BIND: An Indexing Strategy for Big Data Processing

Adib Habbal[1], Fatima Binta Adamu[1], Suhaidi Hassan[1], R. Les Cottrell[2], Bebo White[2], Mustafa Kaiiali[3], Ahmad Samer Wazan[4]

InterNetworks Research Laboratory, School of Computing, Universiti Utara Malaysia, 06010 UUM, Sintok, Kedah, Malaysia.[1]

SLAC National Accelerator Center, CA 94025, USA[2] | Queen's University, Belfast, UK [3] | Paul Sabatier University, Toulouse, France[4]

***Abstract*—With the huge amount of data continuously accumulated and shared by individuals and organizations, it has become necessary to meet the emerging processing and information retrieval requirements associated with these large volumes of data. This could be achieved by indexing the data sets and reducing heavy computational overhead accustomed to most current indexing strategies during processing of very large amounts of data sets. This study proposes a novel Indexing strategy called Big Data INDexing Strategy (BIND), using a concept of high performance parallel computing. BIND supports parallel distribution of data and performs processing in a MapReduce fashion. To develop the BIND strategy, Ian Foster's task-scheduling concept for parallel processing is applied. The proposed indexing strategy was first tested on a 2-node cluster environment where varying sizes of datasets were used to note if the performance improves or declines as the size of the data increases. Subsequently, it was tested on a 3-node cluster to note the performance when the number of computation resources are increased. The results demonstrates that BIND minimizes the processing and query time as compared to the current strategy. The findings have significant implication in efficiently managing Big Data and facilitating data processing and information retrieval for users and organizations that manage Big Data.**

***Index Terms*—Big Data Analytics; Indexing Strategy; MapReduce; Information Retrieval**

I. INTRODUCTION

Big Data is a term used to describe very large data sets that are of different forms or structure (complex), generated at a very high speed, and cannot be managed by traditional database management systems. This definition explains the three (3) main characteristics associated with Big Data: volume, variety and velocity (3Vs), and the value that can be extracted from it is seen as a fourth characteristic (4V's) [1]. Big Data is sourced from many end devices such as Personal Computers (PC), smart phones, sensors, Radio Frequency Identification (RFID) devices, monitoring devices, etc. Also, online applications such as social networks and applications that involve video streaming are great sources that generate Big Data.

According to Zhou et al. in [2], the total size of data generated will surpass 7.9 Zettabytes (ZB) by the end of 2015, and predicted to reach 35ZB in 2020. Significant interest have been taken in Big Data lately – this is due to insights or great value that can be acquired from huge amounts of data sets, which can be useful in decision making for businesses or organizations. Cisco related that organizations such as Facebook, Yahoo, Google, Twitter, etc., accumulate Big Data and retrieve information for analysis and decision making [3]. The Internet of Things (IoT), monitoring tools, and lots of other devices used by organizations for business operations, also accumulates data to enable analysis, information retrieval, and decision making. These operations are becoming difficult to perform because data keeps increasing in volume as time passes by. Current Relational Database Management systems (RDBMS) were built with a scale in mind and for structured data. Hence, they cannot handle processing and information retrieval on very large amount of unstructured data (Big Data). Yet, International Data Corporation (IDC) predicts that the global Big Data will multiply 50 times in the next decade [4], which suggests the continues growth and accumulation of unstructured data. This should be met with efficient processing strategies capable of handling such huge amounts of unstructured data.

Numerous indexing strategies have been proposed [5] (from existing indexing strategies) based on the use of sophisticated tools that can handle the growing amount of data as a solution to the problem. First, indexing strategies can be said to be a non-polynomial process as each relates to the problem it solves [6]. Different Indexing approaches are applied in different domains and on different data types. A popular means of processing Big Data is by indexing using the MapReduce application [7]. MapReduce sorts data sets according to key-value pairs and stores them in an indexed manner. Indexing of data sets facilitates data storage and information retrieval.

This study proposes a solution that expedites Big Data manageability by applying Ian Foster's task-scheduling concept for parallel processing to the MapReduce framework. It also highlights some popular indexing strategies used for Big Data management and exposes the potentials of each as used in previous studies (Section II). Section III discusses and explains the concept of the proposed BIND strategy. Section IV provides an algorithm for the BIND strategy. Experimental analysis and performance evaluation are presented in Section V, and then Section VI concludes the paper.

II. RELATED WORK

The growth of data and accumulation of complex data collections has become a challenge in Big Data management, specifically, processing and information retrieval [8]. A solution to this is in designing indexing strategies to ease in the

978-1-5090-1134-6/17/$31.00 ©2017 IEEE

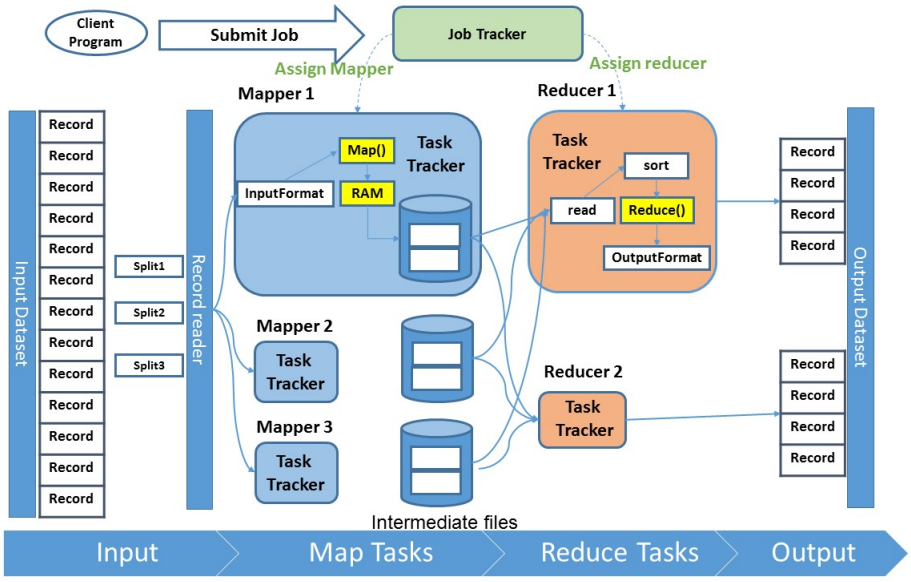

Figure 1. MapReduce Phases

retrieval of data items. In general, indexes or indices are a list of tags, names, subjects, etc. of a group of items which references where the items occur. With this, Big Data indexes can be said to be a list of tags, names, subjects, etc. of a dataset which references where data can be found. An indexing strategy is the design of an access method to a searched item, or simply put, an index. It constitutes all the steps that leads to the creation of indices.

Complex data are collected with metadata that describes their contents. Such datasets can be queried using the metadata of the contents. Instead of searching the whole database (which can be time consuming), a more efficient approach is to search the appropriate group(s) relating to the query. This results in a decrease in information retrieval time, since the search process considers only the content of a specific group(s). Hence, studies on the design of indexing strategies to ease Big Data managebility has become a necessity. Several works have proposed indexing strategies by improving or using existing indexing strategies such as the B-tree, the R-tree, the hash indexing, the inverted indexing strategy, and so on, using the MapReduce framework. Utilizing the MapReduce framework to improve Big Data indexing means improving one or more of the four (4) phases that results in the creation of indices. The phases as illustrated in Figure 1 includes:

Files Input: Files are submitted to the Hadoop Distributed File System (HDFS), which serves as a mechanism for receiving inputs for MapReduce applications. Hadoop MapReduce supports numerous file formats; Sequence- FileInputFormat, NLineInputFormat, FixedLengthInputFormat, TextInputFormat, etc. The InputFormat is used to define how data will be read into the mapper instances. A user can define an InputFormat implementation to format the data or input to be programmed based on personal or organizational requirements. In Inverted Indexing for instance, lines of text files are read by the default TextInputFormat. In the current indexing strategies, the byte offset per line read, is the key produced for each record. The content of the line is the value. The InputFormat also divides the input file or data into fragments (called splits) that will be fed into individual mappers as tasks. The InputSplit is performed before the start of the MapReduce job. Hence in a nutshell, InputFormat performs two tasks:

1) dividing the data sets into splits and allocating to mappers as jobs.
2) sub-dividing the splits (jobs) into records, which are fed one after the other to the mappers as tasks. This is achieved through the RecordReader class.

Mapper: The mapper takes one line at a time, and produces key-value pairs by splitting the lines and sorting the texts. The key is the unique representation of the sorted item(s) in the data sets, and the value is the files in which the keys occur, or the group of data sets in which the searched item can be found.

Reducer: Reducer acquires the keys and list of values from each mapper, combining the keys that conform to each other and their corresponding values. A keyValue pair is emitted as output upon search.

The phases or activities of the proposed BIND strategy is similar to that of the current strategy. However, the current strategies proposed in previous studies concentrate more on the mapper and reducer phase with less consideration to the InputFormat phase. For instance, the study by Zhang *et al* in [9], used Latent Semantic Indexing (LSI) to design an indexing strategies by eliciting the conceptual (semantic) content of data sets (in this case, social media chats), and establishing relationships between terms with similar contexts. Their system categorized or indexed comments made by the audience into audience favors, audience expectations, and the shortcomings of the season (by extracting the meaning of each comment), in the MapReduce phase. This makes it easier for the director to make decisions towards the improvement of the next season, and so on.

Tang *et al.*[10] also used the LSI in their design; like

Zhang. They used the MapReduce to classify the data based on the acceleration information, consisting of the position information and the pure force. Hence, the prediction of the next series (sequence) of motions was dependent on the position informant and the pure force. Again, work on the InputFormat phase was not reported.

Another study by Jaluta *et al.* [11] used the B+tree for their design, and Gollub *et al.* [12] used the inverted indexing strategy to index library queries as keys, and set of library documents as values. Both studies concentrated on the mapper and reducer phase. In the design of the BIND strategy, the study sheds more light on the second phase, which is the InputFormat. Specifically, it concentrates on how the splits are sub-divided before being fed to the mappers (the RecordReader). This is explained in the following section.

III. BIG DATA INDEXING (BIND) STRATEGY OVERVIEW

The proposed strategy called Big Data INDexing (BIND) strategy is designed on the MapReduce framework. MapReduce operates on distributed computers and in a parallel fashion. Some researchers have developed techniques, tools, and algorithms for high performance distributed and parallel computing. One of such known concepts is Ian Foster's Task-Scheduling concept [13]. It is a known fact that Scheduling concepts have been applied in solving job/task assignment related problems.

Hence, Ian Foster's Task-Scheduling concept was applied in the design of the BIND strategy. The concept submitted that in parallel processing, processes that have completed task execution should return to the master for further allocation of task without waiting for other processes to complete task execution. This could be applied to processing of Big Data using the MapReduce framework. As mentioned earlier, Big Data can be processed or indexed in numerous ways, depending on the type of queries to be performed on it. Hence, the design of indexing strategies depends on the type of query that will be performed on the data sets which conform to the sorting of Key-Value pairs.

The design of indexing strategies also depends on four (4) phases which have been highlighted in Section II above. Previous studies have proposed designs that improve one or more of the phases listed. Although, most of the related works concentrate more on the mapper and reducer phase which is where the actual KeyValue pair sorting occurs. With this, less attention is paid to the input phase, which consists of the InputFormat and the RecordReader. The current InputFomat's RecordReader divides the data sets into splits and assigns one record to each individual mapper as a task. The RecordReader assigns more tasks to the mappers simultaneously, after the previous task has been completed. This is done until the data sets have all been allocated to the mappers for processing. The reducer finally collects these keyvalue pairs from all the mappers and combines them as output. This method successfully indexes the data sets, resulting in easy information retrieval. However, computational overhead which is accustomed to larger data sets can be avoided if more attention is paid to the InputFormat phase.

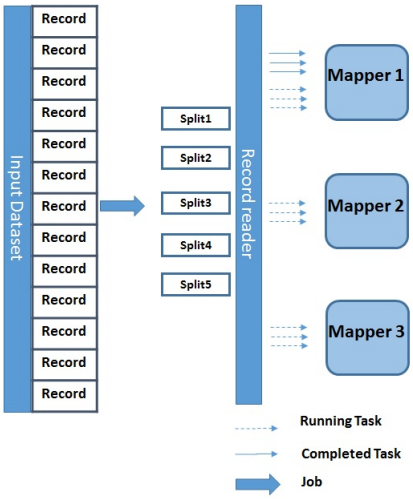

Figure 2. RecordReader for the BIND Strategy

Ian Foster's Task-Scheduling concept could be applied to the way the records are assigned to mappers, hence, reducing processing or task execution time (index creation time), and consequently reducing the time it takes to retrieve information from the data sets. Based on Ian Foster's task-scheduling concept, the research employed the traditional way of feeding data into the mapper (InputFormat and RecordReader) to implement the proposed Indexing Strategy. The study concentrated on customizing the RecordReader, thereby allowing the BIND Strategy to be implemented on records that have been fed into the mapper as tasks. Mappers can receive as many records as possible (as a task) depending on the amount of computational resources available and other settings.

In implementing the BIND strategy, the mapper is set to receive three lines or records from the RecordReader as a task, instead of one line at a time as is accustomed in the traditional or default TextInputFormat. This is performed during the design of the RecordReader. Each mapper with a completed task, is assigned another task (sequentially) without waiting for the other mappers as illustrated in Figure 2. While the JobTracker (or ResourceManager) serves as the master from Ian Foster's concept, the TaskTrackers (a component of the mapper), are responsible for conveying of tasks between the RecordReader and the mapper.

The algorithm for the BIND strategy is presented in Algorithm 1. It was implemented in all phases, with more emphasis on the InputFormat (RecordReader) phase. The number of input splits is dependent on the amount of available resources (CPU cores and RAM). Logical and mathematical comparison are tested against the data as they arrive. When the condition is satisfied, data is indexed and stored. Efficient management of splits before they get to the mapper phase, could mitigate computational overhead. It is expected of the proposed design to improve processing of Big Data by reducing the processing time of data sets, thereby mitigating computational overhead.

Algorithm 1 BIND Algorithm

```
if {
   x ≠0
   split x ;
}
where x = InputFiles
let
   y = split
   m = mapper
   n = lines to process
   t = task
   t = 0,1,2,_ _ _,n
   y = 0,1,2,_ _ _,n
where ∀m ∃y
if {
   y ≠0
   assign t;
}
∀m
if {
   t = 0
   t < n
   t ++ ;
}
let
   k = key or item to be indexed
   v= value associated with key
   l(v) = list of values
   i = 1,2,3,_ _ _,n
   j = 1,2,3,_ _ _,n
   where j > i
do {
   map (k_i, v_j)
   sort (k_i, l(v_j) )
   combine (k_i, l(v_j) )
   reduce (k_i, v_j)
   index k_i
   store;
}
```

IV. PERFORMANCE EVALUATION

In this section, we evaluate the BIND strategy and compare it with the current strategy using Hadoop. The performance of the BIND strategy will be evaluated using:

1) the processing time (index creation time) for varying sizes of data sets, to note if the performance improves or declines as the size of the data increases,
2) the processing time for varying number of computational nodes, to note the performance when the number of computational resources are increased, and
3) the query time for varying sizes of data sets, to check wether the BIND strategy facilitates information retrieval despite the growth of data.

A. Experimental Setup

The experiment was conducted on a 2-node cluster, and then on a 3-node cluster. Each node has four (4) processor cores and 4GB of RAM. All nodes run on the Ubuntu 14.04 operating system, Hadoop 2.6.0, and Java version 1.7.0_79 64bit. Hive 1.2.0 was utilized for data query in the study. One node served as master and also as a slave, and the others as slaves. HDFS is used as the storage mechanism. The replication factor is three (3) in HDFS. The block size is left as the default 64MB. Each experiment is repeated five (5) times and the average result reported.

PingER data [14], [15] was used to verify the efficiency of the BIND strategy. Approximately, every 30 minutes the PingER monitoring host goes through a list of target hosts. For each target host one ping is sent with a 100Byte payload. This is discarded since it is often delayed due to the priming of ARP and DNS caches. This is followed by sending up to 30 100Byte payload pings with one second intervals between the pings, until 10 responses are received or there is a timeout of 30 seconds. This in turn is followed by sending up to 30 1000Byte payload pings with one second intervals between the pings, until 10 responses are received or there is a timeout of 30 seconds.

B. Data Set Analysis

We utilized the PingER analyzed and aggregated data where each record contains the monitor and target and the relevant metric for each hour of the day. The initial state of the data used in the study was in a compressed form stored in flat file formats. Also the size of the files were in bytes and kilobytes (too small to be supported by Hadoop). Hence, needed to be uncompressed and merged (preprocessed) in order to be supported by Hadoop.

The preprocessing of the files (data set) was performed using a shell script. The data sets were grouped into four sets with varying sizes namely D1, D2, D3, D4, and populated with data sizes of 325,650 tuples, 651,950 tuples, 982.150 tuple, and 1,302,600 tuples respectively. Another data set D, was populated with 130,000 tuples and was used as the sample data to test for configuration and program errors. Though, the sample data is far from the size that Big Data is considered to be, it is large enough for potential improvements to be noticed.

Examining the PingER historical log data, the fields are delimited by spaces with the first two fields representing the source address and the destination adress respectively. The values (or dots in some cases) represent the metric value (e.g. Round Trip Time in msecs), and the last two fields also represents the source and destination name, further detail about Pinger log data attributes can be found at [14], [15]. Using the inverted indexing strategy, the pinged addresses were sorted as key, and a wordcount was conducted to verify the performance of the BIND strategy which was compared with the current strategy.

C. Results and discussion

Tables I and II present the average result computed on the index creation time (processing time) after running the experiment several times on a 2-node and a 3-node cluster respectively. The results were obtained using the same number of splits in all cases.

Table I
RESULTS OBTAINED USING 2-NODE CLUSTER

Data Set	Number of Splits	Processing Time (s)	
		Current Strategy	Proposed Strategy
D1	543	2516.6	2379.52
D2	835	3918.2	3724.14
D3	1013	4712.4	4112.8
D4	1181	5860.32	5146.3

Table II
RESULTS OBTAINED USING 3-NODE CLUSTER

Data Set	Number of Splits	Processing Time (s)	
		Current Strategy	Proposed Strategy
D1	543	1852.2	1797.6
D2	835	3128.5	2605.2
D3	1013	3703.96	3333.6
D4	1181	4344.4	3850.37

From Table I, the current strategy returns an average processing time of 2516.6 seconds for data set D1, which is greater than the processing time obtained from implementing the BIND strategy on the same data set D1 (which is 2379.52). This is observed for the other data sets as well, where a processing time of 3918.2 seconds for the current strategy is obtained against a processing time of 3724.14 seconds for the BIND strategy (on the data set D2). A decrease in the average processing time obtained from the proposed strategy as compared to the current strategy implies that the proposed strategy performs better than the current strategy in terms the index creation time (processing time).

Figure 3 illustrates that using a 2-node cluster (small scale), the time taken to process the data sets using the BIND strategy is less than that of the the current strategy. This proves that an increase in performance is obtained from the proposed strategy as compared to the current strategy in terms of the processing time. From Figure 3, it is also observed that the BIND strategy maintains a better performance than the current strategy as the data size increases. This proves the efficiency of BIND for processing Big Data.

In Table II, the processing time obtained from implementing the current strategy on the data set D1 using a 3-node cluster is 1852.2 seconds. Under the same condition, the BIND strategy returns an average processing time of 1797.6 seconds on the data set D1. For the data set D2, the current strategy returns an average processing time of 3128.5 seconds which is less than 2605.2 seconds, obtained from the BIND strategy. The decrease in processing time obtained from the proposed strategy implies that the proposed strategy still maintains a better performance than the current strategy with an increase in the number of nodes.

Figure 4 shows a comparison of the processing time between the current strategy and the proposed strategy using a 3-node cluster. It is observed that the BIND strategy produces lower values as compared to the current strategy. This explains that the performance of the BIND strategy does not decline with increase in nodes. It also shows that the proposed strategy still performes better than the current strategy when the number of nodes are increased.

Figure 5 presents the results obtained from querying the

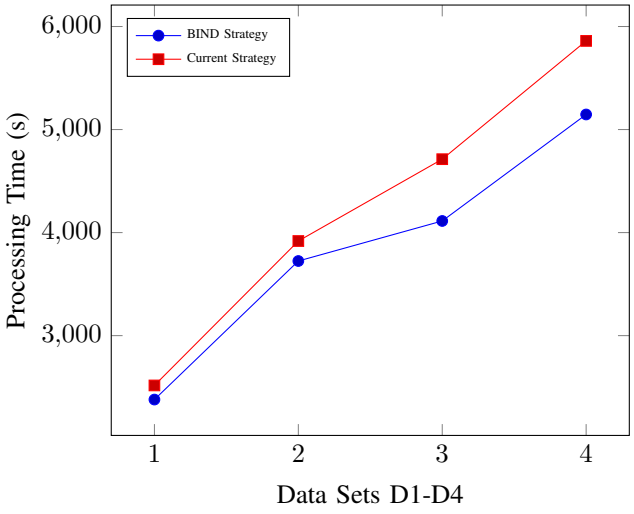

Figure 3. Processing time using 2-node cluster

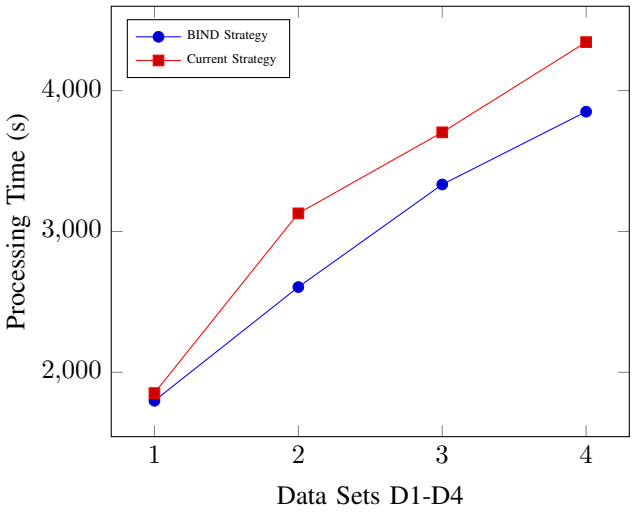

Figure 4. Processing time using 3-node cluster

data sets. As the experiment was conducted on an offline mode, querying the data set as they are processed will bear no significance when comparing the retrieval time between the BIND strategy and the current strategy; unlike on an online mode where it is a known fact that faster index creation time leads to faster availability of data for retrieval. Moreover, the aim of testing for the query time is to verify that the BIND strategy facilitates information retrieval and maintains a consistent performance despite the growth of data. Hence, the results obtained from querying the indexed data sets is compared with the results obtained from querying the unindexed data sets. For future study, it is recommended that the experiment be conducted on an online mode to note the performance of the proposed strategy in terms of information retrieval as compared to the current strategy.

From Figure 5, the average time it took to retrieve information from the unindexed data set D1 is 34.474 seconds and 10.722 seconds from the indexed data set D1 (using the

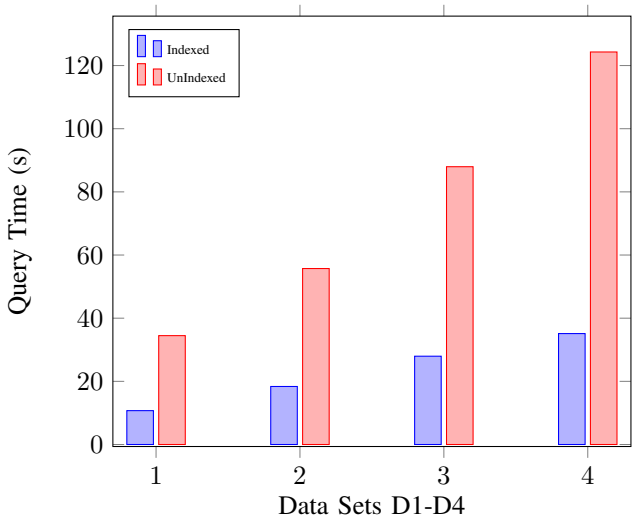

Figure 5. Query Time

BIND strategy). For the data set D2, it took 55.724 seconds to retrieve information from the unindexed data set, and 18.374 seconds from the indexed data set. A significant decrease is observed from the values obtained when the indexed data sets are queried. This indicates that the proposed strategy facilitates information retrieval when implemented on the data sets. Moreover, a consistent decrease in the values obtained from the BIND strategy (the indexed data sets), signifies that the BIND strategy maintains a consistent performance in terms of information retrieval when the size of the data keeps increasing.

In general, it is observed that although the values obtained for the processing time increases with an increase in the size of data sets, the proposed strategy still performs better than the current strategy in all cases. A performance gain was obtained when the proposed strategy was implemented on a 2-node cluster as compared to when the current strategy was applied on a 2-node cluster using the same sizes of data sets. Also, the same performance was obtained when the experiments were conducted on a 3-node cluster. Interestingly, the performance gap between BIND and the current strategy grows as the data set volume increases. These favorable results indicate that BIND reduces information retrieval time.

In addition, a significant decrease in the processing time was observed when a 3-node cluster was used as compared to the processing time obtained from using a 2-node cluster. This is due to the additional computational resources (processing and memory/storage resources) used. The BIND strategy still produced lesser values as compared to the current strategy when the number of nodes were increased. This proves that the performance of the BIND strategy does not decline with increase in size of the data set, as well as nodes (as noted in all cases). Hence, the performance of the proposed strategy improves with increase in nodes, and therefore, could be implemented (and would perform better) on thousands of clusters as could be found in large organizations. In a nutshell, the BIND strategy exceeds the currents strategy in all cases tested.

V. CONCLUSION

The objective of the study is to propose the design of a Big Data indexing strategy called BIND strategy, for processing of Big Data using the MapReduce framework. The proposed strategy facilitates processing of Big data, as well as storage and information retrieval. The strategy was designed and implemented using the Java programming language. The program was executed a number of times on varying data sets (PingER historical log data), as well as varying amount of nodes to compare and analyze the results. The overall results obtained, indicates that the proposed strategy produces lower values than the current strategy. It is noted that the proposed strategy still performs well with increase in nodes, and query is facilitated when the proposed strategy is implemented on data sets. The time taken to process a given amount of data using the proposed strategy has been minimized. This is supported by the results obtained from the study. Users and organizations that manage Big Data will find the BIND strategy helpful in the processing of Big data. For future study, it is recommended that the experiment be conducted on an online mode to note the performance of the BIND strategy in terms of information retrieval as compared to the current strategy.

REFERENCES

[1] C. Liu, R. Ranjan, X. Zhang, C. Yang, D. Georgakopoulos, and J. Chen, "Public auditing for big data storage in cloud computing -," in *A Survey. Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on, (pp. 1128-1135).*, 2013, Dec.

[2] W. Zhou, C. Yuan, R. Gu, and Y. Huang, "Large scale nearest neighbors search based on neighborhood graph," in *Advanced Cloud and Big Data (CBD), 2013 International Conference on*, pp. 181–186, Dec 2013.

[3] H. Nakada, H. Ogawa, and T. Kudoh, "Stream processing with bigdata: Sss-mapreduce," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pp. 618–621, Dec 2012.

[4] T. Chardonnens, "Big data analytics on high velocity streams," Master's thesis, University of Fribourg (Switzerland), June 2013.

[5] A. Gani, A. Siddiqa, S. Shamshirband, and F. Hanum, "A survey on indexing techniques for big data: taxonomy and performance evaluation," *Knowledge and Information Systems*, pp. 1–44, 2015.

[6] F. B. Adamu, A. Habbal, S. Hassan, R. Les Cottrell, B. White, and I. Abdullahi, "A survey on big data indexing strategies," Internet Applications, Protocols and Services (NETAPPS), 2015.

[7] A. Patel, M. Birla, and U. Nair, "Addressing big data problem using hadoop and map reduce," in *Engineering (NUiCONE), 2012 Nirma University International Conference on*, pp. 1–5, Dec 2012.

[8] K. Fasolin, R. Fileto, M. Krugery, D. Kaster, M. Ferreira, R. Cordeiro, A. Traina, and C. Traina, "Efficient execution of conjunctive complex queries on big multimedia databases," in *Multimedia (ISM), 2013 IEEE International Symposium on*, pp. 536–543, Dec 2013.

[9] G. Zhang, J. Wang, W. Huang, C. Li, Y. Zhang, and C. Xing, "A semantic++ mapreduce: A preliminary report," in *Semantic Computing (ICSC), 2014 IEEE International Conference on*, pp. 330–336, June 2014.

[10] Y. Tang and L. Liu, "Multi-keyword privacy-preserving search in personal server networks," *Knowledge and Data Engineering, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[11] I. Jaluta, "Transaction management in b-tree-indexed database systems," in *Information Science, Electronics and Electrical Engineering (ISEEE), 2014 International Conference on*, vol. 3, pp. 1968–1975, April 2014.

[12] T. Gollub, M. Volske, M. Hagen, and B. Stein, "Dynamic taxonomy composition via keyqueries," in *Digital Libraries (JCDL), 2014 IEEE/ACM Joint Conference on*, pp. 39–48, Sept 2014.

[13] I. Foster, "Designing and building parallel programs," 1995.

[14] R. Les Cottrell, "Retrieving pinger historical data," September 2014.

[15] R. Les Cottrell, "Pinger: Ping end-to-end reporting," July 2015.