Investigation of Replication Factor for Performance Enhancement in the Hadoop Distributed File System

Hilmi Egemen Ciritoglu
University College Dublin
School of Computer Science
Dublin, Ireland
hilmi.egemen.ciritoglu@ucdconnect.ie

Leandro Batista de Almeida*
University College Dublin
School of Computer Science
Dublin, Ireland
leandro@utfpr.edu.br

Eduardo Cunha de Almeida
Universidade Federal do Paraná
Curitiba, Brazil
eduardo@inf.ufpr.br

Teodora Sandra Buda
IBM Ireland
Cognitive Computing Group
Innovation Exchange
Dublin, Ireland
tbuda@ie.ibm.com

John Murphy
University College Dublin
School of Computer Science
Dublin, Ireland
j.murphy@ucd.ie

Christina Thorpe
University College Dublin
School of Computer Science
Dublin, Ireland
christina.thorpe@ucd.ie

ABSTRACT

The massive growth in the volume of data and the demand for big data utilisation has led to an increasing prevalence of Hadoop Distributed File System (HDFS) solutions. However, the performance of Hadoop and indeed HDFS has some limitations and remains an open problem in the research community. The ultimate goal of our research is to develop an adaptive replication system; this paper presents the first phase of the work - an investigation into the replication factor used in HDFS to determine whether increasing the replication factor for in-demand data can improve the performance of the system. We constructed a physical Hadoop cluster for our experimental environment, using TestDFSIO and both the real world and the synthetic data sets, NOAA and TPC-H, with Hive to validate our proposal. Results show that increasing the replication factor of the 'hot' data increases the availability and locality of the data, and thus, decreases the job execution time.

KEYWORDS

Hadoop Distributed File System, Performance Testing, Replication Factor.

ACM Reference Format:
Hilmi Egemen Ciritoglu, Leandro Batista de Almeida, Eduardo Cunha de Almeida, Teodora Sandra Buda, John Murphy, and Christina Thorpe. 2018. Investigation of Replication Factor for Performance Enhancement in the Hadoop Distributed File System. In *Proceedings of ACM/SPEC International Conference on Performance Engineering (ICPE'18 Companion).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3185768.3186359

*Also with Universidade Tecnológica Federal do Paraná.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICPE'18 Companion, April 9–13, 2018, Berlin, Germany
© 2018 Association for Computing Machinery.
ACM ISBN ISBN 978-1-4503-5629-9/18/04…$15.00
https://doi.org/10.1145/3185768.3186359

1 INTRODUCTION

In recent years, there has been a rapidly increasing demand for big data utilisation, from data analysis systems to data generated by technologies such as the Internet of Things. Coupled with the growing volume of structured and unstructured data across large enterprises, this has resulted in the significant penetration of big data systems, such as Hadoop. Apache Hadoop is an open source software platform for distributed storage and distributed processing of very large data sets on clusters of computers. As a component of Hadoop, Hadoop Distributed File System (HDFS) forms the foundation of this platform; it is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications [13].

However, despite the growing popularity of Hadoop, it is still not optimal in terms of performance; for example, the work in [12] evaluates the architecture to identify the bottlenecks and portability limitations. Consequently, there has been a lot of activity in the research community to develop new methods to improve the efficiency of the technology, including data placement techniques [8, 16].

Several studies have been carried out to understand the workload of Hadoop clusters. It has been observed that 80% of total jobs are executed on less than 10% of all data, and 80% of data re-accesses occurs within an hour [3]. Also, authors underlined that although there is diversity in workloads, all workloads follow Zipf-like distribution for recurrent data access. This shows that small portions of a data set can be significantly more in-demand than the rest of the data set, i.e., a data set typically has 'hot' data that can vary over time, therefore, an adaptive mechanism is necessary to optimise the provisioning for these workloads. Another study, noted that one of the main concerns for a big data cluster is the access time for the data from different nodes. It has been shown that the network can easily become a system bottleneck if there is remote HDFS access [11]. Thus, improving data locality is also an important consideration for Hadoop performance.

HDFS uses a default replication factor of 3, meaning it stores 3 copies of every file segment (block) in the file system. From a fault tolerance perspective, 3 copies is sufficient to reduce the risk of data loss to negligible levels, however, the replication factor may also be used to increase the availability and locality of the data in the cluster. This paper proposes a modified replication factor to increase the availability of the most in-demand or 'hot' data i.e., files that are used more frequently will have a larger replication factor. We conducted performance tests in order to evaluate the impact of tuning the replication factor. This paper details the test methodology developed and tools used to asses the hypothesis. The future direction for this research will be the development of a dynamic replication factor framework.

In order to validate our proposal, we constructed a physical test bed, consisting of a 20-node Hadoop cluster. Apache Hive [14] was used to manage the data sets residing on the distributed file system using SQL. We defined a comprehensive set of experiments, with different queries executed on three different data sets (TestDFSIO, NOAA, and TPC-H).

Our results show that increasing the replication factor of the most in-demand data increased the availability of that data across the cluster, and thus, decreased the time taken to execute a job.

The remainder of the paper is organised as follows: Section 2 presents the details of the technologies underpinning this research. Section 3 details approach and motivation behind this paper. Section 4 discusses the setup and methodology employed for each of the experiments, Section 5 presents the results of the experiments and provides an in-depth analysis. Section 6 related works. Finally, Section 7 concludes this paper and proposes some future work.

2 BACKGROUND

The concept of Big Data is quite broad, but can be briefly defined as the efficient and scalable analytical processing of large volumes of complex data, produced by possibly several applications. In order to process big data efficiently, new storage and distributed processing techniques have been developed. One of the existing techniques is MapReduce, which was proposed by Google [5] and is based on a distributed file system (GFS). This platform was designed to simplify distributed processing tasks that required linear scalability along with an ecosystem of applications developed from this emerging technology.

2.1 Hadoop Overview

Apache Hadoop [7] is an open source implementation of a distributed file system that was inspired by GFS and a distributed processing manager based on the MapReduce paradigm [5]. Hadoop has a framework that allows the development of MapReduce applications in several programming languages. Programmers can create their own Map and Reduce functions, as well as allowing communication with different systems that can be placed on top of Hadoop to execute large-scale distributed processing. Hadoop is based on a cluster architecture, using conventional, commodity machines. The architecture of a Hadoop system is divided into two main modules: the distributed file system (HDFS - Hadoop Distributed File System) and the distributed processing and job manager (MapReduce v1.0 or YARN).

2.2 Hadoop Distributed File System (HDFS)

Hadoop uses HDFS [13] as a file system, which is responsible for the persistence and consistency of distributed data. HDFS also has a transparency for network failures and data replication. In order for Hadoop to process data, it must be stored in a HDFS directory. HDFS is designed to manage enormous data sets, typically TBs in size (or larger), also providing high availability and fault tolerance even if performing computation on commodity hardware. HDFS can serve different processing/job management systems, such as Spark, Impala, or data warehouse systems that run on MapReduce/YARN jobs [7].

The smallest unit in HDFS is a block and each file in HDFS is divided into blocks with a default block size of 128 MB. The file blocks are replicated and distributed all over the cluster. As HDFS clusters are designed to use commodity machines, the chance of failure is considerable. In this way, the blocks storing the files are replicated in 3 different computers (default configuration), to allow greater tolerance to the failures. The replication factor is a file-level setting, which can be set at the time of file creation and changed later, if necessary. An application can also specify the number of replicas of a file by using the HDFS API.

Hadoop will always try to process the data already found on the node instead of transferring the information. This is done by the principle that "Moving Computation is Cheaper than Moving Data" [7], where Hadoop will copy the executable code of a job to the nodes where the data is, since it is much more expensive to move the data. Having more replicas can help Hadoop to increase availability of data, therefore, the scheduler can run mappers to process data locally instead of requesting data remotely.

2.3 Hive

Hive [14] is a data warehousing software built on top of Hadoop, using its distributed storage and processing capabilities. The main advantage of using Hive is the ability to use SQL (Structured Query Language) to perform searches on data stored in a Big Data system (Hadoop). Using Hive queries, you can avoid writing MapReduce programs, optimising development time. Hive was selected for this research project because it is relatively easy to run SQL queries when compared to writing MapReduce jobs.

3 DYNAMIC REPLICATION FRAMEWORK

3.1 High-level Design of Framework

The motivation behind this research is to understand the relationship between the replication factor and the system performance. This insight will be used to develop an adaptive framework for optimising the replication factor according to file access pattern, that is shown in Figure 1. This system will monitor the overall system during a measurement interval and collect job metrics for each MapReduce job. Query history from each different framework (such as Hive, Pig, SparkSQL etc.) will be recorded and will include information about what data was processed by the query. Each query will be decomposed by table and condition so that the data access pattern can be established by combining this decomposed information with job metrics (e.g., workload of a cluster, data locality, duration and frequency of queries). Hot data will be identified and ranked by density and the frequency of the access. Finally, the framework will optimise the replication factor by the request of each file if the trend of 'hot' data is detected or changes over the time. We are planning to implement detection solution on the

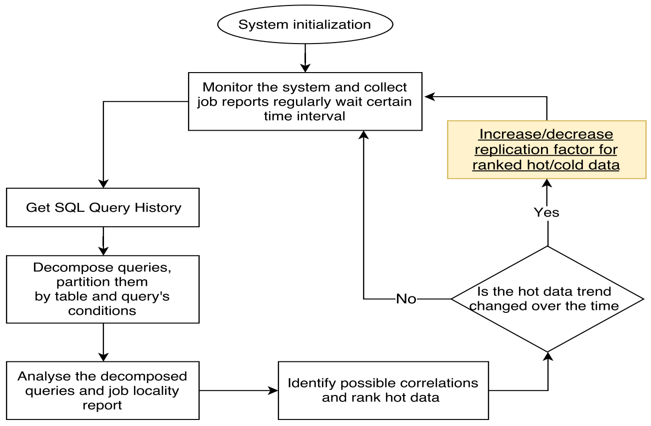
System initialization
Monitor the system and collect job reports regularly wait certain time interval
Increase/decrease replication factor for ranked hot/cold data
Get SQL Query History
Yes
Decompose queries, partition them by table and query's conditions
No
Is the hot data trend changed over the time
Analyse the decomposed queries and job locality report
Identify possible correlations and rank hot data

Figure 1: Architecture of Framework Planned

application layer (top layer of stack). Therefore, the framework does not need to analyse all of the Hadoop logs; it will just need to retrieve information from the application layer only. Considering that Hadoop log files can grow rapidly, this approach will improve the processing efficiency significantly and reduce overhead of the system. This automated learning sensor is left for the future work.

The scope of this paper focuses only on the performance evaluation of increasing the replication factor for 'hot' data, which is highlighted in Figure 1. In accordance with this purpose, a set of experiments has been conducted on a 10 and 20-node Hadoop cluster with varying replication factor.

3.2 Replicating 'Hot' Data

Hot data is the portion of the data which is heavily accessed (e.g., year, region, etc.). In this study, performance tests are conducted to investigate the impact of replication factor on reading time and overall system performance for 'hot' data. Increasing replication factor comes with a price, which is the disk space, network bandwidth, and time to copy the data into the system. However, replicating only a certain portion of data will minimise this overhead. By increasing the replication factor, the resource manager is much more likely to find a node that stores the data. Since the allocated node stores the related blocks, it can process these blocks locally rather than requesting them from another node and waiting for the data to be transferred. Our proposed solution increases the local accesses of jobs and the number of possible parallel mappers; consequently, we expect to see performance improvements.

4 EXPERIMENTS

The experiments were conducted in an isolated laboratory cluster, which was not shared with any other processes. A physical cluster was used instead of a virtual cluster in order to avoid possible deviations in measurements. For example, a virtualised environment would be subject to additional latency for disk reads due to contention between VMs.

4.1 Test Bed

The test bed cluster is composed of 21 'shared-nothing' computers (1 master and 20 slaves), connected by a Gigabit Ethernet Switch. All slave computers have a similar specification, with Intel Core i5 (5th generation) processors, 8 GB and 1 TB hard drive. The master computer has Intel Core i7 (6th generation) processors, 16 GB and 1 TB hard drive. Each machine was monitored using Ganglia during the tests; no errors were detected. A subset of these machines was used to create a smaller cluster of 10 machines to explore the impact of cluster size on the performance gain.

The tests were run on version 2.7.2 of Hadoop, with SQL queries running on a Hive version 1.2.2. Hive jobs ran on YARN using the MapReduce. This configuration was chosen to have a more stable and simple baseline.

4.2 Test Methodology

Read-only workloads have been used in this study as HDFS uses write-once, read-many model. Each test was executed 10 times for statistical soundness, and the average results are presented in Section 5. Before starting the tests, the data is loaded as a csv file (uncompressed) on to the Hadoop cluster, and then the replication factor is increased incrementally. It was observed in early exploratory tests that decreasing the replication factor can cause a imbalance of data in the cluster. So, after the test is finished, the data is deleted from the cluster.

We measured reading performance with TestDFSIO and executed throughput tests in NOAA and TPC-H, testing concurrent users, in order to evaluate the performance gains when multiple users try to access the same data set. This reflects the usual cluster behaviour observed in industry, as the majority of the installed clusters are multi-tenant, supporting several applications and users at same time. The number of users for each test run was defined as ***users* = {32, 64, 128}**. This is a reasonable number of users when we consider multi-tenant systems. Each user is concurrently querying the data with the same query; there is a 1-second interval between the start of each query.

4.3 Benchmarks & Data Sets

TestDFSIO: TestDFSIO is a well-known benchmark for assessing HDFS. It is used to assess the read/write performance for Hadoop clusters. As the read performance is directly related to overall job execution time, we measured the reading throughput performance under different replication factor values and different cluster sizes (10 nodes and 20 nodes). Two different data set sizes were tested (4GB and 8GB) to investigate the impact of the number of blocks on the performance.

NOAA: The NOAA data set is a set of Quality Controlled Local Climatological Data (QCLCD) [9] and represents a real world data set. It contains various pieces of information (42 columns) about weather (eg. measuring station's location, daily avg/max/min temperature, etc). Five years of data is used in this work (2008-2012, 26.3 GB, 217 million records). For the NOAA data set, we explicitly def ined data from one specific year (2010) as hot data. This means that although data from all years contained in the data set is added to the table, we only queried the 2010 year. Therefore, when using a modified replication factor for the data, we specified an increased number of replicas only for portion of data (November of 2010's).

The TPC Benchmark-H (TPC-H): TPC-H [10] is a decision support benchmark and represents a synthetic data set. The query set consists of different query types such as: selection, joining, filtering, grouping, aggregation, etc. The variety is provided in order to satisfy and simulate business operation needs. The TPC-H benchmark was created for evaluating the performance of relational database systems, however, in recent work published in the literature [6], it has been used for Hadoop-based query engines.

The study published in [11] states that the average input size for jobs on the Microsoft production cluster is approximately 13 GB. Additionally, another study on Hadoop clusters' workload on a Facebook cluster trace [3] revealed that "90% of jobs access files of less than a few GB". In light of these studies, we populated a 10 GB data set with the dbgen application and uploaded it to HDFS. The lineitem table used in this work contains approximately 60 million records and 16 different columns per record. For evaluation purposes, we created a table on Hive and set the location of data. Then, we only queried and increased replication factor for particular portion of data (details in Section 4.4).

4.4 Queries

In order to validate our proposal on the NOAA data set, we increased the replication factor for the December 2010 data. This year was exclusively queried using the filter 'WHERE year = 2010 and month = 12' in the Hive query. Although TPC-H has 22 different queries, the 1st and 6th queries were selected for use because they are the most relevant and suitable queries to test our proposal [2]. The 6th query of TPC-H can be represented with only one map-reduce job. Hive does not need to create multiple sub-jobs to accomplish the job, therefore the performance evaluation of the system will be more obvious. The 6th query in the TPC-H benchmark is searching for data between the years 1994 - 1995. In order to remain consistent and to ensure that all queries were executed on hot data, the TPC-H 1st query was modified to only filter on 1 year of data as opposed to the entire data set. As in the 6th query, the hot data was set to be 1994-1995. We consider hot data as the data with a date attribute value between these dates. Only the hot data from each data set is replicated and queried in our experiments. Table 1 details the three queries defined for the NOAA (row 1) and TPC-H data sets (row 2-3).

5 RESULTS

The replication factor has an impact on the data availability, which can be related to reading time, and ultimately, execution time. In this study, we evaluated the performance of Hadoop using an increasing replication factor for hot data.

5.1 Average Job Execution Time

TestDFSIO: We assessed the reading throughput by using TestDFSIO with an increasing replication factor. Figure 2 shows that the replication factor can improve reading throughput significantly. Having more copies of data provides higher data locality and availability, thus improving the reading time, and ultimately improving the job execution time. Two different sets of this data were created. Additionally, two different test beds, one with 10 nodes and another with 20 nodes, were created in order to understand the job's behaviour on different cluster sizes. Regardless of the different cluster size, we see a performance improvement in reading throughput when the replication factor is increased.

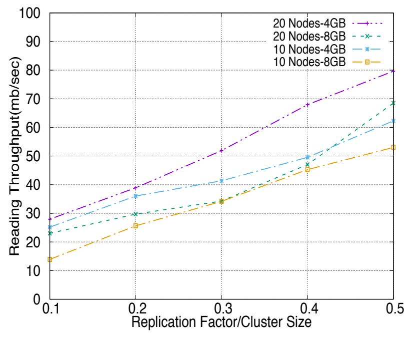

Figure 2: TestDFSIO-Reading throughput on 10/20 nodes

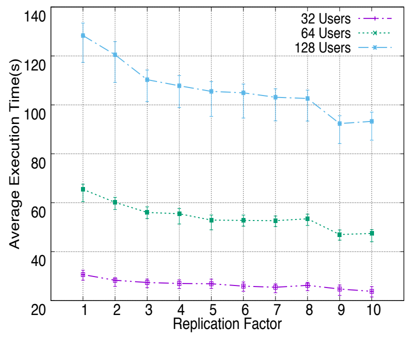

Figure 3: NOAA -Test with concurrent user on 20 nodes

NOAA: Figure 3 plots the results of the experiment on the NOAA data set. Average execution time is reduced from 30s to 23s for 32 users, from 60s to 44s for 64 users and from 133s to 97s for 128 users by changing replication factor from 1 to 10. Therefore, we can see performance gain of approximately 13% when the replication factor for hot data is increased from 3 to 10 for 32 concurrent users. This gain is increased to 16% for 64 users and 17% for 128 users. The decrease in the job execution time can be attributed to an increase in the number of nodes (and the processing slots in each node) available with the data blocks needed for the mapper tasks. This increased availability supports an increased parallelism, and thus, a decrease in job execution. Also, we can see better performance improvements when data is accessed intensively, i.e., when the concurrency is higher.

TPC-H: Figure 4 plots the results of the experiment on TPC-H on the 20 node cluster using the 1st query. The improvement in job execution time is approximately 11% for 32 users, 13% for 64 users, and 15% for 128 users as the number of replicas were increased from default replication factor for hot data.

Figure 5 plots the results of the experiment on TPC-H on the 20 node cluster using the 6th query. It can be seen that the job execution time is reduced by approximately 20% for the test with 128 concurrent users and 17% with 32 and 64 concurrent users when the replication factor were increased from 3.

	Query	Characteristics

NOAA	SELECT COUNT(*) FROM noaa WHERE year = 2010 and month = 12	SELECT
		WHERE

TPC - 1st	SELECT l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price, sum(l_extendedprice * (1 - l_discount)) as sum_disc_price, sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge, avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc, count(*) as count_order FROM lineitem WHERE l_shipdate_year >=1994 AND l_shipdate_year <1995 GROUP BY l_returnflag, l_linestatus ORDER BY l_returnflag, l_linestatus;	SELECT WHERE AGGREGATION GROUPING ORDERING

TPC - 6th	SELECT sum(l_extendedprice * l_discount) as revenue FROM lineitem WHERE l_shipdate_year >= 1994 AND l_shipdate_year <1995 AND l_discount >= 0.05 AND l_discount <= 0.07 AND l_quantity <24;	SELECT
		WHERE
		AGGREGATION

Table 1: Queries defined to search for data

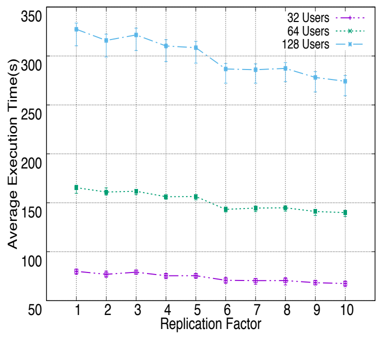

Figure 4: TPC-H Q1 - Test with concurrent user on 20 nodes test bed

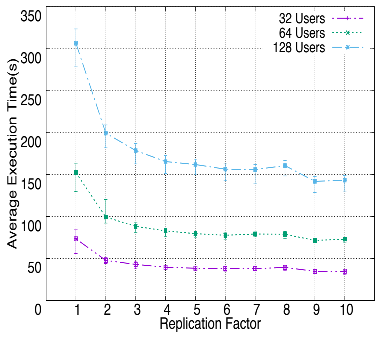

Figure 5: TPC-H Q6 - Test with concurrent user on 20 nodes test bed

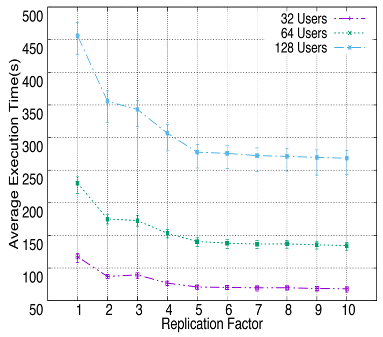

Figure 6: TPC-H Q6 - Test with concurrent user on 10 nodes test bed

In both experiments improvements can be seen until replication factor of 9. It's important to note that, when the data is heavily accessed and requested by concurrent queries, the performance improvement is more compelling. Additionally, the performance gain trend is similar in both experiments. The job execution time plateaus at a replication factor 9 with concurrent users. However, if the data is not concurrently queried or there is only a few jobs running at the same time on the cluster, the performance gain becomes marginal at a replication factor of 4. The results show that when the hot data is frequently and concurrently queried or the cluster processing utilisation is very dense, replication will play a key role in the system performance. Therefore, increasing the replication factor for hot data improves the system performance notably.

Experiment results show that the performance of Hadoop is tightly coupled with the data availability, access latency and executing mappers locally. Although, increasing the replication factor can create a storage overhead, but increasing for only the hot data only will minimise this overhead.

5.2 Impact of Cluster Size

To explore the impact of cluster size, we used 2 different test beds, a 10-node cluster and a 20-node cluster. Both Figure 6 and Figure 5 show TPC-H Q6 results on different cluster size. Results reveal that, the performance improvements of approximately 41% are achieved for the tests on 10 nodes with different number of concurrent user up to replication factor 5; however, these improvements are approximately 53% on 20 nodes, when we increase replication factor from 1 to 9. So the impact of increasing the replication factor has a much more significant effect on system performance when the cluster has more nodes. Therefore, it is reasonable to say that continuing to increase the replication factor on a real production cluster (thousands of nodes), could afford an even greater performance improvement for larger numbers of replicas.

5.3 Data Loading Overhead

When data is replicated on a cluster, it adds an additional overhead. Since increasing replication is the core of our proposal, the data loading time overhead introduced was measured. Creating additional copies of data blocks takes a similar amount of time for every increment of replication, which is always around approximately 11 seconds for this test. On the other hand, each query that runs on hot data, can reduce the execution time up to 40 seconds. When we consider hot data is getting queried simultaneously, the performance improvement of the system will be very significant even if increasing the number of replicas introduces an additional overhead. This overhead is relatively low when we compare against the performance enhancement.

5.4 Summary

During our experiments, we observed a performance improvement on the job's mapping phase when we increased replication factor.

Also, the experiments show that, once the data availability threshold is reached, no more performance gains can be achieved by continuing to increase the replication factor. It is important to note that gain is coupled with cluster size, and our tests are conducted on a 10 - 20 node cluster to demonstrate this. In our case we can see performance improvement up to a replication factor of 9 on a 20 node cluster and 5 on a 10 node cluster. Although, we increase the replication factor for only hot data, the Hadoop job execution takes approximately 18% - 20% less time.

6 RELATED WORK

The replication factor, in addition to the placement of these replicas across data nodes, are two critical subjects related to fault tolerance, data availability and network utilisation in distributed file systems.

Wei et. al. [15] propose CDRM; it is a cost-effective dynamic replication management for cloud storage clusters. CDRM proposes a model to reveal the relationship between the replication factor and the data availability. So, CDRM can determine only lower bound of the replication factor for satisfy the availability requirement. But authors need to modify HDFS caching mechanism and they cached whole file blocks for CDRM implementation. It is obvious that this solution does not perform well for terabytes of data.

Abad et. al. propose DARE [1]; it is an adaptive data replication for efficient cluster scheduling. DARE uses a reactive replication budget technique with probabilistic sampling and an ageing algorithm. Also, it uses existing remote data access; hence, it does not result in extra network consumption.

Cheng et. al. propose ERMS [4]; an elastic replica management system, which increases job locality by replicating hot data more and keeps a minimum number of replicas for the cold data. Another contribution of ERMS is the development of the log parser to analyse data directly from the HDFS level. Since HDFS audit logs are rapidly increasing in size, and it is not easy to process all data, so authors need to propose a new system.

The work in this paper investigates the impact of increasing the replication factor for hot data on the data availability and ultimately the system performance. We plan to extend this work to include an automated hot data sensor that will detect hot data. We do not intend to inspect the HDFS audit logs; therefore, our approach will significantly reduce processing complexity.

7 CONCLUSION

The focus of the research detailed in this paper was to investigate the replication factor employed by HDFS. Specifically, to determine if increasing the replication of the most accessed data can have a positive impact on performance; and test this proposal using a correct methodology. We constructed a physical Hadoop cluster running Hive and executed a comprehensive set of experiments on the TestDFSIO, NOAA, and TPC-H data sets, with different cluster sizes, and varying numbers of concurrent users.

Despite the small size of the cluster, results clearly show that creating more copies of the hot data increases the availability of the data. This increased availability can support additional concurrent mapper jobs, thus, reducing the job execution time. Our cluster consists of only a maximum of 20 nodes, and therefore, the performance gains we achieved are limited. When the replication factor is increased from 3 to 9, we see approximately 18-20% decrease in job execution time, which remains fairly stable after replication factor of 9. The reason for this is that all data nodes in the cluster are well-balanced at 9 replicas, and therefore the number of mapper tasks that can be executed in parallel reached the maximum at this point. However, if the cluster was larger (e.g., thousands of data nodes), it is reasonable to expect the level of parallelism of jobs(specifically mapper tasks) to be much greater and therefore the job execution time to be reduced considerably further.

Future work will include developing an adaptive replication scheme that will include a new mechanism for detecting hot data using the application layer properties (e.g., Hive query analysis), and implement a dynamic replication factor that can be increased automatically as data becomes hot and decreased as data cools down.

ACKNOWLEDGEMENT

This work was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094

REFERENCES

[1] Cristina L Abad, Yi Lu, and Roy H Campbell. 2011. DARE: Adaptive data replication for efficient cluster scheduling. In *Cluster Computing, 2011 IEEE International Conference on.* IEEE, 159–168.

[2] Peter Boncz, Thomas Neumann, and Orri Erling. 2013. TPC-H analyzed: Hidden messages and lessons learned from an influential benchmark. In *Technology Conference on Performance Evaluation and Benchmarking.* Springer, 61–76.

[3] Yanpei Chen, Sara Alspaugh, and Randy Katz. 2012. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proceedings of the VLDB Endowment* 5, 12 (2012), 1802–1813.

[4] Zhendong Cheng, Zhongzhi Luan, You Meng, Yijing Xu, Depei Qian, Alain Roy, Ning Zhang, and Gang Guan. 2012. ERMS: An elastic replication management system for hdfs. In *Cluster Computing Workshops, 2012 IEEE International Conference on.* IEEE, 32–40.

[5] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[6] Avrilia Floratou, Umar Farooq Minhas, and Fatma Özcan. 2014. SQL-on-Hadoop: Full Circle Back to Shared-nothing Database Architectures. *Proc. VLDB Endow.* 7, 12 (Aug. 2014), 1295–1306.

[7] Apache Software Foundation. 2018. Apache Hadoop. (2018). https://hadoop.apache.org.

[8] Hui Jin, Xi Yang, Xian-He Sun, and Ioan Raicu. 2012. Adapt: Availability-aware mapreduce data placement for non-dedicated distributed computing. In *Distributed Computing Systems, 2012 IEEE 32nd International Conference on.* IEEE, 516–525.

[9] NOAA. 2018. NOAA Data set. (2018). https://www.ncdc.noaa.gov/data-access.

[10] Meikel Poess and Chris Floyd. 2000. New TPC benchmarks for decision support and web commerce. *ACM Sigmod Record* 29, 4 (2000), 64–71.

[11] Antony Rowstron, Dushyanth Narayanan, Austin Donnelly, Greg O'Shea, and Andrew Douglas. 2012. Nobody ever got fired for using Hadoop on a cluster. In *Proceedings of the 1st International Workshop on Hot Topics in Cloud Data Processing.* ACM, 2.

[12] Jeffrey Shafer, Scott Rixner, and Alan L Cox. 2010. The hadoop distributed filesystem: Balancing portability and performance. In *Performance Analysis of Systems & Software, 2010 IEEE International Symposium on.* IEEE, 122–133.

[13] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The hadoop distributed file system. In *Mass storage systems and technologies, 2010 IEEE 26th symposium on.* IEEE, 1–10.

[14] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. 2009. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1626–1629.

[15] Qingsong Wei, Bharadwaj Veeravalli, Bozhao Gong, Lingfang Zeng, and Dan Feng. 2010. CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster. In *Cluster Computing, 2010 IEEE International Conference on.* IEEE, 188–196.

[16] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares, and Xiao Qin. 2010. Improving mapreduce performance through data in heterogeneous hadoop clusters. In *Parallel & Distributed Processing, Workshops and Phd Forum, 2010 IEEE International Symposium on.* IEEE, 1–9.