T-FUSE
IMPROVING HADOOP THROUGH FUSIONFS

Alekya Thalari, Krishnaja Kethireddy, Nirmal Kumar Ravi, Prathamesh Mantri
{athalari, kkethire, nravi, pmantri2}@hawk.iit.edu
Department of Computer Science
Illinois Institute of Technology
Chicago, IL

ABSTRACT

Hadoop is an open-source software framework for distributed storage and distributed processing of massive data sets on large computer clusters [1]. Hadoop stores all the files on Hadoop Distributed File System (HDFS [2]). The major drawback of HDFS is that it stores all the metadata of its files on a single node called Name Node. When we have large number of small files the size of metadata might increase in such a way that it may not fit on a single node. And also if there are thousands of meta-data requests from the user, then the server might not be able to handle all of them. So HDFS cannot be used for many modern scientific applications like climatology, astronomy [3] are becoming metadata intensive and requires more support from the storage subsystem [4]. Therefore we use FusionFs [5], a Distributed File System which was designed to handle metadata intensive workloads efficiently. In this paper we are going to use FusionFS as our distributed storage engine and implement Toy Hadoop - a MapReduce framework to process small files efficiently.

Keywords
Hadoop, HDFS, MapReduce, FusionFS, Metadata, Name Node

1.1 INTRODUCTION

The conventional Hadoop doesn't support many modern scientific metadata intensive applications and it has a limitation dealing with something called small files problem. This problem drastically reduces the efficiency of Hadoop when dealing with files of size less than 10MB.

Fig.-1 shows the time taken to run a typical MapReduce-WordCount program on a dataset of 10GB which was divided into equal files of size 1KB, 10KB, 100KB, 1MB, 10MB etc. The observation was that as the size of files decreases, the time taken to run the program increases.

Fig.-2 shows that as the size of files decreases, the no. of mappers assigned by the Job Tracker increases and the communication overhead between Job Tracker and Task Trackers also increases. This degrades the performance of Hadoop significantly. Therefore we came up with a novel approach i.e., implementation of system called ToyHadoop to efficiently deal with small files problem.

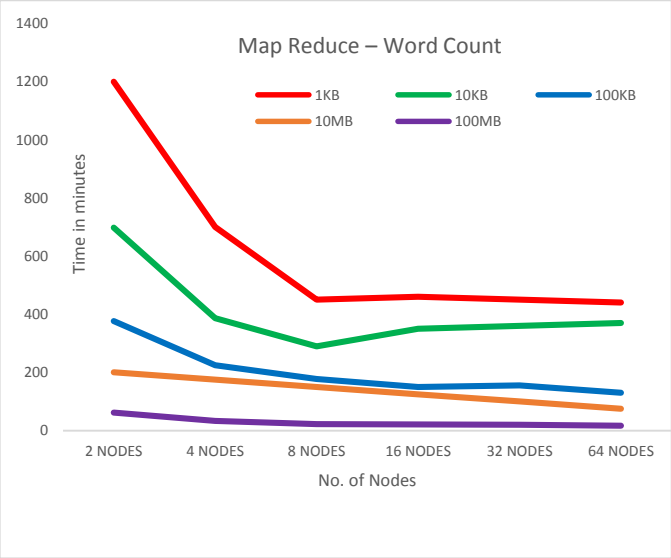

Fig.-1: Effect of Small Files Problem

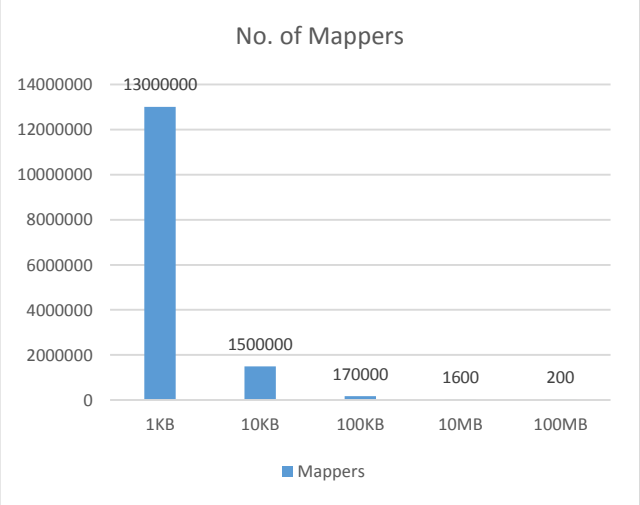

Fig.-2: Reason behind Small Files Problem

1.2 BACKGROUND

Hadoop is an open source implementation of MapReduce [6] which uses the HDFS as its underlying layer. Basically the

Hadoop core is divided into two fundamental layers: MapReduce and HDFS. MapReduce is the computation engine running on top of HDFS, which is its data store manager. An important characteristic of Hadoop is partitioning of data and computation across many thousands of hosts and executing application computations in parallel close to their data.

HDFS, which is an inspiration from Google File System [7], is the file system component of Hadoop. It stores file metadata and application data separately on Name Node and Data Node. Name Node contains the namespaces of hierarchy of files, directories and their mapping. Data Node connects to Name Node and performs handshake to verify NameSpaceID.

Where FusionFS is a Distributed File System designed in such a way that it contains a distributed storage layer local to compute node and enables every compute node to actively participate in both the metadata and data movement. The Client or application will be able to access global namespace of the file system with a distributed metadata service on same node. FusionFS has different data structures for managing regular files and directories. For a regular file, the field addr stores the node where this file resides. For a directory, the field filelist records all the entries under this directory. This filelist field is particularly useful for providing in-memory speed for directory read, the idea is to save extreme amount of data movement between storage and compute node during I/O operations. It a Distributed Hash Table (DHT [8]) called Zero-Hop Distributed Hash Table (ZHT [9]) as its underlying building block for metadata management which is basically a Key-Value Store [10]. It also maintains consistency, because the first replica is strongly consistent with primary copy while the other replicas are asynchronously updated.

1.3 MOTIVATION

In Hadoop, the data is divided into chunks of size 64MB. Suppose if the files sizes are very less, say around 10KB and there are a lot of such files, then the problem is that Hadoop can't handle lots of such files efficiently. It was designed to handle only large datasets. In general, each map task can process only one block of input at a time. Here in our case, as shown in Fig.-3, each map task processes very small input and requires a lot of map tasks. Which results in extra bookkeeping overhead. Moreover, retrieving each small file incurs lots of seeks and hops from data node to data node [11]. The time taken to finish smaller jobs will be higher when compared to larger ones and this is called Small Files Problem in Hadoop. So in this paper we handle this situation in such a way that the performance increases even if the files are small.

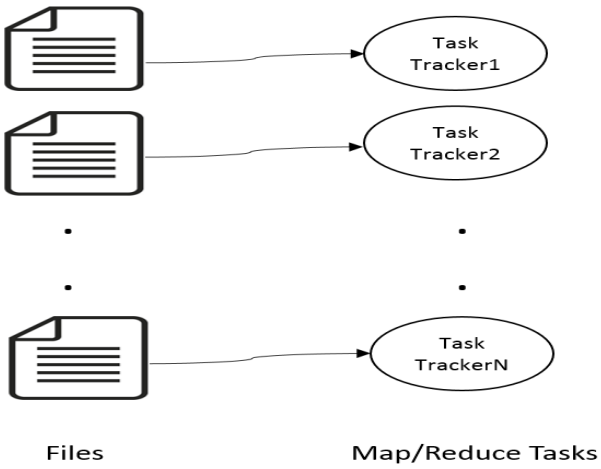

Fig.-3: Small Files Problem

2.1 PROPOSED SOLUTION

In this paper we are going to introduce a new distributed compute engine called ToyHadoop, which can run on any distributed file system. The purpose of this implementation is current Apache Hadoop project suffers handling files with size less than 10 MB. The Apache Hadoop project was implemented with assumption to run on large files, at least of size 64MB. But many scientific applications has files less than this size. To solve this issue we are going to implement our own compute engine called **ToyHadoop (T-Hadoop)**. We call it as ToyHadoop because it doesn't have all the features Apache Hadoop has. For e.g. Hive, Pig, Failure Handling, Security etc. Adding these components to our T-Hadoop will be our future work. To keep our project simple we are going to follow the same design as of Apache Hadoop but, do modifications which makes our T-Hadoop run faster for small files. Our expectation is, T-Hadoop will outperform Apache Hadoop on datasets with smaller file size.

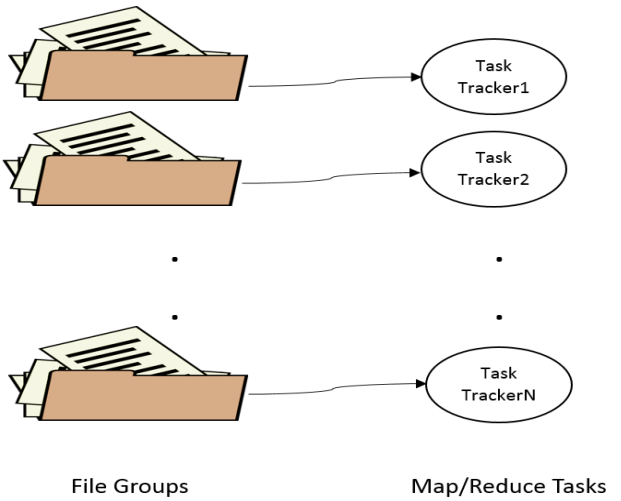

Fig.-4: Grouping Files - Solution to Small Files Problem

We are going to use FusionFs - a distributed file store as our underlying distributed storage system. The reason being FusionFs is that it is built for meta-data intensive applications, as we are going to have large number of small files our compute engine needs to query distributed storage FusionFs, often to get information about files.

2.2 ARCHITECTURE

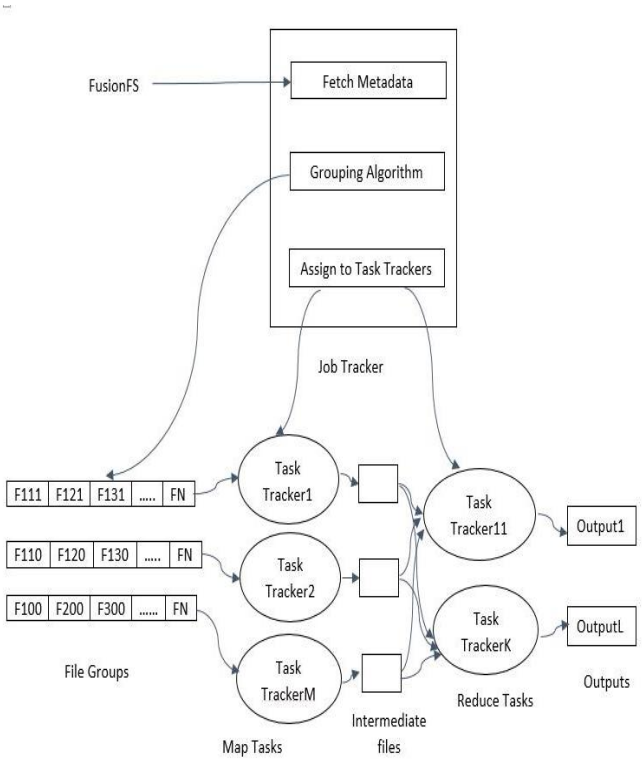

Fig.-5: System Architecture

As stated before, the main goal of T-Hadoop is to deal with small files problem efficiently. In Apache Hadoop implementation mappers are mapped one to one to 64MB file blocks. We can't do the same in this situation as it would put more workload on Job Tracker. So we came up with something called **Grouping Algorithm** to handle this problem. Let's say a user submits a job request to Job Tracker. Next the Job Tracker queries for metadata information from FusionFS. Once it has all information about the files it needs to process, instead of launching one Task Tracker for each file or block, it will assign Map/Reduce tasks on sets of Files, as shown in Fig.-4. This will considerably reduce the traffic between Job Tracker and Task Tracker. Hence Task Tracker will now respond to Job Tracker only after completion of all files assigned to it. This increases the performance of the entire system. The whole Architecture of the system is illustrated in Fig.-5. Once the Task Tracker completes its tasks, then it reports to Job Tracker and waits for assignment of another task.

The three main components of our system are:

a. JOB TRACKER:

This is the Master server of our T-Hadoop. Client submits jobs to Job Tracker and it is responsible for executing the job and produce the output. There will be only one Job Tracker for a cluster.

b. TASK TRACKER:

There can be up to N Task Trackers in a cluster, where N corresponds to number of nodes in a cluster. Task Tracker is responsible for running Map/Reduce on the assigned to them. It reports to Job Tracker after the completion of tasks. Typically a task can be a Mapper or a Reducer task.

c. GROUPING ALGORITHM:

- Query for available nodes.
- Compute Total size of files from the fetched metadata.
- Compute required number of nodes.
- If required nodes < available nodes
  Then pick nodes based on the availability of the data.
- If required nodes > available nodes
  Then perform computation on available nodes and wait until the nodes are available.

IMPLEMENTATION:

The implementation of Toy Hadoop was done from the scratch in Java and FusionFS was implemented in C/C++. So, to integrate Toy Hadoop and FusionFS we had to write a Java wrapper class in order to invoke required methods in FusionFS. And we have used GitHub for version control.

3. EVALUATION

We have evaluated Toy Hadoop against Apache Hadoop using a dataset of 10GB. We did split that dataset into different datasets with files of size 1KB, 10KB, 100KB, 10 MB and 100MB. Then we ran MapReduce-WordCount program on those five datasets on Amazon EC2 on different nodes up to 64.

Test Bed	Amazon EC2
Scaling	2, 4, 8, 16, 32, 64
Instance Type (Master)	M3.large
Instance Type (Worker)	M3.medium

We have also evaluated Toy Hadoop in three categories:

- Apache Hadoop Vs Toy Hadoop on different datasets
- Toy Hadoop with different file systems FusionFS Vs HDFS Vs S3FS
- No. of Mappers in Toy Hadoop Vs Apache Hadoop

1. Apache Hadoop Vs Toy Hadoop on different dataset

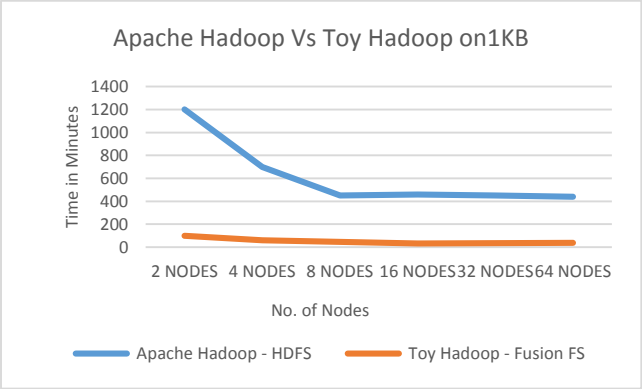

Fig.-6.1: Apache Hadoop Vs Toy Hadoop - 1KB

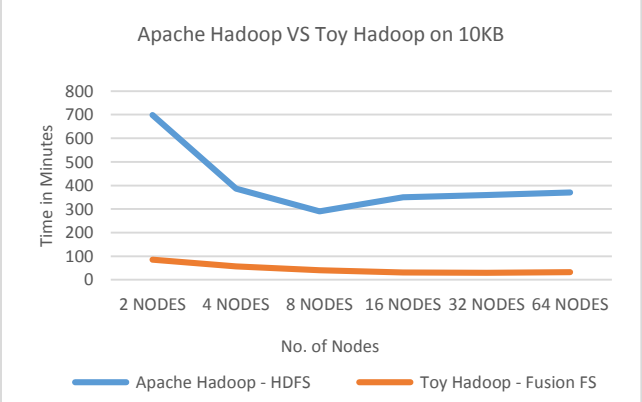

Fig.-6.2: Apache Hadoop Vs Toy Hadoop - 10KB

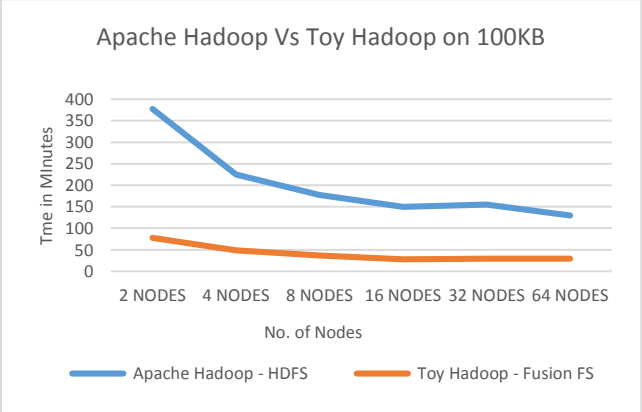

Fig.-6.3: Apache Hadoop Vs Toy Hadoop - 100KB

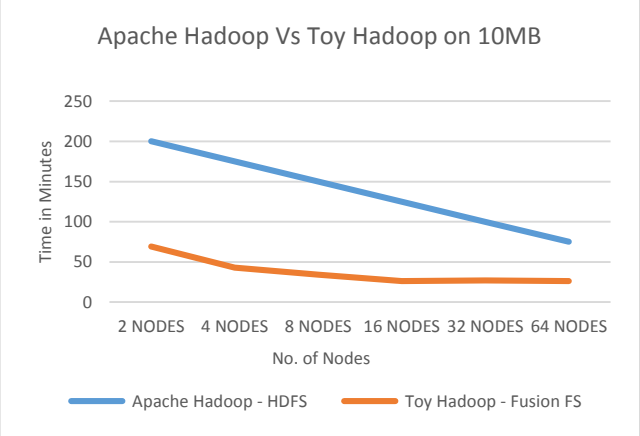

Fig.-6.4: Apache Hadoop Vs Toy Hadoop - 10MB

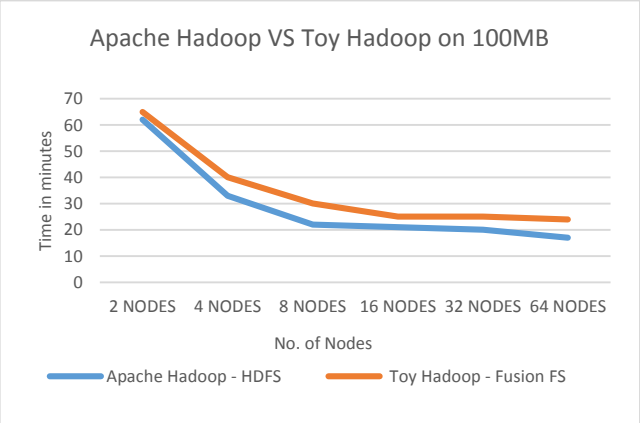

Fig.-6.5: Apache Hadoop Vs Toy Hadoop - 100MB

From Fig.-6.1 we can observe that, when the file size is 1KB, the time taken by Apache Hadoop is 1200 minutes on 2 nodes whereas the Time taken by Toy Hadoop is only 100 minutes. We can observe the same scenario from Fig. [6.2-6.4] as well. The thing to observe here is, as the file size increases, the time taken by Apache Hadoop decreases. In Fig.-6.2 when the file size is 10 KB, it took 698 minutes on 2 nodes. In Fig.-6.3 also, when the file size is 100KB, it took 377 minutes. But in Fig.-6.4 i.e., when the size of files is greater than 64 MB, Apache Hadoop took very less. Therefore it was designed only to handle large datasets efficiently. But if you observe from Fig.-[6.1-6.5], ToyHadoop took almost the same amount of time for files of all sizes. Hence we can say that, our Toy Hadoop works efficiently regardless of the file size. Therefore through our implementation, we handled the small files problem in Apache Hadoop by grouping the files efficiently.

2. Toy Hadoop with different file systems FusionFS Vs HDFS Vs S3FS

Below graphs show the comparison of Toy Hadoop on top of FusionFS, HDFS and S3FS.

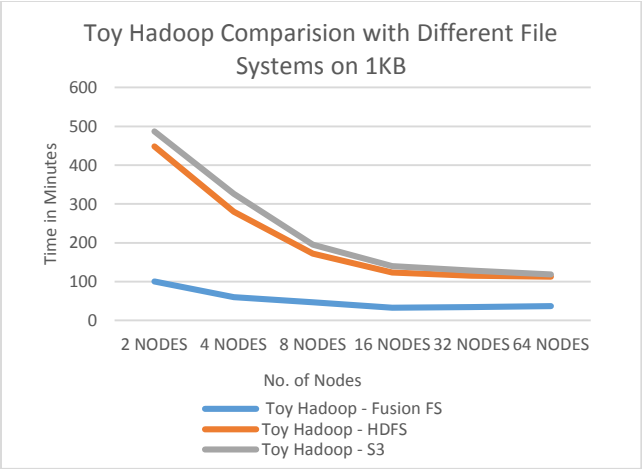

Fig.-7.1: Comparing Toy Hadoop with FusionFS, HDFS, S3FS - 1KB

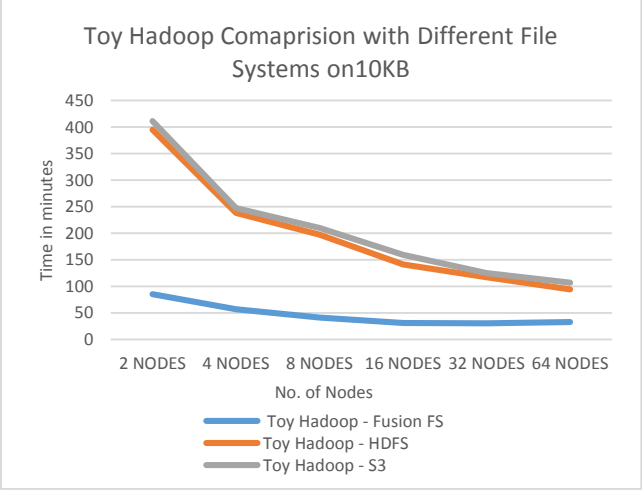

Fig.-7.2: Comparing Toy Hadoop with FusionFS, HDFS, S3FS - 10KB

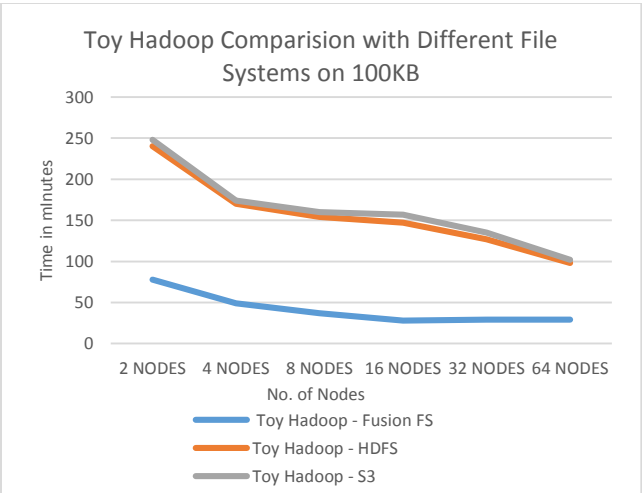

Fig.-7.3: Comparing Toy Hadoop with FusionFS, HDFS, S3FS - 100KB

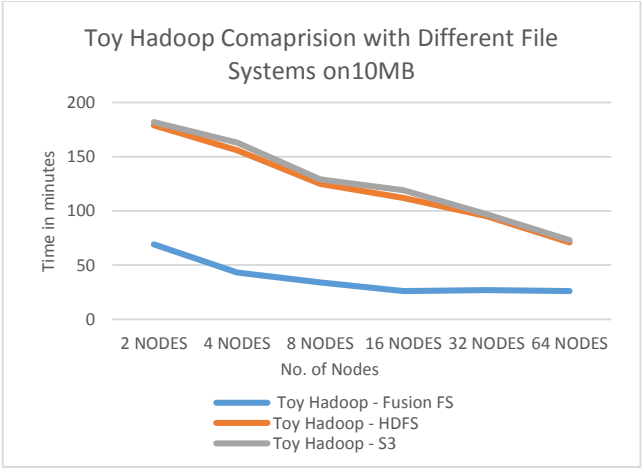

Fig.-7.4: Comparing Toy Hadoop with FusionFS, HDFS, S3FS - 10MB

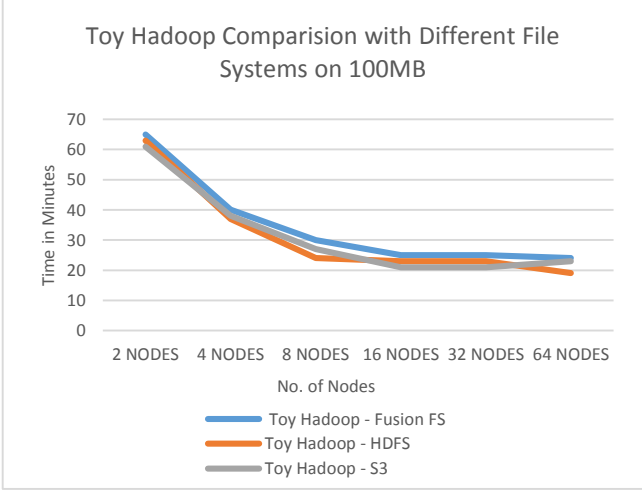

Fig.-7.5: Comparing Toy Hadoop with FusionFS, HDFS, S3FS - 100MB

The graphs from Fig. [7.1- 7.5] shows the time taken to run Map Reduce program on different file system The observation from the above graphs is that Values for Toy-Hadoop – Fusionfs on the following datasets 1kb,10kb,100kb,10 mb works very efficiently on all nodes because Fusion fs has metadata management

But Values of Toy-Hadoop –S3 and Toy-Hadoop-HDFS are similar when compared for all data sets except 100MB because From

100Mb the metadata management is efficient and So T-Hadoop –with all file Systems works similarly.

Implementation	File System	Will Work better for smaller files less than 64MB? And Why?

Toy Hadoop	Fusion FS	Yes. Because of Grouping Algorithm and Metadata intensive file system
Toy Hadoop	HDFS/S3	Yes. Because of Grouping Algorithm
Apache Hadoop	HDFS	No. Because of Fixed 64MB Chunked size.

3. No. of Mappers in Apache Hadoop Vs Toy Hadoop

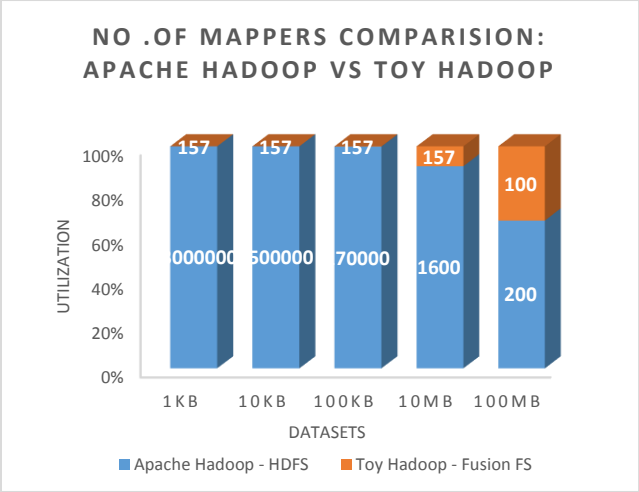

Fig.-8: Comparing No. of Mappers: Apache Hadoop Vs Toy Hadoop

In Fig.-8 we compare number of mappers launched on THadoop vs Apache Hadoop.

The number of mappers launched for Toy Hadoop is very less compared to Apache Hadoop. For a file size of 1 kb our Toy Hadoop launches 157 mappers. This continues for files with size 10 kb, 100 kb and 10 MB. But on the other hand Apache Hadoop launches 13000000 mappers for 1kb files and it get reduced on increase in file size.

For file size of 100MB our Toy Hadoop launches 100 mappers. This directly maps to number of files itself.

In all cases the number of mappers launched by our Toy Hadoop is lesser. This increase performance of the whole system by reducing communication overhead between job tracker and task tracker.

4. RELATED WORK

There are several approaches to deal with small files problem in Hadoop, such as Hadoop Archives (HAR) [12], SequenceFile [13], MapFile [14] etc.

HAR files was introduced to facilitate archiving small files together into HDFS blocks with .har as extension. These HAR files work as a layered file system on top of HDFS. This handles NameNode's memory usage problems but the major issue with HAR files is the Read operation. This makes it less efficient and much slower than HDFS, as it requires two index-file read and two data-file read operations. Moreover, creating an archive file generates a copy of original files and increases the consumption of disk space. Another issue is that, once an archive is created, we can't make any insertions to it and it involves recreation of a new HAR file.

SequenceFile is a persistent data structure for binary key-value pairs where the filename is used as a Key and its contents are used as Value. It converts all the existing small files into sequence files and they can be processed in a streaming fashion. The primary advantage of using SequenceFile is that, they can be split into chunks and can run MapReduce jobs on the chunks independently. Moreover, SequenceFile supports compression and decompression of files unlike HAR. But the problem is that, SequenceFile is Java-Centri and doesn't support cross-platform applications. Converting the existing files into SequenceFile is a time consuming process and there is no way to list all the keys in a SequenceFile.

A MapFile is nothing but a SequenceFile with sorted keys and maintains a partial index. It consists of an Index File and a Data File. The data file stores Key-Value pairs as records and are sorted in key order. The index file stores key location information, which is an offset of the first record containing the key which is located in the data file [15].

As all the above mentioned approaches have their own disadvantages, we came up with our own implementation of Toy Hadoop to handle small files problem.

Since, HDFS has several issues regarding metadata management, we need to use other distributed file systems such as Ceph [16], Luster [17], Sector [18] etc. But these file systems are tightly coupled with MapReduce framework and do not offer a POSIX interface, which is implemented with Fuse framework [19]. Moreover, the file systems that do not support POSIX interface are not designed for metadata-intensive operations.

So, we need other file systems such as xFS [20], FDS [21] etc. that came up with the idea of distributed metadata management. In xFS, though the metadata is distributed, it requires a central manager to locate a particular file. Whereas FDS maintains lightweight metadata server and offloads the metadata to available nodes in a distributed manner. In contrast, the metadata in FusionFS is completely distributed and was designed exclusively for metadata-intensive applications.

Therefore, we implemented Toy Hadoop on top of FusionFs, which is our underlying storage manager.

5. CONCLUSION

In this paper we have proposed a solution to handle small files problem efficiently through the implementation of Toy Hadoop - a MapReduce framework. We have identified the challenges faced by Apache Hadoop in dealing with small files and came up with Grouping Algorithm - a novel approach to group all small files together logically. We have also identified the challenges faced by HDFS in dealing with metadata management and learnt about various other distributed file systems and their approaches for distributed metadata management. Then we have implemented our Toy Hadoop on top of FusionFS to handle the problems faced by Hadoop and HDFS. Finally we have evaluated Toy Hadoop against Apache Hadoop and succeeded in outperforming Apache Hadoop while dealing with small files problem.

As part of our future work, the implementation of Toy Hadoop can be extended by including all the features supported by Apache Hadoop such as Failure Handling, Security Enhancements, Pig, Hive etc.

6. REFERENCES

[1] "Hadoop", https://hadoop.apache.org/

[2] K. Shvachko, H. Kuang, S. Radia and R. Chansler. "The Hadoop Distributed File System", in Proceedings of IEEE Symposium on Mass Storage Systems and Technologies, 2010.

[3] I. Raicu, I.Foster, A. Szalay and G. Turcu, "AstroPortal: A science gate-way for large-scale astronomy data analysis", in TeraGrid Conference, June 2006.

[4] P. Freeman, D. Crawford, S. Kim and J.Munoz, "Cyberinfrastructure for science and engineering: Promises and challenges", Proceedings of the IEEE, vol. 93, no. 3, 2005.

[5] D. Zhao, Z. Zhang, X. Zhou, T. Li, C. Shou and I. Raicu. "FusionFS: A Distributed File System for Extreme Scale Data-Intensive Computing", Under MSST13 review, 2013.

[6] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters", in proceedings of USENIX Sysmposium on Operating Systems Design & Implementation, 2004.

[7] S. Ghemawat, H. Gobioff and S.-T. Leung "The Google File System", in Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, 2003.

[8] T. Li, R. Verma, X. Duan, H. Jin and I. Raicu. "Exploring Distributed Hash Tables in Highend Computing", SIGMETRICS Perform. Eval. Rev., vol. 39, no. 3, Dec. 2011.

[9] T. Li, X. Zhou, K. Brandstatter, D. Zhao, K. Wang, A. Rajendran, Z. Zhang and I. Raicu. "ZHT: A Light-weight Reliable Persistent Dynamic Scalable Zero-hop Distributed Hash Table", in Proceedings of IEEE International Symposium on Parallel and Distributed Processing, 2013.

[10] D. Zhao, K. Burlingame, C. Debains, P. Alvarez-Tabio and I. Raicu. "Towards High-Performance and Cost-Effective Distributed Storage Systems with Information Dispersal Algorithms", in Cluster Computing, IEEE International Conference on, 2013.

[11] T. White. "The Small Files Problem" http://blog.cloudera.com/blog/2009/02/the-small-files-problem/

[12] "Hadoop Archives" http://hadoop.apache.org/docs/r1.2.1/hadoop_archives.html

[13] "SequenceFile" http://wiki.apache.org/hadoop/SequenceFile

[14] "MapFile" https://hadoop.apache.org/docs/current/api/org/apache/hadoop/io/MapFile.html

[15] J. Venner. "Pro Hadoop", Apress. June 209.

[16] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long and C. Maltzahn. "Ceph: A Scalable, High-Performance Distributed File System", in Proceedings of the 7th Symposium on Operating Systems Design and Implementation, 2006.

[17] P. Schwan. "Lustre: Building a file system for 1,000-node clusters", in Proceedings of linux symposium, 2003.

[18] Y. Gu, R. L. Grossman, A. Szalay and A. Thakar. "Distributing the Sloan Digital Sky Survey using UDT and Sector", in Proceedings of IEEE International Conference on e-Science and Grid Computing, 2006.

[19] "FUSE", http://fuse.sourceforge.net/

[20] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli and R. Y. Wang. "Serverless network file systems", in Proceedings of ACM symposium on Operating systems principles, 1995.

[21] E. B. Nightingale, J. Elson, J. Fan, O. Hofmann, J. Howell and Y. Suzue. "Flat Datacenter Storage", in Proceedings of USENIX Symposium on Operating Systems Design and Implementation, 2012.

APPENDIX

Contribution:

- **a. Design:** Krishnaja Kethireddy, Nirmal Kumar Ravi, Prathamesh Mantri
- **b. Implementation:** Toy Hadoop: Nirmal Kumar Ravi, Prathamesh Mantri
  FusionFS: Alekya Thalari
- **c. Evaluation:** Alekya Thalari, Krishnaja Kethireddy, Nirmal Kumar Ravi, Prathamesh Mantri
- **d. Documentation:** Alekya, Krishnaja Kethireddy