Drăghici Marius-Robert (SSA), Manole Filip-George (AAC)

HADOOP CLIENT

***Abstract*- The purpose of this project is to implement a Java client to upload and download files of different sizes into a Hadoop Cluster. The performance of this operations is analyzed by measuring upload/download times and bandwidth.**

Index Terms*- *bandwidth, cluster, directory, download, file, hadoop, Java, speed, upload.

I. INTRODUCTION

This article presents the implementation details of the Java client for uploading and downloading files to a Hadoop Cluster.

The article is divided in the following sections:

1. Introduction
2. Hadoop
3. Implementation details
4. Results
5. Conclusion

II. HADOOP

Hadoop is a collection of open-source software utilities that provide a distributed filesystem and a framework for distributed storage and processing of big data using the MapReduce paradigm. The purpose of the MapReduce paradigm is to process and generate big data sets with a parallel, distributed algorithm on a cluster.

Hadoop has four main components:

1. HDFS (Hadoop Distributed File System) is the filesystem component of Hadoop that is meant to be run on commodity hardware. Additionally, it stores file system metadata and application data separately.

2. Hadoop MapReduce is the Java implementation of the MapReduce paradigm for large scale data processing.

3. Hadoop Common contains libraries and utilities needed by other Hadoop modules.

4. Hadoop YARN manages the computing resources in clusters that store the data and schedules Hadoop jobs.

An important characteristic of Hadoop is the partitioning of data and computation across many (thousands) of hosts, and executing application computations in parallel close to their data. A Hadoop cluster scales computation capacity, storage capacity and IO bandwidth by simply adding commodity servers.

Similar to other distributed file systems such as PVFS, Lustre and GFS, HDFS stores metadata on a dedicated server, called the NameNode. The application data is stored on other servers called DataNodes. All servers are fully connected and communicate using the TCP protocol.

The HDFS client is a code library that exports the HDFS file system interface and is used to allow the user application to access the file. Similar to most conventional file systems, HDFS supports operations to read, write and delete files, and operations to create and delete directories. The user references files and directories by paths in the namespace. However, unlike conventional file systems, HDFS provides an API that exposes the locations of a file blocks. This allows applications like the MapReduce framework to schedule a task to where the data are located, thus improving the read performance. It also allows an application to set the replication factor of a file.

III. Implementation details

The setup for the project is available in the following link: https://github.com/green-spark/bigdata-learning/wiki/Hadoop-setup---Single--node-cluster

For this project, we have used Apache Hadoop 2.7.3.

There are two classes: HDFSFileUpload and HDFSFileDownload. HDFSFileUpload receives as parameters the directory from which it reads the files sequentially into the cluster and the path of the cluster. HDFSFileDownload receives as parameters the directory where the files will be downloaded and the path where the files are stored in the hadoop cluster.

HDFSFileUpload reads each file from the local file system and uploads it to the hadoop file system. It measures the time required for the upload and it calculates the bandwidth by dividing the size of the file by the time needed for its upload.

HDFSFileDownload uses a RemoteIterator in order to iterate through the files stored in the hadoop cluster and to download each of them. It also measures download time and calculates the bandwidth used.

IV. Results

We run experiments using files from size 1Kb to 10 Gb and plotted the results in order to observe the evolution of performance as the file size increases.

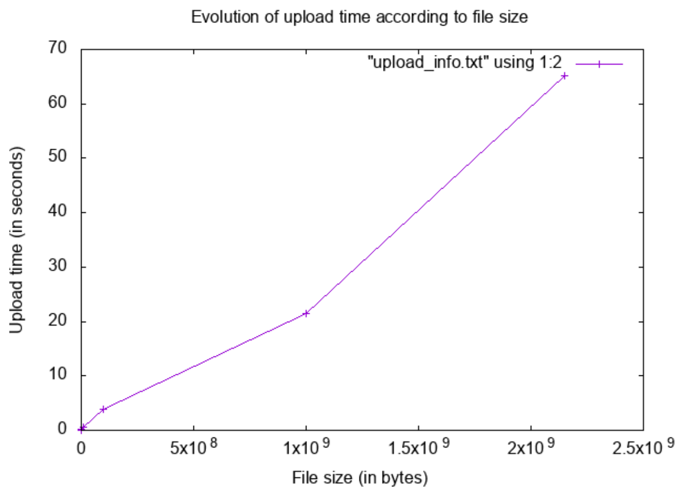

Figure 1

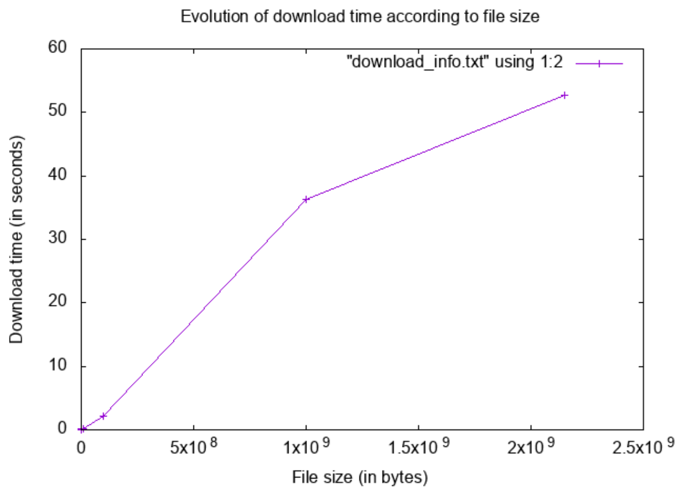

Figure 2

In Figure 1 and Figure 2, we can see that as the size of the file increases so does the time required for its upload or download.

Size in bytes	Upload time	Upload bandwidth
1000	0.083991164	11906.014304076081
10000	0.09941974	100583.64666815664
100000	0.102995589	970915.3660939791
1000000	0.132421695	7551632.683753217
10000000	0.542187824	1.8443793012954123E7
100000000	3.933590525	2.542206652279853E7
1000000000	21.416511069	4.669294624031836E7
2147479552	65.223679203	3.2924845366607673E7

Table 1: Upload experiment results

Size in bytes	Download time	Download bandwidth
1000	0.163051495	6133.031776249583
10000	0.032193563	310621.10149162426
100000	0.014325316	6980648.803837905
1000000	0.008510969	1.1749543442115697E8
10000000	0.181013625	5.524446018911559E7
100000000	2.21467671	4.515331720809039E7
1000000000	36.322654902	2.75310271977101E7
2147479552	52.683881797	4.0761604474677965E7

Table 2: Download experiment results

The data in the 2 tables shows that the difference in upload and download time is small for small files (under 10Mb).

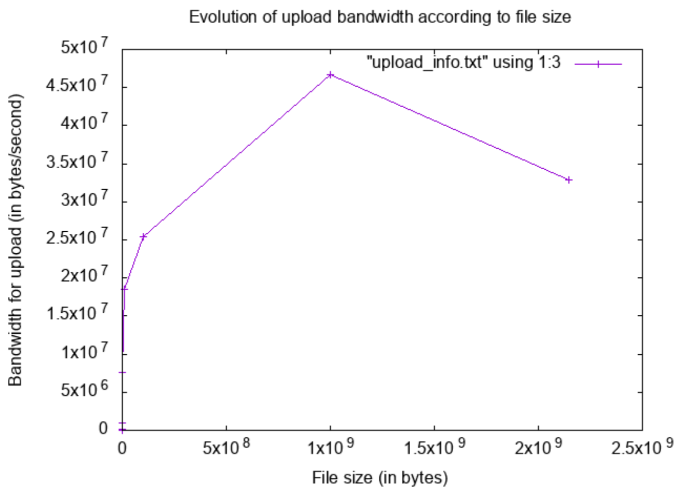

Figure 3

Figure 3 shows that the bandwidth used for uploading the files reaches a maximum point and then begins to decrease.

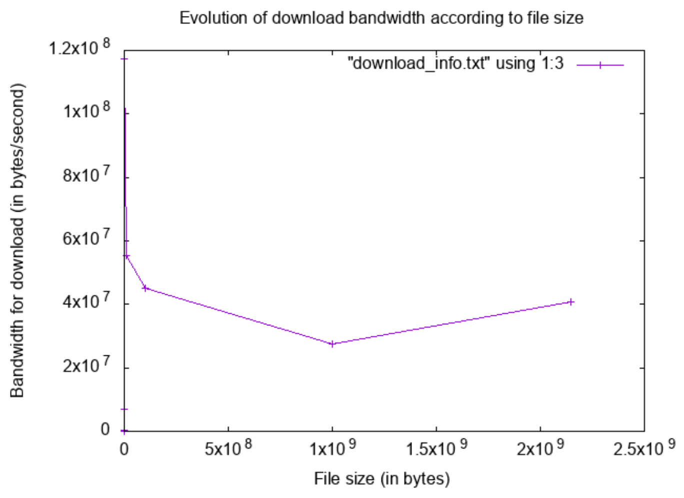

Figure 4

V. Conclusion

In this paper, we have presented an implementation of a Java client for uploading and downloading files to a Hadoop Cluster and shown how some performance metrics such as time and bandwith evolve when the file size increases.