An Improved Approach for Analysis of Hadoop Data for All Files

Heena Jain
Dept. of Information technology
SATI
Vidisha(M.P)

Ajay Goyal
Dept. of Information Technology
SATI
Vidisha (M.P)

ABSTRACT
Here in this paper an efficient Framework is implemented for Hadoop Platform for almost all types of Files. The Proposed Methodology implemented here is based on various algorithms implemented on Hadoop Platform such as Scan, Read, Sort etc. Various Workloads are used for the Analysis of the Algorithms of small and big size such as Facebook, Co-author, and Twitter. The Experimental results show the performance of the proposed methodology. The Methodology provides efficient Running Time, NameNode Memory and Throughput as compared to the existing methodology.

Keywords
Hadoop, HDFS, NameNode, SFReduce, MapReduce, Facebook, Twitter.

1. INTRODUCTION
Hadoop is such an open-source Big Data framework utilized for storing, supervision and evaluating a huge volume of information it is planned to permit distributed processing and storage of information across thousands of machines. The Hadoop environment consists of numerous projects. Two of these Hadoop projects, the distributed computational structure MapReduce [1] and the distributed storage layer Hadoop Distributed File System (HDFS), form the extremely establishment of the Hadoop ecosystem. The Hadoop Distributed File System [2] is an open-source replica of the Google File System (GFS) [4] that is planned to make available high throughput and fault-tolerant storage space on low-cost commodity hardware. In comparison to conventional POSIX distributed file systems (e.g., Lustre, PVFS, AFS, Ceph), HDFS is planned to sustain write-once-read-many (WORM) type of workloads with optimizations for streaming access and huge data sets (e.g., MapReduce [1]). While ordinary HDFS uses replication of data blocks to defend beside hardware failures there are also try to use removal coding methods to make available fault tolerance [3, 5]. MapReduce is a programming model and an connected accomplishment for processing and producing huge data sets. Under this representation, each application is applied as a series of MapReduce operations consisting of a map phase and a decrease phase that procedure a huge number of independent data items.

Big amount of distributed file systems be inclined to split the metadata administration from file read/write operations so that make difficult metadata right to uses will not be in the I/O serious path to block ordinary file I/O operations which also allows parallel executions of meta-data and file I/O operations. HDFS also decouples the metadata supervision from file I/O by means of two self-determining functional parts: a single NameNode that runs the file system namespace and multiple DataNodes that accumulate the real file block data and are dependable for allocation read and write requests from Clients. The single NameNode structural design conversely has long been thinker as the Achilles' heel of HDFS as it not only signifies a single-point-of-failure but also is a most important restrictive factor for the scalability of the complete HDFS cluster. Hadoop [1], an open-source implementation of Google MapReduce, is the most accepted system. It is proposed to run parallel processing on thousands of computing nodes and provide a fault-tolerant and scalable storage service. In Hadoop there are two basic elements, Hadoop MapReduce and Hadoop Distributed File System (HDFS). Hadoop MapReduce uses extremely comparable methods explained in Google MapReduce. HDFS is also an open-source completion of Google File System. Hadoop is not a easy copy of proposals from Google. When Hadoop was released it was extensively utilized not only in industry but also in academy. Such huge scale information is very complicated to development and examine with relational database systems and desktop statistic software's. Both industry and academy require some advancement to deal with this complexity. Google publishes three papers to initiate the associated methods utilized in its own cluster, MapReduce [6], BigTable [7] and the Google File System (GFS) [4].

An essential hypothesis Hadoop system (HDFS) is based on, "moving computation is economical than moving data". It represents that presenting calculations on nodes is more capable than storing the huge data locally. HDFS gives excellent presentation on "commodity" clusters which are economical in environment with comparatively slow network fabrics [8]. HDFS cluster uses a master-slave design consisting of a single NameNode i.e. the master and multiple DataNodes i.e. the slaves frequently one per node in the cluster anticipating high throughput of data right to use to a certain extent than low latency of data access. The NameNode deal with the file method namespace and controls right to use to files by clients, whereas the DataNodes are responsible for serving read and write demands from the file system's clients. In conventional Map/Reduce situations input and output data are accumulated on the HDFS as referred in Figure-1,with transitional data stored in a neighborhood, temporary file method on the Mapper nodes and shuffled as required (via HTTP) to the nodes running the Reducer jobs [9].

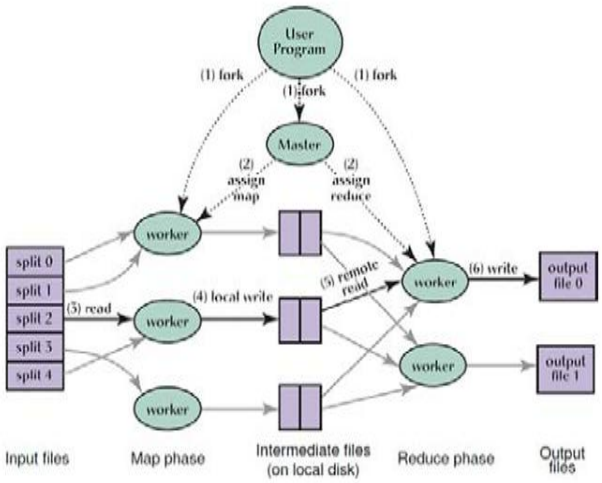

Figure-1: MapReduce Architecture [10]

Two main techniques utilized to put into practice responsibility tolerance in HDFS: i) Data duplication and ii) Checkpoint and recovery [11]. Data duplication consists in duplicating the information in multiple DataNodes as they are distributed. To mark a file to the HDFS client first make contact with the NameNode and then the NameNode proposes a number of i.e. three by default DataNodes exploited to repeat the data. The number of duplications can be enlarged, improving the fault tolerance and the bandwidth in understanding the file. The checkpoint and recovery methods are comparable to the idea of rollback. If a failure take places the scheme rollbacks to the most recent saved synchronization point and the transaction initiates again. This technique is slower than data duplication but alternatively it requires fewer extra resources.

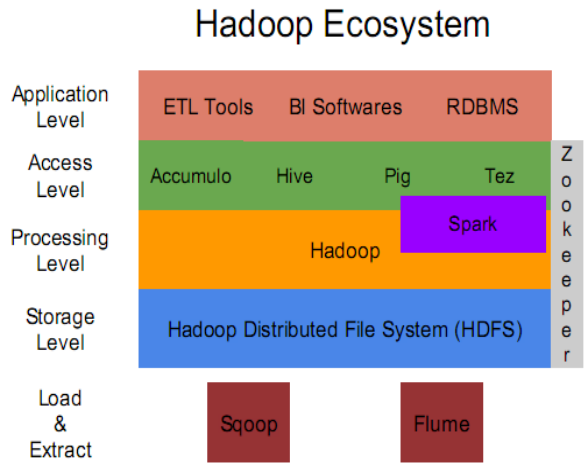

Figure-2: Hadoop Ecosystem

The MapReduce paper available in the OSDI conference, initiates a parallel processing system running on Google cluster. In this paper, authors present the interrelated methods about MapReduce programming model, data type, elements of the scheme, fault tolerance and so on. Map and reduce functions do not become visible first in this paper. In point of fact, map and reduce functions are much admired functions in mainly functional programming languages. Map function executes filtering and sorting. The reduce function executes a review operation. With map and reduce functions clients can simply execute multiple types of conventional operations, such as Wordcount, sort and so on. Approximately Hadoop, groupings of companies engage to extend abundance controlling structures.

Famous IT companies, such as IBM, Facebook, Twitter, Yahoo, Google, Baidu, and soon. Startup companies concentrated on Cloud Computing, such as Cloudera, Hortonworks, MapR, and so on. In Figure 2, we can find that there are four levels in Hadoop ecosystem. They are application level, access level, processing level, and storage level, respectively. Storage level is responsible for keeping and maintaining the data in cloud. Processing level is used to run parallel processing programs in a specified programming model. In access level, developers and users can directly use the frameworks in this level to implement some concrete and high-level operations on the data. The jobs submitted in access level finally are divided into the different jobs running in the processing level. Application level provides users much easier way to execute the operations they need. Users can focus on their needs without concerning about the implementation inside the cloud environment. Before and after using Hadoop, users should use some tools to help load into and extract from Hadoop storage level. In this area, the famous tools include Sqoop, Flume, and so on. In the next subsections, we will introduce some famous and important frameworks in the finer classifications including file system, programming model, SQL, and so on.

2. LITERATURE SURVEY

In this paper [12], they propose a Small File MapReduce Framework (SFMapReduce) that can solve these problems systematically. Two techniques are introduced in our framework, which includes Small File Layout (SFLayout) and Customized MapReduce (CMR). SFLayout is an innovative file layout designed to use in HDFS for solving the storage problem of small files. SFLayout combines small files into an integrated file, which decreases the memory pressure of NameNode. In addition, we design several useful operators to manage the files stored in SFLayout. In order to process the files stored in the form of SFLayout, CMR is proposed to run the related MapReduce jobs. Moreover, CMR avoids the extra overhead and improves the MapReduce performance, compared with the conventional Hadoop. The cost of creating and closing a container keeps constant for any size of files. However, traditional Hadoop MapReduce generates containers for each small file. The cost of containers cannot be ignored in this situation. CMR is proposed to avoid the extra overhead and improve the MapReduce performance. CMR provides two customized components to transfer a SFLayout file to traditional Key/Value pairs in the processes of map and reduce phases. This approach reduces the overhead in running MapReduce programs with lots of small files. CMR also holds a selector that is utilized to choose precise files from SFLayout based on special conditions. SFMapReduce is designed to combine small files into an integrated file with a new layout and then run MapReduce programs based on the new layout.

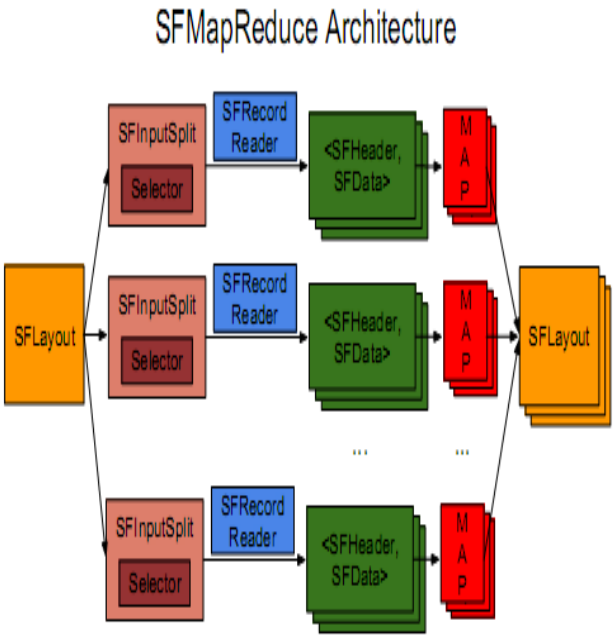

Fig. 3: SFMapReduce Architecture [12].

The experiments' results illustrate [12] the robustness of our framework. SFMapReduce provides better loading throughput than the original MapReduce and HAR file layout by 2.78x and 2.99x, respectively. At the same time, SFMapReduce provides better retrieving throughput than these two frameworks by 1.64x and 1.13x. Furthermore, SFMapReduce also outperforms the MapReduce's processing performance by 14.5x and 20.8x for different benchmarks on average.

In Hadoop MapReduce, reduce tasks issue massive remote I/O operations to copy the transitional effects of map jobs. The operations origin enormous isolated data access interruptions which humiliate the scheme presentation. To feel this difficulty here they propose an execution engine of decrease jobs. The engine separations the execution of reduce jobs into two stages. In the first stage, the engine chooses the nodes to run reduce jobs and then arrange the nodes to prefetch transitional results for the reduce jobs. In the second stage, the preferred nodes assign calculating and memory resource to the reduce jobs and execute these jobs. Due to the fact that transitional results have been prefetched reduce jobs can right to use these results from local nodes and the remote access interruption of the results can be concealed. Here they have employed the engine in Hadoop-0.20.2. They estimated the engine in a Linux cluster. The results give you an idea about that the engine optimized the presentation of Hadoop in most cases.

Tian and Chen [13] propose predicting a given MapReduce application performance from a set of test runs on small input datasets and a small Hadoop cluster. By executing a variety of 25-60 test runs the authors create a training set for building a model of a given application. Once derived, this model is able to predict the future performance of the same application when executed on a larger input and a larger Hadoop cluster. The limitation for this model is that the model it closely tired with the application characteristic, when given a new application, the model has to be rebuilt using another training set created with the new application.

Valvag et al. [14] developed a high-level declarative programming model and its underlying runtime, Oivos, which aims at handling the applications that require running several MapReduce jobs. This framework has two main advantages compared with MapReduce. First, it handles the overhead associated with such type of applications including monitoring the status and development of each work influential when to re-execute a failed situation or initiate the next one and identifying a suitable execution order for the MapReduce jobs. Second, it eliminates the additional synchronization when these applications are performed using the conventional MapReduce structure, i.e., every reduce job in one situation should complete before any of the map tasks in the next job can start.

Potisepp [15] discussed the processing small/regular images of total 48675 by aggregating them into huge data set and development them on Hadoop using MapReduce as sequential files comparable to the one addressed by HIPI. In addition, offered achievability learning as a proof-of-concept test for a particular huge image as blocks and be relating pixels for non-iterative algorithms image processing. on the other hand, no plan, or explanation, or method has been recommended to each to Hadoop or MapReduce for either Image Processing applications or for any other area so that the method efforts for accessible in addition to novel representations under concern.

The above works focused on small files problem by combining the smaller images into large bundle like HIPI does, or they are good for handling the images and lack in performing the image related spatial filters applications as the overlap data among the adjacent blocks is not available. They compared the performance of the single PC system with the Hadoop based clusters, and do not address the data handling and processing mechanisms using Hadoop effectively. In this paper, the issues related to processing large remote sensing images which run into several Megabytes to Gigabytes are addressed, along with the several issues related to data association over HDFS, and processing them by MapReduce using extended HDFS and MapReduce called as XHAMI library. Our proposed XHAMI is applied for describing the applications for remote sensing or geo sciences data perspective. However, the same could be either readily applied or can be extended for other domains such as medical imaging where such similar data dependencies exist in processing.

3. PROPOSED METHODOLOGY

The Proposed Methodology implemented here consists of following steps to be performed for any type of Workload in BigDataBench.

1. First of all Choose BigDataBench Workloads with the following set of assumptions such as a) Paying equal attention to different types of applications: online service, real-time analytics, and offline analytics; b) Covering workloads in diverse and representative application scenarios; c) Including different data sources: text, graph, and table data; d) Covering the representative big data software stacks.
2. Selection of Different Workloads including Wordcount, Scan, Sort, Read, PageRank, Index.

3. On the basis of Different Workloads such as Wordcount Semantic Similarity using Cosine Similarity is computed and hence applying Resource Allocation using MBFD is done.

4. Consider a cloud Computing Environment with D_i as the number of data centers B_i is the number of Brokers of the cloud H_i is the number of hosts and $V_i$is the number of physical virtual machines with N number of Cloudlets C_i and Resources R_i.

1. Cloud consumers can submit their requests for the access of resources to the brokers. Each of the requests from the cloudlets is allocated to their respective brokers who can process their requests.

2. Virtual machines can be dynamically started and stopped on a single physical machine according to the incoming requests, hence providing the flexibility of configuring various partitions of resources on the same physical machine to different requirements of service requests. Multiple VMs can concurrently run applications based on different operating system environments on a single physical machine. By dynamically migrating VMs across physical machines, workloads can be consolidated and unused resources can be switched to a low-power mode, turned off or configured to operate at low-performance levels (e.g. using DVFS) in order to save energy.

3. The underlying physical computing servers provide the hardware infrastructure for creating virtualized resources to meet service demands.

Currently, resource allocation in a Cloud data center aims to provide high performance while meeting SLAs, without focusing on allocating VMs to minimize energy consumption. To explore both performance and energy efficiency, three crucial issues must be addressed. First, excessive power cycling of a server could reduce its reliability. Second, turning resources off in a dynamic environment is risky from the QoS perspective. Due to the variability of the workload and aggressive consolidation, some VMs may not obtain required resources under peak load, and fail to meet the desired QoS. Third, ensuring SLAs brings challenges to accurate application performance management in virtualized environments. All these issues requires effective consolidation policies that can minimize energy consumption without compromising the user-specified QoS requirements.

Allocation of Virtual Machines

Here the allocation of virtual machines is based on the entrance of new requests for the provisioning of Virtual Machines and then allocating of virtual machines on hosts and then optimization of the current allocation of virtual machines. The proposed algorithm implemented here uses Bin backing algorithm which is based on Modified Best Fit Decreasing (MBFD) algorithm in which sorting of all VMs in decreasing order of their current CPU utilizations, and allocate each VM to a host that provides the least increase of power consumption due to this allocation. This allows leveraging the heterogeneity of resources by choosing the most power-efficient nodes first.

Algorithm: Modified Best Fit Decreasing (MBFD)
Input: HostList & VmList
Output: Allocation of VM's

1. First of all sort the list of virtual machine lists in decreasing order of their Utilization.
2. For each of the Virtual machine repeat
3. manpower ← MAX
4. allocatedHost ← NULL
5. for each of the host in HostList do
6. if host has enough resource for VM then
7. power ← estimatePower(host,VM)
8. if power < manpower then
9. allocatedHost ← host
10. manpower ← Power
11. if allocatedHost ≠ NULL then
12. allocated VM to allocatedHost
13. return allocation

- **Data content similarity (SimC)**

It is the Cosine similarity between the term frequency vectors of d1 and d2:

$$\boldsymbol{SimC(d1,d2)} = \frac{\boldsymbol{V_{d1} * V_{d2}}}{\left|\left|\boldsymbol{V_{d_1}}\right|\right| * \left|\left|\boldsymbol{V_{d_2}}\right|\right|} \quad (1)$$

Where Vd is the frequency vector of the terms inside data unit d, ||Vd|| is the length of Vd, and the numerator is the inner product of two vectors.

- **Number of Common Neighbors**

It is defined as the total number of nodes that are connected directly in relationship with node x and y for unweighted network,

$$CN(x,y) = \varphi(x) \cap \varphi(y) \quad (2)$$

Where, $\varphi(x)$is the set of neighbors of node x.

$\varphi(y)$is the set of neighbors of node y.

To calculate link prediction between nodes for unweighted network common neighbors can be calculated as,

$$CN(x,y) = \sum_{z \in \varphi(x) \cap \varphi(y)} w(x,z) + w(y,z) \quad (3)$$

- **Jaccard Coefficient**

It is defined as the highest proportion of common neighbors to the total number of neighbors in the network. The Jaccard Coefficient can also defined for weighted as well for unweighted network.

For unweighted network,

$$JC(x,y) = \frac{\varphi(x) \cap \varphi(y)}{\varphi(x) \cup \varphi(y)} \quad 4)$$

For weighted network,

$$JC(x,y) = \sum_{z \in \varphi(x) \cap \varphi(y)} \frac{w(x,z) + w(y,z)}{\sum_{a \in \varphi(x)} w(a,x) + \sum_{b \in \varphi(y)} w(b,y)} \quad (4.5)$$

Predict the most valuable words from the text documents having most similarity between words.

Skewness algorithm for the avoidance of overload

Here Skewness algorithm is applied for the avoidance of the overload by applying the predicting the unevenness in the allocations of resources. The 'N' number of resources allocation with 'R' number of resources can be applied to their respective servers P on the basis of:

$$skewness(P) = \sqrt{\sum_{i=1}^{n} (\frac{r_i}{\bar{r}})^2}$$

Where, $\bar{\mathbf{r}}$ is the average utilization of all the resources for the'N' number of resources for Servers P. Finallly the mitigation of resource can be done by comparing the threshold. The Utilization for each of the resource can be allocated on the basis of load on each of the server.

4. RESULT ANALYSIS

Table 1. Analysis of NameNode Memory

	NameNode Memory (MB)	
Set ID	**Original Hadoop**	**Proposed Work**
1	5	3
2	50	38
3	800	690
4	7000	6500
5	13000	11500

Table 2. Analysis of Running Time

	Running Time (s)	
SetID	**Original Hadoop**	**Proposed Work**
1	1	1
2	800	500
3	1200	1000
4	1900	1500
5	2500	2000
6	3000	2500
7	3500	3000
8	4000	3500
9	4500	4000
10	5000	4500

Table 3. Analysis of Memory Usage in Namenode

	NameNode Memory (KB)	
Data Size	**Existing Work**	**Proposed Work**
10	10	5
20	50	20
30	70	30
40	80	50
50	90	65
60	100	75

Table 4. Analysis of Loading and Retrieving Throughput

	Throughput (MB/Sec)	
	Existing Work	**Proposed Work**
Load	40	45
Retrieve	80	83

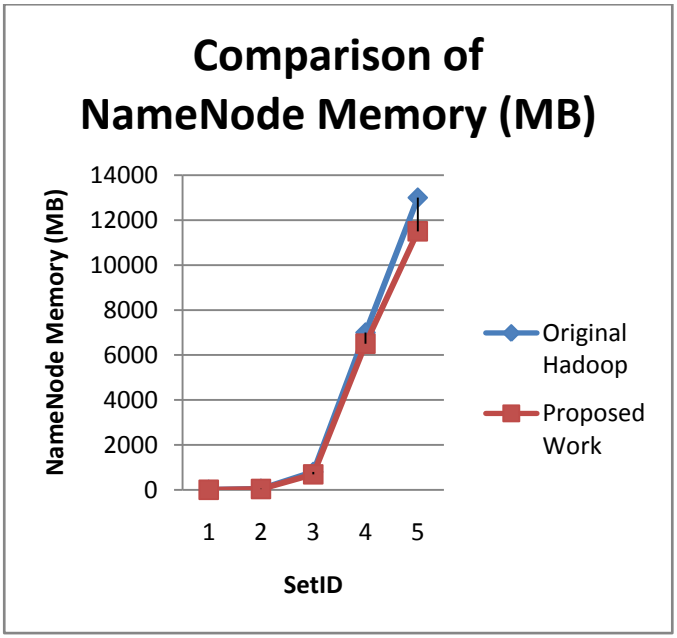

Figure 4. Comparison of NameNode Memory

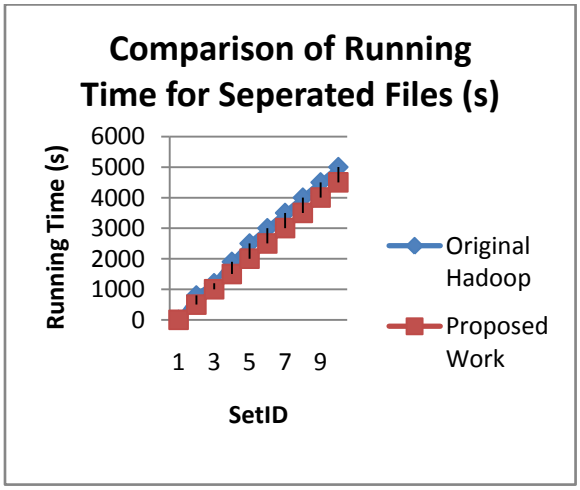

Figure 5. Comparison of Running Time fo Seperated Files

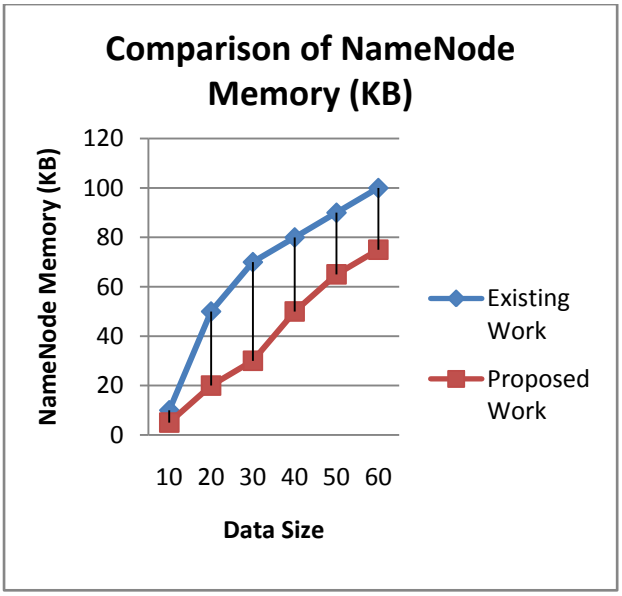

Figure 6. Comparison of NameNode Memory

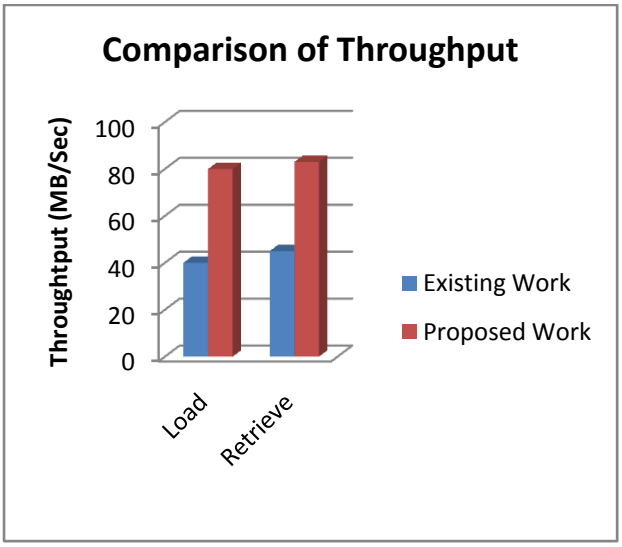

Figure 7. Comparison of Loading & Retrieving Throughput

5. CONCLUSION

Hadoop is a powerful and widely used framework to handle large scale of data. Users and developers can easily use Hadoop to parallelize the processes of data in an available and scalable cloud environment. The demands and requirements vary dramatically in the practical world. One of the most significant demands is to add features to efficiently store and process small files in Hadoop. As we discussed in background section, both HDFS and MapReduce in the original Hadoop cannot support small files well. In order to solve these problems, we propose a new adaptation framework for the analysis of Small and Big Size Files. The various Experimental results shows the Performance of the Proposed Methodology.

6. REFERENCES

[1] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. OSDI'04, Berkeley, CA, USA, 2004. USENIX Association.

[2] Apache.org. Hadoop distributed file system. http://hadoop.apache.org.

[3] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson. Diskreduce: Raid for data-intensive scalable computing. In Proceedings of the 4th Annual Workshop on Petascale Data Storage, PDSW '09, pages 6–10, New York, NY, USA, 2009. ACM.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. SIGOPS Oper. Syst. Rev., (5), 2003.

[5] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. Xoring elephants: novel erasure codes for big data. In Proceedings of the 39th international conference on Very Large Data Bases, PVLDB'13, pages 325–336, 2013.

[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In OSDIn'04, pages 137-150, 2005.

[7] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A.Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26(2):4, 2008.

[8] V.S.Patil, P.D.Soni, Hadoop Skeleton & Fault Tolerance in Hadoop Clusters,International Journal of Application or Innovation in Engineering & Management, Volume 2, Issue 2;February 2013,pp.247-250.

[9] J. Evans, Fault Tolerance in Hadoop for Work Migration, Technical Report CSCI B534 (Survey Paper), Indiana University;November 2011.

[10] J.Dean, S.Ghemawat, MapReduce: Simplified Data Processing on Large Clusters,Communication of The ACM;Jan. 2008,pp. 107-113.

[11] I.Goiri,F. Julià,J.Guitart, J.Torres, Checkpoint-Based Fault-Tolerant Infrastructure for Virtualized Service Providers. IEEE/IFIP Network Operations and Management Symposium,IEEE. Osaka, Japan;April 2010,pp. 455-462.

[12] Fang Zhou Hai Pham Jianhui Yue Hao Zou Weikuan Yu, "SFMapReduce: An Optimized MapReduce Framework for Small Files, IEEE 2015.

[13] Fengguang Tian and Keke Chen. Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds. In Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, CLOUD '11, pages 155–162, 2011.

[14] Steffen Valvag and Dag Johansen. Oivos: Simple and Efficient Distributed Data Proessing. In IEEE 10th International Conference on High Performance Computing and Communications, pages 113–122, Sept. 2008.

[15] K. Potisepp, Large Scale Image Processing Using MapReduce, MSc. Thesis, Institute of Computer Science, Tartu University, 2013.