UNIT IV

Frameworks and Applications: Frameworks: Applications on Big Data Using Pig and Hive, Data processing operators in Pig, Hive services, HiveQL, Querying Data in Hive, fundamentals of HBase and ZooKeeper.

Frameworks: Applications on Big Data Using Pig and Hive, Data processing operators in Pig:

1. Write the difference between SQL and HiveQL? **[7M-R20-SET-2-July 2023] [Remembering]**
2. What are the merits and demerits of Pig . **[7M-R20-SET-2-July 2023] [Remembering]**
3. Explain about the Data processing operators in Pig?

   [7M-R20-SET-4-July 2023] [Understanding]
4. What are the applications of Pig and HIVE on big data?

   [7M-R20-SET-4-July 2023] [Remembering]

Hive services, HiveQL, Querying Data in Hive, fundamentals of HBase and ZooKeeper:

5. Explain about HBase Architecture and it's Use Cases, Components & Data Model.

   [7M-R20-SET-2-July 2023] [Understanding]
6. Explain the Zookeeper workflows. **[7M-R20-SET-2-July 2023] [Understanding]**
7. Explain the creating, dropping and altering databases using Apache Hive.

   [7M-R20-SET-3-July 2023] [Understanding]
8. Explain the Hive services HiveQL, Querying Data in Hive.

   [7M-R20-SET-4-July 2023] [Understanding]
9. Describe in brief about PIG Commands? **[7M-R20-SET-4-July 2023] [Create]**
10. Explain about HBase and ZooKeeper. **[Understanding]**
11. Explain about in briefly HIVE QL. **[Understanding]**

What is Apache Pig?

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with **Hadoop**; we can perform all the data manipulation operations in Hadoop using Apache Pig.

To write data analysis programs, Pig provides a high-level language known as **Pig Latin**. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.

To analyze data using **Apache Pig,** programmers need to write scripts using Pig Latin language. All these scripts are internally converted to Map and Reduce tasks. Apache Pig has a component known as **Pig Engine** that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

WhyDoWeNeedApachePig?

Programmers who are not so good at Java normally used to struggle working with Hadoop,especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

Using **Pig Latin**, programmers can perform MapReduce tasks easily without having to type complex codes in Java.

Apache Pig uses **multi-query approach**, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.

Pig Latin is **SQL-like language** and it is easy to learn Apache Pig when you are familiar with SQL.

Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

Features of Pig

Apache Pig comes with the following features:

Rich set of operators: It provides many operators to perform operations like join, sort, filer, etc.

Ease of programming: Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.

Optimization opportunities: The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of thelanguage.

Extensibility: Using the existing operators, users can develop their own functions to read, process, and write data.

UDF's: Pig provides the facility to create **User-defined Functions** in other programming languages such as Java and invoke or embed them in Pig Scripts.

Handles all kinds of data: Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

Apache Pig Vs MapReduce

Apache Pig	MapReduce
Apache Pig is a data flow language.	MapReduce is a data processing paradigm.
It is a high level language.	MapReduce is low level and rigid.
Performing a Join operation in Apache Pig is pretty simple.	It is quite difficult in MapReduce to performa Join operation between datasets.
Any novice programmer with a basic knowledge of SQL can work conveniently with Apache Pig.	Exposure to Java is must to work with MapReduce.

Apache Pig uses multi-query approach, thereby reducing the length of the codes to a great extent.	MapReduce will require almost 20 times more the number of lines to perform the same task.
There is no need for compilation. On execution, every Apache Pig operator is converted internally into a MapReduce job.	MapReduce jobs have a long compilation process.

Apache Pig Vs SQL

Pig	SQL
Pig Latin is a **procedural** language.	SQL is a **declarative** language.
In Apache Pig, **schema** is optional. We can store data without designing a schema (values are stored as $01, $02 etc.)	Schema is mandatory in SQL.
The data model in Apache Pig is **nested relational**.	The data model used in SQL **is flatrelational**.
Apache Pig provides limited opportunity for **Query optimization**.	There is more opportunity for queryoptimization in SQL.

In addition to above differences, Apache Pig Latin;

- Allows splits in the pipeline.
- Allows developers to store data anywhere in the pipeline.
- Declares execution plans.
- Provides operators to perform ETL (Extract, Transform, and Load) functions.

Apache Pig Vs Hive

Both Apache Pig and Hive are used to create MapReduce jobs. And in some cases, Hive operates on HDFS in a similar way Apache Pig does. In the following table, we have listed a few significant points that set Apache Pig apart from Hive.

Apache Pig	Hive

Apache Pig uses a language called **PigLatin**. It was originally created at **Yahoo**.	Hive uses a language called **HiveQL**. It was originally created at **Facebook**.
Pig Latin is a data flow language.	HiveQL is a query processing language.
Pig Latin is a procedural language and it fitsin pipeline paradigm.	HiveQL is a declarative language.
Apache Pig can handle structured, unstructured, and semi-structured data.	Hive is mostly for structured data.

Applications of Apache Pig

Apache Pig is generally used by data scientists for performing tasks involving ad-hoc processing and quick prototyping. Apache Pig is used;

1.To process huge data sources such as web logs.
2.To perform data processing for search platforms.
3.To process time sensitive data loads.
4.It provides a rich set of operators to perform different operations, such as sort, joins, filter, etc.
5.Apache Pig is considered to be a boon for SQL programmers as it is easy to learn, read, and write.
6. Making user-defined functions and processes is easy
7. Fewer lines of code are required for any process or function
8. Allows the users to perform analysis of both unstructured and structured data
9.Join and Split operations are pretty easy to perform

Apache Pig Architecture

To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded).

After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, itmakes the programmer's job easy. The architecture of Apache Pig is shown below.

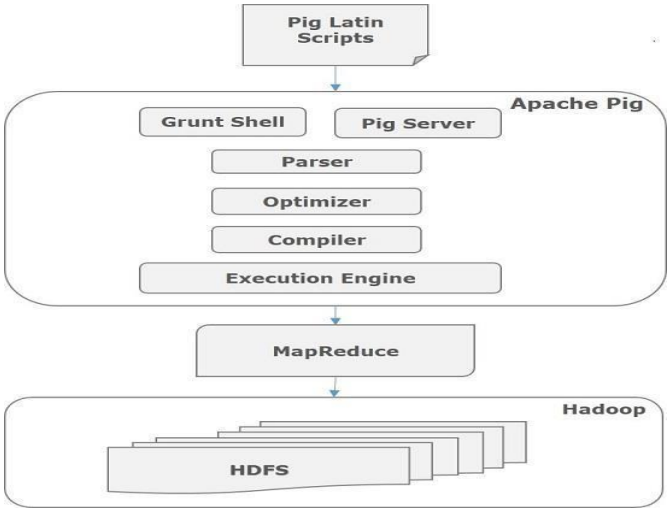


Figure: Apache Pig Architecture

Apache Pig – Components

As shown in the figure, there are various components in the Apache Pig framework. Let us take a look at the major components.

Parser

Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.

In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

Optimizer

The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.

Compiler

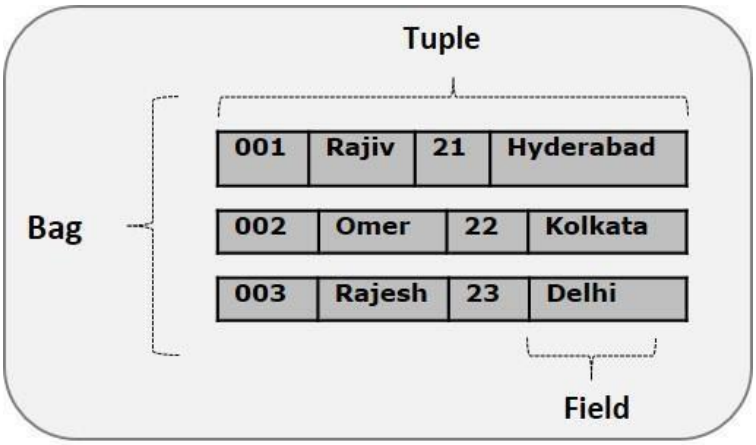
The compiler compiles the optimized logical plan into a series of MapReduce jobs.

Execution engine

Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

Pig Latin – Data Model

The data model of Pig Latin is fully nested and it allows complex non-atomic data types such as **map** and **tuple**. Given below is the diagrammatical representation of Pig Latin's data model.



Atom

Any single value in Pig Latin, irrespective of their data, type is known as an **Atom**. It is stored as string and can be used as string and number. int, long, float, double, chararray, and bytearray are the atomic values of Pig.

A piece of data or a simple atomic value is known as a **field**.

Example: 'raja' or '30'

Tuple

A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type. A tuple is similar to a row in a table of RDBMS.

Example: (Raja, 30)

Bag

A bag is an unordered set of tuples. In other words, a collection of tuples (non-unique) is known as a bag. Each tuple can have any number of fields (flexible schema). A bag is represented by '{}'. It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contain the same number of fields or that the fields in thesame position (column) have the same type.

Example: {(Raja, 30), (Mohammad, 45)}

A bag can be a field in a relation; in that context, it is known as **inner bag**. **Example:** {Raja, 30, **{9848022338, raja@gmail.com,}**}

Relation

A relation is a bag of tuples. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

Map

A map (or data map) is a set of key-value pairs. The **key** needs to be of type chararrayand should be unique. The **value** might be of any type. It is represented by '[]'

Example: [name#Raja, age#30]

Applications of Pig:

1. For exploring large datasets Pig Scripting is used.
2. Provides supports across large data sets for Ad-hoc queries.
3. In the prototyping of large data-sets processing algorithms.
4. Required to process the time-sensitive data loads.
5. For collecting large amounts of datasets in form of search logs and web crawls.
6. Used where the analytical insights are needed using the sampling.

What is Hive

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

Hive is not

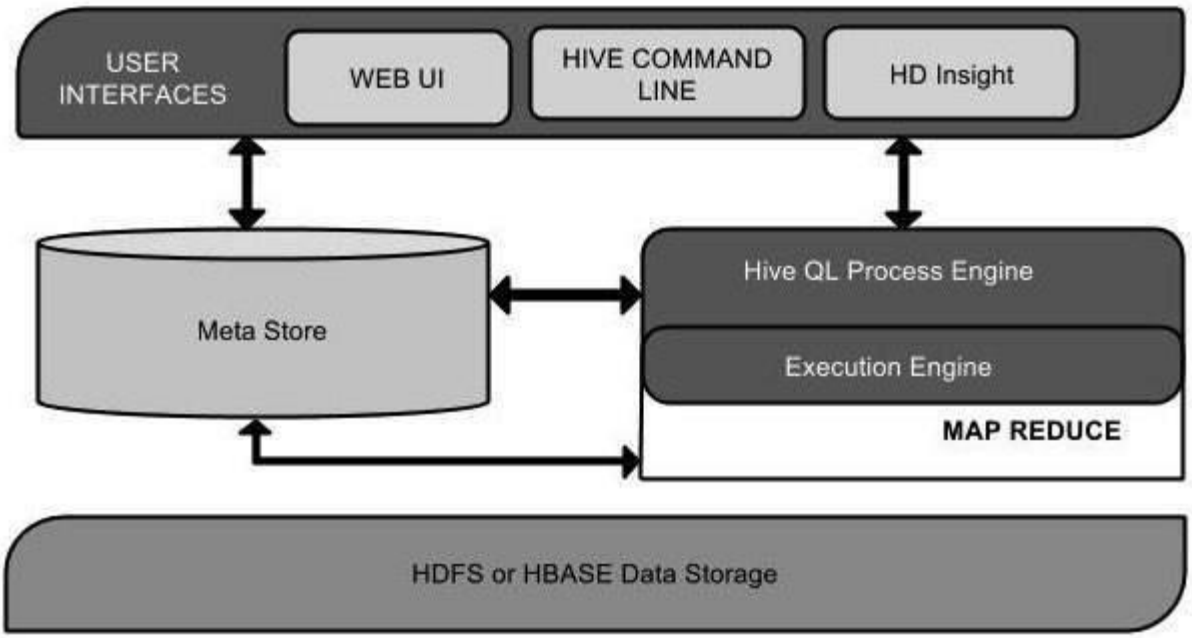
- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

Features of Hive

- It stores schema in a database and processed data into HDFS.
- It is designed for OLAP.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

Architecture of Hive

The following component diagram depicts the architecture of Hive:

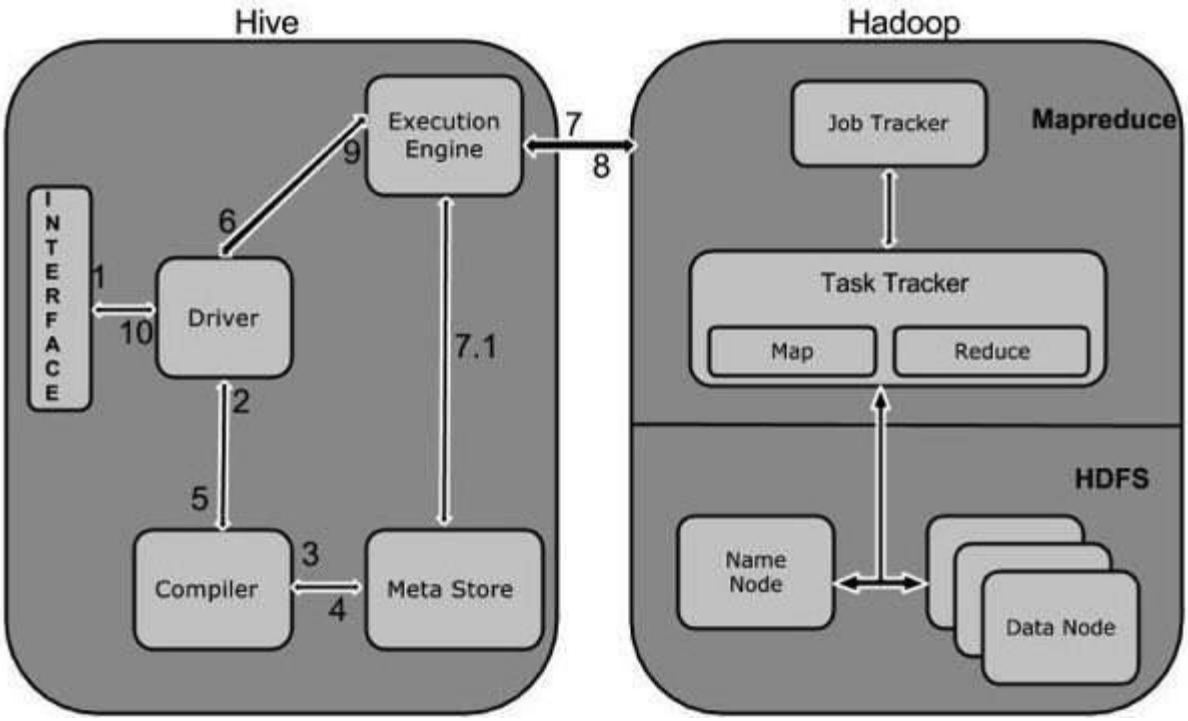


This component diagram contains different units. The following table describes each unit:

Unit Name	Operation
User Interface	Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).
Meta Store	Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.
HiveQL Process Engine	HiveQL is similar to SQL for querying on schema info on 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 conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.
HDFS or HBASE	Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

Working of Hive

The following diagram depicts the workflow between Hive and Hadoop.



The following table defines how Hive interacts with Hadoop framework:

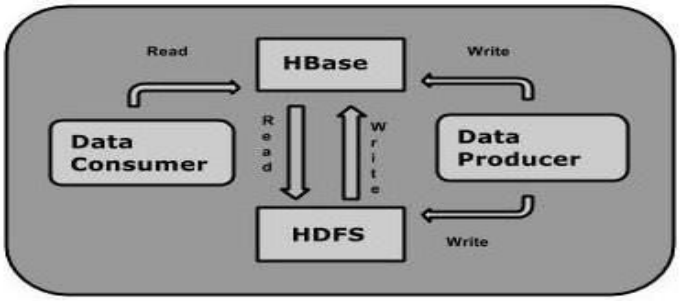
Step No.	Operation
1	**Execute Query** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2	**Get Plan** The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3	**Get Metadata** The compiler sends metadata request to Metastore (any database).
4	**Send Metadata** Metastore sends metadata as a response to the compiler.
5	**Send Plan** The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.

6	**Execute Plan** The driver sends the execute plan to the 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. Here, the query executes MapReduce job.
7.1	**Metadata Ops** Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
8	**Fetch Result** The execution engine receives the results from Data nodes.
9	**Send Results** The execution engine sends those resultant values to the driver.
10	**Send Results** The driver sends the results to Hive Interfaces.

What is HBase?

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable. HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).

It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System. One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.



HBase and HDFS

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

Storage Mechanism in HBase

HBase is a **column-oriented database** and the tables in it are sorted by row. The table schema defines only column families, which are the key value pairs. A table have multiple column families and each column family can have any number of columns. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an HBase:

- Table is a collection of rows.
- Row is a collection of column families.
- Column family is a collection of columns.
- Column is a collection of key value pairs.

Rowid	Column Family			Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3	col1	col2	col3
1												
2												
3												

- Given below is an example schema of table in HBase.\

Column Oriented and Row Oriented

Column-oriented databases are those that store data tables as sections of columns of data, rather than as rows of data. Shortly, they will have column families.

Row-Oriented Database	Column-Oriented Database
It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for small number of rows and columns.	Column-oriented databases are designed for huge tables.

The following image shows column families in a column-oriented database:

Hive Services

The Hive shell is only one of several services that you can run using the hive command. You can specify the service to run using the --service option. Type hive --service help to get a list of available service names;

some of the most useful ones are described in the following list:

cli

The command-line interface to Hive (the shell). This is the default service.

hiveserver2

Runs Hive as a server exposing a Thrift service, enabling access from a range of clients written in different languages. HiveServer 2 improves on the original Hive- Server by supporting authentication and multiuser concurrency

Thrift is a protocol for the application which were developed in a different programming languages to communicate.

HiveServer is an optional service that allows a remote client to submit requests to Hive, using a variety of programming languages, and retrieve results.

Applications using the Thrift, JDBC, and ODBC connectors need to run a Hive server to communicate with Hive. Set the hive.server2.thrift.port configuration property to specify the port the server will listen on (defaults to 10000).

Beeline

A command-line interface to Hive that works in embedded mode (like the regular CLI), or by connecting to a HiveServer 2 process using JDBC.

hwi

The Hive Web Interface. A simple web interface that can be used as an alternative to the CLI without having to install any client software.

jar

The Hive equivalent of hadoop jar, a convenient way to run Java applications that includes both Hadoop and Hive classes on the classpath

Metastore

By default, the metastore is run in the same process as the Hive service. Using this service, it is possible to run the metastore as a standalone (remote) process. Set the METASTORE_PORT environment variable (or use the -p command-line option) to specify the port the server will listen on (defaults to 9083).

Hive clients

If you run Hive as a server (hive --service hiveserver2), there are a number of different mechanisms for connecting to it from applications (the relationship between Hive clients and Hive services is illustrated in Figure 17-1):

Thrift Client

The Hive server is exposed as a Thrift service, so it's possible to interact with it using any programming language that supports Thrift.

JDBC driver

Hive provides a Type 4 (pure Java) JDBC driver, defined in the class **org.apache.hadoop.hive.jdbc.HiveDriver**. When configured with a JDBC URI of the form **jdbc:hive2://host:port/dbname**, a Java application will connect to a Hive server running in a separate process at the given host and port.

ODBC driver

An ODBC driver allows applications that support the ODBC protocol (such as business intelligence software) to connect to Hive. The Apache Hive distribution does not ship with an ODBC driver, but several vendors make one freely available.

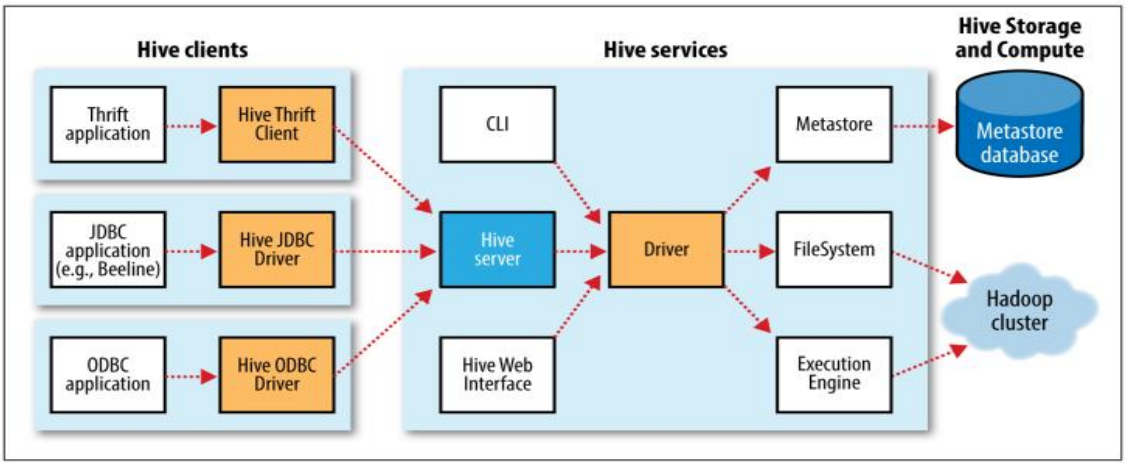


Figure 17-1. Hive architecture

HiveQL

Hive Query Language (HiveQL) is a query language in Apache Hive for processing and analyzing structured data. It separates users from the complexity of Map Reduce programming. It reuses common concepts from relational databases, such as tables, rows, columns, and schema, to ease learning.

Hive's SQL dialect, called HiveQL, is a mixture of SQL-92, MySQL, and Oracle's SQL dialect. Some of Hive's nonstandard extensions to SQL were inspired by MapReduce, such as multitable inserts (see "Multitable insert" on page 501) and the TRANSFORM, MAP, and REDUCE clauses (see "MapReduce Scripts" on page 503).

Most interactions tend to take place over a command line interface (CLI). Generally, HiveQL syntax is similar to the SQL syntax that most data analysts are familiar with. Hive supports four file formats which are: TEXTFILE, SEQUENCEFILE, ORC and RCFILE (Record Columnar File).

Hive provides a CLI for Hive query writing using Hive Query Language (HiveQL). Below are the main types of Built-in Operators in HiveQL:

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Operators on Complex types
- Complex type Constructors

A high-level comparison of SQL and HiveQL.

Table 17-2. A high-level comparison of SQL and HiveQL

Feature	SQL	HiveQL	References
Updates	UPDATE, INSERT, DELETE	UPDATE, INSERT, DELETE	"Inserts" on page 500; "Updates, Transactions, and Indexes" on page 483
Transactions	Supported	Limited support	
Indexes	Supported	Supported	
Data types	Integral, floating-point, fixed-point, text and binary strings, temporal	Boolean, integral, floating-point, fixed-point, text and binary strings, temporal, array, map, struct	"Data Types" on page 486
Functions	Hundreds of built-in functions	Hundreds of built-in functions	"Operators and Functions" on page 488
Multitable inserts	Not supported	Supported	"Multitable insert" on page 501
CREATE TABLE...AS SELECT	Not valid SQL-92, but found in some databases	Supported	"CREATE TABLE...AS SELECT" on page 501
SELECT	SQL-92	SQL-92. SORT BY for partial ordering, LIMIT to limit number of rows returned	"Querying Data" on page 503
Joins	SQL-92, or variants (join tables in the FROM clause, join condition in the WHERE clause)	Inner joins, outer joins, semi joins, map joins, cross joins	"Joins" on page 505
Subqueries	In any clause (correlated or noncorrelated)	In the FROM, WHERE, or HAVING clauses (uncorrelated subqueries not supported)	"Subqueries" on page 508
Views	Updatable (materialized or nonmaterialized)	Read-only (materialized views not supported)	"Views" on page 509

Importing Data

LOAD DATA operation to import data into a Hive table (or partition) by copying or moving files to the table's directory. table with data from another Hive table using an INSERT statement, or at creation time using the CTAS construct, which is an abbreviation used to refer to CREATE TABLE...AS SELECT.

Inserts

Here's an example of an INSERT statement:

```
INSERT OVERWRITE TABLE target
SELECT col1, col2
  FROM source;
```

For partitioned tables, you can specify the partition to insert into by supplying a PARTITION clause:

```
INSERT OVERWRITE TABLE target
PARTITION (dt='2001-01-01')
SELECT col1, col2
  FROM source;
```

The OVERWRITE keyword means that the contents of the target table (for the first ex- ample) or the 2001-01-01 partition (for the second example) are replaced by the results of the SELECT statement.

CREATE TABLE...AS SELECT

The new table's column definitions are derived from the columns retrieved by the SELECT clause. In the following query, the target table has two columns named col1 and col2 whose types are the same as the ones in the source table:

```
CREATE TABLE target
AS
SELECT col1, col2
FROM source;
```

Altering Tables

it is up to you to ensure that the data is changed to reflect the new structure.

You can rename a table using the ALTER TABLE statement:

```
ALTER TABLE source RENAME TO target;
```

In addition to updating the table metadata, ALTER TABLE moves the underlying table directory so that it reflects the new name. In the current example, /user/hive/warehouse/ source is renamed to /user/hive/warehouse/target.

Hive allows you to change the definition for columns, add new columns, or even replace all existing columns in a table with a new set. For example, consider adding a new column:

```
ALTER TABLE target ADD COLUMNS (col3 STRING);
```

Dropping Tables

The DROP TABLE statement deletes the data and metadata for a table. In the case of external tables, only the metadata is deleted; the data is left untouched.

If you want to delete all the data in a table but keep the table definition, use TRUNCATE TABLE. For example:

```
TRUNCATE TABLE my_table;
```

if you want to create a new, empty table with the same schema as another table, then use the LIKE keyword:

```
CREATE TABLE new_table LIKE existing_table;
```

Data processing operators in Pig

Loading and Storing

Data Throughout this chapter, we have seen how to load data from external storage for pro-cessing in Pig.

Filtering Data

Once you have some data loaded into a relation, often the next step is to filter it to remove the data that you are not interested in. By filtering early in the processing pipe- line, you minimize the amount of data flowing through the system, which can improve efficiency

FOREACH...GENERATE

The FOREACH...GENERATE operator is used to act on every row in a relation. It can be used to remove fields or to generate new ones. In this example, we do both:

grunt> DUMP A;
(Joe,cherry,2)
(Ali,apple,3)
(Joe,banana,2)
(Eve,apple,7)
grunt> B = FOREACH A GENERATE $0, $2+1, 'Constant';
grunt> DUMP B;
(Joe,3,Constant)
(Ali,4,Constant)
(Joe,3,Constant)
(Eve,8,Constant)
Here we have created a new relation, B, with three fields. Its first field is a projection of the first field ($0) of A. B's second field is the third field of A ($2) with 1 added to it. B's third field is a constant field (every row in B has the same third field) with the charar ray value Constant.

STREAM
can use built-in commands with arguments. Here is an example that uses the Unix cut command to extract the second field of each tuple in A. Note that the command and its arguments are enclosed in backticks

grunt> C = STREAM A THROUGH `cut -f 2`;
grunt> DUMP C;
(cherry)

(apple)
(banana)
(apple)

The STREAM operator uses PigStorage to serialize and deserialize relations to and from the program's standard input and output streams.

Grouping and Joining Data
the large datasets that are suitable for analysis by Pig (and MapReduce in general) are not normalized, however, joins are used more infrequently in Pig than they are in SQL

JOIN
Let's look at an example of an inner join. Consider the relations A and B:

```
grunt> DUMP A;
(2,Tie)
(4,Coat)
(3,Hat)
(1,Scarf)
grunt> DUMP B;
(Joe,2)
(Hank,4)
(Ali,0)
(Eve,3)
(Hank,2)
```

where each match between the two relations corresponds to a row in the result.

There is a special syntax for telling Pig to use a fragment replicate join:

```
grunt> C = JOIN A BY $0, B BY $1 USING 'replicated';
```

COGROUP

JOIN always gives a flat structure: a set of tuples. The COGROUP statement is similar to JOIN, but instead creates a nested set of output tuples.

This can be useful if you want to exploit the structure in subsequent statements:

```
grunt> D = COGROUP A BY $0, B BY $1;
grunt> DUMP D;
(0,{},{(Ali,0)})
(1,{(1,Scarf)},{})
(2,{(2,Tie)},{(Hank,2),(Joe,2)})
(3,{(3,Hat)},{(Eve,3)})
(4,{(4,Coat)},{(Hank,4)})
```

CROSS

Pig Latin includes the cross-product operator (also known as the Cartesian product), CROSS, which joins every tuple in a relation with every tuple in a second relation (and with every tuple in further relations, if supplied).

The size of the output is the product of the size of the inputs, potentially making the output very large:

```
grunt> I = CROSS A, B;
grunt> DUMP I;
(2,Tie,Joe,2)
(2,Tie,Hank,4)
(2,Tie,Ali,0)
(2,Tie,Eve,3)
(2,Tie,Hank,2)
(4,Coat,Joe,2)
(4,Coat,Hank,4)
(4,Coat,Ali,0)
(4,Coat,Eve,3)
(4,Coat,Hank,2)
(3,Hat,Joe,2)
(3,Hat,Hank,4)
(3,Hat,Ali,0)
(3,Hat,Eve,3)
(3,Hat,Hank,2)
(1,Scarf,Joe,2)
(1,Scarf,Hank,4)
(1,Scarf,Ali,0)
(1,Scarf,Eve,3)
(1,Scarf,Hank,2)
```

When dealing with large datasets, you should try to avoid operations that generate intermediate representations that are quadratic (or worse) in size.

GROUP

Where COGROUP groups the data in two or more relations, the GROUP statement groups the data in a single relation. GROUP supports grouping by more than equality of keys: you can use an expression or user-defined function as the group key.

For example, consider the following relation A:

```
grunt> DUMP A;
(Joe,cherry)
(Ali,apple)
(Joe,banana)
(Eve,apple)
```

Let's group by the number of characters in the second field:

```
grunt> B = GROUP A BY SIZE($1);
grunt> DUMP B;
(5,{(Eve,apple),(Ali,apple)})
(6,{(Joe,banana),(Joe,cherry)})
```

GROUP creates a relation whose first field is the grouping field, which is given the alias group. The second field is a bag containing the grouped fields with the same schema as the original relation (in this case, A).

Combining and Splitting Data

Sometimes you have several relations that you would like to combine into one. For this, the UNION statement is used. For example:

```
grunt> DUMP A;
(2,3)
(1,2)
(2,4)
grunt> DUMP B;
(z,x,8)
(w,y,1)
grunt> C = UNION A, B;
grunt> DUMP C;
(2,3)
(z,x,8)
(1,2)
(w,y,1)
(2,4)
```

C is the union of relations A and B, and because relations are unordered, the order of the tuples in C is undefined. Also, it's possible to form the union of two relations with dif- ferent schemas or with different numbers of fields, as we have done here.

Pig attempts to merge the schemas from the relations that UNION is operating on. In this case, they are incompatible, so C has no schema:

```
grunt> DESCRIBE A;
A: {f0: int,f1: int}
grunt> DESCRIBE B;
B: {f0: chararray,f1: chararray,f2: int}
grunt> DESCRIBE C;
Schema for C unknown.
```

If the output relation has no schema, your script needs to be able to handle tuples that vary in the number of fields and/or types

HBase

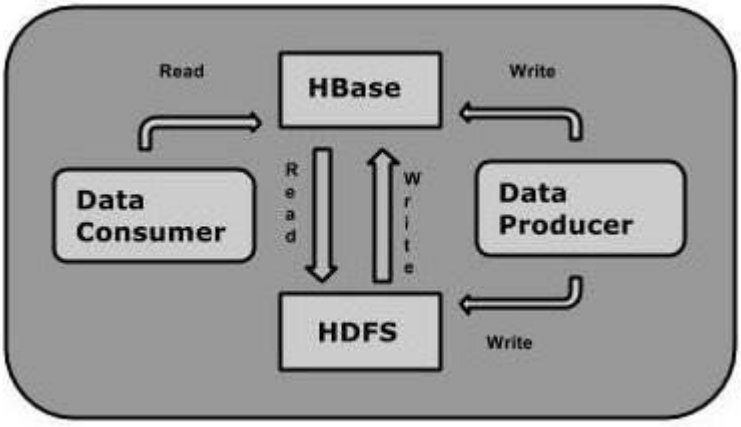
What is HBase?

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.

HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).

It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.



HBase and HDFS

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual	HBase provides fast lookups for larger tables.

record lookups.	
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

Storage Mechanism in HBase

HBase is a **column-oriented database** and the tables in it are sorted by row. The table schema defines only column families, which are the key value pairs. A table have multiple column families and each column family can have any number of columns. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an HBase:

- Table is a collection of rows.
- Row is a collection of column families.
- Column family is a collection of columns.
- Column is a collection of key value pairs.

Given below is an example schema of table in Hbase

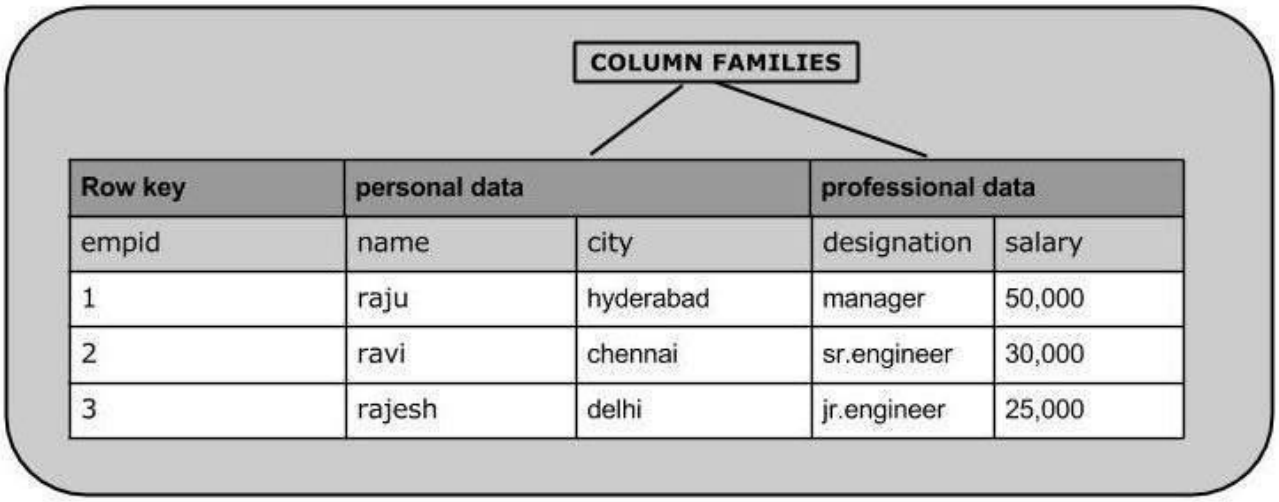
Rowid	Column Family			Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3	col1	col2	col3
1												
2												
3												

Column Oriented and Row Oriented

Column-oriented databases are those that store data tables as sections of columns of data, rather than as rows of data. Shortly, they will have column families.

Row-Oriented Database	Column-Oriented Database
It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for small number of rows and columns.	Column-oriented databases are designed for huge tables.

The following image shows column families in a column-oriented database:



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

HBase and RDBMS

HBase	RDBMS
HBase is schema-less, it doesn't have the concept of fixed columns schema; defines only column families.	An RDBMS is governed by its schema, which describes the whole structure of tables.
It is built for wide tables. HBase is horizontally scalable.	It is thin and built for small tables. Hard to scale.
No transactions are there in HBase.	RDBMS is transactional.
It has de-normalized data.	It will have normalized data.
It is good for semi-structured as well as structured data.	It is good for structured data.

Features of HBase

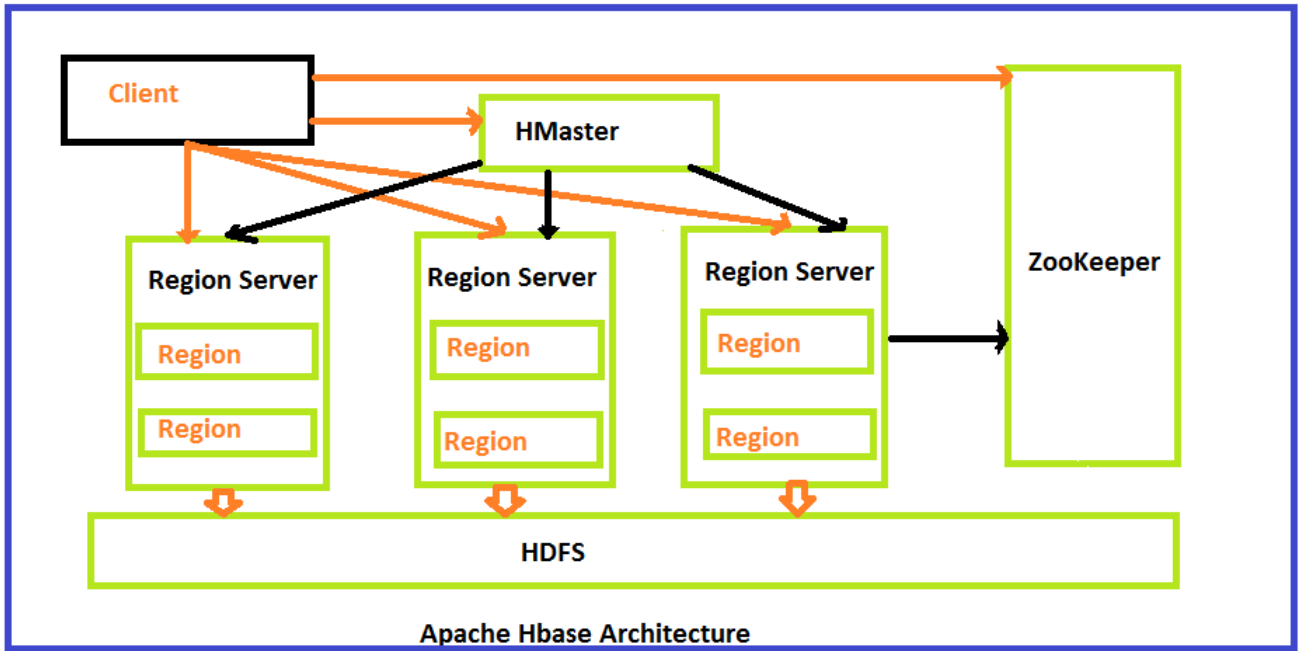
- HBase is linearly scalable.
- It has automatic failure support.
- It provides consistent read and writes.
- It integrates with Hadoop, both as a source and a destination.
- It has easy java API for client.

- It provides data replication across clusters.

Applications of HBase

- It is used whenever there is a need to write heavy applications.
- HBase is used whenever we need to provide fast random access to available data.
- Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

HBase Architecture:



HBase architecture consists mainly of four components

- HMaster
- HRegionserver
- HRegions
- Zookeeper
- HDFS

HMaster in HBase is the implementation of a Master server in HBase architecture. It acts as a monitoring agent to monitor all Region Server instances present in the cluster and acts as an interface for all the metadata changes. In a distributed cluster environment, Master runs on NameNode. Master runs several background threads.

The following are important roles performed by HMaster in HBase.

- Plays a vital role in terms of performance and maintaining nodes in the cluster.
- HMaster provides admin performance and distributes services to different region servers.
- HMaster assigns regions to region servers.

- HMaster has the features like controlling load balancing and failover to handle the load over nodes present in the cluster.

The client communicates in a bi-directional way with both HMaster and ZooKeeper. For read and write operations, it directly contacts with HRegion servers. HMaster assigns regions to region servers and in turn, check the health status of region servers.

HBase Region Servers

When HBase Region Server receives writes and read requests from the client, it assigns the request to a specific region, where the actual column family resides. However, the client can directly contact with HRegion servers, there is no need of HMaster mandatory permission to the client regarding communication with HRegion servers. The client requires HMaster help when operations related to metadata and schema changes are required.

HMaster can get into contact with multiple HRegion servers and performs the following functions.

- Hosting and managing regions
- Splitting regions automatically
- Handling read and writes requests
- Communicating with the client directly

HBase Regions

HRegions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families. It contains multiple stores, one for each column family. It consists of mainly two components, which are Memstore and Hfile.

ZooKeeper

HBase Zookeeper is a centralized monitoring server which maintains configuration information and provides distributed synchronization. Distributed synchronization is to access the distributed applications running across the cluster with the responsibility of providing coordination services between nodes. If the client wants to communicate with regions, the server's client has to approach ZooKeeper first.

Services provided by ZooKeeper

- Maintains Configuration information
- Provides distributed synchronization
- Client Communication establishment with region servers
- Provides ephemeral nodes for which represent different region servers
- Master servers usability of ephemeral nodes for discovering available servers in the cluster
- To track server failure and network partitions

Zookeeper

ZooKeeper is a distributed co-ordination service to manage large set of hosts. Co-ordinating and managing a service in a distributed environment is a complicated process. ZooKeeper solves this issue with its simple architecture and API. ZooKeeper allows developers to focus on core application logic without worrying about the distributed nature of the application.

Why do we need it?

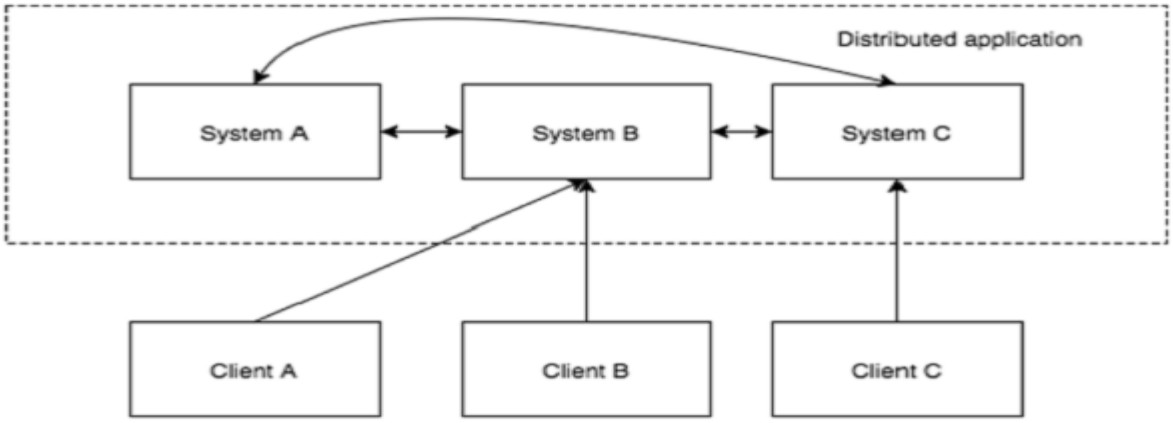
- **Coordination services**: The integration/communication of services in a distributed environment.
- Coordination services are complex to get right. They are especially prone to errors such as race conditions and deadlock.
- **Race condition**-Two or more systems trying to perform some task.
- **Deadlocks**– Two or more operations are waiting for each other.

Distributed Application

A distributed application can run on multiple systems in a network at a given time (simultaneously) by coordinating among themselves to complete a particular task in a fast and efficient manner. Normally, complex and time-consuming tasks, which will take hours to complete by a non-distributed application (running in a single system) can be done in minutes by a distributed application by using computing capabilities of all the system involved.

The time to complete the task can be further reduced by configuring the distributed application to run on more systems. A group of systems in which a distributed application is running is called a **Cluster** and each machine running in a cluster is called a **Node**.

A distributed application has two parts, **Server** and **Client** application. Server applications are actually distributed and have a common interface so that clients can connect to any server in the cluster and get the same result. Client applications are the tools to interact with a distributed application.



Benefits of Distributed Applications

- **Reliability** − Failure of a single or a few systems does not make the whole system to fail.
- **Scalability** − Performance can be increased as and when needed by adding more machines with minor change in the configuration of the application with no downtime.

- **Transparency** − Hides the complexity of the system and shows itself as a single entity / application.

What is Apache ZooKeeper Meant For?

Apache ZooKeeper is a service used by a cluster (group of nodes) to coordinate between themselves and maintain shared data with robust synchronization techniques. ZooKeeper is itself a distributed application providing services for writing a distributed application.

The common services provided by ZooKeeper are as follows −

- **Naming service** − Identifying the nodes in a cluster by name. It is similar to DNS, but for nodes.
- **Configuration management** − Latest and up-to-date configuration information of the system for a joining node.
- **Cluster management** − Joining / leaving of a node in a cluster and node status at real time.
- **Leader election** − Electing a node as leader for coordination purpose.
- **Locking and synchronization service** − Locking the data while modifying it. This mechanism helps you in automatic fail recovery while connecting other distributed applications like Apache HBase.
- **Highly reliable data registry** − Availability of data even when one or a few nodes are down.

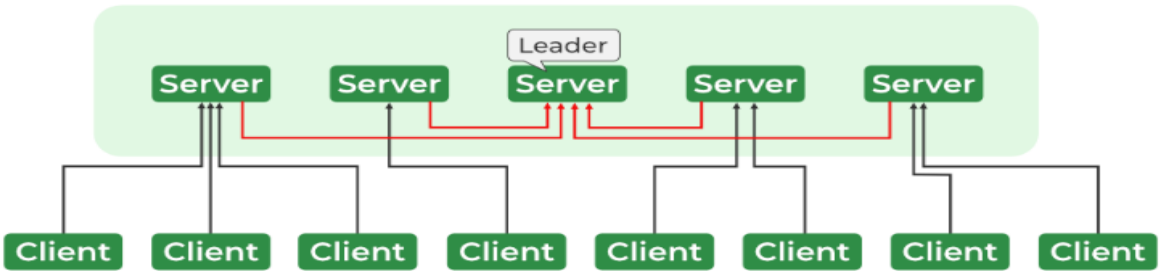
Benefits of ZooKeeper

Here are the benefits of using ZooKeeper −

- **Simple distributed coordination process**
- **Synchronization** − Mutual exclusion and co-operation between server processes. This process helps in Apache HBase for configuration management.
- **Ordered Messages**
- **Serialization** − Encode the data according to specific rules. Ensure your application runs consistently. This approach can be used in MapReduce to coordinate queue to execute running threads.
- **Reliability**

- Atomicity − Data transfer either succeed or fail completely, but no transaction is partial.

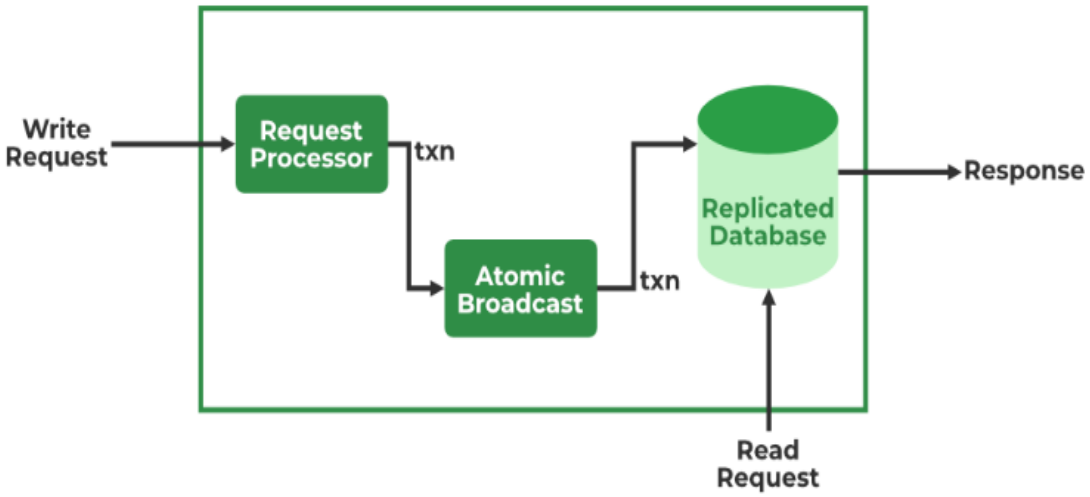
Zookeeper Architecture



The ZooKeeper architecture consists of a hierarchy of nodes called znodes, organized in a tree-like structure. Each znode can store data and has a set of permissions that control access to the znode.

The znodes are organized in a hierarchical namespace, similar to a file system. At the root of the hierarchy is the root znode, and all other znodes are children of the root znode. The hierarchy is similar to a file system hierarchy, where each znode can have children and grandchildren, and so on

Important Components in Zookeeper



- **Leader & Follower**
- **Request Processor** – Active in Leader Node and is responsible for processing write requests. After processing, it sends changes to the follower nodes
- **Atomic Broadcast** – Present in both Leader Node and Follower Nodes. It is responsible for sending the changes to other Nodes.
- **In-memory Databases** (Replicated Databases)-It is responsible for storing the data in the zookeeper. Every node contains its own databases. Data is also written to the file system providing recoverability in case of any problems with the cluster.

Other Components

- **Client** – One of the nodes in our distributed application cluster. Access information from the server. Every client sends a message to the server to let the server know that client is alive.
- **Server**– Provides all the services to the client. Gives acknowledgment to the client.
- **Ensemble**– Group of Zookeeper servers. The minimum number of nodes that are required to form an ensemble is 3.