Architecture Survey

Name of system: Hadoop Distributed File System ****
Reviewer: Zhiyuan Wu
Date: Oct 17th, 2011

§1 Introduction

1.1 *Purpose of system*

The Hadoop Distributed File System complements the popular Hadoop Map-Reduce algorithm. Naturally, they are part of the same Apache project.

Hadoop MapReduce algorithm distributes computation to slave nodes. (See the section 2.1 for details on MapReduce.) In order for slave nodes to have better access to the data it needs, data should be placed as close to the slave node as possible. In short, Hadoop Map-Reduce benefits from file systems that are location-aware. In data center terms, this location awareness is termed rack-awareness. The Hadoop Distributed File System is a rack aware file system, among many that Hadoop supports.

HDFS' importance is attached to Hadoop's own importance. Hadoop's innovation is within the MapReduce itself. The MapReduce framework allows programmers to orchestrate massively distributed computations without having to concern themselves with the multi-threading, synchronization and messaging mechanisms themselves. Thus Hadoop MapReduce has made distributed parallel computing accessible to a less multi-threading savvy audience.

1.2 *Book Chapter*

Author of software: **David Cutting**

Author of book chapter: **Robert Chansler, Hairong Kuang, Sanjay Radia, Konstantin Shvachko, and Suresh Srinivas**

Five star rating of book chapter: ********

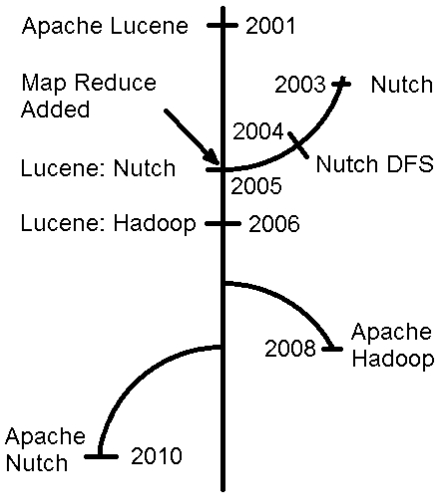
I feel this chapter is well written. It explained the architecture of HDFS quite well. However, since HDFS is bundled with Hadoop, its architecture greatly depends on the nature of the MapReduce algorithm framework. I think the book chapter did not explain enough about MapReduce to demonstrate why HDFS is designed as so.

1.3 *History*

In 2003, David Cutting started developing a webpage crawler/indexer to compete with Google. This software took shape in Nutch. Much to his dismay, David found that he lacked sufficient computational power to topple Google's lead.

To mitigate this problem, David found a cluster to run Nutch on. He found that latency and cross-rack communication was slowing performance. Borrowing a page from Google File System [GGL03], he developed Nutch Distributed File System in 2004. Nutch DFS was rack aware and was quite fast.

In 2005, Nutch was absorbed into Apache Lucene, an indexer and top level Apache project. Later in 2006, Hadoop Map Reduce was created as an additional member of the Lucene project. Around this time, Facebook and Yahoo discovered the usefulness of MapReduce and contributed heavily in money and code. In 2008, Hadoop became its own top level Apache project. HDFS tagged along in this promotion and the rest is history.



1.4 Basic metrics

KLOC	**1.3** MLoc
Project start-up	**2003 (**as **Nutch)**
Number of major releases	**21**
Number of developers	**48**
Size of user community or number of installations	**High**. Some important users are: Yahoo, Powerset, Facebook, Amazon.
Major stakeholders	**Doug Cutting and the user community**
Use of concurrency	**Via the Map-Reduce Framework (see section 2.1)**
Implementation language	**Java**
Supporting software	**Cross Platform Standalone, requires Java JVM** HDFS depends on **Hadoop Core** and **Avro (part of the Hadoop project. See section 2.2)**

§2 Architecture

It does not make sense to talk about the Hadoop Distributed File System architecture without describing motivations from Hadoop itself. Section 2.1 talks about main mechanisms of the MapReduce algorithm itself. Section 2.2 talks about the Hadoop architecture, including the HDFS architecture. Section 2.3 rounds the rear with an explanation of the read scenario in HDFS.

2.1 *Map Reduce Algorithm and Data Control Flows*

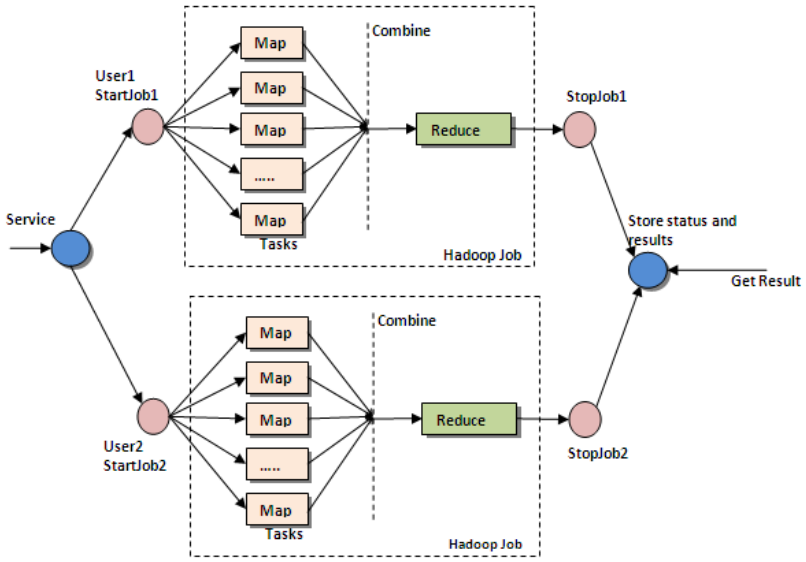
The crown jewel of the Hadoop is the MapReduce system. This algorithm framework was initially published by Google in their famous 2003 paper. [DG04] Like its name implies, MapReduce is composed of two steps: *Map* and *Reduce*. These two steps merely specify the required input and output data structures and that processing on each data entry in the input is independent of other data entries. Thus, data processing can be trivially distributed.

The *Map* step takes as input a map of key-value pairs and for each pair conducts some data processing. The output is a list of key-value pairs. Explicitly:

Map : <key, value> --> List <key, value>

The important element here is that a single key-value pair from the input must be able to be processed *independently* from the other pairs. Thus, this step can be parallelized by design. As such, the input can be and is split into many chunks and each chunk is given to a slave node for processing. This *independence restriction* is important to the Map Step.

The resulting lists of key-value pairs from all slave nodes are combined to make a master list on the master node. At this point the *Reduce* step starts. This master list again is a list of key-value pairs. The keys in this list are not necessarily unique. Given a particular key, there is a list of corresponding values. The input to the Reduce step is a key to list of values mapping. The output is simply a list of values. Explicitly:



Reduce: <key, List<value>> --> List <value>

The algorithm framework has that in the reduce step, processing conducted on each key is *independent* of the other keys. Thus, the Reduce step is also trivially parallelizable via entry-wise distribution. This *independence restriction* is important to the Reduce step.

The ingenuity of this algorithm framework is that the developer is only required to write the *map* and *reduce* functions and the framework takes care of parallelization. Task distribution can be split across keys thanks to the *independence restrictions* placed at each step.

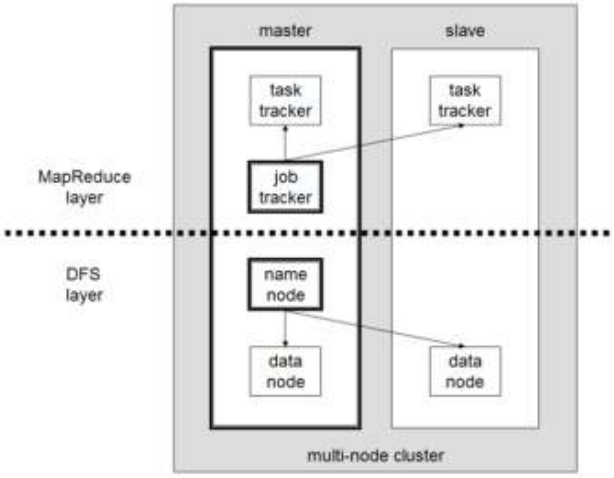
Tying the framework back to the discussion of Hadoop Distributed File System, it is quite intuitive to see the benefits of placing data closer to the slave node that processes it. Rack-awareness reduces cross rack communication.

2.2 *Hadoop architecture*

Hadoop takes advantage of *the independence restrictions* imposed by the Map and Reduce steps and delegates computation tasks to slave nodes.

2.2.1 *High Level Architecture*

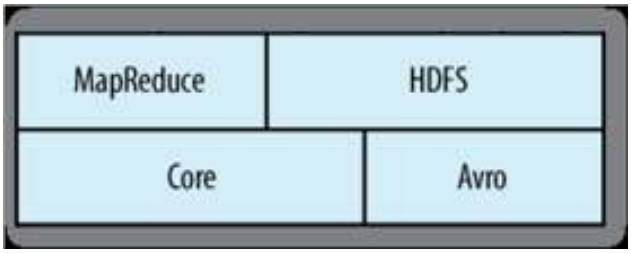
The Hadoop architecture is made simple in the diagram to the right. The MapReduce algorithm (explained in section 2.1) sits on top of a distributed file system. Arrows represent data access. Large enclosing rectangles represent the master and slave nodes. The small rectangles represent functional units.



The file system layer can be any virtualized distributed file system. Hadoop performs best when coupled with the Hadoop Distributed File System because the physical data node, being location/rack aware, can be placed closer to the task tracker that will access this data.

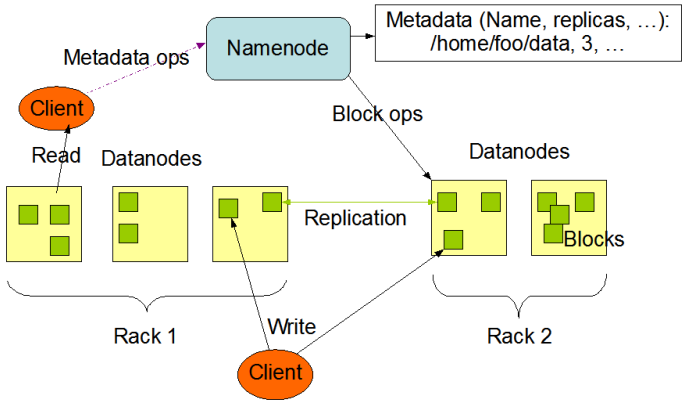
2.2.2 Development View

To the right is a high simplified view of the software dependency stack in the Hadoop Apache project. Core contains utilities like I/O and networking protocols. Avro contains utilities for data serialization and cross-language RPC.



2.2.3 HDFS Concept Architecture

To the right is the concept architecture of the Hadoop Distributed File System. The arrows represent data flow and boxes contain functional units. The ovals are the clients of HDFS, in most cases, Hadoop itself.



For each data access:

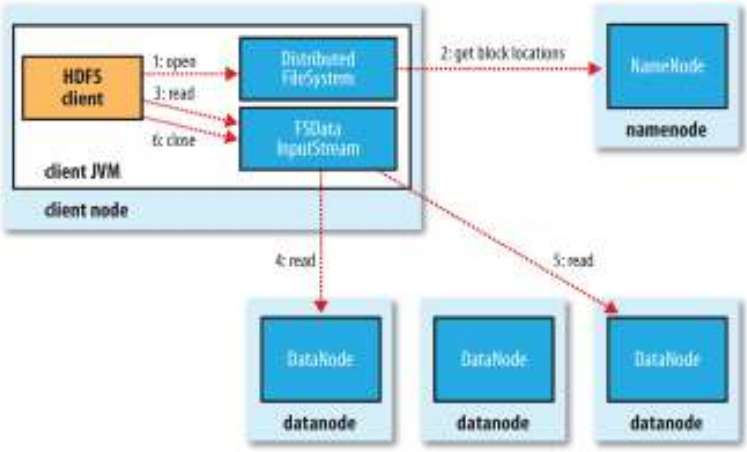
1. A client calls a Namenode to determine which node to access data from.
2. The Namenode looks up the location information from a metadata store
3. The client uses this location data to read/write from/to target rack

4. Writes are replicated. Typically, the two copies of the same data are placed on the same rack and a third copy is placed on a different rack. Placing two copies of data on the same rack allows for better read performance. The third copy is for redundancy.

2.3 High level scenarios

The diagram at the right describes in detail a read scenario in HDFS.

Arrows represent data access. Boxes represent logical entities. The encapsulating light blue boxes represent location.



§3 Style and Methodology

3.1 Architectural style:

At the logic level, MapReduce uses pipeline architecture. HDFS is implemented in Java and thus takes up an OOP style. HDFS follows the implementation pattern of a typical file system. The only difference is that the metadata stores block information rather than file descriptors.

The distribution mechanism for Hadoop follows a master-slave model.

3.2 Major evolutionary changes: [If any. How has architecture changed over time?]

HDFS followed the design outline from the famous Google paper on the Google File System. [GGL03] No evolutionary changes in architecture have occurred.

3.3 Performance bottlenecks:

As with any master-slave system, the master is a single point of failure. However, Hadoop performs mostly data crunching tasks and is not designed for reliability. It is observed that it is normal that a few nodes fail during a computational run. [GGL03] In case of failure, Hadoop reruns a job.

3.4 Real time:

HDFS' rack awareness helps reduce latency during data accesses. Data on the same rack takes less time to access than data from another rack.

3.5 Methodology:

The development methodology for Hadoop is typical of open source projects. It is agile and is based on incremental improvements.

§Appendix: Kruchten's eight context attributes

Size	L **= 1.3** MLoc
Criticality	**Lo** = Computations usually for data aggregation. Usually one or two entries are expected to be erroneous
Age of System	**M~L** = 8 years
Rate of Change	**Med** = low number of major releases
Business Model	**Open Source,** funding from large corporations that use Hadoop
Stable architecture	**Lo** = Hadoop has been based around the simple Map Reduce algorithm. The overall architecture has remain the same
Team distribution	**VH** = team members are often contributors from within large corporations across the world.
Governance	**Lo** = the open source model often has an adhoc governance model based around a chief contributor

§Bibliography

[GGL03] Ghemawat S., Gobioff H., Leung S. T. *The Google file system*. ACM SIGOPS Operating Systems Review - SOSP '03 Volume 37 Issue 5, December 2003

[DG04] Dean, J. and Ghemawat, S. *MapReduce: Simplified Data Processing on Large Clusters* in *OSDI'04: 6th Symp*, 2004.