A New HDFS Structure Model to Evaluate the Performance of Word Count Application on Different File Size

Mohammad Badrul Alam Miah
Dept. of Information and Communication Technology
Mawlana Bhashani Science and Technology University
Santosh, Tangail-1902, Bangladesh

Mehedi Hasan
Dept. of Information and Communication Technology
Mawlana Bhashani Science and Technology University
Santosh, Tangail-1902, Bangladesh

Md. Kamal Uddin
Dept. of Information and Communication Technology
Mawlana Bhashani Science and Technology University
Santosh, Tangail-1902, Bangladesh

ABSTRACT
MapReduce is a powerful distributed processing model for large datasets. Hadoop is an open source framework and implementation of MapReduce. Hadoop distributed file system (HDFS) has become very popular to build large scale and high performance distributed data processing system. HDFS is designed mainly to handle big size files, so the processing of massive small files is a challenge in native HDFS. This paper focuses on introducing an approach to optimize the performance of processing of massive small files on HDFS. We design a new HDFS structure model which main idea is to merge the small files and write the small files at source direct into merged file. Experimental results show that the proposed scheme can improve the storage and access efficiencies of massive small files effectively on HDFS.

Keywords
Hadoop, MapReduce, HDFS, Big data, Cluster

1. INTRODUCTION
The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models to handle bid data. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage [1].

Hadoop is used by Yahoo. Yahoo has more than 100,000 CPUs in >40,000 computers running Hadoop. Facebook use Apache Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting/analytics and machine learning. Currently they have 2 major clusters: A 1100-machine cluster with 8800 cores and about 12 PB raw storage and a 300-machine cluster with 2400 cores and about 3 PB raw storage. A wide variety of websites like Twitter, Amazon, Alibaba, EBay etc. use Hadoop to manage massive amount of data on a daily basis [2].

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this mode [3].

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets [1].

HDFS is designed mainly to streaming access of large files. The default block size of HDFS is 128MB. When data is represented in files significantly smaller than the default block size the performance degrades dramatically. The problem is that HDFS can't handle lots of files. Every file, directory and block in HDFS is represented as an object in the namenode's memory, each of which occupies 150 bytes, as a rule of thumb. So 10 million files, each using a block, would use about 3 gigabytes of memory [4].

Other problem is while running map/reduce jobs. The loading of thousands of small files cause a lot of time, specifically, if small files are in snappy compressed form.

The number of blocks also increase as the number of small files increase. This causes, increase number of map tasks. Though increase in number of map tasks sometimes increase in parallelism and hence, job efficiency, but, too many mappers will cause too much load on the cluster causing slowness of job execution.

2. BACKGROUND
2.1 Hadoop
Hadoop is a platform that provides both distributed storage and computational capabilities. Hadoop was first conceived to fix a scalability issue that existed in Nutch, 2 an open source crawler and search engine. At the time Google had published papers that described its novel distributed file system, the Google File System (GFS), and MapReduce, a computational framework for parallel processing. The successful implementation of these papers' concepts in Nutch resulted in its split into two separate projects, the second of which became Hadoop, a first-class Apache project [5]. Hadoop is an open source framework for writing and running distributed applications that process large amounts of data. Actually Hadoop is a framework of tools. The objective of Hadoop is to support running applications on Big Data & handle it. So, Hadoop is data storage and processing system. It includes two core components:

- Hadoop Distributed File System (HDFS), for storage
- MapReduce, for parallel data processing

2.2 Hadoop Distributed File System (HDFS)

HDFS is the storage component of Hadoop. It's a distributed file system that's modeled after the Google File System (GFS) paper [3]. HDFS is optimized for high throughput and works best when reading and writing large files (gigabytes and larger). To support this throughput HDFS leverages unusually large (for a file system) block sizes and data locality optimizations to reduce network input/output (I/O). Scalability and availability are also key traits of HDFS, achieved in part due to data replication and fault tolerance. HDFS replicates files for a configured number of times, is tolerant of both software and hardware failure, and automatically re-replicates data blocks on nodes that have failed.

Figure 1 shows a logical representation of the components in HDFS: the namenode and the datanode. It also shows an application that's using the Hadoop file system library to access HDFS. Now that you have a bit of HDFS knowledge, it's time to look at MapReduce, Hadoop's computation engine. A set of machines running HDFS and MapReduce is known as a Hadoop Cluster. Individual machines are known as nodes. A cluster can have as few as one node, as many as several thousands.

HDFS, the Hadoop Distributed File System, is responsible for storing data on the cluster. Data files are split into blocks (128MB in size) and distributed across multiple nodes in the cluster. Each block is replicated multiple times. Default is to replicate each block three times. Replicas are stored on different nodes. This ensures both reliability and availability.

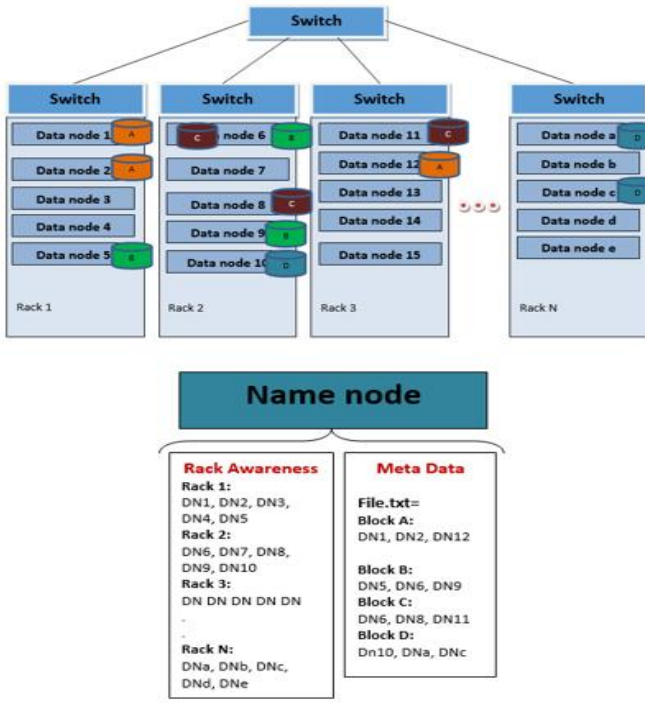

Fig 1: Hadoop cluster model

2.3 MapReduce

MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

Minimally, applications specify the input/output locations and supply map and reduce functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the job configuration. The Hadoop job client then submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client [1].

3. IMPLEMENTATION DETAILS

In this section, implementations of different experiments are discussed.

3.1 Single Node Cluster

In order to evaluate the performance of the word count application on small files, experiments must have been done. Our performance evaluation of word count application on small files is Hadoop based test bed. Hadoop is a framework that has two main components: HDFS and MapReduce. In this section, we describes how we've tried to set up and configure a single-node Hadoop on Ubuntu desktop so that we can execute word count application on many small files on single node Hadoop cluster and evaluate the performance of word count application on small files. It is single node Hadoop cluster with master and slave on same machine. Before implementing single-node Hadoop cluster on Ubuntu desktop, we must need to install: (1) Java^TM 1.6.x or later version; (2) SSH and SSHD must be running to use the Hadoop scripts that manage remote Hadoop daemons [6]. Then we have to create an SSH key for the Hadoop user.

Then we need to create a dedicated Hadoop group and a user in that group and give permission to the specific user to access the Hadoop features. We need to do so for avoiding unusual access to Hadoop cluster from various users. There is a problem with IPv6 on Ubuntu and it is related to Hadoop, so it is better to disable it.

Completing the above steps, we can now setup Hadoop by downloading it from apache site and extract all the files in the downloaded package to any local directory. We have to make sure to change the owner of all the contents to the created user and hadoop group.

We have to set Hadoop environment variable exporting java path, hadoop path, etc. The environment variable that has to be configured for Hadoop is JAVA_HOME. Open hadoop-env.sh, set the JAVA_HOME environment variable.

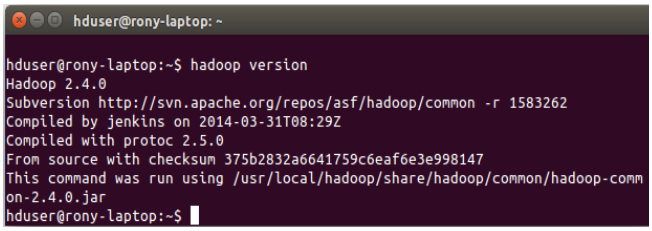

Fig 2: Command prompt

To set the directory where HDFS will store its data blocks, specify temporary directory, specify the port of our MapReduce job tracker, etc. we must tell Hadoop these by configuring its some configuration files named core-site.xml, mapred-site.xml, hdfs-site.xml, etc. To verifying single node Apache Hadoop Setup, commanding 'hadoop version' on command prompt, the following result in the figure will appear.

Now we've got Hadoop up and running on our Ubuntu desktop. Hadoop comes with several web interfaces which provide concise information about what's happening in your Hadoop cluster such that: The name node web UI shows you a cluster summary including information about total/remaining capacity, live and dead nodes; he JobTracker web UI provides information about general job statistics of the Hadoop cluster, running/completed/failed jobs and a job history log file; The task tracker web UI shows you running and non-running tasks [7].

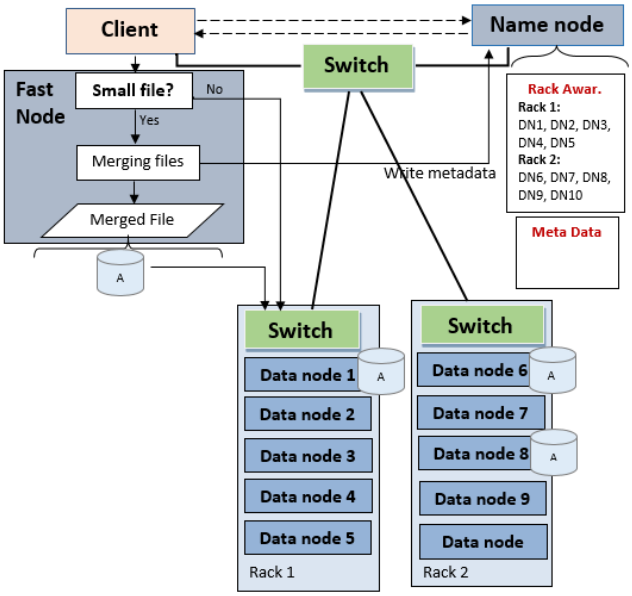

Fig 3: HDFS structure model

To start our cluster we need to run all the Hadoop daemon such as namenode, datanode, jobtracker and a tasktracker on the machine through commanding 'start-dfs.sh' and 'start-yarn.sh' on the command prompt. The Java process status tool 'jps' list all processes in Hadoop.

3.2 HDFS Structure Model

We design a new HDFS structure model which main idea is to merge the small files and write the small files at source direct into merged file and then replicates the merged file through data pipeline. The new HDFS structure is shown as Figure.

In the previous related work to handle massive small files, everyone tried mainly to merge small file that are already placed in datanode in different ways. In native HDFS the data files are written to datanode directly without considering the size of data files. So we introduce structure model where the data files are gone through an intermediate step in which data files are first find out to be small files or large file. If the data files are small files then the small files are merged to a large merged file. A local index file is built for a merged file, and is also merged into the merged file. Then the merged file is written to HDFS. If data files are large enough than default HDFS block size then the general data pipeline is maintained. Thus the efficiency of accessing massive small file is significantly increased, when reading files. The client needs to query namenode for file metadata. According to the metadata, the client connects with appropriate datanodes where blocks locate. When the local index file is firstly read, based on the offset and length, the requested file is split from the block, and is returned to the client.

In this scheme we merge the small files first then we want to write merged file to datanode. For merging process, a computational machine is needed. So we introduce a fast-node to do these merging step. This fast-node is highly configured machine that easily merge the small files in short time.

Table 1. File size vs run time

F (KB)	T_m (ms)	T_R (ms)
10	3671	3501
50	3854	3594
100	3994	3651
200	4221	3846
500	5715	4713

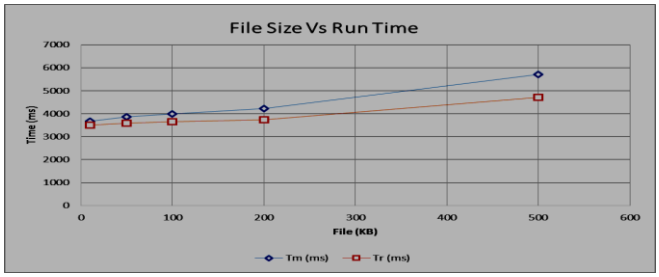

Fig 4: Plloting graph for different file size of file and their run time

4. EVALUATION

In order to evaluate performance of word count application on small files on single node cluster, experiments have been performed within single node cluster. To investigate the efficiency of the proposed HDFS structure model, we also show some mathematical and graphical explanations. The test perform on single node cluster and this paper has mainly tested read and write operations.

My machine's specifications are Intel Core i5 2.26 GHz, 64-bit, 2GB of RAM and running Ubuntu 14.04 LTS operating system. Hadoop 2.4.0 version is installed on it with default settings.

Performing word count MapReduce application in single cluster Hadoop for various file size we get the following times that are spent by map task and reduce task

Here, F, Tm and Tr indicate text file size in kilobyte, total time spent by all map tasks in millisecond and total time spent by all reduce tasks in millisecond, respectively. Plotting the previous table values, we get the following two curves in the graph.

From the experiment resulted graph, we can say that the total spent times by both map and reduce task are insignificantly increased according to increasing file size.

Our next experiment is to run word count application to count distinct words occurring times on 1000 files with the size of 100bytes each and then again run word count application on a whole one merged file of the size 100KB. In such way, we've also perform the test for 200KB and 300KB size. The resulted data is given bellow in the table.

Table 2. Comparison of reading time

File Size	1000 files (Sec.)	1 file (Sec.)
100KB	1956	7.7
200KB	2257	8.1
300KB	2548	8.9

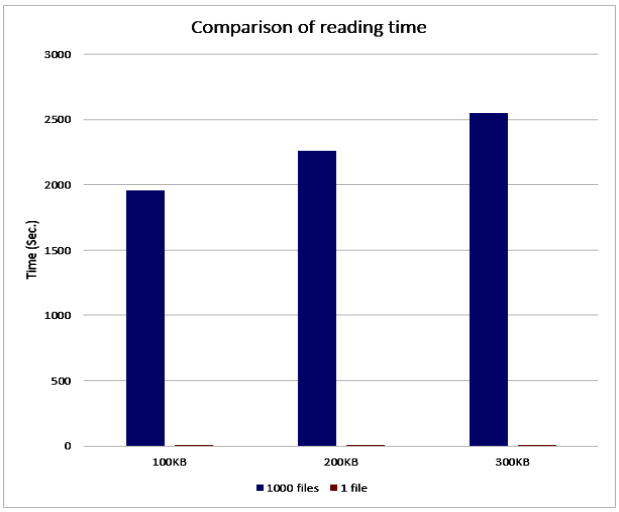

Fig 4: Plotting graph for comparison of reading time

We plot the resulted data and compare the reading time for 1000 files and for 1 file of the total same size. The graph shows that how the processing of massive small files affect the system greatly and degrade the performance of HDFS. It also indicates that the processing of the large merged file consumes very lower reading time. Thus, our proposed HDFS structure model can efficiently improve the storage and access efficiencies of massive small files effectively on HDFS.

Lastly, in our proposed HDFS structure model, we design a data pipeline that merge the small files first and then write merged file to datanode directly unlike storing the small files into HDFS first after then merged it to merged file which takes double times to maintain. First time, by storing small files on HDFS already creates burden on namenode uploading metadata. Writing massive small files on HDFS would be very slow. Second time, merging massive small files locally and store them again in HDFS would take higher time than our proposed method. In our proposed method, it takes high time to writing massive small files than the native HDFS but it reduces the total time needed by two steps in native HDFS. It also reduce the burden on namenode greatly uploading very smaller metadata.

5. CONCLUSION AND FUTURE WORK

We identified the small file problem in HDFS (Hadoop Distributed File System) and proposed a HDFS structure model that can process massive number of small files better. The experimental analysis indicates that the proposed method can efficiently reduce the burden of HDFS and improve the read and write performance of massive small files.

Our future research will focus on compressing the data on Hadoop cluster using different compression algorithms. In order to improve the performance of heterogeneous cluster, we will also research various data placement techniques.

6. ACKNOWLEDGMENTS

The authors are grateful to the participants who contributed to this research.

7. REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org. [Last accessed: 20th December 2014]

[2] Hadoop Wiki. https://wiki.apache.org/hadoop/PoweredBy#M. [Last accessed 20th December 2014]

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[4] Cloudera. http://blog.cloudera.com/blog/2009/02/the-small-files-problem. [Last accessed: 20th December 2014]

[5] Chuck Lam, "Hadoop in Action", Manning Publications, 2011.

[6] Apache Hadoop. http://hadoop.apache.org/docs/stable2/hadoop-project-dist/hadoop-common/SingleNodeSetup.html. [Last accessed: 10th April 2014].

[7] "Running Hadoop On Ubuntu Linux (Single-Node Cluster) - Michael G. Noll." [Online]. Available: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/. [Last accessed: 20th May 2014].

[8] N. Mirajkar, S. Bhujbal, and A. Deshmukh, "Perform wordcount Map-Reduce Job in Single Node Apache Hadoop cluster and compress data using Lempel-Ziv-Oberhumer (LZO) algorithm," *arXiv:1307.1517 [cs]*, July 2013.

[9] N. Mirajkar, S. Bhujbal, and A. Deshmukh, "Perform wordcount Map-Reduce Job in Single Node Apache Hadoop cluster and compress data using Lempel-Ziv-Oberhumer (LZO) algorithm," *arXiv:1307.1517 [cs]*, July 2013.

[10] B. Dong, Q. Zheng, F. Tian, K.-M. Chao, R. Ma, and R. Anane, "An optimized approach for storing and accessing small files on cloud storage," *Journal of Network and Computer Applications*, vol. 35, no. 6, pp. 1847–1862, Nov. 2012.

[11] Y. Zhang and D. Liu, "Improving the Efficiency of Storing for Small Files in HDFS," in *2012 International Conference on Computer Science Service System (CSSS)*, 2012, pp. 2239–2242.

[12] "Welcome to Apache™ Hadoop®!". http://hadoop.apac [Last accessed: 20th December 2014]

[13] he.org/docs/r2.4.0/. [Last accessed: 5th July 2014]