing the length of the codes significantly	Require almost 20 times more the number of lines to perform the same task
No need for compilation for execution; operators convert internally into MapReduce jobs	Long compilation process for Jobs
Provides nested data types like tuples, bags and maps	No such data types

Differences between Pig and SQL:

Pig	SQL
Pig Latin is a procedural language	A declarative language
Schema is optional, stores data without assigning a schema	Schema is mandatory
Nested relational data model	Flat relational data model
Provides limited opportunity for Query optimization	More opportunity for query optimization

Differences between Pig and Hive:

Pig	Hive
Originally created at Yahoo	Originally created at Facebook
Exploits Pig Latin language	Exploits HiveQL
Pig Latin is a dataflow language	HiveQL is a query processing language
Pig Latin is a procedural language and it fits in pipeline paradigm	HiveQL is a declarative language
Handles structured, unstructured and semi-structured data	Mostly used for structured data

Pig Architecture:

- Firstly, Pig Latin scripts submit to the Apache Pig Execution Engine.
- The figure shows Pig architecture for scripts dataflow and processing in the HDFS environment.

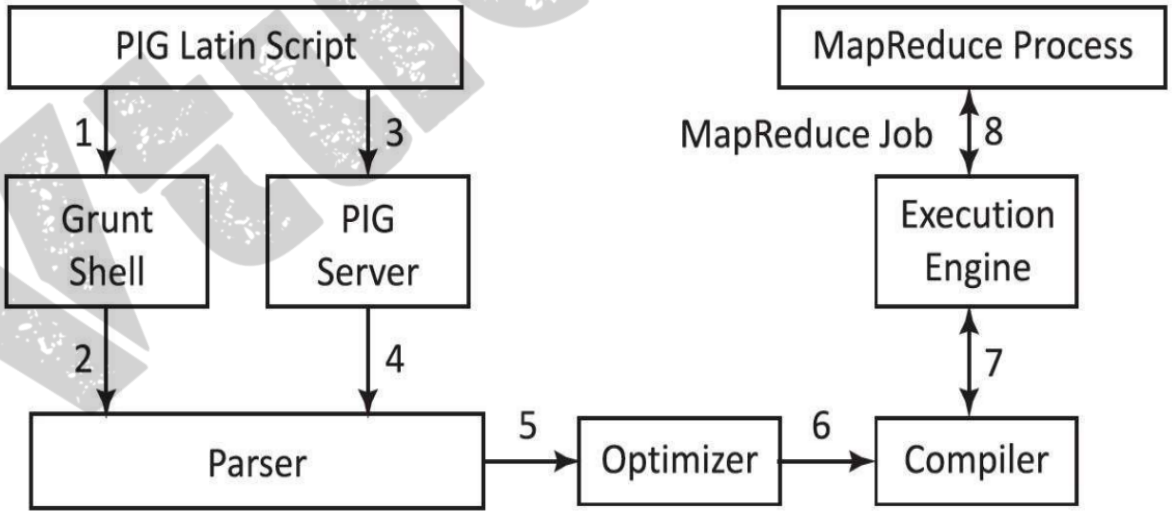


Figure:Pig architecture for scripts data flow and processing

The three ways to execute scripts are:

1. Grunt Shell: An interactive shell of Pig that executes the scripts.
2. Script File:Pig commands written in a script file that execute at Pig Server.
3. Embedded Script: Create UDFs for the functions unavailable in Pig built-in operators. UDF can be in other programming languages. The UDFs can embed in Pig Latin Script file.

Parser:

- A parser handles Pig scripts after passing through Grunt or Pig Server.
- The Parser performs type checking and checks the script syntax.
- The output is a Directed Acyclic Graph (DAG).
- Acyclic means only one set of inputs are simultaneously at a node, and only one set of output generates after node operations.
- DAG represents the Pig Latin statements and logical operators.
- Nodes represent the logical operators.
- Edges between sequentially traversed nodes represent the data flows.

Optimizer:

- The DAG is submitted to the logical optimizer.
- The optimization activities, such as split, merge, transform and reorder operators execute in this phase.
- The optimization is an automatic feature.
- The optimizer reduces the amount of data in the pipeline at any instant of time, while processing the extracted data.
- It executes certain functions for carrying out this task, as explained as follows:

Push Up Filter: If there are multiple conditions in the filter and the filter can be split, Pig splits the conditions and pushes up each condition separately.

Selecting these conditions at an early stage helps in reducing the number of records remaining in the pipeline.

Push Down For Each Flatten: Applying flatten, which produces a cross product between a complex type such as a tuple, bag or other fields in the record, as late as possible in the plan.

This keeps the number of records low in the pipeline.

Column Pruner:

Omits never used columns or the ones no longer needed, reducing the size of the record.

This can be applied after each operator, so that the fields can be pruned as aggressively as possible.

Map Key Pruner:

Omits never used map keys, reducing the size of the record.

Limit Optimizer:

If the limit operator is immediately applied after loador sort operator, Pig converts the load or sort into a limit-sensitive implementation, which does not require processing the whole dataset.

Compiler:

- The compiler compiles after the optimization process.
- The optimized codes are a series of MapReduce jobs.

Execution Engine:

- Finally, the MapReduce jobs submit for execution to the engine.
- The MapReduce jobs execute and it outputs the final result.

Apache Pig – Grunt Shell:

- Main use of Grunt shell is for writing Pig Latin scripts.
- Any shell command invokes using sh and ls.
- Syntax of sh command is:

  grunt>sh shell command parameters

- Syntax of ls command:

grunt>shls

- Grunt shell includes a set of utility commands. Included utility commands are clear, help, history, quit, and set.
- The shell includes commands such as exec, kill and run to control the Pig from the Grunt shell.

InstallingPig:

Following are the steps for installing Pig:

1. Download the latest version from -https://pig.apache.org/
2. Download the tar files and create a Pigdirectory
   $ cd Downloads/
   $ tar zxvf pig-0.15.0-src.tar.gz
   $ tar zxvf pig-0.15.0.tar.gz
   $ mv pig-0.15.0-src.tar.gz/* /home/Hadoop/Pig/
3. Configure thePig
   export PIG HOME =/home/Hadoop/Pig
   export PATH = $PATH:/home/Hadoop/pig/bin
   export PIG_CLASSPATH = $HADOOP_ HOME/conf

Pig Latin DataModel:

- Pig Latin supports primitive data types which are atomic or scalar data types.
- Atomic data types are int, float, long, double, char [], byte [].
- The language also defines complex data types.
- Complex data types are tuple, bag and map.
- The below table gives data types and examples.

Table: Data types and examples

Data type	Description	Example
Bag	Collection of tuples	{(1,1), (2,4)}
Tuple	Ordered set of fields	(1,1)
map (data map)	Set of key-value pairs	[Number#1]

Int	Signed 32-bit integer	10
Long	Signed 64-bit integer	10L or 101
Float	32-bit floating point	22.7F or 22.7f
Double	64-bit floating point	3.4 or 3.4e2 or 3.4E2
Char array	Char [], Character array	data analytics
Byte array	BLOB (Byte array)	ff00

Tuple:

- Tuple is a record of an ordered set of fields.
- A tuple is similar to a row in a table of RDBMS.
- The elements inside a tuple do not necessarily need to have a schema associated to it.
- A tuple represents by '()' symbol. For example, (1, Oreo, 10, Cadbury).
- The below figure shows Pig data model with fields, Tuple and Bag.

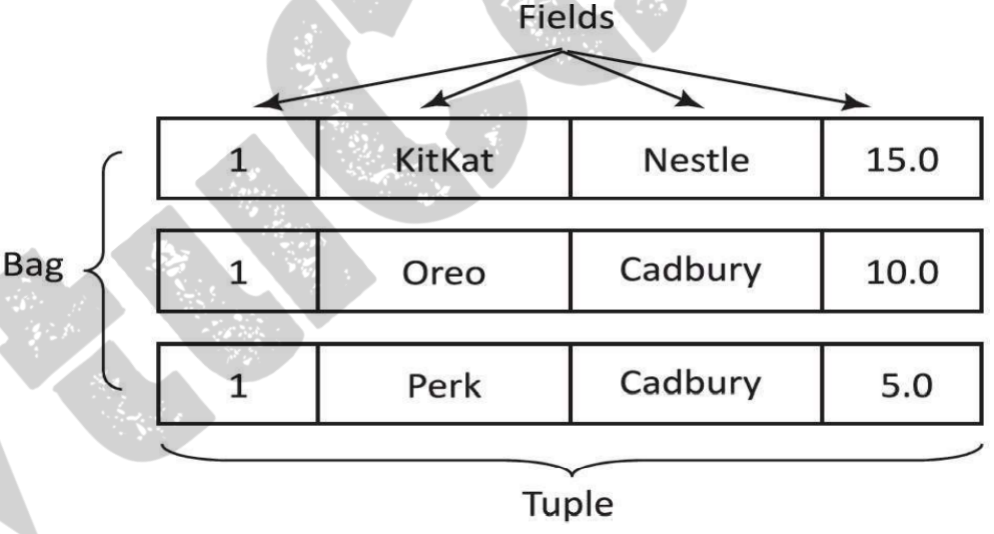


Figure: Pig Data Model with fields, Tuple and Bag

Bag:

A bag is an unordered set of tuples.

A bag can contain duplicate tuples as it is not mandatory that they need to be unique.

Each tuple can have any number of fields (flexible schema).

A bag can also have tuples with different data types.

{} symbol represents a bag.

- It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contains the same number of fields or that the fields in the same position (column) have the same type.
- For example, {(Oreo, 10), (KitKat, 15, Cadbury)}

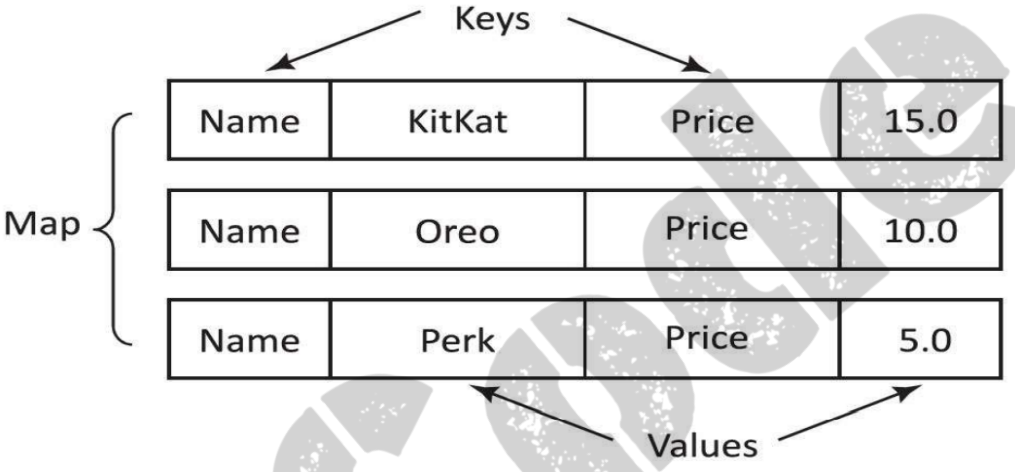


Figure: Relation and corresponding keys and their values (key-value pairs)

Pig Latin and Developing Pig Latin Scripts:

- Pig Latin enables developing the scripts for data analysis.
- A number of operators in Pig Latin help to develop their own functions for reading, writing and processing data.
- Pig Latin programs execute in the Pig run-time environment.

Pig Latin:

Statements in Pig Latin:

1. Basic constructs to process the data.
2. Include schemas and expressions.
3. End with a semicolon.

4. LOAD statement reads the data from file system, DUMP displays the result and STORE stores the result.
5. Single-line comments begin with - - and multiline begin with /* and end with */
6. Keywords (for example, LOAD, STORE, DUMP) are not case-sensitive.
7. Function names, relations, and paths are case-sensitive.

- The below figure shows the order of processing Pig statements -Load, dump and store.

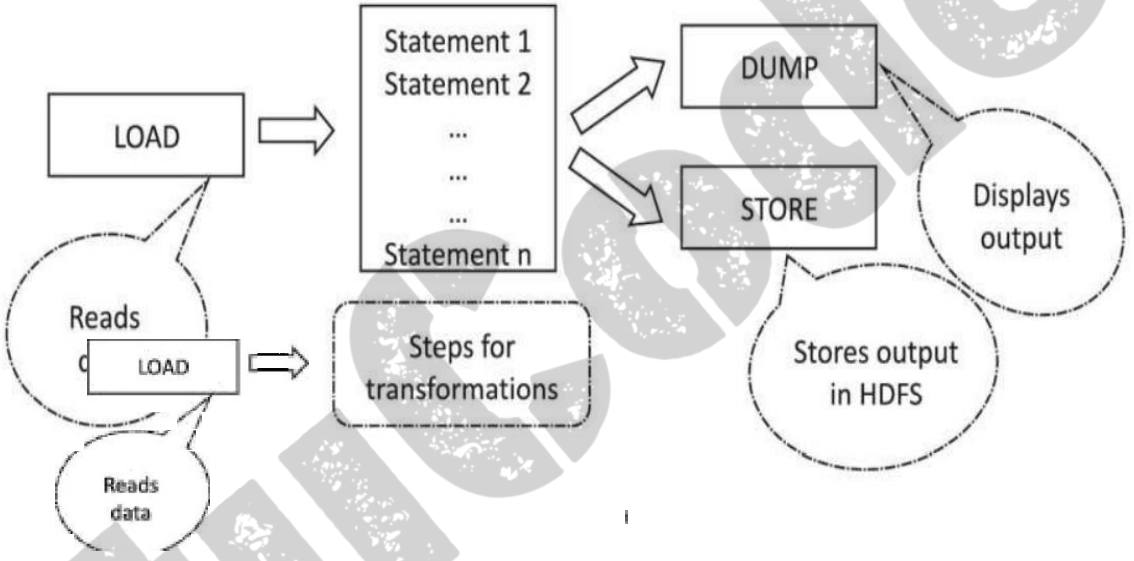


Figure: Order of processing Pig statements - Load, dump, andstore

Arithmetic Operators	+	-	*	/	%
Used for	Addition	Subtraction	Multiplication	Division	Remainder

Comparison Operators	==	!=	<	>	≤	≥
Used for	Equality	Not equal	Less than	Greater than	Less than and equal to	Greater than and equal to

Apache Pig Execution:

Pig Execution Modes:

1. Local Mode:
   - All the data files install and run from a local host using the local file system.
   - Local mode is mostly used for testing purpose.
   - COMMAND: pig -x local
2. MapReduce Mode:
   - All the data files load or process that exists in the HDFS.
   - A MapReduce job invokes in the back-end to perform a particular operation on the data that exists in the HDFS when a Pig Latin statement executes to process the data.
   - COMMAND: pig -x map reduce or pig

Pig Latin Script Execution Modes:

- Interactive Mode - Using the Grunt shell.
- Batch Mode - Writing the Pig Latin script in a single file with .pig extension.
- Embedded Mode – Defining UDFs in programming languages such as Java, and using them in the script.

Commands:

- To get the list of pig commands: pig-help;
- To get the version of pig: pig-version.
- To start the Grunt shell, write the command: pig

1. LOAD Command

- The first step to data flow is to specify the input. Load statement in Pig Latin loads the data from Pig Storage.
- To load data from HB as e: book = load 'MyBook' using HBase Storage();
- For reading the CSV file, Pig Storage takes an argument that indicates which character to use as a separator. For example, book = LOAD 'Pig Demo/Data/Input/myBook.csv' USING Pig Storage(,);
- For reading text data line by line: book=LOAD 'Pig Demo/Data/Input/myBook.txt' USING Pig Storage( ) AS (lines: char array);
- To specify the data-schema for loading: book=LOAD 'MyBook' AS (name, author, edition, publisher);

2. STORE Command

- Pig provides the store statement for writing the processed data after the processing is complete. It is the mirror image of the load statement in certain ways.
- By default, Pig stores data on HDFS in a tab-delimited file using Pig Storage:

STORE processed into'/Pig Demo/Data/Output/Processed';

- To store in HBase Storage with a using clause: STORE processed into 'processed' using HBase Storage( );

3. DUMP Command

- Pig provides a dump command to see the processed data on the screen.

This is particularly useful during debugging and prototyping sessions. It can also be useful for quick Adhoc jobs.

- The following command directs the output of the Pig script on the display screen:

  DUMP processed;

Relational Operations:

The relational operations provided at Pig Latin operate on data. They transform data using sorting, grouping, joining, projecting, and filtering. Followings are the basic relational operators:

1. **For each**

- FOREACH gives a simple way to apply transformations based on columns. It is Pig's projection operator. The table gives examples using FOREACH.

Table: Applying transformations on columns using FOREACH operator

Load an entire record, but then remove all but the name and phone fields from each record	A = load 'input' as (name: char array, rollno: long, address: char array, phone: char array, preferences: map[]); B = for each A generate name, phone;
Tuple projection using the dot operator	A=load 'input' as(t:tuple(x:int, y:int)); B = for each A generate t.x,t.$1;
Bag projection	A=load 'input' as(b:bag{t:(x:int, y:int)}); B = for each A generate b.x;
Bag projection	A=load 'input' as(b:bag{t:(x:int, y:int)}); B = for each A generate b.(x,y);
Add all integer values	A=load 'input' as (x:chararray,y:int, z:int); Al=for each A generate x,y+zasyz; B = group A1 by x; C = for each B generate SUM(Al.yz);

2. Filter

- FILTER gives a simple way to select tuples from a relation based on some specified conditions (predicate). It is Pig's select command.

Loads an entire record, then selects the tuples with marks more than 75 from each record	A = load 'input'as (name: char array, rollno: long, marks float); B = filter A by marks>75.0;
Find a name (char array) that does not match a regular expression by preceding the text without a given character string. Output is all names that do not start with.	A = load 'input' as(name : char array, rollno : long, marks : float) ; B = filter A by not name matches 'P.*';

3. Group

GROUP statement collects records with the same key. There is no direct connection between group and aggregate functions in Pig Latin, unlike SQL.

Collects all records with the same value for the provided key into a bag. Then it can pass to the aggregate function, if required, or do other things with that.	A = load 'input' as (name: char array, rollno: long, marks: float); grpd = group A by marks; B = for each grpd generate name, COUNT(A);

4. Order by

ORDER statement sorts the data based on a **specific** field value, producing a total order of output data.

The syntax of order is similar to the group. Key indicates by which the data sort.	A=load 'input' as(name: char array, rollno: long, marks: float); B = order A by name;
To sort based on two or more keys (For example, the first sort by, then sort by), indicates a set of keys by which the data sort. No parentheses around the keys when multiple keys indicate in order	A=load 'input' as (name: Char array, rollno: long, marks: float); B = order A by name marks;

5. Distinct

- DISTINCT removes duplicate tuples. It works only on entire tuples, not on individual fields:

Removes the tuples having the same name and city.	A = load 'input' city: as (name: char array, char array); B = distinct A;

6. Join

- JOIN statement joins two or more relations based on values in the common field.
- Keys indicate the inputs. When those keys are equal, two tuples are joined. Tuples for which no match is found are dropped.

Join selects tuples from one input to put together with tuples from another input.	A = load 'input1' as (name: char array, rollno: long); B = load 'input2' as (rollno: long, marks: float); C = join A by rollno, B by rollno
Also based on multiple keys join. All cases must have the	A = load 'input l' as (name: char array, father name: Char aaray, rollno: long);

same number of keys, and they must be of the same or compatible types.	B=load‘input2’as(name: char array, rollno: long, marks: float); C = join A by (name, rollno),B by (name, rollno)

7. Limit : LIMIT gets a limited number of results.

Outputs only the first five tuples from the relation.	A=load ‘input’ as (name: char array, city: char array); B=Limit A 5;

8. Sample

- SAMPLE offers to get a sample of the entire data. It reads through hall of the data but returns only a percentage of rows on a random basis.
- Thus, the results of a script with a sample will vary with every execution. The percentage it will return is expressed as a double value, between 0 and 1. For example, 0.2 indicates20%.

Outputs only 10% tuples from the relation.	A=load ‘input’ as (name: char array, city: char array); B=sample A 0.1;

9. Split

- SPLIT partitions a relation into two or more relations

Outputs A relation A splits into two relations P and Q	A = load ‘input’ as(name: char array, rollno: long,marks: float); Split A into P if marks >50.0, Q if marks <=50.0;

10. Parallel

- The PARALLEL statement is for parallel data processing.
- Any relational operator in Pig Latin can attach PARALLEL. However, it

controls only reduce-side parallelism, so it makes sense only for operators that force a reduce phase, such as group, order, distinct, join or limit.

Generating MapReduce job with 10 reducers	A=load 'input' as (name: char array, marks: float); B=group A by marks parallel 10;

11. EVAL Functions

Following are the evaluation functions:

Function Name	Description
AVG	Compute the average of the numeric values in a single- column bag.
SUM	Compute the sum of the numeric values in a single-column bag.
MAX	Get the maximum of numeric values or char arrays in a single-column bag.
MIN	Get the minimum of numeric values or char arrays in a single-column bag.
COUNT and COUNT_STAR	Count the number of tuples in a bag.
CONTACT	Concatenate two fields. The data type of the two fields must be the same, either char array or byte array.
DIFF	Compare two fields in a tuple.
IsEmpty	Check if a bag or map is empty (has no data).
SIZE	Compute the number of elements based on the data type.
TOKENIZE	Split a string and output a bag of words.