2012 International Symposium on Cloud and Services Computing

A Scalable Cloud Platform using Matlab Distributed Computing Server Integrated with HDFS

Rahul Dutta[1], Annappa B[2]

[#]*Department of Computer Science & Engineering, National Institute of Technology Karnataka, Surathkal, India*

[1]dutta.rd@gmail.com, [2]annappa@ieee.org

***Abstract*— The Hadoop Distributed File System (HDFS) is a large data storage system which exhibits several features of a good distributed file system. In this paper we integrate Matlab Distributed Computing Server (MDCS) with HDFS to build a scalable, efficient platform for scientific computations. We use an FTP server on top of HDFS for data transfer from the Matlab system to HDFS. The motivation of using HDFS for storage with MDCS is to provide an efficient, fault-tolerant file system and also to utilize the resources efficiently by making each system serve as both data node for HDFS and worker for MDCS. We test the storage efficiency of HDFS and compare with normal file system for data transfer operations through MDCS.**

***Keywords*— HDFS, Matlab, Distributed Computing, PaaS.**

I. INTRODUCTION

In today's cloud computing scenario, there is a need to achieve high computational speed and efficient storage. The Hadoop Distributed File system (HDFS) demonstrated several desirable features for scalable cloud storage. With the parallel computation tool and Distributed Computing Server in Matlab, it offers a highly efficient and scalable platform for computationally intensive applications. Matlab provides high-performance computational routines for complex calculations and an easy-to-use scripting language. Hence an integration of Matlab, along with HDFS would provide an excellent platform for applications where scalability is an issue in terms of both computation speed and storage.

HDFS achieves efficient storage space utilization through compression of stored data. The HDFS architecture also provides fast access to large amount of data distributed across several systems and provides high fault tolerance and scalability. Matlab, on the other hand allows fast and efficient distributed processing of computationally intensive applications. With parallel computing toolbox, the task of parallel processing of scientific computations becomes easy.

Integration of HDFS with Matlab will provide an ideal platform for applications which has very high storage and computation speed requirement. Examples of such applications are large-scale image processing, geo-spatial data processing and bioinformatics.

The Hadoop Distributed File System was originally designed to work with the MapReduce framework for huge data analysis. However, HDFS demonstrates several features which are desirable for distributed storage of enormous data. Thus, HDFS can be a good choice for use with other applications for distributed and fault-tolerant storage, over ordinary file system. However, the performance of such a system needs to be carefully analysed.

The rest of the paper is arranged as follows. Section II provides an overview of HDFS and MDCS architecture. Section III introduces our concept of the integrated system. Section IV analyses and compares the results of the system followed by the concluding remarks in section V.

II. HDFS AND MDCS ARCHITECTURE

This section discusses the architecture of HDFS and MDCS and highlights the features of these architectures. We look at the factors that determine a good storage system and how HDFS tries to achieve them. We then discuss about the MDCS architecture and functioning.

A. HDFS Architecture

The HDFS architecture[11] consists of the following components:

1. Name Node – The name node stores the metadata information of the cluster. It contains the metadata along with enumeration of blocks of HDFS and a list of data nodes in the cluster.
2. Data Nodes - The data nodes are the actual sites where the HDFS blocks are stored. The data nodes are connected to the name node and periodically send heartbeat signals to inform their status to the name node.
3. Secondary Name Node – The secondary name node contains a snapshot of the primary name node and in case of name node failure, the secondary name node is used.

The architecture of HDFS provides many desirable features of a good distributed file system.

1. Fault tolerance – Each data block in HDFS is replicated on multiple data nodes across the HDFS cluster. The replication factor determines the number of replica of each data block. In case of a node failure, HDFS is able to switch to different data nodes to retrieve the data. This provides reliability and availability.
2. Efficient storage space utilization – All data blocks in HDFS are compressed using Gzip algorithm by default. This reduces the size of data being stored in HDFS and better utilization of storage space.
3. Fast I/O operations – The HDFS blocks are indexed at the name node, and the name node contains the

location of the data nodes. Thus they can be located very easily and retrieved faster.

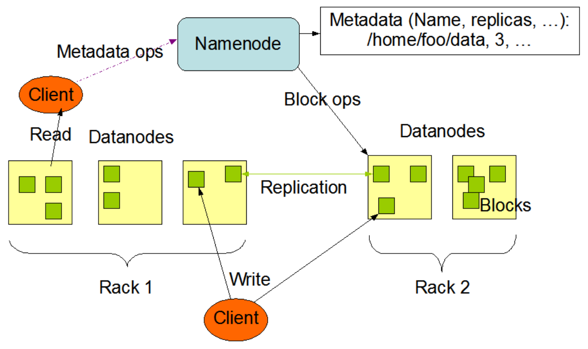

Fig. 1 HDFS Architecture

B. *MDCS Architecture*

The Matlab Distributed Computing Server[9] consists of the following components.

1. Matlab Client with parallel computing toolbox – This is the node where the application is written and executed. It must have the MDCS service installed.
2. Matlab job Scheduler – This scheduler schedules various Matlab jobs to the available worker nodes. The scheduler is normally configured on the Matlab client machine.
3. Worker nodes – These are the computers in the cluster that perform the computing. All the worker nodes must have MDCS installed and running. Multiple worker nodes can be configured on a single computer with multi-core CPU. A maximum of 12 workers can be configured on a single PC.

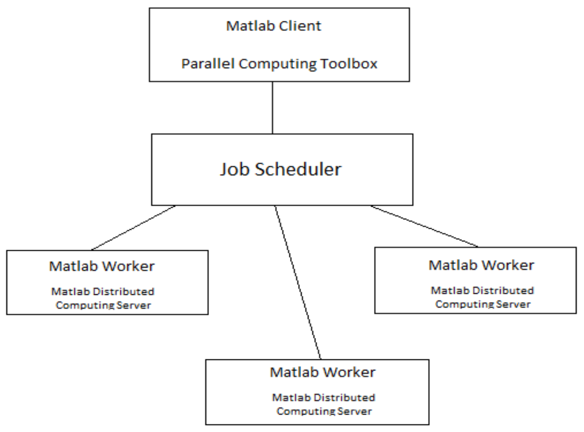
Matlab Client
Parallel Computing Toolbox
Job Scheduler
Matlab Worker
Matlab Distributed Computing Server
Matlab Worker
Matlab Distributed Computing Server
Matlab Worker
Matlab Distributed Computing Server

Fig. 2 MDCS Architecture

III. PROPOSED ARCHITECTURE

In our proposed architecture we configure the HDFS cluster and implement an FTP server on the name node. This FTP server provides an interface for data transfer by outside applications. Applications written in Matlab will access HDFS through this FTP server.

In order to integrate Matlab with HDFS, a common interface for data exchange is needed. For both the systems to work with each other, a communication channel is required that will allow Matlab to retrieve and store data into HDFS clusters. The solution to this is to use FTP as a channel for communication. HDFS can be accessed over FTP and Matlab can be used to store and retrieve files using FTP. The FTP server will typically runs on the name node machine of HDFS. The Matlab system will act as the FTP client and access data via the FTP server. In this system, Matlab will be able to directly interact with HDFS for data retrieval and update.

In our proposed system, we configure data nodes of the HDFS cluster and the Matlab workers on the same systems. Such configuration allows better utilization of resources as well. The name node, FTP server, Matlab client and the job scheduler are configured on the same system. This is done for better utilization of resources. However, this configuration can be changed and each component can be located on a different system. However, since the FTP server interacts directly with the name node, these two should run on the same system to avoid performance issues. This architecture is shown in fig. 3.

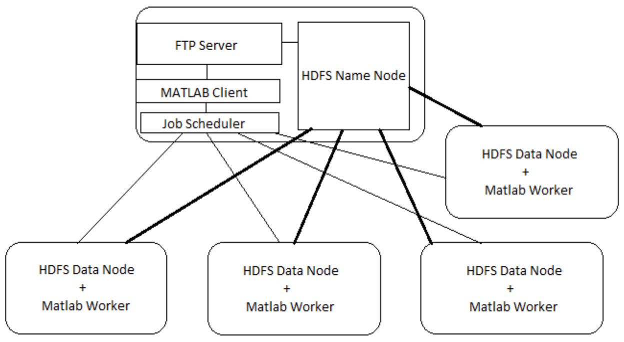
FTP Server
HDFS Name Node
MATLAB Client
Job Scheduler
HDFS Data Node + Matlab Worker
HDFS Data Node + Matlab Worker
HDFS Data Node + Matlab Worker
HDFS Data Node + Matlab Worker

Fig. 3 Proposed Architecture

IV. IMPLEMENTATION

The proposed system has been implemented and tested in two ways. In the first case, we configure all the components on a single node cluster and test the I/O characteristics. In the second case we configure four nodes as HDFS Data Node + Matlab worker and carry out the testing. Each of the nodes consist of Core i7 2.93 GHz CPU, 8 GB RAM, 500 GBx2 Hard disk and Ubuntu Linux 12.04 LTS. For network connections, we use D-Link DES-3526 Fast Ethernet switch. For implementation we use Hadoop v1.0.3 and Matlab R2012a. In each of the tests, we considered the following three types of data.

1) *Single data file of medium size.*

2) *Single data file of large size.*

3) *800 image files (approx. 1 MB each) total 1 GB.*

Data transfer rates and the size of the data in HDFS are compared with normal (ext4) file system. The FTP hosts are mounted as file systems using curlftpfs [6] and the Matlab functions 'textscan' and 'imread' for reading and 'fprintf' and 'imwrite' to write data from Matlab.

A. Test Case 1 (single node)

In this test, all the components (name node, data node, Matlab cluster, job scheduler) are configured on a single system. We set the replication factor as 1 in HDFS (since it is a single node cluster) and perform data transfer (read and write) operations from Matlab. We consider the above mentioned types of loads and repeat each experiment for 10 times. We repeat the experiment for different block size of HDFS and measure the time for data transfer and size of data in HDFS. We also conduct experiment based on normal file system with Matlab to compare and contrast with HDFS.

B. Test Case 2 (multiple nodes)

We repeat the same experiments as in test case 1 for 4 data nodes + Matlab workers. Replication factor in HDFS is set to 3. All the nodes are connected using D-Link Fast Ethernet 10/100 Mbps switch. We also conduct experiment with FTP server on remote system and compare the result with HDFS. We compare the file sizes and data transfer times. We calculate the performance improvement over normal file system.

V. Result and Analysis

Table 1 shows the file size for different block sizes in HDFS and comparison with the file size in normal (ext4) file system. This result is common for both test case 1 and test case 2.

TABLE 1
SIZE OF FILES ON HDFS AND EXT4

File type	Size of single replica		
	HDFS (16 MB block size)	HDFS (512 MB block size)	EXT4 file system
Data file (single)	685.29	685.29	718.6 MB
Data file (single)	4.97 GB	4.97 GB	5.3 GB
800 image files	955 MB	955 MB	1 GB

The amounts of compression achieved in the three cases are 4.63%, 6.22% and 6.73%. The compression ratio is invariant to the following factors:

1. *HDFS block size*
2. *Cluster size*

Thus, using HDFS instead of normal file system leads to better space utilization. The compression ratio is best for small files.

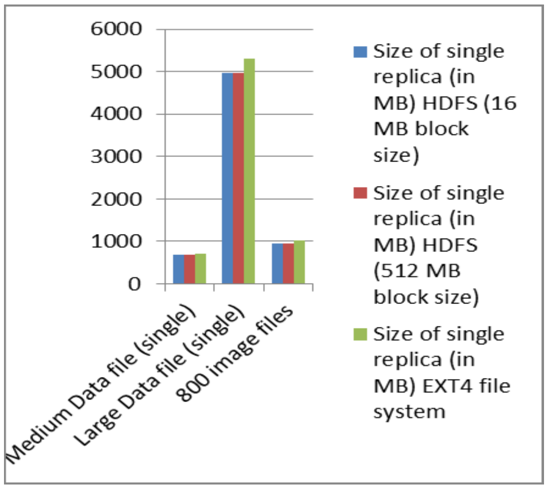

Fig.3 File size comparison between HDFS and EXT4

The time taken for data transfer operations in both the test cases is shown below. Table II lists the time for read operations and Table III lists the time taken for write in single node and cluster of size 4 respectively. All results are average over ten trials of each experiment.

TABLE II
TIME TAKEN FOR READ OPERATIONS

	File type	Read Time (in seconds)		
		HDFS (16 MB block size)	HDFS (512 MB block size)	Remote EXT4 file system
Single Node	Single data file of 718.6 MB	12.20	11.64	12.35
	Single Data file of 5.3 GB	86.47	82.56	87.54
	800 image files (1 GB)	19.25	16.33	15.36
4 Nodes Cluster	Single data file of 718.6 MB	62.55	61.99	64.33
	Single Data file of 5.3 GB	465.42	460.66	469.43
	800 image files (1 GB)	70.06	68.61	58.8

From Table II, we observe that HDFS gives slightly better performance for large files but performance is not better in case of multiple smaller files. A larger HDFS block size improves the read time than smaller block size. In case of large files, bigger block size reduces the total number of blocks and this improves the reading time of the file. Since each data block is located on a single node, larger block size reduces the total number of blocks and hence improves the reading time. Smaller block size causes lot of fragments of large files and retrieving each fragment from different data nodes cause overhead which reduces the performance.

In case of both single node and 4 node cluster, the highest performance gain is observed for single data file of 5.3 GB. The read time is reduced by 5.6 % in single node system and for the 4 node cluster; it is reduced by 1.8 %.

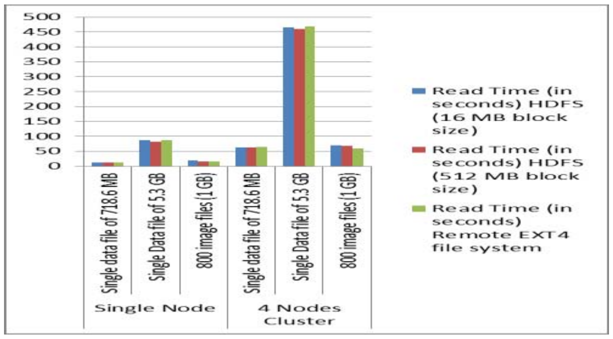

Fig.5 Comparison between HDFS and EXT4 for read operation

TABLE IIII
TIME TAKEN FOR WRITE OPERATIONS

	File type	Write Time (in seconds)		
		HDFS (16 MB block size)	HDFS (512 MB block size)	Remote EXT4 file system
Single Node	Single data file of 718.6 MB	13.39	12.09	13.03
	Single Data file of 5.3 GB	100.5	93.52	110.36
	800 image files (1 GB)	18.26	16.54	15.34
4 Nodes Cluster	Single data file of 718.6 MB	86.28	78.52	124.02
	Single Data file of 5.3 GB	624.81	580.96	623.71
	800 image files (1 GB)	122.55	121.93	97.4

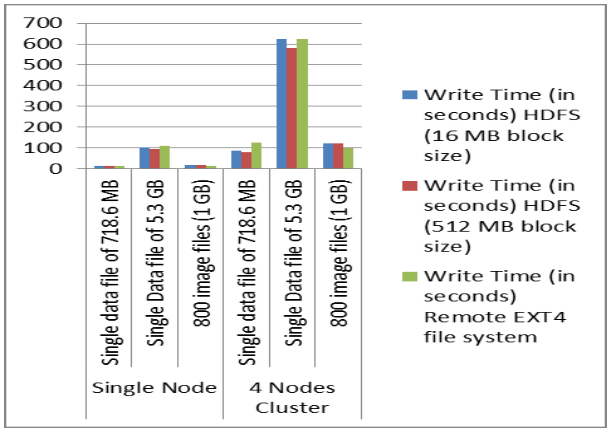

Fig.6 Comparison between HDFS and EXT4 for write operation

Compared to the read operations, in case of write operation, HDFS gives better performance over EXT4. The main reason for this is that HDFS writes out the data blocks concurrently on different data nodes. For a single node system as well as 4 node cluster, HDFS give best performance for single data file of 5.3 GB. In case of a single node, it gives 15.34 % better performance and for the 4 node cluster, it gives 6.85 % better performance.

VI. Conclusions

Our experiments show that HDFS provides better I/O speed with large datasets and files in both standalone and cluster configuration. The performance is however, marginally poor for small files. HDFS provides high degree of fault tolerance, which is transparent to the application (Matlab) running over it. In case of any node failure, Matlab can also detect the worker node failure and continue using other worker nodes. HDFS provides efficient storage space utilization using compression for all types of data and file. This is an essential feature when using large datasets for processing. The proposed system can be easily scaled-up or down by adding nodes and configuring them. It does not need any changes to be made to the existing Matlab applications or data. The addition or removal of nodes or storage space is transparent to the Matlab applications. The proposed system is thus elastic in nature. Our proposed system can be used as a platform for very large dataset processing in scientific computations.

In future it may be possible to further improve the I/O characteristics by modifying various parameters of HDFS. It may also be possible to have better compression by changing the underlying compression methods in HDFS.

References

[1] Tom White, “Hadoop: The Definitive Guide”, First Edition, Yahoo press, June 2009.
[2] Cong Wang, et. al, “Towards Secure and Dependable Storage Services in Cloud Computing”, IEEE transactions on service computing, Vol. 5, No. 2, pp. 220-232, April-June 2012.

[3] Yongqiang He, et. al, “RCFile: A Fast and Space-efficient Data Placement Structure in MapReduce-based Warehouse Systems,” *IEEE ICDE conference.*, 2011.

[4] Da-Wei Zhang, “Research on hadoop-based enterprise file cloud storage system,” 3rd International Conference on Awareness Science and Technology, pp. 434-437, Sept. 2011.

[5] Yanmei Huo, “A Cloud Storage Architecture Model for Data-Intensive Applications,” International Conference on Computer and Management, pp. 1-4, May 2011.

[6] Curlftpfs (Oct. 2012) website. [Online]. Available: http://curlftpfs.sourceforge.net/

[7] Mathworks. (2012). [Online]. Available: http://www.mathworks.in/support/product/DM/installation/ver_current/

[8] Choy, R., “Parallel MATLAB: Doing it Right”, Proceedings of the IEEE, Vol. 93, Issue 2, pp. 331-341.

[9] Matlab (2012). [Online]. Available: Mathworks. (2012). [Online]. Available: http://www.mathworks.in/support/product/DM/installation/ver_current/

[10] Raeth, P.G, “Parallel MATLAB Using Standard MPI Implementations”, High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), pp. 438-441, June 2010.

[11] Apache Hadoop project (2012). [Online]. Available: http://hadoop.apache.org/

[12] Kapil Bakshi, “Considerations for Big Data: Architecture and Approach”, IEEE Aerospace Conference, pp. 1-7, March 2012.