IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 5, Ver. VI (Sep. - Oct. 2016), PP 48-58
www.iosrjournals.org

Implementation and Performance Analysis of Apache Hadoop

Song Feng Lu, Husein S Naji Alwerfali
School of Computer Science Huazhong University of Science and Technology, Wuhan, China, 430074

Abstract: *With the Internet and data growth increasing trends, big data is becoming an extremely important and challenging problem for Data Centers. Many platforms and frameworks are working to bring a cutting edge technology to this problem. Apache Hadoop is a software framework addressing the big-data processing and storing on clusters, providing reliability, scala-bility and distributed computing. Hadoop has a distributed file system to store vast amount of data in distributed environments, and uses Map reduce algorithm to perform the computations and process large amount of data, by parallelizing the workload and storage. In comparison to other relational database systems, Hadoop works well with unstructured data. Our work is focused on performance evaluation of benchmarks of Hadoop, which are crucial for testing the infrastructure of the clusters. Taking into consideration the sensitiveness and importance of data, it's inevitable testing the clusters and distributed systems before deploying. The benchmark results can lead to optimizing the parameters for an enhanced performance tuning of the cluster. In this paper, we are motivated to study and evaluate the performance of Hadoop and a comprehensive listing of bench-marks used to test Hadoop, while providing detailed information for their appliance and procedures to run them. We construct a distributed hadoop cluster simulation based on VmWare Workstation consisted of multiple nodes running hadoop. we have conducted a measurement study on the performance evaluation of hadoop cluster simulation under multiple scenarios. Our results demonstrates the trade-off between performance and flexibility. This evaluation study is focused on the throughput performance comparisons under different scenarios. The Hadoop performance has been evaluated for different size of data sets in this simulation. To measure the performance we set up a Hadoop cluster with many nodes and use the fileTestDFSIO.java of the Hadoop version 1.2.1 which gives us the data throughput, average I/O rate and I/O rate standard deviation. Our results demonstrates that the HDFS writing performance scales well on both small and big data set. The average HDFS reading performance scales well on big data set where it is, however, lower than on the small data set. The more nodes a writing/reading operation is run on, the faster its performance is.*

Index Terms: *Hadoop; Big data; Benchmark; Distributed file system*

I. Introduction

Cloud computing and big data networks has become one of the key part of the content delivery technology since the past decade. The demands on cloud for functionality and scalability are growing due to the rapid proliferation of new network devices and applications which are generating huge amount of data to be stored and processed at the cloud.

The enormous changes in technology and electronic user devices requires the efficient and fast data retrieval from the data centers as well as the reduced processing time. It is expected that annual global data center IP traffic will reach 10.4 zettabytes (863 exabytes [EB] per month) by the end of 2019, up from 3.4 zettabytes (ZB) per year (287 EB per month) in 2014. Global data center IP traffic will grow 3-fold over the next 5 years according to Cisco statistics[1]. Overall, data center IP traffic will grow at a compound annual growth rate (CAGR) of 25 percent from 2014 to 2019. This dramatic increase in traffic suggests to improve the cloud in terms of reliability and performance. Hadoop is a software used for distributed infrastructures with particular focus on big data, ensuring scalability and reliability for distributed data centers. Hadoop has its own file system, Hadoop File system or HDFS that differs from other filesystem due its large block size, which was designed to be robust to many problems that other Distributed File systems couldnt handle such as fault tolerance, reliability and scalability. Distributed systems have existed before, but with Hadoop the data and work are automatically distributed across machines and the CPU usage are parallelized. As most distributed file system, HDFS is based in an architecture where data is separated from the namespace [2]. HDFS is a best effort solution to fault tolerance in a very large data center. Its purpose is to distribute large amount of data across many machines (nodes). As an input Hadoop receives a very large file and divides it into chunks called blocks, which then are stored and replicated across different machines. So when a machine in the distributed environment will fail due to some problem, HDFS will provide the missing blocks which were replicated in some other machines. In this way HDFS ensures reliability so end users won't be affected by a single machine failure since data is distributed. Although Hadoop is a recently developed framework, there are great expectations in its quick development and spreading in the near future [3]. First hand users of Hadoop have

confessed that there are still many areas of improvement to the software in regards to security, administration, availability and others. However, the open source community is continuously addressing these issues and is also steadily contributing to improving Hadoop products which leads to anticipating greater capabilities and usage of the software in the next years.

II. Background

Todays most successful internet applications, from search engine to social network, greatly rely on large scale data processing. Meanwhile, in academia, sheer volume data sets analysis has become a popular methodology and the fourth paradigm for modern scientific discoveries [4]. Moreover, the Big Data processing capacity has become the killer ability of some startup internet companies, especially for those providing the social network applications and business intelligence services.

A. Hadoop Distributed File System (HDFS)

Hadoop is a project develops open-source software for reli-able, scalable, distributed computing. It consists four module: 1) Map Reduce, a system for parallel processing of large data sets; 2) HDFS, a distributed file system that provides high throughput access to application data; 3)YARN, a framework for job scheduling and cluster resource management; 4) Com-mon, the common utilities that support the other Hadoop modules.

The Hadoop Distributed File System (HDFS), an Apache Hadoop subproject, is a highly fault-tolerant distributed file system designed to provide high throughput access to appli-cation data and is suitable for applications with extremely large data files (Petabytes and up) on a distributed network of computers[5].

A HDFS cluster consists two kinds of nodes: (a) a single Name Node that works on master server and maintain the meta-data structure of HDFS namespace includes opening, saving, retrieving, closing and renaming files and directories. It de-termines the mapping between files and blocks to Data Nodes, and (b) a number of Data Nodes manage storage attached to the nodes. The Data Nodes serve read and write requests from the HDFS clients. Data file is split into one or more blocks, and these blocks are stored in a set of different Data Nodes. The Data Nodes also perform block creation, deletion, and replication upon instruction cooperating with the Name Node. There are four operation of typical jobs involving HDFS: 1) client send a read comment to the Name Node, the Name Node then send back metadata to the client in order to confirm the process; 2) Name Node send metadata to the Data Nodes; 3) Data Nodes read and write the metadata; 4) requested data is sent back from the Data Nodes to the client via network. The critical factors in the process need to be probed and adjusted in order to improve the total performance in cloud computing.

B. Map Reduce Framework

Map Reduce was developed by Google as a new framework for processing large data sets. The original Map Reduce soft-ware framework is not publicly available, but several open-source alternatives such as Hadoop Map Reduce do exist and is available for the public to use [6]. The Hadoop Map Reduce is the parallel computation mechanism that is used to realize the transparency about the complexity of parallelization, data distribution, fault-tolerance, and load balancing to developers, which can be used to focus on developing the actual algorithm.

Two major functions, Map and Reduce are specified in user program respectively. Although neither function is explicitly parallel, many instances of these functions are allocated and executed concurrently by the framework. A master node allocates Map and Reduce tasks to worker nodes for execute client program. The input files, stored in HDFS, are fetched and split into data chunks. The following steps are then required for Map Reduce framework to complete the requested jobs: 1) data chunks are distributed to parallel Map tasks for processing, then key/value records are generated and stored in intermediate files. No communication is allowed due to parallel Map processing tasks, 2) the output intermediate file are copied to Reduce tasks nodes, 3) according to the key/value, Sort function is then executed, 4) the Reduce function is accomplished, multiple outputs are then finally created. Data locality of Map/Copy/Sort/Reduce function is the one of critical factors in Map Reduce processing, such as distance between storage and compute nodes that is across networks, racks or nodes. For Map executing phase, the node executing the Map tasks should be close to the node that stores the input data (preferably local to the same node). For Sort phase, it is required to move the intermediate data generated by the Map tasks to the Reduce tasks as their input. For Reduce phase, the node executing Reduce tasks should be close to the Map tasks nodes which generate the intermediate file used as Reduce tasks input. The data locality issues can cause a massive

III. Related Work

Much of the work concentrates on Meta data for taking scheduling decisions. However, the inherent knowledge gained during scheduling is not considered much for improving data related throughput and response time. Hadoop is originally designed and configured for batch oriented jobs. Due to the widespread adoption of

Hadoop by various industries and academia for simplicity and scalability, several real-time user facing applications are executed on Hadoop platform. Main-taining fixed number of replicas for blocks leads to heavy load on popular blocks which affects the jobs response time. To provide better user experience, the availability of blocks is to be maintained at high level. Sometimes the terms file and block are used interchangeably. [7] proposed a method to increase the availability of Hadoop through metadata replication. To avoid single point of failure, all required metadata of critical nodes are replicated into backup node(s). This work only concentrating on metadata replication to overcome from failure and does not consider the replication of applications data. In [8], two heuristics are proposed to solve the file allocation problem in parallel I/O systems. The load balance across all disks and variance of the service time at each disks are simultaneously minimized to achieve better response time. The product of file access rate and service time, called heat of the file, is used as an objective function. In case of HDFS, files are stored as a fixed size blocks and hence, the service time may probably same for all blocks. The metrics such as service times are not suitable in HDFS and the work only considers the problem of file allocation not replication. Jiong Xie et al. [8] presented a data placement method to balance the processing load among the nodes in heterogeneous Hadoop clusters. However, replication is not considered in their work. Wenhao Li et al. [10] proposed an incremental replication strategy to meet reliability requirement and reduce the storage cost. This work aims to meet required reliability and works well for temporary data or data with low reliability requirement. The high availability requirements of popular data blocks and load balancing are not considered. Q. Wei et al. [9] proposed a model to capture the relationship between availability and replica number. This method dynamically maintains required number of replicas to meet a given availability requirement. Sai-Qin et al. proposed a multi-objective replication strategy for cloud storage cluster [10] which is closest to our work. The objective function includes mean file unavailability, mean service time, load variance, energy consumption and mean latency. The artificial immune algorithm is used to finding replication factor and replica placement. The main problem here is setting proportionate values of objectives for getting an optimal solution. This work also does not consider the dynamic workload and load balancing. Several other works [11] are presented to optimize the replication in distributed file systems. Some of them aim to optimize the replica number and some of them concentrates on replica placement with respect to various goals such as load balancing, availability, reliability and energy efficiency. Providing fault-tolerance with techniques other than replication such as erasure codes [12], are not suitable for Hadoop Framework. Because, replication is not only useful for fault-tolerance service, but also increases the availability of the data which is essential for Hadoop like systems. The performance of Hadoop is also based on various other factors, such as block placement, other than replication. For the sake of simplicity, they are not considered and considering the factors other than replication is also beyond the scope of this work.

IV. Test Bed

Benchmarking in distributed architecture systems where the performance of your clusters is affected by many hardware and software components is crucial. Benchmark is the evaluation of the capacity and performance, measured in many parameters which are yielded as outcome of benchmarking tests. Based on the results of these parameters we can decide how to tune Hadoop Cluster for best performance. The aim of Hadoop benchmarks is to push the workload to the limit and find the bottlenecks of cluster, by estimating the cost of tasks.

Test DFSIO Benchmark is used for testing I/O performance of Hadoop. DFSIO or Distributed Filesystem Input/Output writes or reads into a specified number of files and sizes. Test DFSIO is used to evaluate the performance of the through-put, by putting it on a stress test. Stress testing (sometimes called torture testing) is a form of deliberately intense or thorough testing used to determine the stability of a given system or entity. It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results. This benchmark uses a Map reduce Job to read and

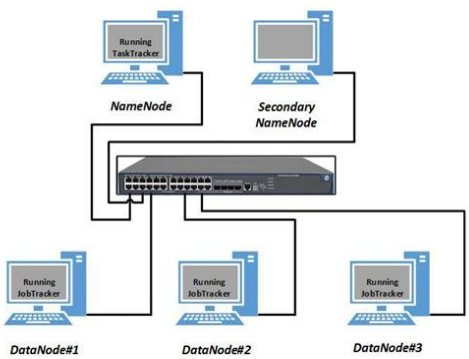


Fig. 1: Topology Diagram

write files into separate map tasks, whose output is used for collecting statistics that are accumulated in the reduce tasks to produce a summary result. The Benchmark data then is appended to a local file named Test DFSIO.log and written to standard output.
In short, the four key parameters in their architecture:
Nr Files.
File Size.
result File Name.
Buffer Size.
Expected Output:
Total Throughput Mbps
Average Input/output rate Mbps
Input/output rate standard deviation
Test execution time in seconds

We applied the benchmarks listed above on a Hadoop installed machine, to be able to evaluate the performance of these benchmarks. Normally, evaluating benchmarks must be done in a real distributed environment, to be able to extract the key factors when adjusting a Hadoop performance, but in this document we focused on parameters which can be applied in a single machine and yield some reasonable outcome, based on which we can conclude their performance. The benchmarking tests were performed in a virtual machine environment, with a Single Node Hadoop Cluster installed.

Processors: 2 (Intel Core I5- 3532QM CPU 2.50 GHZ) Memory: 2GB
JAVA VERSION: 1.7.0
HADOOP VERSION: 1.2.1 VM TYPE: 64 bit

In this section we will go into the details of the benchmark program TestDFSIO.java in order to understand how a client creates and submits a Map reduce job in the queue and how the job is processed on the HDFS. hadoop-1.2.1/bin/hadoop jar TestDFSIO.jar
Usage: Test FDSIO -read j -write j -clean [-nrFiles N] [-file Size MB] [-res File result File- Name] [-buffer Size Bytes]

1) Client creates control files: At first the client receives the input parameters and creates control files on the HDFS depending on the parameter -nrFiles (default = 1) with the function createControlFile(fs, fileSize, nrFiles) 38 . The names of the control files are default in file test io x (x = 0,1,..,N). They are unique and consist of the file size internally. Accord-ing to these files the client is able to know about the tasks (map tasks) and input filesize.
2) Client creates and submits a job: The design is that the number of the map tasks are overwritten by the number of input files (-nrFiles) and each map task performs the operation completely on Data node, which means the file will be written completely on one Data node. The map function Mapper used here implements an I/O operation as well as gathers the values of tasks (number of map tasks), size (filesize), time (executing time), rate and sqrate of each corresponding map tasks and sends them to the Reducer. The reduced function Reducer counts all the immediate values and save a reduced output file named part-00000 on the HDFS. The Function analyzeResult handles this file and prints out the final values of data throughput, average IO rate and IO rate standard deviation. After the Reducer receives the outputs of the Mapper, it sums the intermediate values, calculates Data Throughput (Mb/s), Average IO (Mb/s), Standard Deviation, etc. and creates reduced output files on the HDFS according to the number of reduced tasks. We only want to have a single reduced output file on the HDFS consisting all the values we need. So this is the meaning why the developers code the number of reduced tasks equals 1. Furthermore the client collects other attributes via files like control files, hadoop-site.xml, hadoop-default.xml, etc. to create job.jar, job.xml and job.split. Here are the meanings of these files: job.jar includes binary java files for the test, in this case it is for the class TestDFSIO.java. job.xml includes attributes for the test, e.g. mapred.map.tasks, mapred.reduce.tasks, dfs.block.size, etc. job.split includes the path to the control files and the java file used for splitting the input file. These files are useful for creating the job. Then the client deletes these files and sends the job into the HDFS queue.
3) Master handles jobs via queue: Job trackers and Name nodes daemons are threads running in the background on HDFS after we start it 18. There is one Thread, namely JobInitThread. This thread gets the job in sequence from the queue and handles it. According to the number of Map reduce tasks in the job the Job tracker contacts to the Name node for the nodes on where it should start the Map reduce tasks. The Job tracker is intelligent to make the job working even the job configuration is bad. For example we have only m map tasks (configured in hadoop-site.xml), but the number of split data sets is n (n¡m). Each map task can only work with one split data set.

If the Job tracker starts all of m map tasks, there is m-n map tasks which do nothing. So its wasted and can occur problems on the HDFS. To avoid it the Job tracker sets them equals as default. Each split data set has usually many files. The map task will call (or create) for each file a Mapper and this Mapper handles this file. Its analog to the reduced task. How the Mapper and Reducer work is already described in step 2.

V. Results

The objective of our simulation is to analyze the perfor-mance of small cluster thereby evaluating the feasibility of this network. We present a measurement study to understand how it performs under different workloads and types of traffic. The objective is to understand the practicality of the system. The test was accomplished on the cluster with nine nodes with the configuration: Two nodes for masters (one for Name node, one for Job tracker) and the remaining nodes are Data nodes.
Each Map reduce task will run on one Data node and the task distribution is made by the Job-Tracker.

A. Test scenarios

The tests deliver the writing/reading performance with the small (512 MB) /big (2 and 4 GB) data set with the block size 64/128 MB
Write / Read 512 MB with block size 64/128 MB
Write / Read 2 GB with block size 64/128 MB
Write / Read 4 GB with block size 64/128 MB

The measure and belongs to one (map-) tasks. That means if we have five tasks and the measure and is equal to X Mb/s we will have altogether 5*X MB/s. For all test scenarios the writing/reading performance is tested three times and a median value will be compared to the other to avoid outliers. To know about the locations of blocks we can run the fsck tool on the Name node, for instance: hadoop-1.2.1/bin/hadoop fsck path to file -blocks -files –locations Algorithm

B. Write Operation Throughput Evaluation

1) 512 MB with Block size of 64 MB: The figure 2 refers to the graph for 3 tests taken with a replication factor of 1 and dateset of 512 MB with a block size of 64 MB. The test is repeated 3 times for taking throughput and the input/output performance of the system thereby taking the standard deviation and the mean value.The graph in 2 shows that the average throughput achived is nearly 34 Mbps for the given dataset and replication factor of 1. A very low standard deviation value fo 0.006 shows that there is not much difference of throughput among the test values.
2) 2GB with Block size of 64 MB: After the first evaluation test with the 512 MB dataset we increase the dataset to 2GB and evaluate it against the same block size of 64 MB and the replication factor of 1. The figure 3 refers to the graph for 3 tests taken with a replication factor of 1 and dateset of 2GB with a block size of 64 MB. The test is repeated 3 times for taking throughput and the input/output performance of the system thereby taking the standard deviation and the mean value.The graph in 3 shows that the average throughput achived is nearly 33 Mbps which is a little lower the the previous test results mentioned in 2 for the given dataset and replication factor of 1. The standard deviation value of 1.20 is also comparatively higher for the bigger size of the dataset.

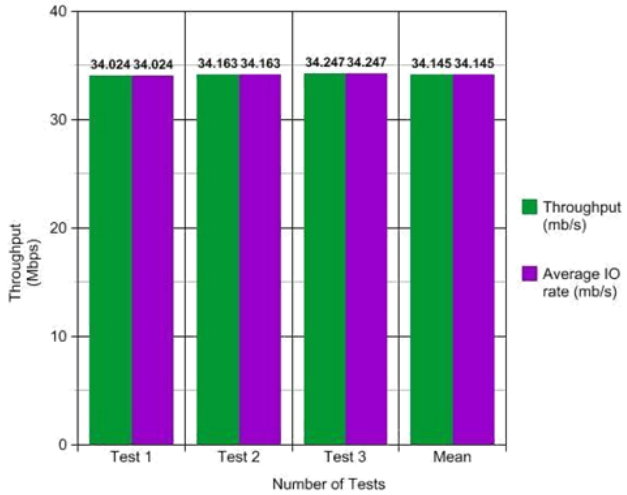


Fig. 2: Write Operation Throughput Evaluation for 512 MB, Block size of 64 MB

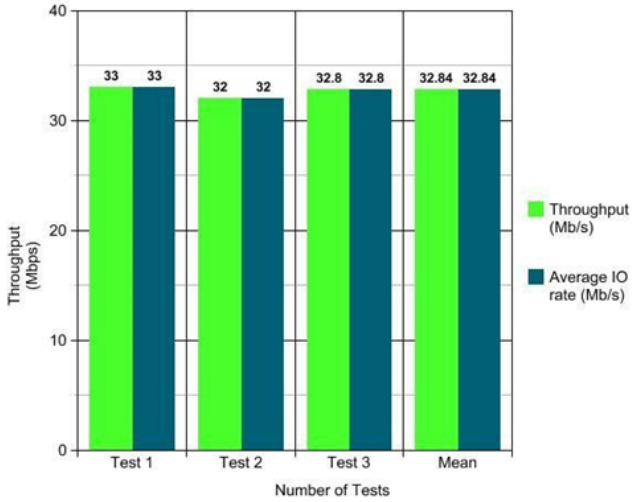


Fig. 3: Write Operation Throughput Evaluation for 2GB, Block size of 64 MB

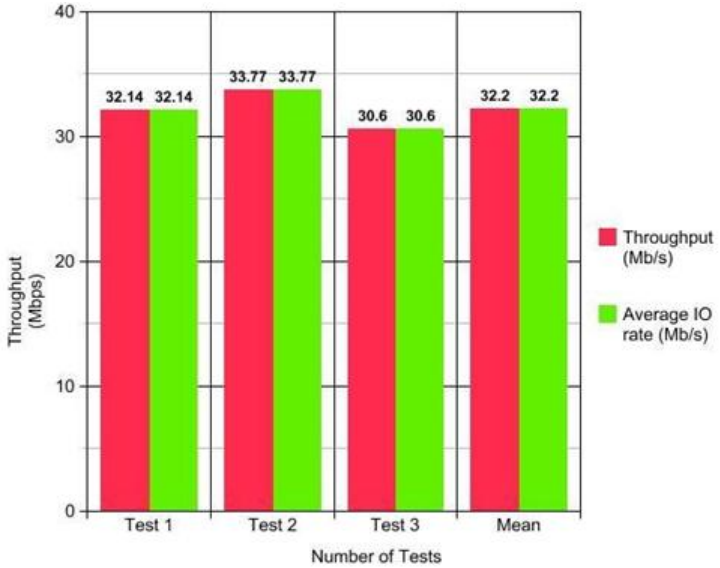


Fig. 4: Write Operation Throughput Evaluation for 4GB, Block size of 64 MB

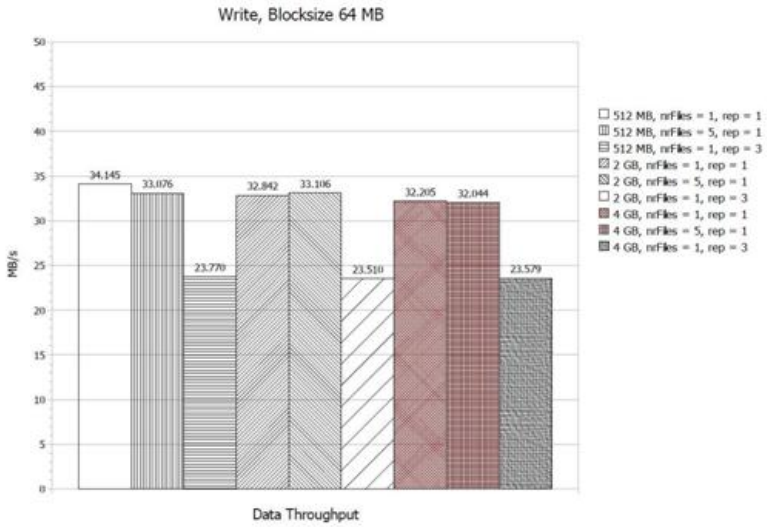


Fig. 5: Write Operation Throughput Evaluation Summary

3) 4GB with Block size of 64 MB: Now we increase the replication factor to 3 and evaluation test with the 4GB dataset. We evaluate it against the same block size of 64 MB. The figure 4 refers to the graph for 3 tests taken with a replication factor of 3 and dateset of 4GB with a block size of 64 MB. The test is repeated 3 times for taking throughput and the input/output performance of the system thereby taking the standard deviation and the mean value.The graph in 4 shows that the average throughput achived is nearly 32.2 Mbps which is a little lower the the previous test results mentioned in 2 and 3 for the given dataset and replication factor of 1.
4) Comparison of Write Performance: Hadoop is designed for clients, which dont run on a Hadoop daemon itself. If a client performs a writing operation, normally this operation will run on the Name node and it will split the input data set and spread the split parts across the Data nodes. Otherwise if we want to perform the writing operation for some reason on any Data nodes internally, this operation will only performed locally to avoid congestion on the network. The writing performance with both blocksize 64 MB and 128 MB looks similar to each other. It scales very well with both the small as well as big data set. Writing with a replicated file logically produces a slower performance. The 5 shows a comparison of all the performed test to evaluate the write performance. The writing performance of Hadoop scales better than reading with small data sets. But it doesnt matter because Hadoop is designed for the batch processing on huge data sets. So in this case its quite fine with the scalability. Furthermore the writing and reading performance are fast. If we write or read a data set with 20 GB distributed on 5 nodes, we will end up with approximately 160 MB/s and 181 MB/s. The more data nodes we have, the faster it is.

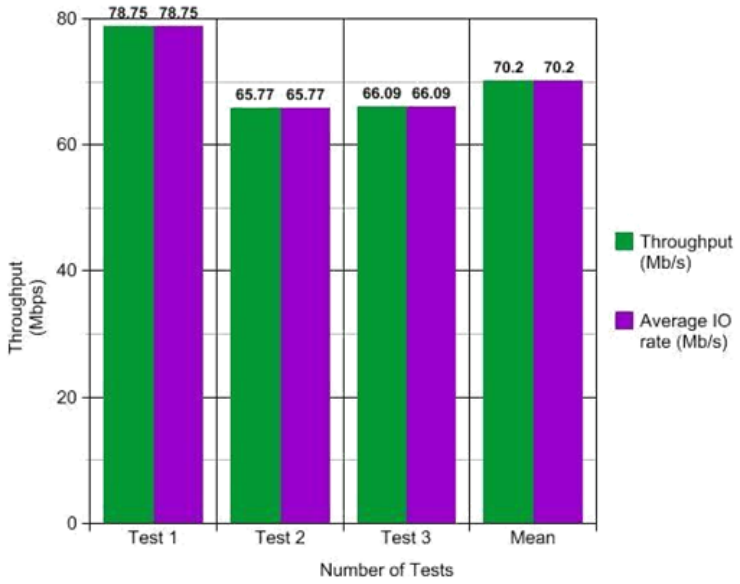


Fig. 6: Read Operation Throughput Evaluation for 512 MB, Block size of 64 MB

C. Read Operation Throughput Evaluation

1) 512 MB with Block size of 64 MB: The figure 6 refers to the graph for 3 tests taken with a replication factor of 1 and dateset of 512 MB with a block size of 64 MB. The test is repeated 3 times for taking throughput and the input/output performance of the system thereby taking the standard deviation and the mean value. The graph in 6 shows that the average throughput achieved is nearly 70.2 Mbps for the given dataset and replication factor of 3. A very low standard deviation value of 0.012 shows that there is not much difference of throughput among the test values.
2) 2GB with Block size of 64 MB: After the first evaluation test with the 512 MB dataset we increase the dataset to 2GB and evaluate it against the same block size of 64 MB and the replication factor of 3. The figure 7 refers to the graph for 3 tests taken with a replication factor of 1 and dateset of 2GB with a block size of 64 MB. The test is repeated 3 times for taking throughput and the input/output performance of the system thereby taking the standard deviation and the mean value.The graph in 7 shows that the average throughput achieved is nearly 67 Mbps which is a little lower the the previous test results mentioned in 6 for the given dataset and replication factor of 1. The standard deviation value of 0.011 is also comparatively higher for the bigger size of the dataset.
3) 4GB with Block size of 64 MB: Now we keep the replication factor to 3 and evaluation test with the 4GB dataset. We evaluate it against the same block size of 64 MB. The figure 8 refers to the graph for 3 tests

taken with a replication factor of 3 and dateset of 4GB with a block size of 64 MB. The test is repeated 3 times for taking throughput and the input/output performance of the system thereby taking the standard deviation and the mean value.The graph in 8 shows that the average throughput achived is nearly 53 Mbps which is lower the the previous test results mentioned in 6 and 7 for the given dataset and replication factor of 3.

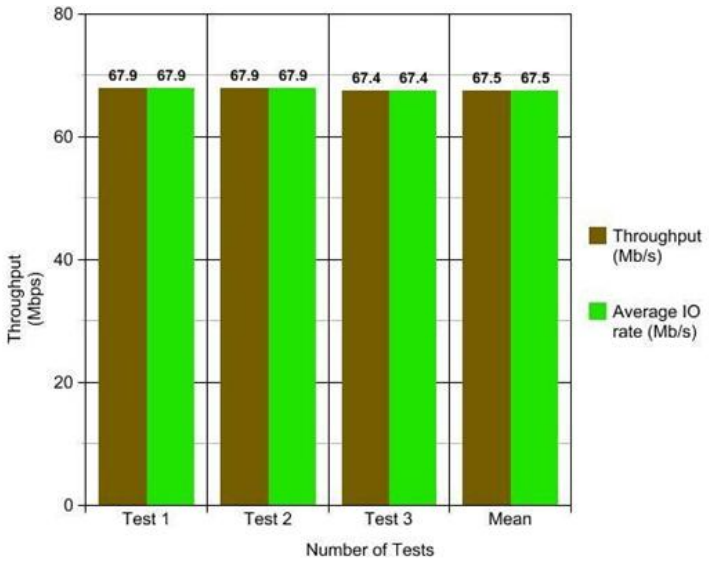
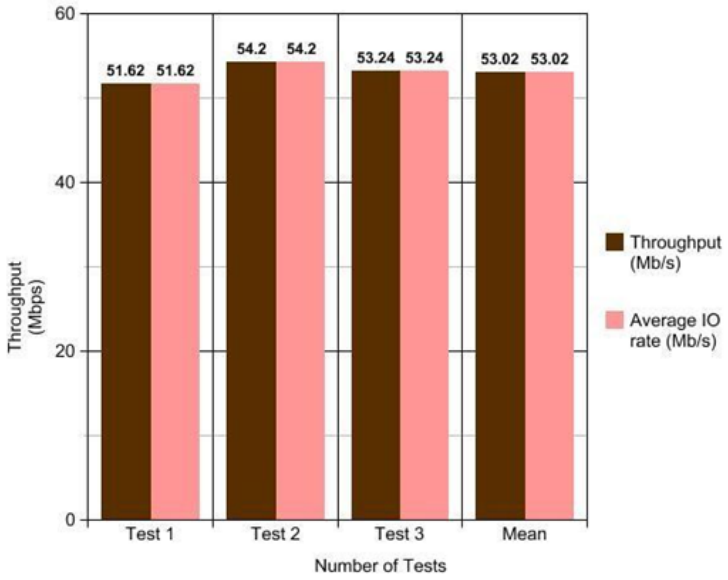


Fig. 7: Read Operation Throughput Evaluation for 2GB, Block size of 64 MB



51

Fig. 8: Read Operation Throughput Evaluation for 4GB, Block size of 64 MB

4) 512 MB, 2GB, 4GB with Block size of 128 MB: In the following experiments we analyse the results by comparison with the increased block size of 128 MB. The figures 9, 10, 11 refers to the graph for 3 tests taken with a replication factor of 3 and dateset of 512 MB, 2GB and 4GB with a bigger block size of 128 MB. The test is repeated 3 times for taking throughput and the input/output performance of the system thereby taking the standard deviation and the mean value. The graph in 9, 10, 11 is 68, 65 and 58 Mbps which is showing a gradual decreasing trend in the throughput values and shows that the average throughput achieved is nearly 65 Mbps for the given dataset and replication factor of 3. A very low standard deviation value of 0.009 shows that there is not much difference of throughput among the test values.

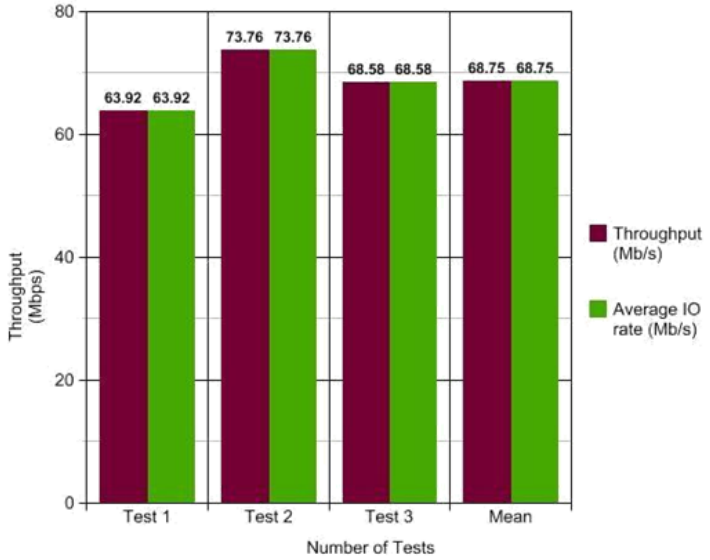


Fig. 9: Read Operation Throughput Evaluation for 512 MB, Block size of 128 MB

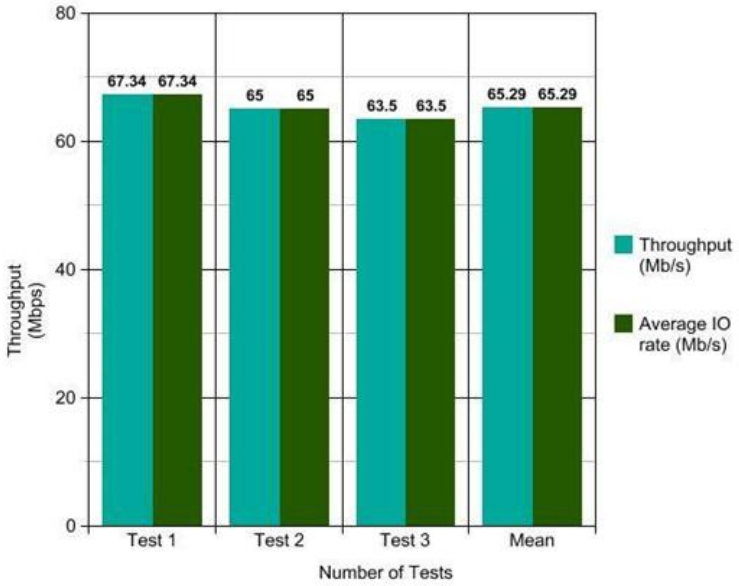


Fig. 10: Read Operation Throughput Evaluation for 2GB, Block size of 128 MB

5) Read Operation performance Evaluation: The reading performance with both block size 64 MB and 128 MB looks similar to each other too and faster than the writing. The reading performance with small files (e.g. 512 MB) is faster than with the big data set (e.g. 2 and 4 GB). Reading with a replicated file logically produces a slower performance. The 12 describes a comparison of all the read tests in the results and produces and graphs of throughput with respect to dataset size and the block size.

VI. Conclusion

In this paper, we have evaluated the performance of an a small hadoop cluster. We built a simulated environment of a hadoop cluster comprising of name nodes and data nodes. To measure the performance we set up a Hadoop cluster with 9 nodes and use the fileTestDFSIO.java of the Hadoop version 1.2.1 which gives us the data throughput, average I/O rate and I/O rate standard deviation. The HDFS writing performance scales well on both small and big data set. The average HDFS reading performance scales well on big data set where it is - however - lower than on the small data set. The more nodes a writing/reading operation is run on, the faster its performance is.

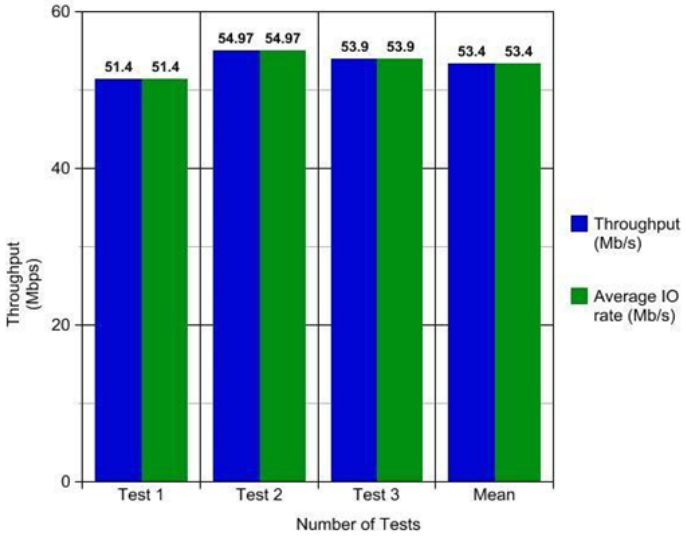


Fig. 11: Read Operation Throughput Evaluation for 4GB, Block size of 128 MB

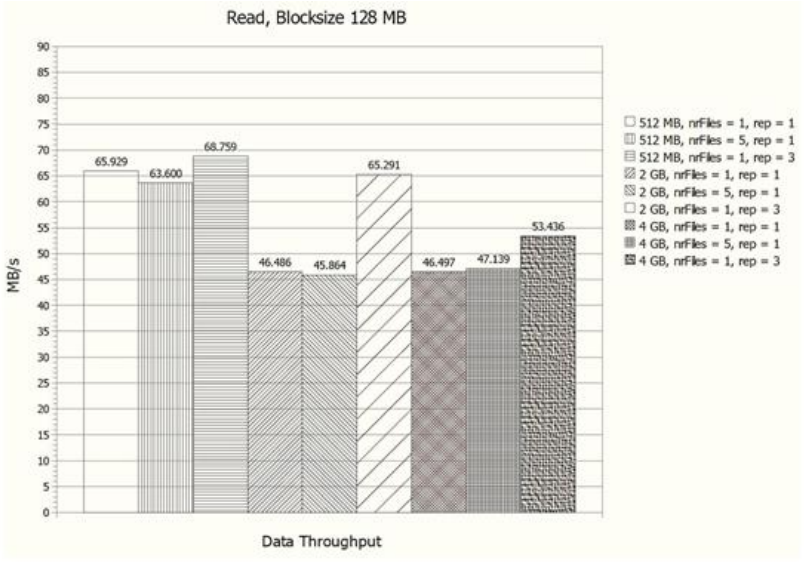


Fig. 12: Read Operation Throughput Evaluation Summary

Our results show that the writing performance of Hadoop scales better than reading with small data sets. But it doesnt matter because Hadoop is designed for the batch processing on huge data sets. So in this case its quite fine with the scalability. Furthermore the writing and reading performance are fast. If we write or read a data set with 20 GB distributed on 5 nodes, we will end up with approximately 160 MB/s and 181 MB/s. The more data nodes we have, the faster it is. In comparison with the local file system on the cluster the HDFS writing/reading performance is lower approximately 25 to 30 percent. The loss of HDFS performance is caused by the HDFS management and maybe Java IO overhead. Hadoop allows writing/reading parallel on all data nodes like other distributed file system. In addition, with Map reduce it is possible to perform Map reduce operations parallel and flexibly depending on users purposes.

References

[1] C. V. N. Index, "Forecast and methodology, 2014-2019 white paper," Retrieved 23rd September, 2015.
[2] P. Zikopoulos, C. Eaton et al., Understanding big data: Analytics for enterprise class hadoop and streaming data. McGraw-Hill Osborne Media, 2011.
[3] J. Dittrich and J.-A. Quiane´-Ruiz, "Efficient big data processing in hadoop mapreduce," Proceedings of the VLDB Endowment, vol. 5, no. 12, pp. 2014–2015, 2012.
[4] T. Hey, S. Tansley, K. M. Tolle et al., The fourth paradigm: data-intensive scientific discovery. Microsoft research Redmond, WA, 2009, vol. 1.
[5] H. Liao, J. Han, and J. Fang, "Multi-dimensional index on hadoop distributed file system," in Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference on. IEEE, 2010, pp. 240– 249.
[6] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1,

pp. 107–113, 2008.
[7] F. Wang, J. Qiu, J. Yang, B. Dong, X. Li, and Y. Li, "Hadoop high availability through metadata replication," in Proceedings of the first international workshop on Cloud data management. ACM, 2009, pp. 37–44.
[8] L.-W. Lee, P. Scheuermann, and R. Vingralek, "File assignment in parallel i/o systems with minimal variance of service time," IEEE Transactions on Computers, vol. 49, no. 2, pp. 127–140, 2000.
[9] W. Li, Y. Yang, and D. Yuan, "A novel cost-effective dynamic data replication strategy for reliability in cloud data centres," in Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth Interna-tional Conference on. IEEE, 2011, pp. 496–502.
[10] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "Cdrm: A cost-effective dynamic replication management scheme for cloud storage cluster," in 2010 IEEE international conference on cluster computing. IEEE, 2010, pp. 188–196.
[11] S.-Q. Long, Y.-L. Zhao, and W. Chen, "Morm: A multi-objective optimized replication management strategy for cloud storage cluster," Journal of Systems Architecture, vol. 60, no. 2, pp. 234–244, 2014.
[12] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench bench-mark suite: Characterization of the mapreduce-based data analysis," in New Frontiers in Information and Software as Services. Springer, 2011, pp. 209–228.