Algorithms for Intelligent Systems

Series Editors: Jagdish Chand Bansal · Kusum Deep · Atulya K. Nagar

Dinesh Goyal · Valentina Emilia Bălaş ·
Abhishek Mukherjee ·
Victor Hugo C. de Albuquerque ·
Amit Kumar Gupta *Editors*

Information Management and Machine Intelligence

Proceedings of ICIMMI 2019

Springer

Algorithms for Intelligent Systems

This book series publishes research on the analysis and development of algorithms for intelligent systems with their applications to various real world problems. It covers research related to autonomous agents, multi-agent systems, behavioral modeling, reinforcement learning, game theory, mechanism design, machine learning, meta-heuristic search, optimization, planning and scheduling, artificial neural networks, evolutionary computation, swarm intelligence and other algorithms for intelligent systems.

The book series includes recent advancements, modification and applications of the artificial neural networks, evolutionary computation, swarm intelligence, artificial immune systems, fuzzy system, autonomous and multi agent systems, machine learning and other intelligent systems related areas. The material will be beneficial for the graduate students, post-graduate students as well as the researchers who want a broader view of advances in algorithms for intelligent systems. The contents will also be useful to the researchers from other fields who have no knowledge of the power of intelligent systems, e.g. the researchers in the field of bioinformatics, biochemists, mechanical and chemical engineers, economists, musicians and medical practitioners.

The series publishes monographs, edited volumes, advanced textbooks and selected proceedings.

More information about this series at http://www.springer.com/series/16171

Dinesh Goyal · Valentina Emilia Bălaş ·
Abhishek Mukherjee ·
Victor Hugo C. de Albuquerque ·
Amit Kumar Gupta
Editors

Information Management and Machine Intelligence

Proceedings of ICIMMI 2019

Springer

Editors
Dinesh Goyal
Poornima Institute of Engineering
and Technology
Jaipur, Rajasthan, India

Abhishek Mukherjee
CISCO Technologies
Milpitas, CA, USA

Amit Kumar Gupta
Poornima Institute of Engineering
and Technology
Jaipur, Rajasthan, India

Valentina Emilia Bălaş
Department of Automatics
and Applied Informatics
Aurel Vlaicu University of Arad
Arad, Romania

Victor Hugo C. de Albuquerque
Universidade de Fortaleza
Fortaleza, Brazil

ISSN 2524-7565 ISSN 2524-7573 (electronic)
Algorithms for Intelligent Systems
ISBN 978-981-15-4935-9 ISBN 978-981-15-4936-6 (eBook)
https://doi.org/10.1007/978-981-15-4936-6

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Contents

Performance Improvement of Heterogeneous Cluster of Big Data Using Query Optimization and MapReduce

Pankaj Dadheech, Dinesh Goyal, Sumit Srivastava, Ankit Kumar, and Manish Bhardwaj

1 Introduction

Hadoop is scalable and capable of managing substantial data volumes with same or homogeneous kind of clusters since the data movement and processing capacities of servers remain same all around the network. In case of heterogeneous clusters, each node comprises of a host with different speeds of storage and processing capabilities. The task of processing tasks then is changed from slow-end nodes into high-end nodes. This strategy works good once the number of information to be processed is less or job load is low, but it fails to improve the rate of job processing on Hadoop heterogeneous clusters if processing and data involves huge volumes of data sets. The improvement of the hardware product is another issue where processors can be improved but it reveals quite expensive. Another solution for this matter is to improve performance of those heterogeneous clusters using different analytical techniques to analyze structured and unstructured information [1]. The MapReduce algorithm

P. Dadheech (✉) · A. Kumar
Department of Computer Science & Engineering, Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur, Rajasthan, India
e-mail: pankajdadheech777@gmail.com

A. Kumar
e-mail: iiita.ankit@gmail.com

D. Goyal · M. Bhardwaj
Poornima Institute of Engineering & Technology, Jaipur, Rajasthan, India
e-mail: dinesh8dg@gmail.com

M. Bhardwaj
e-mail: manishbhardwaj.it@gmail.com

S. Srivastava
Manipal University Jaipur, Jaipur, Rajasthan, India
e-mail: sumit.310879@gmail.com

D. Goyal et al. (eds.), *Information Management and Machine Intelligence*,
Algorithms for Intelligent Systems,
https://doi.org/10.1007/978-981-15-4936-6_9

divided the data classification activities and assigned those jobs to different computers on the system. When every task was computed and finished processing, it was gathered and introduced as the final outcome. The MapReduce algorithm was shown to be very effective as now, large data sets could be readily processed and that also in a reduced time period. The group of Mike Caferella and Doug Cutting with additional members in 2005 employed the MapReduce algorithm and laid the base of Hadoop. It is able to deal with large amount of data sets as in big data and provide insights and meaning of those data statistically. The data generated today can be in any kind of format, be it a simple text message, an audio clipping, video or even a file with information of customers. Broadly speaking data may be user generated, machine generated, structured form where data is organized to offer some significance, or unstructured in which information is in raw format. An analytical survey shows that 33% of information generated digitally can be useful for different situations. But today on 0.5% of information is used to seriously analyze the situation. The limited analytical capabilities of the existing algorithmic approaches a very major chunk of information insights and meanings have been missing. The usage of MapReduce in data warehousing systems implies that analytic queries aren’t sufficient for the data outburst experienced now. A quick look on information generation and classification reveals that the rate of ability of data movement on/off hard drive is slow compared to the online data development. Though the capacities of hard drives are also advancing day by day yet, processor speeds or functionality have comparable growth rates as of hard drives. The major bottle neck of this data throughput mismatch occurs due to the motion of information on and off the hard disk drive [2]. Hadoop is also an open-source framework which is used to store in addition to process information using different approaches in a distributed environment. It can manage data flow in a distributed manner across a various bunch of computers in such a way that data access gets fast, reliable, and secure. It entails use of computing and programming models for handling data storage or distribution over large and widespread geographical areas. Going with the conventional databases was quite a good alternative with predefined software written to utilize them. It might be used to analyze information in consumer in addition to business standpoint but it worked well only till the data was confined to some less quantity. With larger volumes of information, processing large number of data became an impossible task. The processing and managing of information within a single or bunch of computers has been also a tedious task as there were risks of node failures, data reduction, and slow processing times. The conventional database servers proved to be a massive risk to store and process data when the amount of data doubled exponentially [3].

2 Literature Review

Another such implementation of good clustering algorithm is conducted by the authors of the paper, [4] where they have used K-means and TF-IDF on MapReduce for clustering of distributed documents. The authors have proposed that as data is accumulating over Internet rapidly, the amount of documents or data to be processed has starting causing overheads for Web servers. The authors have proposed a new type algorithm to cluster the data they are taking in form of documents. They have initiated a preprocessing phase where they calculate the respective TF-IDF weights in the documents for further processing. The experiments conducted reveal that the precision of clustering has improved by incorporating K means and TF-IDF algorithm [5]. It can help in processing thousands of documents processed concurrently at high pace and in stipulated time period. As mentioned in earlier works of distinct authors it is evident that big data requires a lot of mining along big data sets to find the right keyword, but it comes with a cost that lots of organizations have to pay as to finding a difficult pattern in large sets is already a tedious task. This problem is raised as a concern in the paper by Rong [6]. The algorithm implemented here is introduced in MapReduce framework to map the memory as and when required the data items from big data. The authors have accessed and analyzed few of the Web logs spanning over simple databases and reached to the conclusion that traditional database do not offer a good solution in managing and accessing because data cannot reside on a single machine and data movement over long distances is not feasible enough. The cost of computing also increases manifold when the data items in a cluster are in peta or zeta bytes. Many developers and researchers have shifted their focus on developing algorithms in MapReduce framework as it supports a wide variety of options and can contend with real case scenarios which was lacking in earlier implementations [7, 8].

As per the authors [9] deep neural networks (DNN) are a good way to represent the complexity of speech recognition systems. The authors have used singular value decomposition (SVD) in order to maintain and restructure the data model for learning effects of speech recognition in complex systems. The weights of each sub-point are collected and applied SVD upon. The paper gives a general idea on how the complexity of speech recognition systems could be managed by fine tuning, modeling the sparseness and training the recovery systems in manners so as they could be recognized and worked upon faster calculations.

The expenses of big data are relevantly written by [10] in his paper. He explains the scenario of testing tools, hardware, and software counterparts of the Hadoop and the big data that can help organizations to discover new ways to use the day and provide good monetization. He also advocates in finding the performance bottlenecks and what could be the solution to the idealistic features to improve them for further proceedings.

Another such analysis conducted by the author [11] in his paper suggests that clustering can really help large organizations process data that are not labeled and help them manifest all the resources effectively. They have used the method of filtering, uploading and then applying K-means to find effective results. The document clustering can be improved by using methods that distinguishably divided/filters or prunes documents and then held them in the Hadoop account. The subsequent steps of managing them into a cluster can help the document to be utilized at its fullest and reduce the data management jobs. The author has incorporated Davis–Bouldin index that measures the quality of cluster that is generated post Hadoop implementation step.

An application is designed around the architecture that once a document is fed into a system, it is cleansed from less important words by using Porter Stemmer, then for calculating the weighted frequency, TF-IDF is applied [12, 13]. The clustering algorithm used here, K-means, require inputs in numerical forms; therefore, the weight of important words are given as an input to the K-means algorithm after the document intermediary is uploaded on Hadoop ecosystem. Then, the subsequent steps are utilized to check the quality of clusters where closeness in clusters depicts similarity in documents and distance in clusters depicts the dissimilarity of clusters [14, 15]. One big issue that arises here is that the larger the data set the more effective the implementation begins. It is applicable for MapReduce frameworks which work well in highly complex environments.

The cluster quality is measured in Davis–Bouldin metrics which provides a statistical analysis of the document clustering done so far. The more the value of a cluster is nearer to 1, the bad the quality of cluster is. In this way, it is easier to manage cluster quality by large organizations and the impact of clustering in distributed documentation [16, 17]. The preprocessing steps help in determining the needful from whole lot of documents. The document clustering is one of the main headaches for companies that involve lot of go through in data sets. While big data continues to thrive on Internet, it is becoming more complex and expensive to maintain sanity among data processing. The need of such preprocessing clusters is highly requited so that it can be solve and managed efficiently without introducing the costs to storage, computation, and other means. The intent of clustering document has attracted many eminent researchers. One such research group have studied and designed algorithm for text clustering [18].

3 Proposed Work

3.1 Methodology

In this work, we enhance the performance of heterogeneous clusters on Hadoop by minding a set of policies that increase the input/output signal queries, enhances the routing of the algorithm into heterogeneous clusters, so improving the operation of processing of inquiries in such a fashion that they are readily attached with the ideal portion of implementation at minimal period because we could without raising the values at calculating degree [19]. This work includes that scenario in which we processed the complex information through clustering. We handle the clustering processing through query improvisation.

3.1.1 Solving Problems Through Iteration and Phasing

The information employed here for doing experiments have been all log files of sites that will be in the form of unstructured information. It is the very good method through which we could establish the efficacy of processes and it is also closely associated with the real-world environment where information is recorded in such condition. The information thus additionally assists in imagining the situation of true use of procedures which are closely associated with the real-world situation where one needs to take care of data that may arrive in any shape and shape [16, 20]. The information since log files have assisted in obtaining the question entry procedure for an e-commerce site that must take care of countless search key words, necessitates clustering of information items to purchase and supplying precise results in rather short periods of life. Within this paper breaking up the question processing system for MapReduce in successive pragmatic stages such as filtering, clustering, and enhancing the question for quicker execution procedures. Initially, the stage involves iterative actions to clean out the query and discover dependencies one of the question so that after a query is implemented, task tracker does not need to count on calculating query dependencies again after it had been calculated at the first actions. To lessen the query execution time, enhance the efficacy of clusters and such as the pragmatic processing in which the questions are enhanced through clustering them to comparable measures, employing TF-IDF, calculating weights, and handling the similarity matrix and ultimately submitting the question for implementation [8].

The procedure used here for calculating minimum rate and supplying output in comparatively smaller timeframe is represented Fig. 1.:

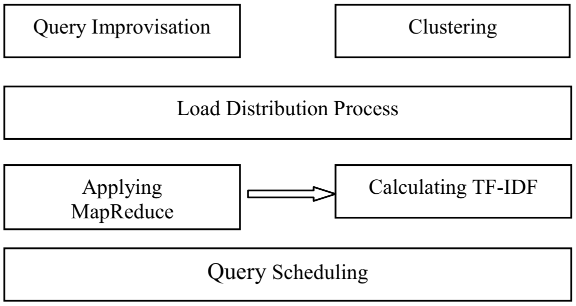


Fig. 1 Phasing methodology for heterogeneous clusters

3.1.2 Query Improvisation

Query improvisations are actually just a procedure through which people strengthen the performance of both clustering [6, 21]. Clustering can be employed when queries are brought into the parser and semantic analyzer they always compute the dependencies among these queries. But the moment this parsed question has been sent into the Hadoop's MapReduce to do the exact dependencies which have been calculated for your Hive query is dropped between your alterations. Once the dependencies are calculated they are used for semantics extractions in Hive QL chip. In the second phase, we are able to use these dependencies like logical predicates and input tables to method that the dependencies among distinct questions to become closely attached with each other during transition [9, 22]. When the semantic gap between both Hive and also Hadoop is bridged from these intermediary measures, they are sometimes readily utilized for clustering equivalent inquiries at query level, and thus by minute steps improving the query job performance (Fig. 2).

Steps of Query Improvisation Process which use the following property:

1. **Selection of Query**: A select with conjunctive conditions on the attribute list is equivalent to a cascade of selects upon selects:

$$QA1^\wedge A2^\wedge \ldots An(R) \equiv QA1(QA2(\ldots(QAn(R))\ldots))$$

2. **Calculating the Commutative property of Query**: We use select operation by commutative

$$A1(QA2(R)) \equiv QA2(QA1(R))$$

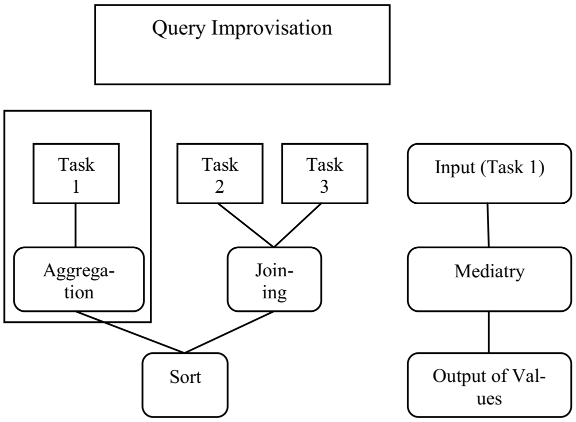


Fig. 2 Query improvisation process

3. **Calculating the value of cascade of p**: The value of cascade is the of project operations is equivalent to the last project operation of the calculated value:

$$\text{pAList1 (pAList2(}\ldots\text{(pAListn}(R))\ldots))$$
$$\equiv \text{pAList1}(R)$$

4. **Calculating the Commuting Query with the value of *p***: Given a P's and Q's attribute list of $A1, A2\ldots An$. The P and Q selection operations can be commuted:

$$\text{Commutative of } Q : A1, A2\ldots An(Qc(R))$$
$$\equiv Qc(P, A1, A2\ldots An(R))$$

5. **Calculate the value of *X***: We merge the join and Cartesian product operations which are commutative:

$$RXQ \equiv QXR \text{ and } RXS \equiv QXR$$

6. **Commuting Query with or *X***: Select can be commuted with join (or Cartesian product) as follows:

   a. If all of the attributes in the select's condition are in relation *R,* then Query, *c* (*R* Query) ≡ (Query *c* (*R*)) Query
   b. Given select the condition c composed of conditions $c1$ and $c2$, and $c1$ contains only attributes from *R*, and $c2$ contains only attributes from Query, then Query *c*(*R* Query) ≡ (Query $c1$ (*R*)) (Query $c2$ (Query))

7. **Commutative of set operations (*U*, ∩, −)**: Union and intersection operations are commutative; but the difference operation is not:

$$RUQ \text{ Selection} \equiv QUR, R \cap Q$$
$$\equiv S \cap Q, R - Q \neq Q{-}R$$

8. **Associativity of *X*, *U* and ∩**: All four of these operations are individually associative. Let θ be any one of these operators, then:

$$(R\theta Q)\theta T \equiv R\theta(Q\theta T)$$

9. **Commuting s with set operations (*U*, ∩, −)**: Let θ be any one of the three set operators, then:

$$Qc(R\theta Q) \equiv (Qc(R))\theta(Qc(S))$$

10. **Commuting P with U**: Project and union operations can be commuted:

$$\text{PAList}(RUQ) \equiv (\text{PAList}(R))U(\text{PAList}(Q))$$

3.1.3 Applying Clustering in Initial Phases

A lot of applications and systems use clustering as a part of dividing similar and dissimilar data items together. Clustering of data sets here can help achieve focusing on the items that are useful and discarding items which are generally not useful [23]. Looking from the view of a log file and applying clustering on it, we get clusters of user prone data that categorize the behavior of the item buying selections. The interactions of a user with an e-commerce Web site can identify good buying patterns easily. When this single end user data is collected from the multi-million users of the Web site and analyzed, the results tell a whole lot of insight on the consumer behavior, selection, buying options, and fields which need improvisation.

The log files used here symbolizes the unstructured data category and recording the data from different fronts, a presentation of heterogeneous Hadoop clusters [24, 25]. In this iterative clustering model when the data is captured from user end, it is optimized to remove static and dynamic parts of it in the clustering process itself where static parts are the decisions where dynamic parts are variants of time stamping, invoice numbers, delimiters, etc. It could be done through iteration process but applying a standard clustering algorithm with changes improves the efficiency as well.

3.1.4 Proposed Approach Algorithm

Step 1: We will read the log transition.
Step 2: We store the transition value in Q_i, Where $i = 1, 2, 3 \ldots n$. Where n is equal to the log transition value.
Step 3: We store the query which is request from client and store in array by get Query () method.
$IQ = I$ get query. Where I is number of query request. We store the query in array to create cluster. By method table put (null, array, parameter1, parameter 2 …. n);
To convert the array in object they are $Q_i =$ Table—get query ().
Step 4: Merge the pair of most similar queries (q_i, q_j) that does not convert the same queries. If (Q_i is not in C_i) then store the frequency of the item and the increase the value.
Step 5: Compute the simulating Matrix if Q_i in C_i.
Step 6: If Q_i is not C_i then compute new cluster $IQ =$ New (C_i).
Step 7: Go to Step 3.
Step 8: Step go to Step 2.

Table 1 System specification

Nodes	Specifications		
	RAM (in GB)	Processors	Memory (in GB)
NameNode	2	1	30
DataNode 1	1	1	30
DataNode 2	1	1	30
JobTracker	2	1	30

4 Experimental Setup

4.1 System Specifications

The system deployed here to test the proposed work contains data nodes, name nodes, job trackers, and task trackers on virtual machines [26, 27]. The machines have different capacity and hardware configurations. The nodes are created in such a manner that each node has its own specialty and try to resemble to real-world environment (Table 1).

4.2 Load Distribution to High- and Slow-End Clusters

Once the data is sorted initially, it can be forwarded to the clusters for processing one by one. Here, comes the tricky part because ideally Hadoop does not contain same type of hardware architecture underneath [28]. There can be nodes that operate as high-end clusters with good processing speeds but there also can be slow-end clusters with lower speeds than their high-end counterparts. When a job or couple of jobs is fed into Hadoop ecosystem, they are sent to both types of processors depending on the availability. When a high-end cluster finishes job processing, the data waiting to be processed is sent to the high-end systems. This data movement introduces cost to the processing, data storage, and security as well.

4.3 Applying MapReduce

As demonstrated in previous sections, when initially the system requirements, job processing on clusters, and queries to implement are decided, it becomes whole lot easier to deal with the unstructured data formats [29, 30]. The cleansed, clustered, and diversified data items can now be sent to MapReduce to extract the key value pairs for further processing. In MapReduce, the applications can be used to classify unstructured documents into a set of meaningful terms which can provide value to the clusters. The MapReduce algorithm can help in retrieving important information from a pool of data which cannot be achieved manually.

Implementation of MapReduce:

Map:

Step 1: Prepare the Map() input

Input unstructured data file: (log_file.txt)

Step 2: Run the Map() code–Map() is run exactly once for

each $K1$ key value, generating output organized by key

values $K2$.

Function that divides data sets:

(log_file_intermediate)

Splitthefile(log_filetranstation.txt):(lft) + (lft2) + (lft3)….+

(lftn)

Step 3: Collect Output: (log_file_output; lfIdt, 1)

Reduce:

C is the Count of the individual terms.

Step 1: Input file which was output from map function:

(log_file_output; lfIdt, $[c]$)

Step 2: Run the Reduce() code Function to summarize the

intermediate terms and give a final value to all the

detected valid terms in the module: $S = \text{Sum}\ (c)$

Step 3: Provide output to sum up the file values.

(log_part, countId, S)

5 Results and Discussion

5.1 Results

The subsequent results have been established by directing the proposed methodology in the system. The heterogeneous Hadoop clusters were used to test the working of big data sets which emulated the *IO* throughput, average rate of execution, standard deviations and finally the test execution time.

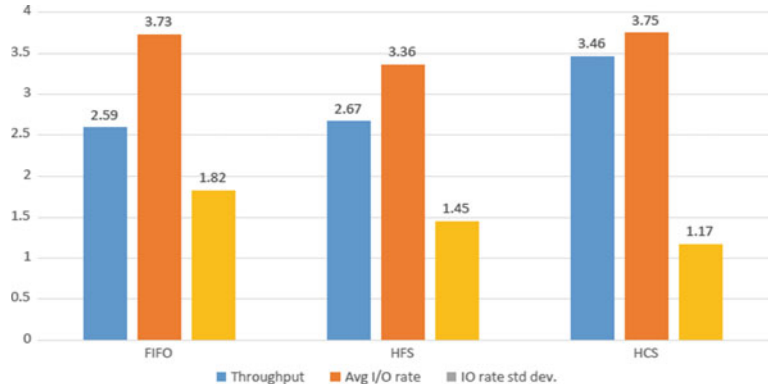


Fig. 3 Comparison of different Hadoop schedulers

5.1.1 Comparison of Productivity Among Hadoop Schedulers

Comparison among the most efficient schedulers used in Hadoop, i.e., FIFO, HFS, and HCS. The following graphs establish the point that schedulers can overtake other schedulers which are used in HDFS and internal mechanisms to share a file or job over the several default systems to execute and store them reliably (Fig. 3).

The figure shows a trade-off among various schedules which may lead to optimized performance of systems in HDFS for clustering big data. The response time caused by the data loads, which were actually the logs of an e-commerce Web site and the response time took by the schedulers in mapping and reducing them to store them on the DFS, manually extract as well as manage them. It also establishes the fact that surmounting amount of data as is general in big data sets; Hadoop can optimize the processing instead of data loading and unloading frequently to get better insights about data. This also proves the fact that once data is accepted by a node, the average time to consume and process it may vary depending upon the type of scheduler being used.

5.1.2 Comparative Study of FIFO/HCS/HFS

See Table 2; Figs. 4, 5 and 6.

5.2 Performance Evaluation

The clusters that were created for this work are working and derived results as formulated. The average *I*/*O* rate of data movement, standard deviations, and time

Table 2 Efficient job scheduling for MapReduce clusters [16]

Scheduling algorithm	Data size	Previous methodology	Proposed methodology
		Execution time	Execution time
FIFO	500 MB	4020	3958
	1 GB	7700	7156.356
	2 GB	9900	9165
	5 GB	14,500	11,130
	10 GB	Not consider	12,780
HFS	500 MB	3950	3925
	1 GB	7200	7065
	2 GB	9200	9085
	5 GB	13,500	10,980
	10 GB		12,400
HCS	500 MB	3900	3875
	1 GB	7000	6998
	2 GB	9000	8912
	5 GB	12,000	10,260
	10 GB		12,610

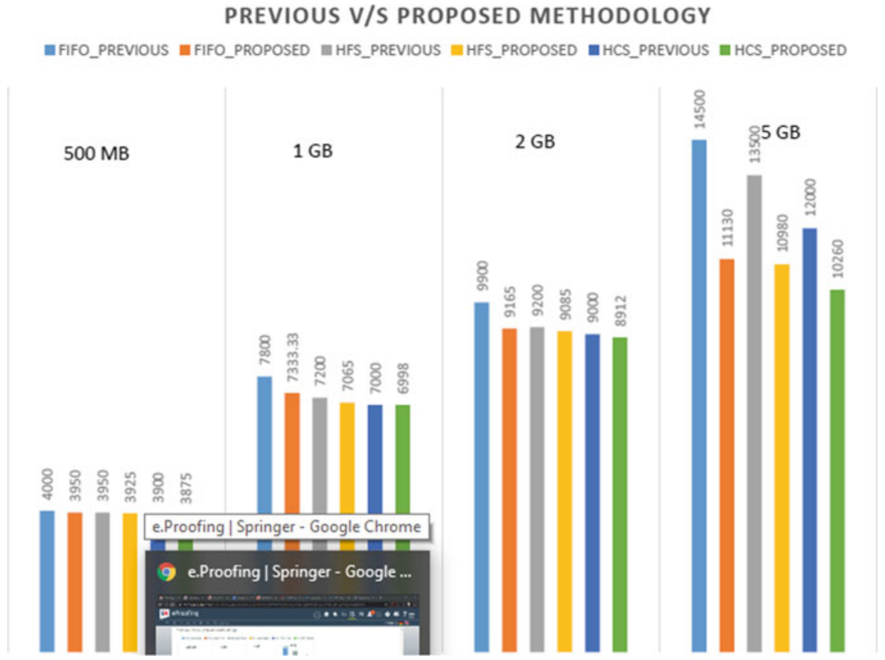
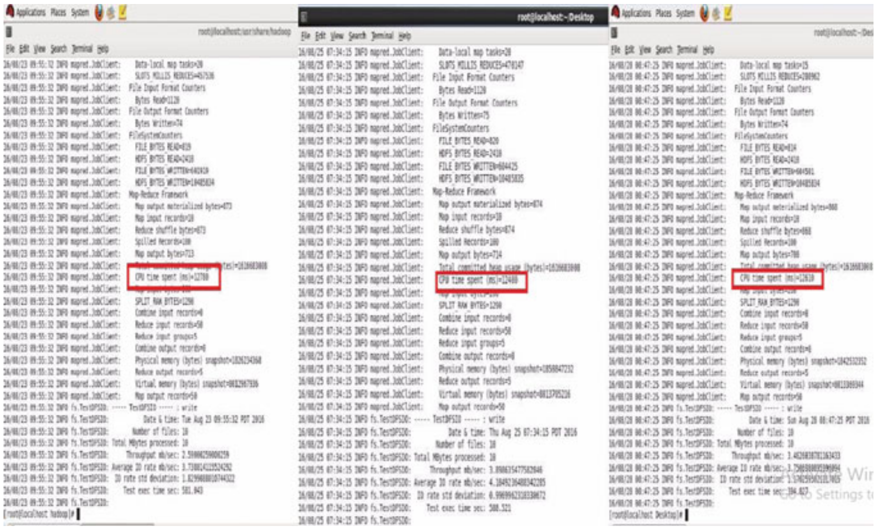


Fig. 4 Previous versus proposed methodology



```
root@localhost:/usr/share/hadoop
16/08/23 09:55:32 INFO mapred.JobClient:     Data-local map tasks=20
16/08/23 09:55:32 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=457536
16/08/23 09:55:32 INFO mapred.JobClient:   File Input Format Counters
16/08/23 09:55:32 INFO mapred.JobClient:     Bytes Read=1128
16/08/23 09:55:32 INFO mapred.JobClient:   File Output Format Counters
16/08/23 09:55:32 INFO mapred.JobClient:     Bytes Written=74
16/08/23 09:55:32 INFO mapred.JobClient:   FileSystemCounters
16/08/23 09:55:32 INFO mapred.JobClient:     FILE_BYTES_READ=819
16/08/23 09:55:32 INFO mapred.JobClient:     HDFS_BYTES_READ=2438
16/08/23 09:55:32 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=640819
16/08/23 09:55:32 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=10485834
16/08/23 09:55:32 INFO mapred.JobClient:   Map-Reduce Framework
16/08/23 09:55:32 INFO mapred.JobClient:     Map output materialized bytes=873
16/08/23 09:55:32 INFO mapred.JobClient:     Map input records=10
16/08/23 09:55:32 INFO mapred.JobClient:     Reduce shuffle bytes=873
16/08/23 09:55:32 INFO mapred.JobClient:     Spilled Records=100
16/08/23 09:55:32 INFO mapred.JobClient:     Map output bytes=713
16/08/23 09:55:32 INFO mapred.JobClient:     [illegible]tes)=1818083008
16/08/23 09:55:32 INFO mapred.JobClient:     CPU time spent (ms)=12700
16/08/23 09:55:32 INFO mapred.JobClient:     [illegible]
16/08/23 09:55:32 INFO mapred.JobClient:     SPLIT_RAW_BYTES=1290
16/08/23 09:55:32 INFO mapred.JobClient:     Combine input records=0
16/08/23 09:55:32 INFO mapred.JobClient:     Reduce input records=50
16/08/23 09:55:32 INFO mapred.JobClient:     Reduce input groups=5
16/08/23 09:55:32 INFO mapred.JobClient:     Combine output records=0
16/08/23 09:55:32 INFO mapred.JobClient:     Physical memory (bytes) snapshot=1826234368
16/08/23 09:55:32 INFO mapred.JobClient:     Reduce output records=5
16/08/23 09:55:32 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=8812967936
16/08/23 09:55:32 INFO mapred.JobClient:     Map output records=50
16/08/23 09:55:32 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
16/08/23 09:55:32 INFO fs.TestDFSIO:            Date & time: Tue Aug 23 09:55:32 PDT 2016
16/08/23 09:55:32 INFO fs.TestDFSIO:        Number of files: 10
16/08/23 09:55:32 INFO fs.TestDFSIO: Total MBytes processed: 10
16/08/23 09:55:32 INFO fs.TestDFSIO:      Throughput mb/sec: 2.5900025900259
16/08/23 09:55:32 INFO fs.TestDFSIO: Average IO rate mb/sec: 3.7380142253424292
16/08/23 09:55:32 INFO fs.TestDFSIO:  IO rate std deviation: 1.8299808010744322
16/08/23 09:55:32 INFO fs.TestDFSIO:     Test exec time sec: 581.043
16/08/23 09:55:32 INFO fs.TestDFSIO:
[root@localhost hadoop]#
```

```
root@localhost:~/Desktop
16/08/25 07:34:15 INFO mapred.JobClient:     Data-local map tasks=20
16/08/25 07:34:15 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=478147
16/08/25 07:34:15 INFO mapred.JobClient:   File Input Format Counters
16/08/25 07:34:15 INFO mapred.JobClient:     Bytes Read=1128
16/08/25 07:34:15 INFO mapred.JobClient:   File Output Format Counters
16/08/25 07:34:15 INFO mapred.JobClient:     Bytes Written=75
16/08/25 07:34:15 INFO mapred.JobClient:   FileSystemCounters
16/08/25 07:34:15 INFO mapred.JobClient:     FILE_BYTES_READ=820
16/08/25 07:34:15 INFO mapred.JobClient:     HDFS_BYTES_READ=2418
16/08/25 07:34:15 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=604425
16/08/25 07:34:15 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=10485835
16/08/25 07:34:15 INFO mapred.JobClient:   Map-Reduce Framework
16/08/25 07:34:15 INFO mapred.JobClient:     Map output materialized bytes=874
16/08/25 07:34:15 INFO mapred.JobClient:     Map input records=10
16/08/25 07:34:15 INFO mapred.JobClient:     Reduce shuffle bytes=874
16/08/25 07:34:15 INFO mapred.JobClient:     Spilled Records=100
16/08/25 07:34:15 INFO mapred.JobClient:     Map output bytes=714
16/08/25 07:34:15 INFO mapred.JobClient:     Total committed heap usage (bytes)=1818083008
16/08/25 07:34:15 INFO mapred.JobClient:     CPU time spent (ms)=12480
16/08/25 07:34:15 INFO mapred.JobClient:     [illegible]
16/08/25 07:34:15 INFO mapred.JobClient:     SPLIT_RAW_BYTES=1290
16/08/25 07:34:15 INFO mapred.JobClient:     Combine input records=0
16/08/25 07:34:15 INFO mapred.JobClient:     Reduce input records=50
16/08/25 07:34:15 INFO mapred.JobClient:     Reduce input groups=5
16/08/25 07:34:15 INFO mapred.JobClient:     Combine output records=0
16/08/25 07:34:15 INFO mapred.JobClient:     Physical memory (bytes) snapshot=1858847232
16/08/25 07:34:15 INFO mapred.JobClient:     Reduce output records=5
16/08/25 07:34:15 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=8813705216
16/08/25 07:34:15 INFO mapred.JobClient:     Map output records=50
16/08/25 07:34:15 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
16/08/25 07:34:15 INFO fs.TestDFSIO:            Date & time: Thu Aug 25 07:34:15 PDT 2016
16/08/25 07:34:15 INFO fs.TestDFSIO:        Number of files: 10
16/08/25 07:34:15 INFO fs.TestDFSIO: Total MBytes processed: 10
16/08/25 07:34:15 INFO fs.TestDFSIO:      Throughput mb/sec: 3.898635477582846
16/08/25 07:34:15 INFO fs.TestDFSIO: Average IO rate mb/sec: 4.184923648834285
16/08/25 07:34:15 INFO fs.TestDFSIO:  IO rate std deviation: 0.996996231833306 72
16/08/25 07:34:15 INFO fs.TestDFSIO:     Test exec time sec: 388.321
16/08/25 07:34:15 INFO fs.TestDFSIO:
```

```
root@localhost:~/Des
16/08/28 08:47:25 INFO mapred.JobClient:     Data-local map tasks=15
16/08/28 08:47:25 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=208962
16/08/28 08:47:25 INFO mapred.JobClient:   File Input Format Counters
16/08/28 08:47:25 INFO mapred.JobClient:     Bytes Read=1128
16/08/28 08:47:25 INFO mapred.JobClient:   File Output Format Counters
16/08/28 08:47:25 INFO mapred.JobClient:     Bytes Written=74
16/08/28 08:47:25 INFO mapred.JobClient:   FileSystemCounters
16/08/28 08:47:25 INFO mapred.JobClient:     FILE_BYTES_READ=814
16/08/28 08:47:25 INFO mapred.JobClient:     HDFS_BYTES_READ=2418
16/08/28 08:47:25 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=604501
16/08/28 08:47:25 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=10485834
16/08/28 08:47:25 INFO mapred.JobClient:   Map-Reduce Framework
16/08/28 08:47:25 INFO mapred.JobClient:     Map output materialized bytes=868
16/08/28 08:47:25 INFO mapred.JobClient:     Map input records=10
16/08/28 08:47:25 INFO mapred.JobClient:     Reduce shuffle bytes=868
16/08/28 08:47:25 INFO mapred.JobClient:     Spilled Records=100
16/08/28 08:47:25 INFO mapred.JobClient:     Map output bytes=708
16/08/28 08:47:25 INFO mapred.JobClient:     Total committed heap usage (bytes)=1818083008
16/08/28 08:47:25 INFO mapred.JobClient:     CPU time spent (ms)=12610
16/08/28 08:47:25 INFO mapred.JobClient:     [illegible]
16/08/28 08:47:25 INFO mapred.JobClient:     SPLIT_RAW_BYTES=1290
16/08/28 08:47:25 INFO mapred.JobClient:     Combine input records=0
16/08/28 08:47:25 INFO mapred.JobClient:     Reduce input records=50
16/08/28 08:47:25 INFO mapred.JobClient:     Reduce input groups=5
16/08/28 08:47:25 INFO mapred.JobClient:     Combine output records=0
16/08/28 08:47:25 INFO mapred.JobClient:     Physical memory (bytes) snapshot=1842532352
16/08/28 08:47:25 INFO mapred.JobClient:     Reduce output records=5
16/08/28 08:47:25 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=8811089944
16/08/28 08:47:25 INFO mapred.JobClient:     Map output records=50
16/08/28 08:47:25 INFO fs.TestDFSIO: ----- TestDFSIO ----- : write
16/08/28 08:47:25 INFO fs.TestDFSIO:            Date & time: Sun Aug 28 08:47:25 PDT 2016
16/08/28 08:47:25 INFO fs.TestDFSIO:        Number of files: 10
16/08/28 08:47:25 INFO fs.TestDFSIO: Total MBytes processed: 10
16/08/28 08:47:25 INFO fs.TestDFSIO:      Throughput mb/sec: 3.4026838781163433
16/08/28 08:47:25 INFO fs.TestDFSIO: Average IO rate mb/sec: 3.7500080053906964
16/08/28 08:47:25 INFO fs.TestDFSIO:  IO rate std deviation: 1.5740259562101765
16/08/28 08:47:25 INFO fs.TestDFSIO:     Test exec time sec: 394.827
16/08/28 08:47:25 INFO fs.TestDFSIO:
[root@localhost Desktop]#
```

Fig. 5 Performance analysis of FIFO/HCS/HFS with query improvisation

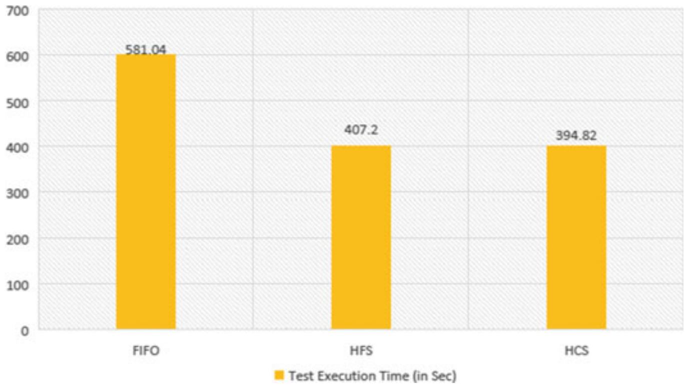


Fig. 6 Test execution time by schedulers

taken to execute the loaded files prove that the formulated method is working as expected.

5.3 Schedulers at Work

5.3.1 First in First Out (FIFO)

The scheduler used here is FIFO which is used to schedule the resources of big data sets in the heterogeneous clusters and concurrently running TestDFSIO as benchmarking.

5.3.2 Hadoop Fair Scheduler

The test results are the resultant of the working of fair scheduler algorithm on the big data sets for heterogeneous clusters in Hadoop environment.

6 Conclusion

The conducted experimentation also establishes the fact that if a practice is described to handle different usage case scenarios, one could complete lessen the amount being spent on computing and may gain on counting upon dispersed programs for rapidly executions.

The major points of thought that were resulted as conduction of this work are as following:

Calculating question dependencies: This really may be lowered by keeping an index or question addiction desk which cannot only cut back time but affect that the total calculating in an excellent phrase. The processor could be utilized to method an increasing number of data within the dispersed nodes of Hadoop. The heterogeneous clusters used in the Hadoop can reap by simply in carrying the relevant and shedding the insignificant jobs to method.
Job scheduling mechanics: The default scheduler of Hadoop and also MapReduce could be your FIFO scheduler (First in First Out) which seems important as so when Hadoop project arrives. However, at a dynamic setting of big data if the payable size of data is to be processed, using in everything that comes could become unfruitful and squander of reference, because while in the end to make money from these kinds of big chunks of data, then it must be sorted at the past. Therefore, scheduling the tasks that process this info therefore the clusters using high-performance can manage big numbers sets and clusters with slow-ends can assist in additional processing systems. The study experiments and work conducted under this work have emulated quite surprising results, many of them function as selection of schedulers to program tasks, placement of data in the similarity matrix, clustering before scheduling inquiries and moreover, iterative, mapping along with reducing and binding the inner dependencies collectively to steer clear of query stalling and execution times.

7 Limitation

The big data set has many probing worries for computing and data storage. Since the major part of big data analytics is related to dynamic nature of data, data is deleted, manipulated, and retrieved frequently. This ad hoc-based processing of data includes streaming data in and out of the storage systems as the requirement but this also introduces a large chunk of processing.

References

1. Liu, Z. (2015). Efficient storage design and query scheduling for improving big data retrieval and analytics, Dissertation, Auburn University, Alabama.
2. Zongben, X., & Shi, Y. (2015). Exploring big data analysis: Fundamental scientific problems. *Springer Annals of Data Science, 2*(4), 363–372.
3. Tinetti, F. G., Real, I., Jaramillo, R., & Barry, D. (2015). Hadoop scalability and performance testing in heterogeneous clusters. In *The proceedings of the 2015 international conference on parallel and distributed processing techniques and applications (PDPTA-2015), Part of WORLDCOMP'15* (pp. 441–446).
4. Wan, J., Yu, W., & Xu, X. (2009). *Design and implement of distributed document clustering based on MapReduce*, ISBN 978-952-5726-07-7, 2009.
5. Kamtekar, K., & Jain, R.. (2015). *Performance modeling of big data* (pp. 1–9). Washington University in St. Louis.
6. Das, T. K., & Mohan Kumar, P. (2013). Big data analytics: A framework for unstructured data analysis. *International Journal of Engineering and Technology (IJET)*, *5*(1), 153–156. ISSN: 0975-4024.
7. Liu, F. H., Liou, Y. R., Lo, H. F., Chang, K. C., & Lee, W. T. (2014). The comprehensive performance rating for hadoop clusters on cloud computing platform. *International Journal of Information and Electronics Engineering, 4*(6), 480–484.
8. Rong, Z., & De Knijf, J. (2013). Direct out-of-memory distributed parallel frequent pattern mining, ACM, BigMine'13. In *Proceedings of the 2nd international workshop on big data, streams and heterogeneous source mining: Algorithms, systems, programming models and applications* (pp. 55–62). ISBN: 978-1-4503-2324-6, https://doi.org/10.1145/2501221. 2501229.
9. Li, B., & Guoyong, Y. (2012). Improvement of TF-IDF algorithm based on Hadoop framework. In *The 2nd international conference on computer application and system modeling* (pp. 0391–0393), Paris, France: Atlantis Press.
10. Kamtekar, K. (2015). *Performance modeling of big data*, May 2015.
11. Jagtap, A. (2015). Categorization of the documents using K-Means and MapReduce. *International Journal of Innovative Research in Science, Engineering and Technology*, ISSN: 2319-8753, 2015.
12. Das, T. K., & Kumar, P. M. (2013). BIG data analytics: A framework for unstructured data analysis. *International Journal of Engineering and Technology (IJET)*, *5*(1), 153–156. ISSN: 0975-4024.
13. Novacescu, F. (2013). Big data in high performance scientific computing. In *International Journal of Analele Universităţii "Eftimie Murgu* (vol. 1, pp. 207–216). "Eftimie Murgu" University of Resita, ANUL XX, NR.
14. Rao, B. T., Sridevi, N. V., Reddy, V. K., & Reddy, L. S. S. (2011). Performance issues of heterogeneous Hadoop clusters in cloud computing. *Global Journal of Computer Science and Technology*, *XI*(VIII).

15. Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A., & Qin, X. (2010). Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In *Proceedings of the 19th international heterogeneity in computing workshop* (pp. 1–9), Atlanta, Georgia.
16. Liu, J., et al. (2015). An efficient job scheduling for MapReduce clusters. *International Journal of Future Generation Communication and Networking, 8*(2), 391–398.
17. Fahad, A., Alshatri, N., Tari, Z., Alamri, A., Khalil, I., Zomaya, A. Y., Foufou, S., & Bouras, A. (2014). A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE Transactions on Emerging Topics in Computing*, *2*(3), 267–279. Digital Object Identifier https://doi.org/10.1109/tetc.2014.2330519.
18. MonaP. EMC Corporation (2014). Virtualizing Hadoop in large-scale infrastructures.
19. Aggarwal, C., & Han, J. (2014). An introduction to frequent pattern mining. In *Frequent pattern mining*, Springer. ISBN 978-3-319-07820-5 (Print) 978-3-319-07821-2 (Online), https://doi.org/10.1007/978-3-319-07821-2.
20. Victor, G. S., Antonia, P., & Spyros, S. (2014). CSMR: A scalable algorithm for text clustering with cosine similarity and MapReduce. In *IFIP international conference on artificial intelligence applications and innovations*, *AIAI 2014: Artificial intelligence applications and innovations* (pp. 211–220), AICT 437.
21. Novacescu, F. (2013). Big data in high performance scientific computing. EFTIMIE MURGU RESITA, ANUL XX, NR. 1, (pp 207–216). ISSN 1453–7397.
22. Xue, J., Li, J., & Gong, Y. (2013). *Restructuring of deep neural network acoustic models with singular value decomposition* (pp. 2365–2369), ISCA, INTERSPEECH.
23. Akdere, M., Cetintemel, U., Riondato, M., Upfal, E., & Zdonik, S. B. (2012). Learning based query performance modeling and prediction. In *IEEE 28th international conference on data engineering* (pp. 390–401).
24. Thirumala Rao, B., Sridevi, N. V., Krishna Reddy, V., & Reddy, L. S. S. (2011). Performance issues of heterogeneous Hadoop clusters in cloud computing. *Global Journal of Computer Science and Technology*, *XI*(VIII).
25. Kumar, A., Goyal, D., Dadheech, P. (2018). A novel framework for performance optimization of routing protocol in VANET network. *Journal of Advanced Research in Dynamical & Control Systems*, *10*(02), 2110–2121. ISSN: 1943-023X.
26. Dadheech, P., Goyal, D., Srivastava, S., & Kumar, A. (2018). A scalable data processing using Hadoop & MapReduce for big data. *Journal of Advanced Research in Dynamical & Control Systems*, *10*, (02), 2099–2109. ISSN: 1943-023X.
27. Dadheech, P., Goyal, D., Srivastava, S., & Choudhary, C. M. (2018). An efficient approach for big data processing using spatial boolean queries. *Journal of Statistics and Management Systems (JSMS), 21*(4), 583–591.
28. Dadheech, P., Kumar, A., Choudhary, C., Beniwal, M. K., Dogiwal, S. R., & Agarwal, B. (2019). An enhanced 4-way technique using cookies for robust authentication process in wireless network. *Journal of Statistics and Management Systems, 22*(4), 773–782. https://doi.org/10.1080/09720510.2019.1609557.
29. Kumar, A., Dadheech, P., Singh, V., Raja, L., & Poonia, R. C. (2019). An enhanced quantum key distribution protocol for security authentication. *Journal of Discrete Mathematical Sciences and Cryptography, 22*(4), 499–507. https://doi.org/10.1080/09720529.2019.1637154.
30. Kumar, A., Dadheech, P., Singh, V., Poonia, R. C., & Raja, L. (2019). An improved quantum key distribution protocol for verification. *Journal of Discrete Mathematical Sciences and Cryptography, 22*(4), 491–498. https://doi.org/10.1080/09720529.2019.1637153.