Map Reduce Framework Report

**Arjun Puri & Karan Sharma**

22nd November 2014

**Contents**

**Overview**

Distributed File System (DFS)

**MapReduce engine supplies files given by user**

**Registers Data Nodes**

**Updating health status**

**Distribute Files**

**Handshake initialization**

**Heartbeat**

**Initialize**

The DFS is the backbone of the MapReduce Framework. It provides a distributed data storage facility for all participants of the framework. The DFS consists of a few central components: The DFSNameNode, DFSDataNode, DFSConnectionManager, and the DFSHealthMonitor.

The DFS is initialized with the NameNode being created upon the creation of a MapReduceMaster host. This consequently instantiates a DFSConnectionManager and DFSHealthMonitor in separate threads to aid the facility of the DFSNameNode. Their functionality is explained further on. The DataNodes are created on separate machines (NOTE: must be separate from the NameNode). These DataNodes are created upon instantiation of the MapReduce Slaves.

Here is a better description of the respective parts of the DFS:

**DFSNameNode**

The DFSNameNode must be assigned before the assignment of any MapReduce jobs. The purpose of the DFSNameNode is to facilitate the partitioning and transfer of files to the DFSDataNodes that it receives from the MapReducer client. It also manages the status of all participants with the help of the DFSHealthMonitor. It provides RMI Services for other DFSDataNodes to query for information about where files are located across the system. In the case of a DFSDataNode failure, the DFSNameNode will ensure in maintaining the replication factor by transferring those files to another DFSDataNode, while also balancing the load appropriately. This is done with aid of FIFO queue.

**DFSDataNode**

The DFSDataNode facilitates the storage of file block replicas. To clarify- on the DFSNameNode, any given file is split up into chunks and then replicated across the file system. The DFSDataNode maintains the availability of space and also provides it’s own RMI services for the name node to transfer file blocks. The data node is also responsible for sending a heartbeat at a fixed interval defined in an internal configuration class. This heartbeat is sent to the DFSHealthMonitor.

**DFSConnectionManager**

The DFSConnectionManager runs in a separate thread from the DFSNameNode. It exists to handle all incoming connection requests from any new DFSDataNode. On receiving a handshake from a data node, the DFSConnectionManager will register this DFSDataNode with the DFSNameNode, and the DFSHealthMonitor. One this is done, the DFSDataNode is officially part of the active system.

**DFSHealthMonitor**

The DFSHealthMonitor is responsible for ensuring that all participating DFSDataNodes are fully responsive. The DFSHealthMonitor stores an initial 100 health for all DFSDataNodes that are initiated into the system. When a data node skips a heartbeat, it loses a certain amount of health until it’s health reaches 0 at which point it is deemed inactive/dead. The reason this is done is to allow a data node to go offline for a momentary period then join back with losing all it’s files to other data nodes.

MapReduce

**Flush input files to NameNode to distribute**

**Master creates job handler for specific job.**

**handshake**

**runJob- serializes and sends class files across as a config package**

**Created in application programmer’s code which calls runJob**

The MapReduce framework provides a platform for application programmers to perform MapReduce jobs in an efficient and succinct manner. The first point of interaction is in the application programmer’s code which creates a MapReduceClient object to run a job. The programmer provides the input files, a Map class and a Reduce class to the framework in the form of a configuration abstraction called the MapReduceConfig. It must be noted that the DFS and the MapReducerMaster and Slaves must be running prior to running the programmer’s code. The MapReduceClient will send all the require information to the MapReducerMaster which will start, schedule and monitor the job across the slaves.

**It is extremely important to note that a MapReduceClient MUST run on a machine that is already hosting a DFSDataNode. This allows the framework to maximize efficiency through a certain amount locality.**

Below is a more detailed description of the components of the MapReduce Framework.

**MapReducerClient**

The MapReducerClient is where the application programmer can interact with the framework. In the code that the programmer writes- he must create an instance of this, and then calls the runJob function. The MapReducerClient will take in a configuration that the programmer will write that contains the class files, the input files etc. that the programmer has customized. The MapReducerClient will then locate the input files and will flush it to the DFS by handing byte arrays to the DFSNameNode for partitioning and distribution. The MapReducerClient will also forward the Mapper and Reducer Class as byte arrays to the Master. The MapReducerClient must handshake with the Master before this is possible.

**Master**

The Master on the MapReduce framework provides the central RMI facility to communicate the DFSNameNode and all other it’s respective components such as the ScheduleManager etc. The Master on construction will start a DFSNameNode and a ScheduleManager. The MapReducerClient will handshake with the Master which puts the client in a position to call the createJob method. This will construct the necessary components for a job- jobID, add the class byte arrays to the Master, and a JobHandler for that job. Starting the job will forward execution to the JobHandler. The Master also has some interactive capabilities in the terminal.

**TaskManager**

The TaskManager is local to the particular slave that it is working on. It runs as a Thread while working with the local files on the corresponding DFSDataNode. The TaskManager establishes a Thread pool based on the number of cores the current machine which is determined at run time.

**JobHandler**

**ScheduleManager**

**MRCollector**

**MapExecutor**

**ReduceExecutor**

**Limitations**

**System Requirements**

**Framework Features**

Distributed File System (DFS)

MapReduce

**Improvements**

Distributed File System (DFS)

MapReduce

**Building and Deploying**

**Testing with Examples**