MODULE 4

MapReduce, Hive and Pig

Syllabus to Discuss

MapReduce, Hive and Pig: Introduction, MapReduce Map Tasks, Reduce Tasks and MapReduce Execution, Composing MapReduce for Calculations and Algorithms, Hive, HiveQL, 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 API's of Programs such as **MapReduce** and **Spark.**

- ✓ The application support layer includes HBase which creates column-family data store using other formats such as key-value pairs or JSON file.
- ✓ HBase stores and processes the columnar data after translating into MapReduce tasks to run in HDFS.
- ✓ The support layer also includes Hive which creates SQL-like tables. Hive stores and processes table data after translating it into MapReduce tasks to run in HDFS.
- ✓ Hive creates SQL-like tables in Hive shell. Hive uses HiveQL processes queries, ad hoc (unstructured) queries, aggregation functions and summarizing functions, such as functions to compute maximum, minimum, average of selected or grouped datasets. HiveQL is a restricted form of SQL.
- ✓ The support layer also includes Pig. Pig is a data-flow language and an execution framework.
- ✓ Pig enables the usage of relational algebra in HDFS. MapReduce is the processing framework and YARN is the resource managing framework.

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

The smallest unit of data that can be stored or retrieved from the disk is a block. HDFS deals with the data stored in blocks.

The Hadoop application is responsible for distributing the data blocks across multiple nodes. The tasks, therefore, first convert into map and reduce tasks.

This requirement arises because the mapping of stored values is very important. The number of map tasks in an application is handled by the number of blocks of input files.

Reduce task uses those values for further processing such as counting, sorting or aggregating.

Application sub-task assigned for processing needs only the outputs of reduce tasks. For example, a query needs the required response for a data store.

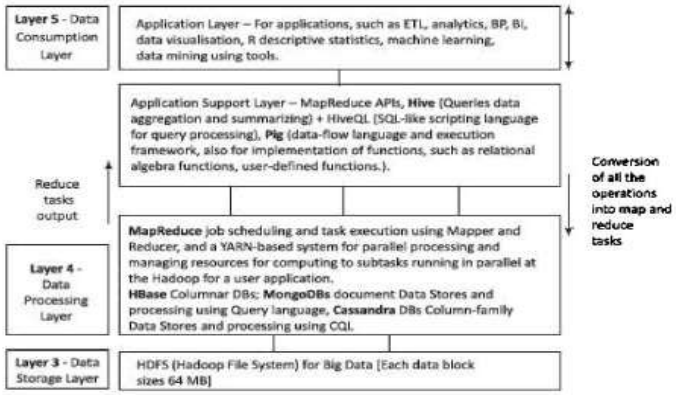


Figure 4.1 Big Data architecture design layers

MapReduce programming model refers to a programming paradigm for processing Big Data sets with a parallel and distributed environment using map and reduce tasks.

YARN refers to provisioning of running and scheduling parallel programs for map and reduce tasks and allocating parallel processing resources for computing sub-tasks running in parallel at the Hadoop for a user application.

The YARN resources management enables large-scale data analytics using multiple machines (data nodes) in the HDFS cluster.

Script refers to a small program (codes up to few thousand lines of code) in a language used for purposes such as query processing, text processing, or refers to a small code written in a dynamic high-level general-purpose language, such as Python or PERL.

SQL-like scripting language means a language for writing script that processes queries similar to SQL. SQL lets us: (i) write structured queries for processing in DBMS, (ii) create and modify schema, and control the data access, (iii) Create client for sending query scripts, and create and manage server databases, and (iv) view, query and change (update, insert or append or delete)databases.

A theorem known as CAP (Consistency, Availability and £,artitions) states that out of three properties, at least two must be present for the application/service/process.

NoSQL relies upon another model known as the BASE model. This model has three principles: Basic availability (the availability of data even in the presence of multiple failures), Soft state (data consistencyis the developer's problem and should not be handled by the database).

Eventual consistency (when no new changes occur on existing data, eventually all accesses to that data will return the last updated value).

Data-architecture patterns refer to formats used in NoSQL DBs. The examples are Key-Value Data Stores, Object Data Stores, Column family Big Data Stores, Tabular Data Stores and Document Stores.

Key-Value Data Store refers to a simplest way to implement a schema-less database. A string called key maps to values in a large data string or BLOB (basic large object)..

Object Data Store refers to a repository which stores the (i) objects (such as files, images, documents, folders and business reports), (ii) system metadata which provides information such as filename, creation_date, last_modified, language_used, access_permissions, supported Query languages, and (iii) Custom metadata which provides information such as subject, category and sharing permission.

Tabular Data Store refers to table, column-family or BigTable like Data Store.

Column family Big Data store refers to storage in logical groups of column families. The storage may be similar to columns of sparse matrix. They use a pair of row and column keys to access the column fields.

BigTable Data Store **is a popular column-family based Data Store.**

Row key,column key and timestamp uniquely identify a value. Google BigTable, HBaseand Cassandra DBs use the BigTable Data Store model.

Document Store means a NoSQL DB which stores hierarchical information in a single unit called document. Document stores data in nested hierarchies, for example in XML document object model, JSON formats data model or machine-readable data as one BLOB.

Tuple means an ordered list of elements. An n-tuple relates to set theory, a collection (sequence) of "n" elements. Tuples implement the records.

Collection means a well-defined collection of distinct objects in a set, the objects of a set are the elements. A collection may be analogous to a table of RDBMS. A collection in a database also refers to storage of a number of documents.

Aggregate refers to collection of data sets in the key value, column family or BigTable data stores which usually require sequential processing.

Aggregation function refers to a function to find counts, sum, maximum, minimum, other statistical or mathematical function using a collection of datasets, such as column or column-family.

Sequence refers to an enumerated collection of objects, (the repetitions can be there) which contain members similar to a set. Sequence length equals the number of elements (can also be infinite). Sequence should reflect an order which matters, unlike a set.

Document refers to a container for the number of collections. The container can be a unit of storing data in a database, such as MongoDB.

Natural join is where two tables join based on all common columns. Both thetables must have the same column name and the data type.

MAPREDUCE MAP TASKS, REDUCE TASKS ANDMAPREDUCE EXECUTION

Big Data Processing employs the Map Reduce Programming Model. A job means a Map Reduce Program. Each job consists of several smaller unit, called MapReduce Tasks.

A software execution framework in MapReduce programming defines the parallel tasks.

The Hadoop MapReduceimplementation usesJava framework.

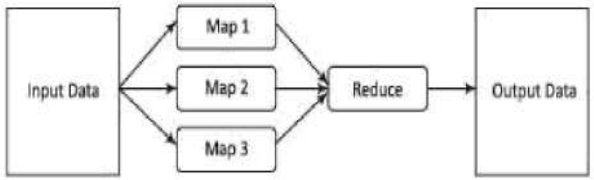


Figure 4.2 MapReduce Programming Model

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

Map takes input data set as pieces of data and maps them on various nodes for parallel processing.

The reduce task, which takes the output from the maps as an input and combines those data pieces into a smaller set of data. A reduce task always run after the map task (s).

Many real-world situations are expressible using this model.

Inner join is the default natural join. It refers to two tables that join based on common columns mentioned using the ON clause. Inner Join returns all rows from both tables if the columns match.

Node refers to a place for storing data, data block or read or write computations.

Data center in a DB refers to a collection of related nodes. Many nodes form a data center or rack.

Cluster refers to a collection of many nodes.

Keyspace means a namespace to group multiple column families, especially one per partition.

Indexing to a field means providing reference to a field in a document of collections that support the queries and operations using that index. A DB creates an index on the _id field of every collection.

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

The compute nodes and the storage nodes are the same at a cluster, that is, the MapReduce program and the HDFS are running on the same set of nodes.

.

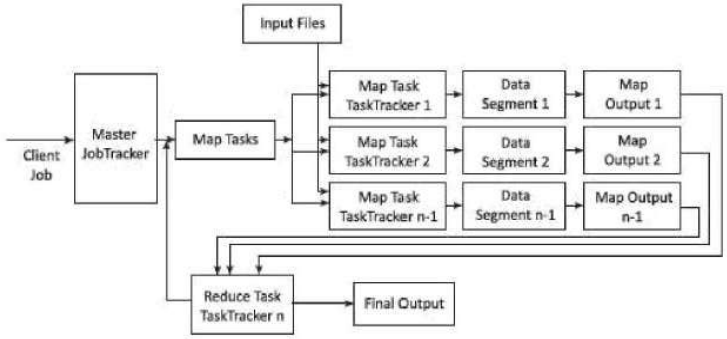


Figure 4.3 MapReduce process on client submitting a job

Figure 4.3 shows MapReduce process when a client submits a job, and the succeeding actions by the JobTracker andTaskTracker.

JobTracker and Task Tracker 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.

The MapReduce framework operates entirely on key, value-pairs. The framework views the input to the task as a set of (key, value)pairs and produces a set of (key, value) pairs as the output of the task, possiblyof different types.

Map-Tasks

Map task means a task that implements a map(), which runs user application codes for each key-value pair **(kl, vl).** Key **kl** is a set of keys. Key **kl** maps to group of data values (Section 3.3.1). Values **vl** 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. Figure 4.4 shows logical view of functioning of map().

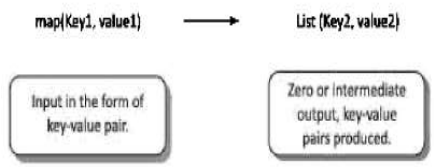


Figure 4.4 Logical view of functioning of map()

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

The Sample Code for Mapper Class

public clase SampleMapper extends Mapper<kl, Vl, k2, *v2*>

{

void map (kl key, Vl value, Context context) throwe IOException, InterruptedException

{..}

Individual Mappers do not communicate with each other.

Number of Maps The number of maps depends on the size of the input files, i.e., the total number of blocks of the input files.

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 Nmap can be explicitly set by using *setNumMapTasks(int).* Suggested numberis nearly 10-100 maps per node. Nmap can be set even higher.

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 ispassed 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().

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 readingby the Mapper.

RecordReader uses TextInputFormat by default for converting data into key-value pairs.

RecordReader communicates with the InputSplit untilthe file is read.

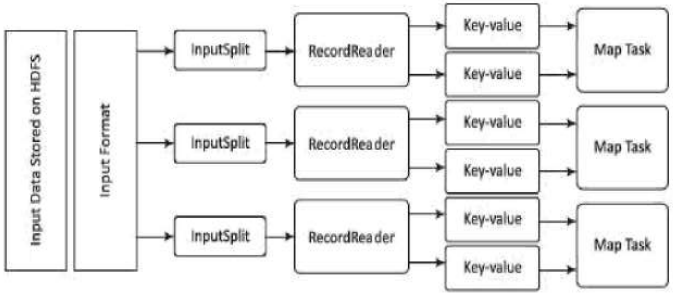


Figure 4.5 Key-value pairing in MapReduce

Figure 4.5 shows the steps in MapReduce key-value pairing.

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

All pairs with the same group key **(k2)** collect and group together, creatingone group for each key.

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.

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.
- ✓ The same codes implement the Partitioner, Combiner as well as reduce() functions.
- ✓ Functions forPartitioner and sorting functions are at the mapping node.
- ✓ The main function ofa 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 optimize MapReduce task that locally aggregates before the shuffle and sort phase.

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

The main function of a Combiner is to consolidate the map output records with the same key.

The output (key-value collection) of the combiner transfers over the network to the Reducer task as input.

This limits the volume of data transfer between map and reduce tasks, and thus reduces the cost of data transfer across the network. Combiners use grouping by key for carrying out this function.

The combiner works as follows:

- ✓ It does not have its own interface and it must implement the interface at reduce().
- ✓ It operates on each map output key. It must have the same input andoutput key-value types as the Reducer class.
- ✓ It can produce summary information from a large dataset because itreplaces the original Map output with fewer records or smaller records.

Reduced 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 ExarrpleReducer extends Reducer<k2, v2, k3, v3>

    void reduce (k2 key, Iterable<V2> values, Context context) throws
    IOBxception, InterruptedBxception

    { ... }
```

Details of Map Reduce processing Steps.

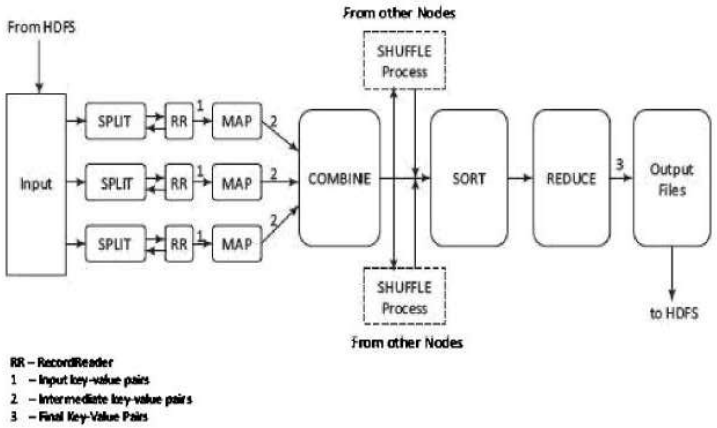


Figure 4.6 MapReduce execution steps

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

Figure 4.6 shows the execution steps, data flow, splitting, partitioning and sorting on a map node and reducer on reducer node.

Copying with Node Failure

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

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:

- ✓ 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
- ✓ Reduce TaskTracker failure:
  - Only in-progress tasks are reset to idle
- ✓ Master JobTracker failure:
  - Map-Reduce task aborts and notifies the client (in case of one master node).

COMPOSING MAPREDUCE FOR CALCULATIONS ANDALGORITHMS

MapReduce program composition in counting and summing, algorithms for relational algebraic operations, projections, unions, intersections, natural joins, grouping and aggregation, matrix multiplicationand other computations.

Composing Map-Reduce for Calculations

The calculations for various operations compose are:

Counting and Summing

- ✓ The number of alerts or messages generated during a specific maintenance activity of vehicles need counting for a month.
- ✓ From 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 thelist of ls 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 alsoin business analytics field.

Sorting

- ✓ From figure 4.6 illustrated MapReduce execution steps, i.e., dataflow, splitting, partitioning and sorting on a map node and reduce on a reducer node.
- ✓ *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 totalnumber of unique values in each field for each set of records that belongs to the same group.

Two solutions are possible:

- ✓ The *Mapper* emits the dummy counters for each pair of field and groupld,and the *Reducer* calculates the total number of occurrences for each suchpair.
- ✓ The *Mapper* emits the values and groupld, 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 arefound in data validation, log analysis and querying of datasets.
- ✓ *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.
- ✓ A set of nodes stores the data and codes at a network. Each node contains a listof neighboring 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 stateagain 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 ofitems is N, then the total number of values= N x 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 whobuy item x tend to also buy y).

If N x 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:

- ✓ 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.

- ✓ 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, mergesthem and emits the same result as in the pairs approach.

The grouping:

- ✓ Generates fewer intermediate keys. Hence, the framework has less sortingtodo.
- ✓ Greatly benefits from the use of combiners.
- ✓ In-memory accumulation possible.
- ✓ Enables complex implementations.
- ✓ Results in general, faster computations using stripes than "pairs".

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,

$c_i = \sum_{j=1}^{n} a_{ij} b_j$. An example of calculations is given below.

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

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

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

Algorithm for using MapReduce: The Mapper operates on A and emits row-wise multiplication of each matrix element and vector element (aij x bj 'V i). The Reducer executes sum() for summing all values associated with each i and emits the element ci. Application of the algorithm is found in linear transformation.

Relational – Algebra Operations

Selection

Consider the attributenames (ACVM_ID, Date, chocolate_flavour, daily_sales). Consider relation

R = {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72), (525 2122017, KitKat, 82), (525, 12122017, Oreo, 72), (526, 12122017, KitKat, 82), (526, 12122017, Oreo, 72)}.

Selection *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 *chocolate_flavour= Oreo* selects the subset {(524, 12122017, Oreo, 72), (525, 12122017, Oreo, 72), (526, 12122017, Oreo, 72)}.

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.

Projection

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

Consider relation R = {(524, 12122017, KitKat, 82), (524, 12122017, Oreo, 72)}. Projection II *ACVM_m (R)* selects the subset {(524)}.

Projection, II *chocolate_flavour, o.s* daily_sales* selects the subset {(KitKat, 0.5x82), (Oreo, 0.5 X 72)}

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

Union

Consider,

Rl = {(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 Rl and R3 is:

Rl 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.

Intersection

Consider, Rl = {(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

Rl n R3 = {(12122011, 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:

Rl = {(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_l is Rl - R3 = (12122017, Oreo, 72) and

set_2 is R3 - Rl = (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 to set_l.

Symmetric Difference

Symmetric difference (notation is *A fl. B* (or *A* e B)] is another relational entity. It means the set of elements in exactly one of the two relations A or B. R3 e Rl =

(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 this tuple belongs to).

The *Reducer* transfers only tuples that belong to neither set_l or set_2.

Natural Join

Consider two relations Rl and R2 for tuples a, band 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, (Rl, a)) for each tuple (a, b) of Rl, 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 Rl and the other with first component R2, say (Rl, 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 (Rl, a) and (R2, c) are present in the input list of values.

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().

Assume *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)}.

Chocolate_flavour i count ACVM_ID, sum (daily_sales (chocolate_flavour))

will give the output (524, KitKat, sale_month), (525, KitKat, sale_month), ... and (524, Oreo, sale_month), (525, Oreo, sale_month), ... for all **ACVM_IDs.**

The *Mapper* finds the values from each tuple for grouping and aggregates them. The *Reducer* receives the already grouped values in input for aggregation.

Matrix Multiplication

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

$$A = \begin{matrix} a_{11} & a_{12} & \cdot\cdot & a_{1j} \\ a_{21} & a_{22} & \cdot\cdot & a_{2j} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{i1} & a_{i2} & \cdot\cdot & a_{ij} \end{matrix} \quad B = \begin{matrix} b_{11} & b_{12} & \cdot & b_{1k} \\ b_{21} & b_{22} & \cdot & b_{2k} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ b_{j1} & b_{i2} & \cdot & b_{jk} \end{matrix} \quad C = \begin{matrix} c_{11} & c_{12} & \cdot & c_{1k} \\ c_{21} & c_{22} & \cdot & c_{2k} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ c_{i1} & c_{i2} & \cdot & c_{ik} \end{matrix}$$

A.B = C; Each element evaluates as follow:

$$C_{ik} = \text{Sum}\,(a_{ij} \times b_{jk})_{j=1 \text{ to } j}.\ v_a = a_{ij} \text{ and } v_b = b_{jk}$$

First Row of C

- ✓ C first column element= ($a_{11}b_{11}$+ $a_{12}b_{21}$ +.... + a_{1j}'b_{1}).
- ✓ Second column element= ($a_{11}b_{12}$+ $a_{12}b_{22}$+ + a_{1j}•b_{jz}),
- ✓ The kth column element= ($a_{11}b_{1}$ + $a_{12}b_{2}$ +.... + a_{1j} b_{jk}).

Second row of C

- ✓ C first column element = ($a_{21}b_{11}$+ $a_{22}b_{21}$ +.... + a_{2j} b_{1}).

- ✓ Second column element = (a21b12+ a22b22+ + azj•bjz),
- ✓ The kth column element= ($a_{21}b_{1k}$ + $a_{22}b_{2k}$ + + azj· bjk).

The ith row of C

- ✓ C first column element = ($a_{i1}b_{11}$+ $a_{i2}b_{21}$ +.... + $a_{ij}.b_{j1}$).
- ✓ Second column element = ($a_{i1}b_{12}$+ $a_{i2}b_{22}$+.... + $a_{ij}.b_{j2}$).
- ✓ The kth column element= ($a_{i1}b_{1k}$ + $a_{i2}b_{2k}$ + ……..+aij.bjk).

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.

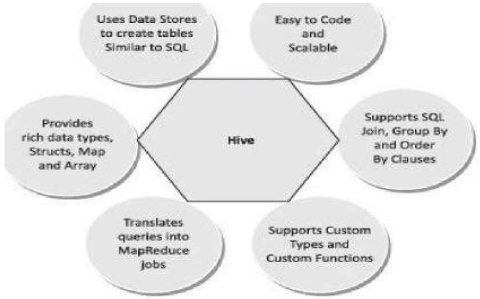


Figure 4.9 Main features of Hive

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-browserclients, can access the Hive DB server.
- ✓ Provides an SQL dialect (Hive Query Language, abbreviated HiveQL or HQL).

Results of HiveQL Query and the data load in the tables which store at the Hadoop cluster at HDFS.

Limitations of Hive is:

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

HIVE ARCHITECTURE

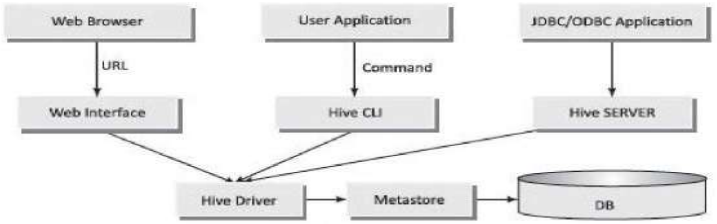


Figure 4.10 Hive architecture

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 toexecute HiveQL statements.
- ✓ **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.
- ✓ **Web Interface** - 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, 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.

Comparison with RDBMS

Hive is a DB system which defines databases and tables. Hive analyzes structured data in DB. Hive has certain differences 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

Hive Data Types and File Formats

Hive defines various primitive, complex, string, date/time, collection data typesand file formats for handling and storing different data formats. The following Table gives primitive, string, date/time and complex Hive data types and their descriptions.

Data TypeName	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 0ava .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

The following Table gives Hive three Collection data types and their descriptions.

File Format	Description
Text file	The default file format, and a line represents a record. The delimiting characters separate the lines. Text file examples are CSV, TSV,JSON and XML(Section 3.3.2).
Sequentialfile	Flat file which stores binary key-value pairs, and supports compression.
RCFile	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).

HIVE Data Model

Name	Description
Database	Namespace for tables
Tables	Similar to tables **in** RDBMS Support filter, projection, join and union operationsThe 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 columnin the table.Stored as a file in the partition directory.

Hive Integration and Workflow Steps

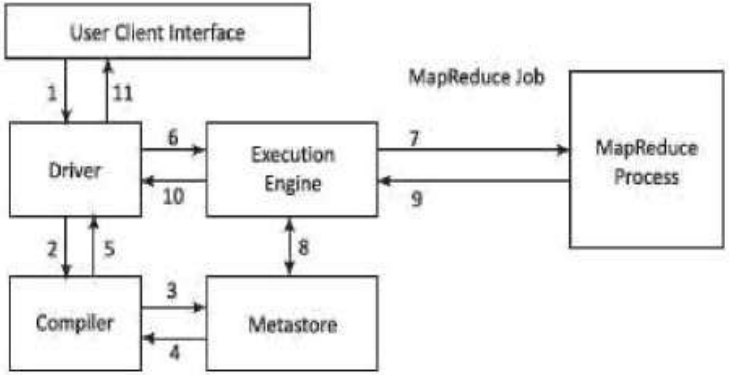


Figure 4.11 Dataflow sequences and workflow steps

The workflow steps are as follows :

Execute Query: Hive interface (CLI or Web Interface) sends a query to DatabaseDriver to execute the query.
Get Plan: Driver sends the query to query compiler that parses the query to checkthe syntax and query plan or the requirement of the query.

Get Metadata: Compiler sends metadata request to Metastore (of any database, such as MySQL).
Send Metadata: Metastore sends metadata as a response to compiler.
Send Plan: Compiler checks the requirement and resends the plan to driver. The parsing and compiling of the query is complete at this place.
Execute Plan: Driver sends the execute plan to execution engine.
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.
Metadata Operations: Meanwhile the execution engine can execute the metadata operations with Metastore.
Fetch Result: Execution engine receives the results from Data nodes.
Send Results: Execution engine sends the result to Dr iver.
Send Results: Driver sends the results to Hive Interfaces.

Hive Built-in Functions

Return Type	Syntax	Description
BIGINT	round(doublea)	Returns the rounded BIGINT (8 Byte integer) value of the 8 Byte double-precision floating point number a
BIGINT	floor(doublea)	Returns the maximum BIGINT value that is equal to or less thanthe 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 O to 1 and that changes in each row. Integer seed ensuredthat random number sequence is deterministic.
string	concate(string strl, string str2, ...)	Returns the string resulting from concatenating strl 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 ofstr to upper case.
string	lower(string str), lcase(stringstr)	Returns the string resulting from converting all characters ofstr to lower case.
string	trim(stringstr)	Returns the string resulting from trimming spaces from both ends. trim ('12A34 56') returns '12A3456'

string	ltrim(string str); rtrim(stringstr)	Returns the string resulting from trimming spaces (only one end, left or right hand side or right-handside spaces trimmed). ltrim('12A34 56') returns '12A3456' and rtrim(' 12A34 56 ')returns '12A3456'.
string	rtrim(stringstr)	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.

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 supports a large base of SQL users who are acquainted with SQL to extract information from data warehouses.

HiveQL Process Engine	HiveQL is similar to SQL for querying on schema information at the Metastore. It is one of the replacements of traditional approach for MapReduce program . Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.
Execution Engine	The bridge between HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results same as MapReduce results. It uses the flavor of MapReduce.

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 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>]

- ✓ A command is

CREATE DATABASEISCHEMA [IF NOT EXISTS] <database name>;

IF NOT EXISTS is an optional clause. The clause notifies the user that a database with the same name already exists. SCHEMA can be also created in place of DATABASE using this command

A command is written to get the list of all existing databases. SHOW DATABASES;

A command is written to delete an existing database.

DROP (DATABASEISCHEMA) [RESTRICT I CASCADE]; [IF EXISTS] <database name>

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 ...)]

LOCAL is an identifier to specify the local path. It is optional. OVERWRITE is optional to overwrite the data in the table. PARTITION is optional. vall is value assigned to partition column 1 (partcoll) and val2 is value assigned to partition column 2 (partcol2).

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, *having*

presence of an entity or selecting specific entity using GroupBy .

SELECT [ALL I DISTINCT] <select expression>, <selectexpression>, ...

FROM <table name>

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

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

[LIMIT number];

PIG

- ✓ It is an abstract over MapReduce
- ✓ It 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

- ✓ Analyzing large datasets
- ✓ Executing tasks involving adhoc processing
- ✓ Processing large data sources such as web logs and streaming online data
- ✓ Data processing for search platforms. Pig processes different types of data
- ✓ Processing time sensitive data loads; data extracts and analyzes quickly.

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 requireexposure to Java language
Uses multi-query approach, therebyreducing 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, bagsand maps	No such data types

Differences between Pig and SQL

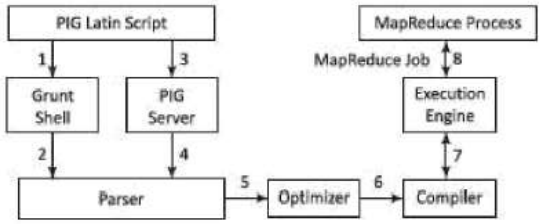
Pig	SQL
Pig Latin is a procedural language	A declarative language
Schema is optional, stores data without assigning a schema	Schema is mandatory
Nested relational data model	Flat relational data model
Provides limited opportunity for Query optimization	More opportunity for query optimization

Pig and Hive codes, both create MapReduce jobs when execute. Hive in some cases, operates on HDFS in a similar way Apache Pig does.

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 processinglanguage
Pig Latin is a procedural language and it fits in pipelineparadigm	HiveQL is a declarative language
Handles structured, unstructured and semi-structureddata	Mostly used for structured data

Pig Architecture



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).

Acylic 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 dataflows.

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:

PushUpFilter: 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.

PushDownf or EachFlatten: 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.

ColumnPruner: Omitts 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.

MapKeyPruner: Omitts never used map keys, reducing the size of the record.

Limit Optimizer: If the limit operator is immediately applied after *load* or *sort* operator, Pig converts the load or sort into a limit-sensitive implementation, which does not require processing the whole dataset. Applying the limit earlier reduces the number of records.

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> sh ls

Pig Latin Data Model

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.

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#l]
int	Signed 32-bit integer	10
long	Signed 64-bit integer	lOL 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
chararray	Char [], Character array	data analytics
bytearray	BLOB (Byte array)	ffoo

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

- ✓ Basic constructs to process the data.

- ✓ Include schemas and expressions.
- ✓ End with a semicolon.
- ✓ LOAD statement reads the data from file system, DUMP displays the resultand STORE stores the result.
- ✓ Single line comments begin with - - and multiline begin with/* and end with*/
- ✓ Keywords (for example, LOAD, STORE, DUMP) are not case-sensitive. Function names, relations and paths are case-sensitive.

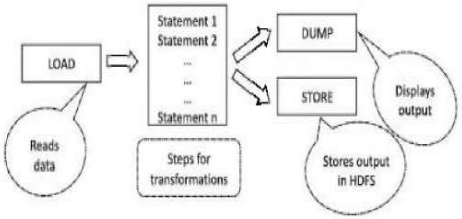


Figure 4.15 Order of processing Pig statements—Load, dump, and store

Apache Pig Execution

Pig Execution Modes 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
```

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 thedata.

COMMAND:

```
pig -x mapreduce 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*

LOAD Command The first step to a dataflow is to specify the input.

Load statement in Pig Latin loads the data from PigStorage.

To load data from HBase: book load 'MyBook' using HBaseStorage();

For reading CSV file, PigStorage **takes an argument which indicates which character to use as a separator.**

For example,

book = LOAD 'PigDemo/Data/Input/myBook.csv' USING PigStorage (,);

To specify the data-schema for loading: book = LOAD 'MyBook' AS (name, author, edition, publisher);

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 PigStorage:

STORE processed into '/PigDemo/Data/Output/Processed';

To store in HBaseStorage with a *using* clause: STORE processed into 'processed' using HBaseStorage();

To store data as comma-separated text data, PigStorage takes an argument to indicate which character to use as a separator: STORE processed into 'processed' using PigStorage(',');

Dump Command Pig provides 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:

Foreach FOREACH gives a simple way to apply transformations based on columns. It is Pig's projection operator.