Comparing MapReduce and Spark in Computing the PCC Matrix in Gene Co-expression Networks

Nagwan Abdel Samee[1], Nada Hassan Osman[2], Rania Ahmed Abdel Azeem Abul Seoud[3]
Information Technology Department, College of Computer & Information Sciences[1]
Princess Nourah bint Abdulrahman University, Riyadh, 11461 Saudi Arabia[1]
Computer Engineering Department, Misr University for Science and Technology, Giza, 12511 Egypt[1]
Department of Communication &Electronics Engineering, Fayoum University, Egypt[2, 3]

***Abstract*—Correlation between gene expression profiles across multiple samples and the identification of inter-gene interactions is a critical technique for Co-expression networking. Due to the highly intensive processing of calculating the Pearson's Correlation Coefficient, PCC, matrix, it often takes too much processing time to accomplish it. Therefore, in this work, Big Data techniques including MapReduce and Spark have been employed in a cloud environment to calculate the PCC matrix to find the dependencies between genes measured in high throughput microarray. A comparison between the running time of each phase in both of MapReduce and Spark approaches has been held. Both these techniques can dramatically speed up the computation allowing users to work with highly intensive processing. However, Spark has yielded a better performance than the MapReduce as it performs the processing in the main memory of the worker nodes and avoids the unnecessary I/O operations with the disks. Spark has yielded 80 times speed up for calculating the PCC of 22777 genes, however the MapReduce attained barely 8 times speed up.**

***Keywords*—Pearson's correlation; Hadoop; MapReduce; spark; gene co-expression networks; GCN; Affymetrix microarrays**

I. INTRODUCTION

Gene co-expression networks (GCN) [1] are gaining attention nowadays as useful representations of biologically interesting interactions among genes. Finding the interactions with significant genes [2] can help in understanding their molecular pathways. Constructing the similarity matrix between genes in GCN is the most complex part as the complexity rises quadratically. So, the most computationally demanding all pairwise combinations must be analyzed.

The analysis of correlated genes can help in finding other gene functions or relationships. The correlation between genes can be estimated based on their expression values and can be visualized via networks that reveal the interactions between co-expressed genes. Utilizing such gene expression values is currently effortless using the public accessible genomics data banks for RNA-seq, and high throughput microarrays. However, the genomics data is public and available, still the analysis of such data needs powerful platforms and algorithms for its processing. The parallel computing technology plays an essential role in processing and analyzing such huge amount of data. Even though, there are many paradigms and platforms from the parallel computing technology have been intensively reviewed and compared in previous studies[3],[4], [5],[6], there still an open question in utilizing the big data techniques in the processing of the gene expression profiles and finding their relationships.

High throughput technologies such as the Affymetrix microarrays have turned molecular biology into a data-intensive discipline that requires the usage of high-performance computing resources[7]. Flexible framework is needed to cover the resources which are required in highly intensive processing, and to help in data storage and processing. This requires a huge investment in both money and manpower. This problem can be overcome by cloud computing which has emerged as an additional technology offering virtualized environments[8], [9].

High throughput microarrays contain a huge number of genes. Determining the relationships between all these gene experiments proved to be very useful in biological analyses[10]. It has helped in understanding the molecular basis of complex disease traits as well as the prediction of treatment responses of individual subjects. Several methods existed to construct correlation or similarity matrix, i.e., a two-dimensional triangular matrix, where each value is the similarity coefficient of one gene pair. Some examples of those methods are Pearson's[11],Spearman[12],Theil-Sen[13] and Kendall[14] correlations.

Computation between gene expression profiles across multiple samples and the identification of inter-gene interactions is a critical technique for Co-expression networking, which usually relies on all-pairs correlation (or a similar measure). In this respect, Pearson's Correlation Coefficient (PCC) is one of the techniques that have been widely used for gene co-expression network construction. All pairs PCC computation has recently been widely used in Bioinformatics; yet, it is computationally demanding large numbers of gene expression profile. In the present work, it is important to calculate the PCC matrix to find the dependencies between all huge numbers of genes measured in our high throughput microarray. It requires an enormous amount of computation, resulting in slow data processing and takes more days to finish the calculation because it is considered very highly intensive processing. So, new approaches are needed to calculate and accelerate such a complicated process.

There are technologies that show great promise in bioinformatics, such as MapReduce[15], Hadoop[16] and Spark[17] which can contribute to solving the intensive computations. MapReduce and Spark are widely used high performance parallel frameworks that can solve the problem of

the Pearson's Correlation Coefficient matrix[18]. Apache Spark is an open source designed to enhance the computational speed in highly intensive processing. MapReduce works on the file system commonly known as Hadoop Distributed File System (HDFS), whereas Spark works in memory data processing engine. Whenever any operation is performed, Hadoop reads the data from the disk and uses the MapReduce to perform the task. While Spark keeps the data in memory and performs operation at a faster speed than Hadoop. However, the main drawback of MapReduce lies in its relatively high runtime for input datasets consisting of thousands of genes. This prevents the wide adoption of this method by the scientific community especially in intensive processing computation.

The present work focuses on holding a comparison between two Big Data techniques: MapReduce and Spark, which are considered parallel tools that accelerate the construction of intensive processing of pairwise correlation matrix between genes. Multithreading Programming Model [19] in both techniques are employed in this study to achieve efficient performance. The rest of the paper is organized as follows: Section II presents previous works related to the parallelization of the algorithm to calculate the Pearson's Correlation for intensive processing data to find the dependencies between all the huge numbers of genes. Section III describes the parallel implementation in MapReduce and Spark. Section IV provides the experimental evaluation regarding runtime efficiency. Section V presents the results and discussion. Finally, the conclusion is presented in Section VI.

II. Literature Review

This section shows recent work for determining the PCC matrix in Bioinformatics[20],[21] and Non-bioinformatics [22][23]applications. In[20], a parallel approach has been introduced to analyze the correlation between genes. In [24], a parallel implementation of transcription networks via GPUs, Graphical Processing Units, has been developed. In [25] and [26] a Message Passing Interface (MPI) implementation for the parallel construction of similarity matrices on multicore cluster processing have been provided. They have used MI (Mutual Information) instead of Pearson's Correlation for inferring the interactions between genes. Although the MI can detect non-linear connections better than Pearson's Correlation, some experiments have shown that it is not relevant in the case of gene co-expression networks. In [27], a parallel tool for the construction of GCN using GPUs was introduced. In [28], a distributed approach for computing the PCC matrix on Intel Xeon cluster has been proposed. A hybrid approach of MPI and OpenMP to compute the PCC matrix has been provided in [29]. A parallel approach on a multicore cluster using MPI, MPIGeneNet, has been introduced in [30] which uses GSL (GNU Scientific Library) and MKL (Math Kernel Library) libraries to perform mathematical functions. These libraries are in continuous evolution, so MPIGeneNet will benefit from future library updates without requiring any modification in its code. In [31], an algorithm for constructing the GCN using MapReduce in a cloud environment has been developed. In that algorithm, an approach from the information theory, ARACNE [21], has been employed. In [22] a parallel implementation of the Support Vector Machine, SVM, algorithm using the OpenMP for the Multicore platform has been developed. In [32], a parallel approach using GPU to compute the Pcc matrix in a Magnetic Resonance Imaging, MRI, images to estimate the functional interactions in human's brain. Another work for calculate the PCC matrix using a hybrid approach of the MPI, and the Compute Unified Device Architecture, CUDA has been developed in [33]. And in [23], a model for estimating the scalability of the parallel algorithms in the Cluster platform has been presented. Recently, in [34], a Parallel MapReduce (PMR) framework was proposed to compute bioinformatics applications and reduce the computation cost.

Each of the afore-mentioned frameworks provides a different paradigm of parallel programming and has their own strong and weak points. However, the performance of the Big Data techniques in the construction of similarity matrix in GCN still needs more examination. In this research, a comparison is held between the MapReduce, and Spark to compute the PCC matrix in real data of time series microarrays of Hepatocellular Carcinoma, HCC, containing a massive number of genes. The comparison has been done in a cloud environment which is more inexpensive, and flexible than the on-premises computing resources [31]. Cloud computing model has achieved an incredible performance for many applications in bioinformatics [35],[36].

III. Methods

The Pearson correlation coefficient is one of the most popular approaches in measuring the intensity of a linear association between two genes in a Gene Co-expression Network; hence, it has been applied here as a measure of dependencies between interacting genes. Let(X&Y) considered as a pair of gene expression profiles & (n) is the number of pairs in a gene expression data then the PCC can be calculated as shown in (1).

$$PCC = \frac{\sum_{i=1}^{n} XY - \frac{\sum_{i=1}^{n} X \sum_{i=1}^{n} Y}{n}}{\sqrt{(\sum_{i=1}^{n} X^2) - (\frac{(\sum_{i=1}^{n} X)^2}{n})} \sqrt{(\sum_{i=1}^{n} Y^2) - (\frac{(\sum_{i=1}^{n} Y)^2}{n})}} \quad (1)$$

The computation of the Pearson's Correlation Coefficient between genes expressed in a gene expression matrix has a quadratic complexity. Therefore, we are suggesting here a parallel algorithm that will break the entire calculation of PCC matrix into components. Each component represents an independent computation. Fig. 1 and Fig. 2 depict a flowchart for calculating the PCC matrix using the MapReduce and Spark.

Each technique receives an input matrix comprising the gene expression values for each gene in different conditions, samples. These expression data are saved in a numeric matrix, with n columns, the number of genes, and m rows, the number of samples.

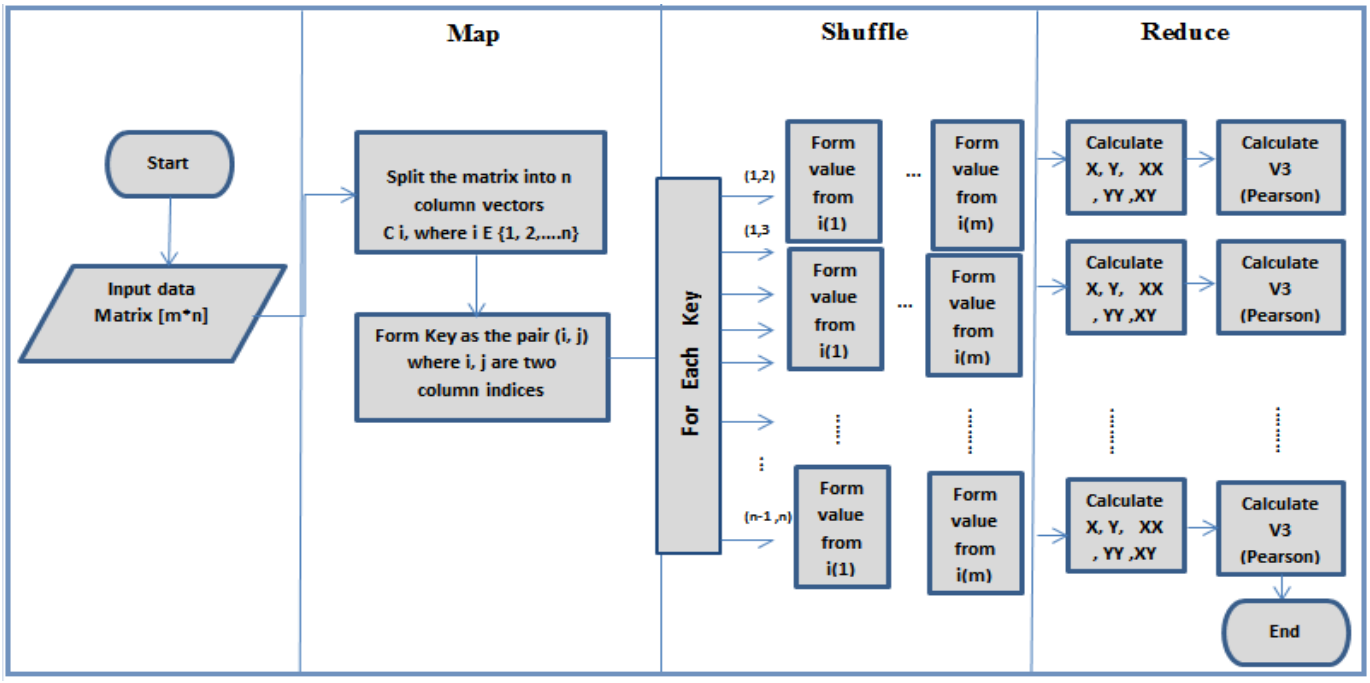

Fig. 1. Flowchart for Calculating the PCC Matrix between Genes Expressed in a Gene Expression Matrix using MapReduce.

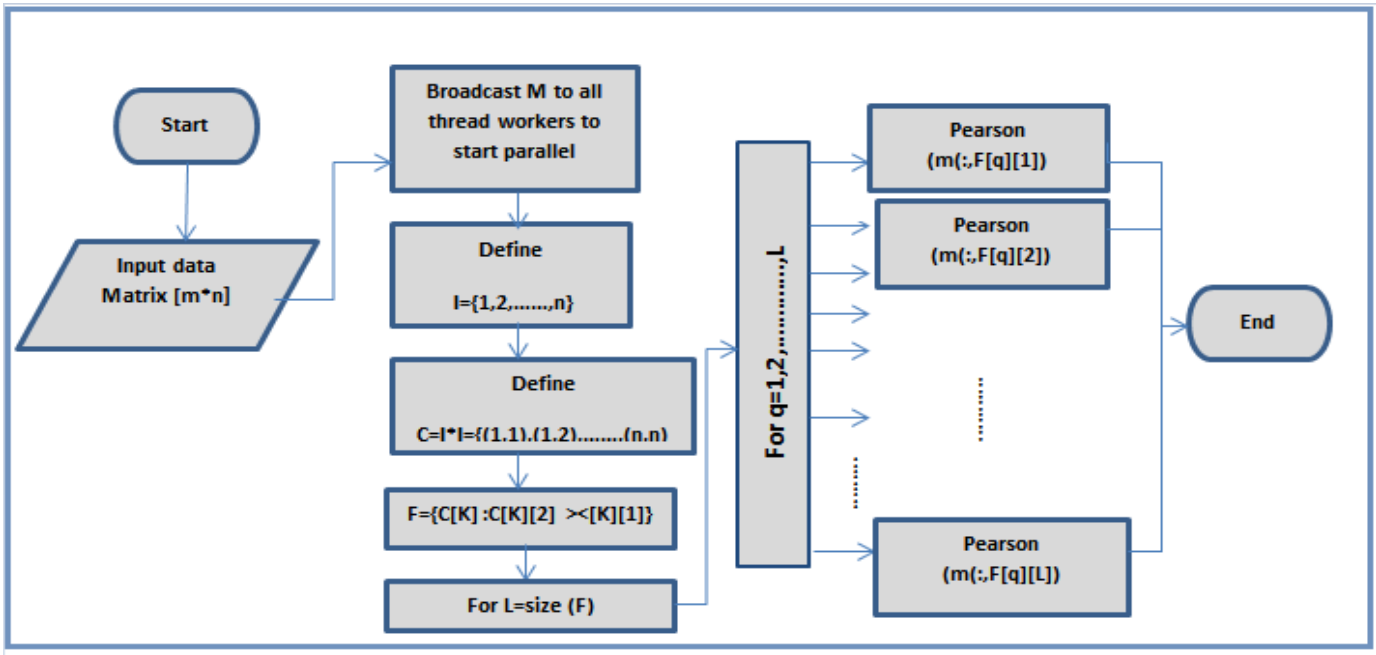

Fig. 2. Flowchart for Calculating the PCC Matrix between Genes Expressed in a Gene Expression Matrix using Spark.

MapReduce consists of mappers which perform a large portion of work and reducers which perform a relatively small amount of computation which achieves the best performance. In the implementation of the Multithreaded Mapper implementation, threads from a thread pool invoke a queue of key value pairs in parallel. Multiple threads running a map task can help to speed up the tasks, based on availability of cores in the system. The map and shuffle task only receive a key-value pair input <k1, V1> and get outputs with other key-value pair in parallel. However, the reduce task receives the input from shuffle<K2, List<V2>>, which is a key and a list of values associated with that key. It gets all this pairs of values associated with the i-th and j-th columns/variables and compute Pearson correlation. The output of the Reduce phase is (K3,V3) as the PCC between a pair of genes, the i-th and j-th genes.

Spark integrates the whole functionality in one program. This makes the tool easier to work with the users having only to launch the application once and avoid writing/reading from disks because it is based on memory which makes it is faster than MapReduce. Spark uses a hybrid approach that combines

MPI processes and threads. Each MPI process launches multiple threads to efficiently exploit the cores available on each node and to reduce the memory requirements. Regarding the workload distribution, as the PCC must be calculated for all gene pairs, the workload of this step can be represented with a 2D matrix, where both axes x and y include all genes. Further improvement in the performance of the proposed algorithm has been done by dividing the data into multiple partitions based on the number of threads and execute on available cores on multiple nodes and also only half of the matrix must be calculated Concretely sum (i, i=1 to N) = N * (N + 1) / 2 which called triangular numbers (Since cor(A,B)=cor(B,A)) which adding more time for saving. So, every pair of genes has been handled as a compute job, key them with a unique index, send to any compute unit available, and put them back into the result matrix using their key.

The detailed algorithm to construct the PCC matrix using the Map Reduce and Spark are shown below.

Algorithm: PCC using MapReduce

Require: Input data matrix A[m,n] where n length of A and split the matrix into n column ci where I {1,2,…..n} . (i,j) are two cloumn indicies. N as the number of values in array V -> N = length of array V.

```
Function Map(Array A)
for each thread
        for i = 0....A.length
                for j = i....A.length
                        Let k = pair(i, j)
                        Let v = pair(A[i], A[j])
                        Send the key (k) and the value pair (v)
Function Reduce(key k, value pairs V)
        Let X = 0
        Let Y = 0
        Let XX = 0
        Let YY = 0
        Let XY = 0
        for i=1 to N
                Let X = X + V[i][1]
                Let Y = Y + V[i][2]
                Let XX = XX + (V[i][1])^2
                Let YY = YY + (V[i][2])^2
                Let XY = XY + V[i][1] * V[i][2]
Let P = (XY - (X * Y / N)) / sqrt((XX - (X^2 / N)) * (YY - (Y^2 / N))
Compute P
end for
```

Algorithm: PCC using Spark

Require: Input data matrix A[m,n] where n length of A ,Number of threads=8, Obtain indices -> I[i], where i=1 to n.

```
Broadcast matrix to all thread workers
for each thread
Let I be an array of size n
for i = 1....I.size
        I[i] = i
Le let C be an array of pairs, where each element of C is holds an element of the cac cartesian product of the elments of I -> C[i], where i =1 to n^2
Let F be an empty array
Let j = 0
for i = 1....C.size
        if C[i][1] is more than or equal C[i][2]
                F[j] = C[i]
                Let j = j + 1
Let P be an array of C.size
For i = 1....F.size -> where (i=1 to n*(n + 1)/2)
        Compute the Pearson correlation M[:, P[i][1]] and M[:, P[i][2]]
end for
```

IV. Experimental Setup

A. Material

The data employed in this research is a non-benchmarking dataset for Liver cancer, Hepatocellular Carcinoma (HCC). It is a real data downloaded from GEO, Gene Expression Omnibus data bank [37] and contains thirty-five Microarray samples of HCC that have been downloaded from. These samples have been collected using the Affymetrix HG-U133A 2.0 platform. HCC is a complication of HCV (Hepatitis C virus) cirrhosis. The raw data has been preprocessed by the Affy package which is provided by the Bioconductor [38].

B. Platform

A cloud platform from IBM, IBM Analytics Engine, IAE[39] has been utilized in this research. The IAE offers a parallel infrastructure for MapReduce and Spark on the IBM cloud. It permits users to upload their data in a layer called the IBM Cloud Object Storage and provides clusters of computing nodes to work on the uploaded data. The separation of the computing and storage layers helps in having more scalability and flexibility in analyzing. Analytics libraries and open-source packages has been employed. More details about the hardware and software employed in each technique are listed in Table I.

TABLE I. CHARACTERISTICS OF THE CLOUD PLATFORM USED IN THE EXPERIMENTAL EVALUATION

Specification	Hadoop	Spark
Software package	AE 1.2 (Analytic Engine IBM)	
Version	Hadoop 3.1.1	Apache Spark 2.3.2
Nodes	2	
Cores	32	
Memory	128GB	
Driver Memory	50GB	
Executor Memory	20GB	
Spark driver max Result size	Should be more than or equal 25GB	
Thread	8	

V. RESULTS AND DISCUSSION

The performance of the employed techniques has been evaluated by calculating the whole running time on different chunks of data including 400, 1000, 4000, 8000, 12000, 16000, and 22777genes. Multithreading for each technique splits tasks into threads to execute them at the same time in parallel.The retrieved running time using the proposed MapReduce, and Spark algorithms is shown in Fig. 3, and Fig. 4 correspondingly. As illustrated in Fig. 3, the running time for finding the correlation between 4000 genes was an hour and twenty minutes and it increases exponentially until it reaches four hours and fifty minutes for the whole number of genes, 22777. The exponential increase in the running time reveals a high time complexity in using the MapReduce approach. That complexity is due to storing the whole dataset to HDFS after running each job. In addition, the dataset is replicated three times in HDFS by default. Thus, the time complexity of Map task in intrinsic operations (sorting, shuffling, sending data, etc.) is O(n^2) operations per line and O(n) in the Reduce task.

As noticed from Fig. 3, a long processing time has been retrieved using the MapReduce. Such long processing time can be clarified as follows:

- The size of the whole dataset employed in this work is small (3.7 MB) which is less than the minimum block size 64 MB of HDFS. This means that if a block size of HDFS is 64 MB with 3 replication and we have 1 MB file, we do not lose (63 MB * 3) 189 MB. Since physically just three 1 MB files are stored with the standard block size of the underlying file systems and Hadoop will typically try to spawn a mapper per block. So, if we have 40 blocks with10 KB files, then Hadoop may end up spawning 40 mappers eventually per block even if the size of file is small.
- The number of mappers and reducer tasks are directly proportional to the input splits, which depend on DFS block size (no. split= no. Map =input size/block size) which helps in working all tasks in parallel well. In our present work, we had one Map task, and one reduce. So, using HDFS in intensive parallel processing with small dataset was not helpful in MapReduce. This is a real wastage overhead time when using MapReduce.

On the other side, as illustrated in Fig. 4, the running time for the Spark algorithm is less than the time consumed using the MapReduce algorithm. The consumed time for retrieving the correlation between 4000 genes was just five minutes and fifty-four seconds and it increases exponentially until it reaches twenty-nine minutes for the whole number of genes, 22777.

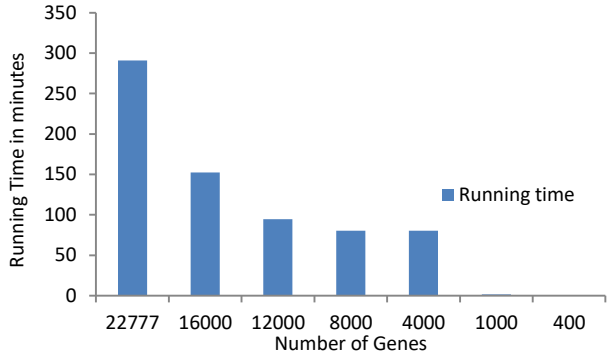

Fig. 3. The Running Time of MapReduce Algorithm on different Chunks of Data on IBM Analytic Engine.

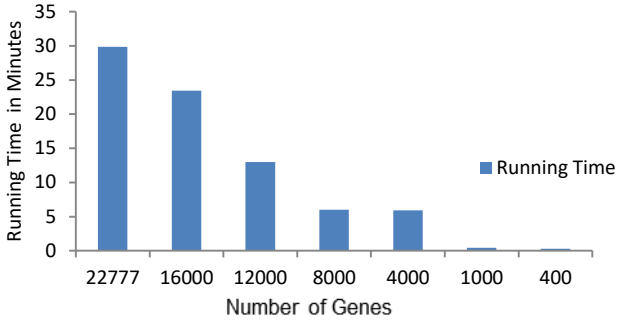

Fig. 4. The Running Time of Spark Algorithm on different Chunks of Data on IBM Analytic Engine.

In Spark, Resilient Distributed Datasets (RDD) is employed to manage the data through partitions. RDD helps to distribute the data processing with negligible network traffic for sending data between executors in parallel. The best way to decide the number of partitions into smaller chunks of data is to make the number of partitions equal to the number of cores in the cluster; so that all the partitions will process in parallel, and the resources will be utilized in an optimal way.

We used 32 partitions like the number of Virtual cores and 8 threads in the cluster to achieve the best performance as we can. Compared to MapReduce, Spark does not work the entire dataset in the HDFS until completing each task as done in MapReduce.

To sum up the comparison between the MapReduce and Spark in calculating the correlation coefficient between genes in a gene expression matrix, a bar chart for the retrieved running times using them is depicted in Fig. 5.

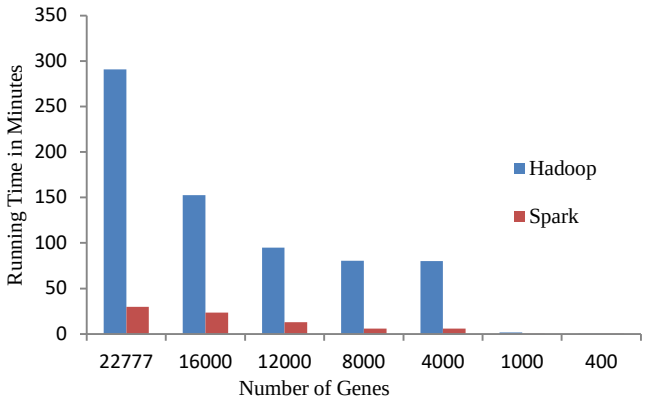

Fig. 5. A Comparison between the Retrieved Running Time in Calculating the PCC Matrix on different Chunks of Genes in using MapReduce and Spark.

A. *Comparing the Performance*

In this section, the performance between MapReduce and Spark algorithms is compared to a single CPU processing of the PCC. We have calculated the speedup for different data sizes as listed in Table II. In Table II, N, and M represents the number of genes and number of samples. It can be noticed that the maximum speed up of Spark, 304.1x, is considerably greater than the corresponding one on the MapReduce, 83.12x, for a data size of 1000 genes represented in 15 samples.

In addition, we have performed more investigation to the performance of the employed techniques, MapReduce, and Spark, by analyzing the total running time. The running time in the MapReduce technique is composed of three components including the Mapping, Shuffle & Merge, and Reduce. However, in the Spark, it is composed only of the Mapping, and Reduce phases. As depicted in Table III, long time is consumed in the Mapper phase compared to the Reduce phase.

Long time for the phases of the MapReduce technique can be clarified as follows. Reading the data from the disk and then running the mappers has consumed too much time, four hours. The Generation of a lot of keys has taken a lot of time to sort them. Storing the output of mappers back on disk also has consumed around 6 minutes. Then the reading/storing of the data from the disk in Shuffling phase has contributed to the increased processing time, 3 minutes. Finally, the Reduce phase has spent 36 minutes.

Six times the disk is accessed to complete one job in the MapReduce which slowdown its processing. Thus, MapReduce has a big drawback since it must operate with the entire set of data in the HDFS on the completion of each task, which in turn increases the time and the cost of processing data, so we found that Spark is faster than MapReduce with 10.2613%.

Although, Spark has two phases including transformation (Map) and reduce tasks as in the MapReduce approach. Spark has spent only 23 minutes and the other is for action (reduce) which spent only 2 minutes. An interpretation to the retrieved performance of Spark can be given as follows:

- In Spark, a Directed a cyclic graph (DAG)[40] is created when it starts executing a job.
- Looking at the DAG, remembering the steps in the DAG. The Spark's DAGs enable optimizations between steps compared to Hadoop which does not have any cyclical connection between MapReduce steps. That is, at that level, no performance tuning can occur which proven that Spark is much faster for applications.
- Having eight steps for transformations but does not really go to disk to perform the transformations.

TABLE II. COMPARING THE PERFORMANCE BETWEEN MAPREDUCE & SPARK.

Matrix Size (N*M)	**Serial PCC Time(s) / Speedup**	**MapReduce Time(s) / Speedup**	**Spark Time(s) / Speedup**
400*10	826.05/ 1x	25.98/ 31.79 x	17.652/ 46.77x
1000*15	7,773.29/ 1x	93.516/ 83.12 x	25.56198/ 304.1x
4000*20	22034.14/ 1x	4813.2/ 4.58 x	355.98/ 61.88x
8000*25	66402.9/ 1x	4830/ 13.75x	360/ 184.45x
22277*35	143,382.74/ 1x	17443.98/ 8.22x	1789.98/ 80.1x

TABLE III. A COMPARISON BETWEEN THE RUNNING TIME OF EACH PHASE IN BOTH OF MAPREDUCE AND SPARK APPROACHES IN CALCULATING THE CORRELATION MATRIX BETWEEN GENES IN A HIGH THROUGHPUT MICROARRAY

Technique	**Phase**	**Phase Time**
MapReduce	Mapping phase	4 hours + 6 minutes+53 seconds
	Shuffle & merge phase	3 minutes + 48 seconds
	Reduce Phase	36 minutes+ 46 seconds
Spark	Mapping phase	23 minutes
	Reduce phase	2 minutes

At this point, a Spark job goes to disk, performs the first transformation, keeps the result of transformation in memory, performs the second transformation, keeps the result in memory and so on until all the steps are completed; so only two accesses to disk to write the output of the job which makes Spark faster than MapReduce to access the disk. The higher efficiency on Spark can be explained as follows. In Spark, each MPI process launches multiple threads to efficiently exploit the available cores on each node and to reduce the memory requirement. The launched threads have a shared memory space on the cluster cores and allowing parallel execution with up to 32 working processes. This approach can be implemented easily with Spark which makes it faster than MapReduce.

VI. Conclusion

In the present paper, we have introduced a parallel implementation for an algorithm for computing the PCC matrix in GCN using Spark and MapReduce approaches. Spark has yielded a better performance than the MapReduce as it performs the processing in the main memory of the worker nodes and prevents the unnecessary I/O operations with the disks. So, the memory in the Spark cluster should be at least as large as the amount of data needed to process. However, Spark is more expensive when setting up the cluster because it requires more RAM compared to MapReduce. MapReduce is highly fault-tolerant because it was designed to replicate data across many nodes but in Spark, data is replicated across executor nodes, and can generally be corrupted if the node or phase between executors and drivers fails.

As a future work for this study, we are recommending the implementation of PCC matrix using GPUs and comparing its performance to the results retrieved in this research.

Acknowledgment and Funding

This research was funded by the Deanship of Scientific Research at Princess Nourah bint Abdulrahman University through the Fast-track Research Funding Program.

References

[1] N. M. A. Samee, N. H. Solouma, and Y. M. Kadah, “K4. Gene network construction and pathways analysis for high throughput microarrays,” Natl. Radio Sci. Conf. NRSC, Proc., no. April, pp. 649–658, 2012, doi: 10.1109/NRSC.2012.6208578.

[2] N. M. Abdel Samee, N. H. Solouma, and Y. M. Kadah, “Detection of biomarkers for Hepatocellular Carcinoma using a hybrid univariate gene selection methods,” Theor. Biol. Med. Model., vol. 9, no. 1, p. 1, 2012, doi: 10.1186/1742-4682-9-34.

[3] A. O, K. AR, and A. A, “Parallel Algorithms for Inferring Gene Regulatory Networks: A Review,” Curr. Genomics, vol. 19, no. 7, pp. 603–614, Jun. 2018, doi: 10.2174/1389202919666180601081718.

[4] H. Shi, B. Schmidt, W. Liu, and W. Müller-Wittig, “Parallel mutual information estimation for inferring gene regulatory networks on GPUs,” BMC Res. Notes, vol. 4, p. 189, 2011, doi: 10.1186/1756-0500-4-189.

[5] F. F. Borelli, R. Y. de Camargo, D. C. Martins, Jr, and L. C. Rozante, “Gene regulatory networks inference using a multi-GPU exhaustive search algorithm,” BMC Bioinformatics, vol. 14, no. Suppl 18, p. S5, 2013, doi: 10.1186/1471-2105-14-S18-S5.

[6] W.-P. Lee, Y.-T. Hsiao, and W.-C. Hwang, “Designing a parallel evolutionary algorithm for inferring gene networks on thecloud computing environment,” BMC Syst. Biol., vol. 8, no. 1, p. 5, Jan. 2014, doi: 10.1186/1752-0509-8-5.

[7] O. Spjuth et al., “Experiences with workflows for automating data-intensive bioinformatics,” Biol. Direct, vol. 10, no. 1, pp. 1–12, 2015, doi: 10.1186/s13062-015-0071-8.

[8] M. C. Schatz, B. Langmead, and S. L. Salzberg, “Cloud computing and the DNA data race,” Nat. Biotechnol., vol. 28, no. 7, pp. 691–693, 2010, doi: 10.1038/nbt0710-691.

[9] L. D. Stein, “The case for cloud computing in genome informatics,” Genome Biol., vol. 11, no. 5, 2010, doi: 10.1186/gb-2010-11-5-207.

[10] P. Minguez and J. Dopazo, “Assessing the biological significance of gene expression signatures and co-expression modules by studying their network properties,” PLoS One, vol. 6, no. 3, 2011, doi: 10.1371/journal.pone.0017474.

[11] A. Gobbi and G. Jurman, “A null model for pearson coexpression networks,” PLoS One, vol. 10, no. 6, 2015, doi: 10.1371/journal.pone.0128115.

[12] J. Nie et al., “TF-Cluster: A pipeline for identifying functionally coordinated transcription factors via network decomposition of the shared coexpression connectivity matrix (SCCM),” BMC Syst. Biol., vol. 5, 2011, doi: 10.1186/1752-0509-5-53.

[13] H. Peng, S. Wang, and X. Wang, “Consistency and asymptotic distribution of the Theil-Sen estimator,” J. Stat. Plan. Inference, vol. 138, no. 6, pp. 1836–1850, 2008, doi: 10.1016/j.jspi.2007.06.036.

[14] F. Gómez-Vela, C. D. Barranco, and N. Díaz-Díaz, “Incorporating biological knowledge for construction of fuzzy networks of gene associations,” Appl. Soft Comput. J., vol. 42, pp. 144–155, 2016, doi: 10.1016/j.asoc.2016.01.014.

[15] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008, doi: 10.1145/1327452.1327492.

[16] Tom White, “Hadoop: The Definitive Guide [Book],” O’Reilly Media, Inc. https://www.oreilly.com/library/view/hadoop-the-definitive/9780596521974/ (accessed Jul. 20, 2020).

[17] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” 2nd USENIX Work. Hot Top. Cloud Comput. HotCloud 2010, 2010.

[18] S. Wang et al., “Optimising parallel R correlation matrix calculations on gene expression data using MapReduce,” BMC Bioinformatics, vol. 15, no. 1, pp. 1–9, 2014, doi: 10.1186/s12859-014-0351-9.

[19] K. Kavi, “Multithreading Implementations The University of Texas at Arlington,” no. September 1998, 2013.

[20] M. M. Zhu and Q. Wu, “Transcription network construction for large-scale microarray datasets using a high-performance computing approach,” BMC Genomics, vol. 9, no. SUPPL. 1, pp. 1–10, 2008, doi: 10.1186/1471-2164-9-S1-S5.

[21] P. Zoppoli, S. Morganella, and M. Ceccarelli, “An information theoretic approach to reverse engineering of regulatory gene networks from time-course data,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 6160 LNBI, pp. 97–111, 2010, doi: 10.1007/978-3-642-14571-1_8.

[22] H. Zhu, P. Li, P. Zhang, and Z. Luo, “A high performance parallel ranking SVM with OpenCL on multicore and many-core platforms,” Int. J. Grid High Perform. Comput., vol. 11, no. 1, pp. 17–28, 2019, doi: 10.4018/IJGHPC.2019010102.

[23] L. B. Sokolinsky, “BSF: A parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems,” J. Parallel Distrib. Comput., vol. 149, no. May, pp. 193–206, 2021, doi: 10.1016/j.jpdc.2020.12.009.

[24] J. Ingram and M. Zhu, “GPU accelerated microarray data analysis using random matrix theory,” Proc.- 2011 IEEE Int. Conf. HPCC 2011 - 2011 IEEE Int. Work. FTDCS 2011 -Workshops 2011 Int. Conf. UIC 2011-Work. 2011 Int. Conf. ATC 2011, pp. 839–844, 2011, doi: 10.1109/HPCC.2011.119.

[25] J. Zola, M. Aluru, A. Sarje, and S. Aluru, “Parallel information-theory-based construction of genome-wide gene regulatory networks,” IEEE Trans. Parallel Distrib. Syst., vol. 21, no. 12, pp. 1721–1733, 2010, doi: 10.1109/TPDS.2010.59.

[26] S. Misra, K. Pamnany, and S. Aluru, “of Genome-Scale Networks on the Intel Ò Xeon Phi TM Coprocessor,” vol. 12, no. 5, pp. 1008–1020, 2015.

[27] M. Liang, F. Zhang, G. Jin, and J. Zhu, "FastGCN: A GPU accelerated tool for fast gene co-expression networks," PLoS One, vol. 10, no. 1, pp. 1–11, 2015, doi: 10.1371/journal.pone.0116776.

[28] Y. Liu, T. Pan, and S. Aluru, "Parallel Pairwise Correlation Computation on Intel Xeon Phi Clusters," Proc. - Symp. Comput. Archit. High Perform. Comput., pp. 141–149, 2016, doi: 10.1109/SBAC-PAD.2016.26.

[29] J. González-Domínguez and M. J. Martín, "Fast Parallel Construction of Correlation Similarity Matrices for Gene Co-Expression Networks on Multicore Clusters," Procedia Comput. Sci., vol. 108, pp. 485–494, 2017, doi: 10.1016/j.procs.2017.05.023.

[30] J. Gonzalez-Dominguez and M. J. Martin, "MPIGeneNet: Parallel calculation of gene co-expression networks on multicore clusters," IEEE/ACM Trans. Comput. Biol. Bioinforma., vol. 15, no. 5, pp. 1732–1737, 2018, doi: 10.1109/TCBB.2017.2761340.

[31] Y. Abduallah, T. Turki, K. Byron, Z. Du, M. Cervantes-Cervantes, and J. T. L. Wang, "MapReduce Algorithms for Inferring Gene Regulatory Networks from Time-Series Microarray Data Using an Information-Theoretic Approach," Biomed Res. Int., vol. 2017, 2017, doi: 10.1155/2017/6261802.

[32] T. Eslami and F. Saeed, "Fast-GPU-PCC: A GPU-based technique to compute pairwise pearson's correlation coefficients for time series data—fMRI study," High-Throughput, vol. 7, no. 2, 2018, doi: 10.3390/ht7020011.

[33] E. Kijsipongse, S. U-Ruekolan, C. Ngamphiw, and S. Tongsima, "Efficient large Pearson correlation matrix computing using hybrid MPI/CUDA," Proc. 2011 8th Int. Jt. Conf. Comput. Sci. Softw. Eng. JCSSE 2011, no. May, pp. 237–241, 2011, doi: 10.1109/JCSSE.2011.5930127.

[34] A. A. Al-Absi, N. A. Al-Sammarraie, W. M. S. Yafooz, and D. K. Kang, "Parallel MapReduce: Maximizing cloud resource utilization and performance improvement using parallel execution strategies," Biomed Res. Int., vol. 2018, 2018, doi: 10.1155/2018/7501042.

[35] T. Nguyen, W. Shi, and D. Ruden, "CloudAligner: A fast and full-featured MapReduce based tool for sequence mapping," BMC Res. Notes, vol. 4, 2011, doi: 10.1186/1756-0500-4-171.

[36] S. Zhao et al., "Rainbow: A tool for large-scale whole-genome sequencing data analysis using cloud computing," BMC Genomics, vol. 14, no. 1, 2013, doi: 10.1186/1471-2164-14-425.

[37] R. Edgar, M. Domrachev, and A. E. Lash, "Gene Expression Omnibus: NCBI gene expression and hybridization array data repository," Nucleic Acids Res., vol. 30, no. 1, pp. 207–210, 2002, doi: 10.1093/nar/30.1.207.

[38] R. C. Gentleman et al., "Bioconductor: open software development for computational biology and bioinformatics.," Genome Biol., vol. 5, no. 10, 2004, doi: 10.1186/gb-2004-5-10-r80.

[39] A. G. Bromley, "Babbage's Analytical Engine Plans 28 and 28a - The Programmer's Interface," IEEE Ann. Hist. Comput., vol. 22, no. 4, pp. 5–19, Oct. 2000, doi: 10.1109/85.887986.

[40] H. Wang, S. Li, X. Li, and H. Zhong, Microstructure and thermoelectric properties of doped p-type CoSb3 under TGZM effect, vol. 466. 2017.