Hadoop Distributed File System

History

Hadoop Distributed File System (HDFS) is an open-source implementation of the Google's GFS architecture as developed by Apache Software Foundation.

The development was initiated by Yahoo in 2006 being inspired by the Google's GFS and MapReduce papers and was looking to develop an open-source based system to fulfill their storage requirements.

They decided to pass the storage and data processing parts of their 'Nutch' search engine project to Apache Foundation and form the Hadoop as an open-source project.

Features

HDFS is a **Java-based** distributed file system that provides scalable and reliable data storage.

HDFS is also designed to run on **large clusters of commodity servers**.

It is a sub-project of the **Apache Hadoop project**.

It is extremely **fault-tolerant** and provides **high output access to application data**.

The file system is available for consumers on the **Amazon EC2 cloud platform**.

HDFS is designed to reliably store very large files across multiple machines in a large cluster.

HDFS cluster contains two types of nodes as one single master node called as **NameNode** and other slave nodes called as **DataNodes**.

Files are broken into sequence of blocks of reasonably bigger size (64 MB or 128 MB)

These blocks are stored on DataNodes commodity servers.

Fault Tolerance

Consider a scenario of node failure in HDFS

To increase fault tolerance of the system, it replicates blocks over multiple DataNodes.

By default it uses **3 replicas**. The block size and the replication factor are configurable.

During read operation, data is fetched from any one of the replicas.

During write operation, data is sent to all of the DataNodes containing replicas of the file.

Master node usually stores metadata about the blocks.

Master-Slave Architecture

Every server in a HDFS cluster have **data node** and a **task tracker** associated with them.

The single **name node** stays in a master server that manages the file system and **stores metadata about the data nodes**.

The master server also has a **job tracker** that coordinates all of the activities across a cluster.

Every server, master or slave both, have **MapReduce** function implemented into them.

Every node has a **database engine** also.

The name node and data node are actually pieces of software developed in Java that generally run on Linux operating system.

Usages of portable language like java ensure that the software can be deployed on broad range of commodity hardware.

Generally in real-life cases, one data node is created on one server although the HDFS architecture does not prevent running multiple data nodes on the same server.

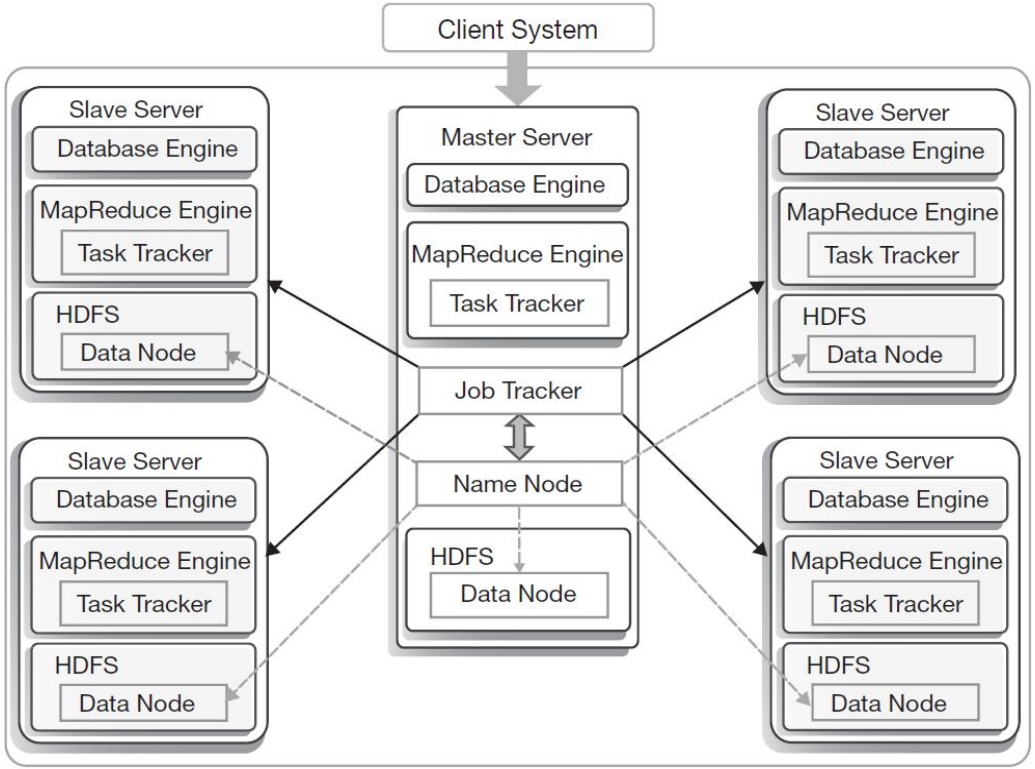

FIG 13.2: A model of HDFS cluster

The goals of HDFS

- Fast recovery from hardware failures
- Access to streaming data
- Accommodation of large data sets
- Portability


Pig Latin
High Level
Language

Introduction

Pig Latin is a high-level data flow language developed by Yahoo! that has been implemented on top of Hadoop in the Apache Pig project.

Pig Latin, Sawzall and DryadLINQ are different approaches to building languages on top of MapReduce and its extensions

Table 6.7 Comparison of High-Level Data Analysis Languages

	Sawzall	Pig Latin	DryadLINQ
Origin	Google	Yahoo!	Microsoft
Data Model	Google protocol buffer or basic	Atom, Tuple, Bag, Map	Partition file
Typing	Static	Dynamic	Static
Category	Interpreted	Compiled	Compiled
Programming Style	Imperative	Procedural: sequence of declarative steps	Imperative and declarative
Similarity to SQL	Least	Moderate	A lot!
Extensibility (User-Defined Functions)	No	Yes	Yes
Control Structures	Yes	No	Yes
Execution Model	Record operations + fixed aggregations	Sequence of MapReduce operations	DAGs
Target Runtime	Google MapReduce	Hadoop (Pig)	Dryad

Table 6.8 Pig Latin Data Types

Data Type	Description	Example
Atom	Simple atomic value	'Clouds'
Tuple	Sequence of fields of any Pig Latin type	('Clouds', 'Grids')
Bag	Collection of tuples with each member of the bag allowed a different schema	{ ('Clouds', 'Grids') ('Clouds', ('IaaS', 'PaaS') }
Map	A collection of data items associated with a set of keys; the keys are a bag of atomic data	['Microsoft' → { ('Windows') ('Azure') } 'Redhat' → 'Linux']

Table 6.9 Pig Latin Operators

Command	Description
LOAD	Read data from the file system.
STORE	Write data to the file system.
FOREACH GENERATE	Apply an expression to each record and output one or more records.
FILTER	Apply a predicate and remove records that do not return true.
GROUP/COGROUP	Collect records with the same key from one or more inputs.
JOIN	Join two or more inputs based on a key.
CROSS	Cross product two or more inputs.
UNION	Merge two or more data sets.
SPLIT	Split data into two or more sets, based on filter conditions.
ORDER	Sort records based on a key.
DISTINCT	Remove duplicate tuples.
STREAM	Send all records through a user-provided binary.
DUMP	Write output to stdout.
LIMIT	Limit the number of records.

Example

Given below is a Pig Latin statement, which loads data to Apache Pig.

```
grunt> Student_data = LOAD 'student_data.txt' USING PigStorage(',')as
   ( id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray );
```