Towards a Full Big Data Based Solution for Computationally Intensive Problems

Solving Motif Finding Problem as a Case Study

M. M. Al-Qutt*[γ], H. Khaled*, H. M. Faheem*
*Computer Systems Department, Faculty of Computer & Information Science,
Ain Shams University, Cairo 11566, Egypt,
mmalqutt@cis.asu.edu.eg, heba.khaled@cis.asu.edu.eg, hmfaheem@cis.asu.edu.eg
[γ] Corresponding Author

Rania ElGohary[β]
[β]Information Systems Department, Faculty of Computer & Information Science,
Ain Shams University, Cairo 11566, Egypt, Email: dr.raniaelgohary@fcis.asu.edu.eg

Abstract— **The Solution scalability of computationally intensive problems constitutes a major challenge to researcher in the High Performance Computing realm. Cloud platform services have been explored for storage and analytics purposes and taking into consideration the huge volume of data, tremendous cloud storage platforms that currently exist and provide a scalable, distributed storage service, Therefore, the challenges of implementing (MF) Motif Finding problem based on the services provided by the cloud storage platform, are addressed. In this work, Apache Hadoop is picked for Big Data processing which is an open source Platform that provides enormous number of clusters that is used for parallel implementations. MF solution was implemented using two different Big Data frameworks: MapReduce and Apache Spark, they have different implementation schemes and resources usage plans. The results are collected and analyzed against speedup value. From experiments Spark achieves much more speedup than Mapreduce. This finding is expected due to Mapreduce I/O operations overheads. The main purpose of this step is to accomplish effective parallelization and evaluates the performance of such an integrated solution.**

Keywords— Motif Finding (MF), High Performance Computing, Big Data, Spark, Mapreduce.

I. INTRODUCTION

The fact that (MF) problem is NP-complete problem is already established [1]. This categorization as NP-complete reveals the fact the performance is not bounded whenever the dataset size increases. As many algorithms and techniques have been proposed with high accuracy and measurable speedup yet, they could not bind the performance when the real-world data set size grows. In this work, we perceive the MF problem as a big data problem and we proposed a solution for (MF), as a case study, using big data platforms exploiting the cloud-based services.

This paper considered Motif finding problem as case study for a bioinformatics computationally intensive problem. Motifs are defined as short patterns that have remarkable number of occurrences in the DNA sequences' set. Discovering of these patterns helps in gene function exploration and constructing regulatory networks. Mutations could be exploited in random positions of the genome and these patterns are also subject to variations, promoting the problem more challenging. Motif Finding (MF) aims to locate unknown motifs that are expected to be common in a set of sequences. A well-known (MF) variant is called planted (l; d) models the locating of these subtle motif patterns in the DNA.

In this paper we employ "SKIP Brute-Force" by Faheem[2] to solve planted (MF) (15, 4).

The rest of this paper is organized as follows: section 2 is concerned with problem definition and the related work has been reviewed in section 3, while section 4 exploits the scientific background. The implementation of the proposed system and a discussion of its performance are illustrated in section 5 and 6. Section 7 summarizes the conclusion.

II. PROBLEM DEFINITION

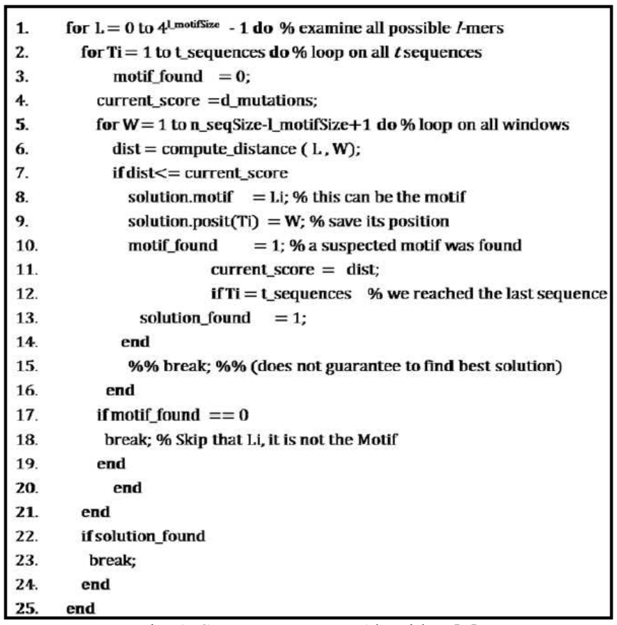
Sustaining the scalability of high performance paradigms based solutions of computationally intensive bioinformatics problems is considered a rich filed for researchers in high performance computing and cloud computing fields.

The concept of "Parallel Computing" has gained a successful history and achievements; it focuses on "number munching". Wide-used programs and applications were tightly coupled with CPU intensive power. The main weak point for those solutions is they are tricky and hard to build beside other hard targets to accomplish "fault tolerance" and "Scalability". In contrast, a recent class of parallel and distributed systems has appeared, "Cloud Computing".

This work focus on MF problem as a case study, It is a fact that (MF) problem is an NP-complete problem which is already proven by Jones et al [1]. Techniques used for solving motif finding problem could be categorized into either pattern or profile based techniques. In the pattern based category the main objective is to reach a global exact optimal solution, while in the based techniques it concerns more with probabilistic models of motif locations. Pattern based techniques is better used when targeting short motifs [3].in the other hand the profile based techniques are more suitable to locate longer and general motifs but this doesn't guarantee global optimality of the results [4].

This work is addressing an optimized version of Brute-Force algorithm which is called "SKIP Brute-Force" (SKIP BF) Algorithm [5, 6], Fig 1. The main privilege of SKIP Brute-Force is its ability of early detection of useless iteration that will not lead to correct solution and skip them. Many reasons favored SKIP BF over other algorithms such as: it can reach an exact optimal solution that guarantees the global optimality; also, the algorithm is parallelizable by nature.

In this paper, SKIP BF algorithm is parallelized and implemented using two different Big Data frameworks: MapReduce and Apache Spark, they have different implementation schemes and resources usage plans.



```
for L = 0 to $4^{L\_motifSize}$ - 1 do % examine all possible l-mers
  for Ti = 1 to t_sequences do % loop on all t sequences
      motif_found   = 0;
    current_score = d_mutations;
    for W = 1 to n_seqSize-l_motifSize+1 do % loop on all windows
      dist = compute_distance ( L , W);
      if dist<= current_score
        solution.motif    = Li; % this can be the motif
        solution.posit(Ti) = W; % save its position
        motif_found      = 1; % a suspected motif was found
                  current_score =  dist;
                  if Ti = t_sequences   % we reached the last sequence
          solution_found    = 1;
        end
        %% break; %% (does not guarantee to find best solution)
     end
    if motif_found  == 0
     break; % Skip that Li, it is not the Motif
    end
      end
  end
  if solution_found
   break;
  end
end
```

Fig. 1: SKIP Brute-Force Algorithm [5]

III. Related Work

Different researchers exploited Big Data tools, platforms and programming frameworks in bioinformatics problems in general and (MF) specifically. Bioinformatics tool in [7] has been launched to build genome pipeline in Scala and for RNA and DNA sequence analysis. The main target was to criticize the scalability and performance by analyzing large scale bio datasets: genome, protein, and DNA. MapReduce model was proposed and implemented for parallel and distributed execution in Spark. Nordberg et al. [8] developed "BioPig" which is employed in processing and analyzing large-scale sequence datasets. This system gained a remarkable level of scalability, portability and programming simplicity and portability. The undeniable limitation of the proposed system is the latency in startup of Hadoop. This problem is solved by Spark. Sun et al. [9] exploited Hadoop MapReduce to implement "Mapping of long sequence" using Bwasw-cloud algorithm which resolves the performance issues those exist in other algorithms such as: BLAST, SOAP, etc…

IV. Big Data and Cloud Computing Platform

The technology of Cloud computing is the consequence of the convergence of these technologies:

- Virtualization.
- Grid computing.
- Service Oriented Architecture (SOA).

The main objective of cloud computing is to provide various services based on a virtualized parallel back-end platform. Those services are categorized based on the offered resource. The categories are:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

The concept of big data is the key word nowadays in distributed and parallel paradigms. As the term implies, it can be defined as the collection of sets of extensive amount of data, its size is measured in terabytes, petobytes etc…. These data sets are very large and complex to process, analyze or even manage using traditional database management tools [10]. It is obvious that the traditional Relational Database Management Systems (RDBMS) could not handle the unbounded huge volume, diversity, unstructured nature and heterogeneity of Big Data.

The main characteristic of any big data application relies on the 3 factors which include:

- Data Velocity
- Data Volume
- Data Variety.

Data velocity refers to the data arrival rate, while volume indicates availability hugeness of the data and the multiple sources/formats of the data could be described as data variety [11]. Big Data is not only about the data size but also involves data diversity and velocity [12, 13] and analytics of Big data is defined as the process of exploiting deep analysis solutions, scripts and algorithms running upon powerful platforms to reveal the potentials hidden insights in big data, such as patterns or unknown correlations. From the processing time requirement perspective, big data analytics can be divided into two alternative types:

First, Batch Analytics: data are stored first and are analyzed later. (MapReduce) is considered one of the dominant batch-processing models in the last decade. MapReduce main concept is that data are separated into small elementary chunks. Then, those chunks are processed in parallel and distributed manner to compute the intermediate stage results (Map Step). Final results are calculated by performing aggregating all over the intermediate results (Reduce Step) [14, 15]. The MapReduce model is straightforward and commonly employed in bioinformatics, web mining, and heavy machine learning.

Second, Streaming Processing: The key theory of the streaming processing is the fact that some applications potential value and results correctness relies on data freshness. This paradigm processes and analyzes data once it is available to conclude the results. In data continuous arrival in streams, due to the stream speed and size, only a pre-defined portion of the stream is stored in bounded memory. The streaming processing paradigm is commonly employed in various online applications [16].

Apache Hadoop is a very common framework used to provide services for the processing of large data sets in distributed computing environments [17, 18]. Apache Hadoop ecosystem in its current version consists of:

- Hadoop Kernel
- Mapreduce
- Hadoop Distributed File System (HDFS)
- Add-on components such as Apache Hive, HBase and Zookeeper, pig, Oozie, and spark.

As it is obvious, Hadoop has many components; in this work we are concerned more with Hadoop Distributed File System (HDFS) and (MapReduce and Spark) as storage platform and processing services.

HDFS is a fault tolerant storage system as it ensures data replication [19]. HDFS manages and plans the storage over a set of machine, distributed File system. It is very useful whenever the data size is too much to be handled by a single machine.

The main purpose of HDFS is to divide data into smaller blocks and perform blocks distribution over the cluster. One node is called "The name node" which is responsible for storing the metadata for other nodes. Also, Name Nodes monitors the state of the Data Nodes beside other file system operations, Fig. 2 [20, 21].

A. MapReduce

MapReduce has obtained this outstanding popularity when Google employed it successfully as data processing mechanism [22]. MapReduce can be defined as a programming framework provides services for the fault tolerance and being simple and scalable and this forms the main reason behind being preferred through both industry and research communities in additions it supports parallel processing based programs [19]. The main idea of MapReduce is to divide input data set into chunks that are independent and processed in a parallel way [23]. From, the start, (HDFS) splits data in different machines and data is organized as pairs of (key, value). At the master machine (Single Node), the map function is called to perform job mapping which converts the inputs into intermediate data set and lastly, the reduce function takes these intermediate results and aggregates these data into a smaller set of outputs. Fig. 3 illustrates Hadoop MapReduce architecture

Mapreduce program contains 2 main processing stages: (1) Map phase, (2) Reduce phase. The real MapReduce process executes in task tracker and between both stages, map and reduce, there exist an intermediate phase. During the intermediate steps, some steps such as (shuffle and sorting) will be executed in the intermediate stage results are stored in local file system [24] which causes a lot of I/O operations overheads.

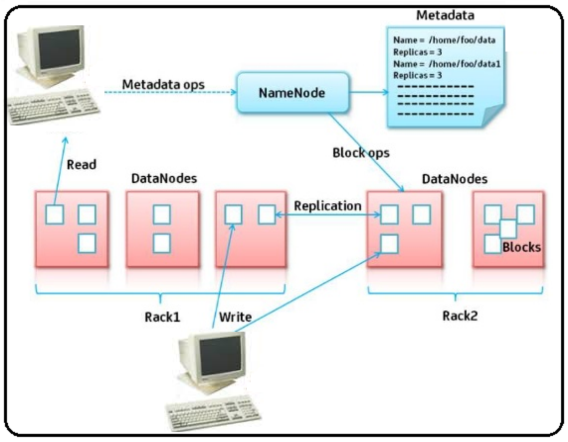


Fig. 2: HDFS Architecture

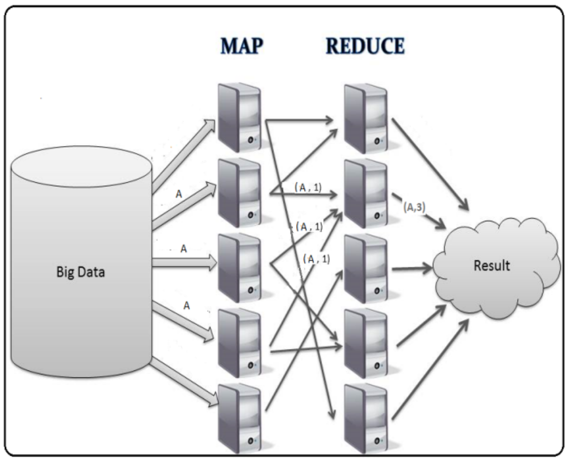


Fig. 3: Mapreduce Architecture

B. Spark

Apache Spark is an exponentially growing cluster computing technology which is dedicated for fast computation. Spark is the future paradigm for big data analytics, management and processing. It is an alternative to Hadoop Mapreduce and it is designed to resolve the disk I/O operations issue and enhance the overall performance. The main feature of Spark that makes it distinguishable is, it's able to perform inside-memory operations without disk I/O swapping. It permits the data memory caching and it eliminates the disk overhead limitation for repeated tasks. Spark can be viewed as an analytics general-purpose engine for processing of large scale data; it supports different programming languages such as Java, Scala, R and Python. From performance perceptive, it can achieve speedup of more than 100× over Hadoop MapReduce in certain cases that data fit in the memory [25].

Spark can run as an upper layer over (Hadoop Yarn Manager) and can read input data from HDFS which makes it adaptable to run on various platforms. It achieves the fault tolerance over a distributed file system which facilitates file sharing over memory. There are 3 methods to deploy spark:

- Standalone: Spark reserves the place on top of HDFS and space is allocated for HDFS. In that case, MapReduce and Spark will run side by side
- Hadoop Yarn: spark runs on Hadoop Yarn without any pre-installation or requiring for root access.
- Spark in MapReduce: this scheme is used to start and control spark job in addition to standalone deployment. In that case, client can launch Spark and exploits its shell without any administrative access.

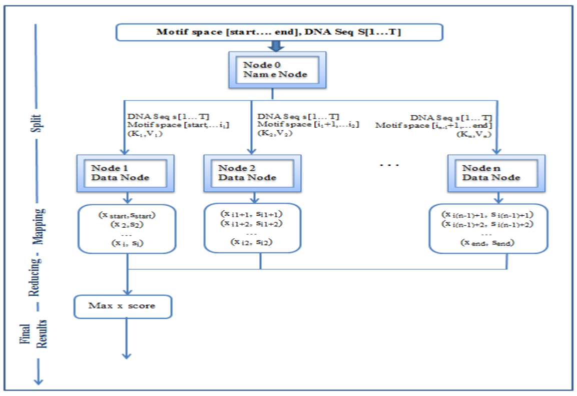
V. Proposed Motif Finding Acceleration on Hadoop Cluster

In this section a solution of motif finding acceleration on Hadoop cluster is proposed, Hadoop is a big data framework that provides a distributed data infrastructure; it is used to distribute a massive amount of data within a cluster. MapReduce is a big data processing tool enables big data processing and analytics; it is integrated within the Hadoop framework unlike Spark while it is a big data processing tool also but it needs to be integrated within file management system such as Hadoop.

Although that Spark isn't a built in tool within Hadoop but it generally a way faster than MapReduce, this comes from that fact that park operates on the whole dataset at once and do all the operations in memory and near real-time while Mapreduce operates in steps and it requires I/O operation with intermediates results. MapReduce is preferred with the mostly static operations but if we need to work with applications that required real-time processing requirements Spark will be more suitable

A. *Mapreduce Based Motif Finding Solution*

The proposed Mapreduce solution is illustrated in Fig. 4 including different operations: Split, Mapping, and Reducing over. The map operation receives the split data (the splitting is for Motif space). It measures the score for each motif in its assigned space. The output of this function is a list of pairs (key, Value) of (Motif, Score), this list computed on data nodes, Fig. 5. Finally, the reduce step computes the final output by comparing the output scores and retrieves the motif with the maximum score.


Motif space [start.... end], DNA Seq S[1...T]
Node 0 Name Node
DNA Seq s[1...T] Motif space [start,...i₁] (K₁,V₁)
DNA Seq s[1...T] Motif space [i₁+1,...i₂] (K₂,V₂)
DNA Seq s[1...T] Motif space [iₙ₋₁+1,...end] (Kₙ,Vₙ)
Node 1 Data Node
Node 2 Data Node
Node n Data Node
(x start, Sstart) (x 2, S2) ... (x i, Si)
(x i1+1, Si1+1) (x i1+2, Si1+2) ... (x i2, Si2)
(x i(n-1)+1, S i(n-1)+1) (x i(n-1)+2, S i(n-1)+2) ... (x end, Send)
Max x score
Split
Mapping
Reducing
Final Results
Figure 4: Proposed Mapreduce Architecture

```
Function Map (Motif Space MS[i..j], DNA
Sequence DS[a..b])
BEGIN
motif ← 0
MaxScore ← 0

FOR ALL motif (m) in MS [i→j]
        BEGIN
            score ← score (m,s)
            IF Score > Maxscore THEN
                motif ← m
                MaxScore ← score
            ENDIF
        END
return list(motif, MaxScore)
END
```

Figure 5: Map Function

B. *Spark Based Motif Finding Solution*

Another solution has been implemented using Spark which offers multiple privileges over Mapreduce:

- It's simpler to program and different programming languages can be used such as python, R and Scala meanwhile, Mapreduce is more complex and only java can be used to code.
- Spark runs in an interactive mode but Mapreduce works only for batch processing mode.
- Spark usually runs batch processing jobs even 10 to 100 times faster than MapReduce.
- Spark defines an abstraction called Resilient Distributed Datasets (RDD) which enriches Spark features meanwhile map reduce doesn't contain any abstraction
- Spark minimizes the latency by introducing results caching partial/complete across different nodes whereas MapReduce is completely dependent on disk-based.

Figure 6 illustrates the main spark function that solves MF problem; the implementation was coded using Scala.

The Spark implementation has several points to discuss:

First, it supports what's called (RDD Lineage) which is a dependency graph of the entire parents RDDs of a given RDD. It is built as a result of applying transformations to the RDD and proposes a logical parallel execution plan. It means that, spark gives the facility of operation abstraction and spark framework would pick the optimal execution plan and parallel implementation based on task and the available resources (Nodes).

Second, there are two types of Spark operations on RDD: Transformations and Actions. Transformation is an operation that generates new RDD from the already existing RDDs. When an action is triggered the execution plan starts to put in action to retrieve the result. In the proposed implementation there are 2 transformations used:

MaxScore ← ApplyMaxTransform(Score)
OutputMotif ← ApplyRetrieveTransform(MaxScore)
These transformations generate new RDDs from existing

RDDs. These transformations are built in analytical functions in Spark.

Third, the computation of motif scoring based on the equation mentioned before, is done using User-Define Function (UDF)

Score←ApplyScoreUDF(RDD)

Finally, the final operation is an action "Collect" which retrieves the final motif value and score to the master node.

VI. EXPERIMENTAL RESULTS

The experiments were conducted basically on a Hadoop cluster (Databricks cloud account) version (3.0.1) with the following specifications:

- HDFS capacity is 21.9 TB
- Ingestion per day (10 GB).
- The replication factor (3).
- 10 hosts (8 data nodes 2 master nodes)

The Apache Spark implementation is based on Spark ver 2.3.0 and Scala ver 2.11.

The main experiment that has been conducted is measure the speedup achieved by both by MapReduce and Spark, the baseline is the sequential implementation over a single core. The experiments have been conducted using different number of data nodes (2, 5 and 8). Table 1, Table 2 and Table 3 illustrates the speedup of both MapReduce and Spark, on different number of nodes

From Table 1, Table 2 and Table 3, Spark achieves much more speedup than Mapreduce. This finding is expected due to Mapreduce I/O operations overheads; it saves the intermediate results on nodes disks. Meanwhile Spark overcomes this issue by caching the data in nodes memories. An interesting note is that when data sizes increases, the speedup growth rate of mapreduce outperforms the growth rate of spark speedup (Not the speedup value) Fig 7, Fig 8 and Fig 9. These findings come from the fact that when dataset partition does not fit in nodes' memories, spark performance starts to degrade from the expected. From this experiment we conclude that Spark is much better than Mapreduce once the data partitions fit in memory of nodes. If data sizes starts to increase to the extent that partitions don't fit in memory, mapreduce for certain large scale sizes would outperform spark.

```
Main   Spark_MFP(Motif   Space   MS[i..j],   DNA
Sequence DS[a..b])
BEGIN
   motif ← 0
   MaxScore  ← 0
   RDD ← LoadParallelRDD()
   Score← ApplyScoreUDF(RDD)
   MaxScore ←ApplyMaxTransform(Score)
   OutputMotif ←
           ApplyRetrieveTransform(MaxScore)
   Output← CollectResult(OutputMotif)
 END
```

Fig. 6: Spark Implementation

TABLE 1: MAPREDUCE AND SPARK SPEEDUP VALUES FOR ACCELERATION OF MOTIF FINDING PROBLEM ON 2 CLUSTER NODES.

T*N Number of Sequences x Sequence Length	**MapReduce Speedup Value**	**Spark Speedup Value**
20*1200	12	88
24*1200	15	102
30*1200	25	143
40*1200	29	209
60*2400	42	253
80*2400	58	269
120*2400	72	276
160*2400	93	282
240*2400	131	293

TABLE 2: MAPREDUCE AND SPARK SPEEDUP VALUES FOR ACCELERATION OF MOTIF FINDING PROBLEM ON 5 CLUSTER NODES.

T*N Number of Sequences x Sequence Length	**MapReduce Speedup Value**	**Spark Speedup Value**
20*1200	21	101
24*1200	29	121
30*1200	35	155
40*1200	41	221
60*2400	49	259
80*2400	58	273
120*2400	73	288
160*2400	97	301
240*2400	143	305

TABLE 3: MAPREDUCE AND SPARK SPEEDUP VALUES FOR ACCELERATION OF MOTIF FINDING PROBLEM ON 8 CLUSTER NODES.

T*N Number of Sequences x Sequence Length	**MapReduce Speedup Value**	**Spark Speedup Value**
20*1200	38	165
24*1200	54	192
30*1200	62	201
40*1200	72	262
60*2400	98	304
80*2400	137	331
120*2400	175	359
160*2400	192	381
240*2400	243	400

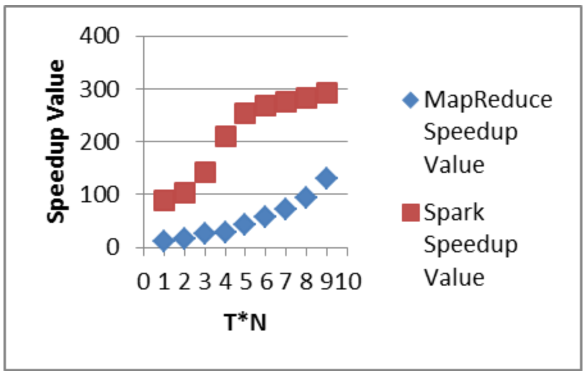


Fig. 7: MapReduce and Spark Speedup values for acceleration of motif finding problem on 2 cluster nodes.

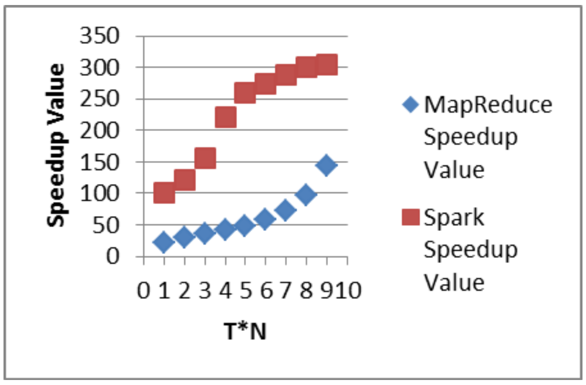


Fig. 8: MapReduce and Spark Speedup values for acceleration of motif finding problem on 5 cluster nodes.

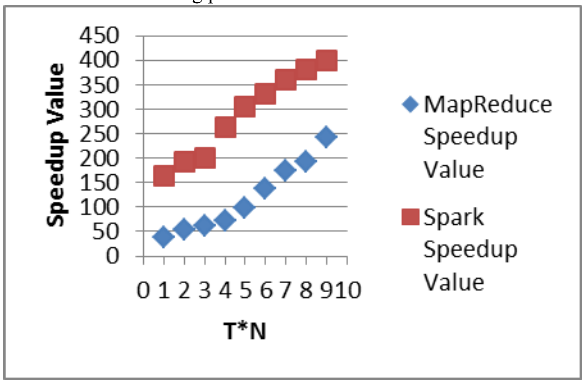


Fig. 9: MapReduce and Spark Speedup values for acceleration of motif finding problem on 8 cluster nodes.

VII. CONCLUSION

This work extends the previous work in [26] that exploited a multicore CPU based and GPU based methods to accelerate (MF) in DNA sequences using Skip-Brute Force algorithm. The main contribution of this work is presenting the bioinformatics problems solutions on a cloud based paradigms to maintain scalability of the solutions.

This work Implements different programming models for big-data for the purpose of accomplishing efficiency and maintaining the scalability. Apache Hadoop, MapReduce and Spark platforms and frameworks are exploited in migrating the intensive problem for being a big-data. From experiments Spark achieves much more speedup than Mapreduce. This finding is expected due to Mapreduce I/O operations overheads. An interesting note is that when data sizes increases, the speedup growth rate of mapreduce outperforms the growth rate of spark speedup (Not the speedup value) this result is due to the fact that when dataset partition does not fit in nodes' memories, spark performance starts to degrade from the expected. From this experiment we conclude that Spark is much better than Mapreduce once the data partitions fit in memory of nodes. If data sizes starts to increase to the extent that partitions don't fit in memory, mapreduce for certain large scale sizes would outperform spark.

REFERENCES

[1] N. Jones, P. Pevzner,\An Introduction to Bioinformatics Algorithms", Massachusetts Institute of Technology Press, 2004

[2] H.M.Faheem, "Accelerating Motif Finding Problem using Grid Computing with enhanced Brute Force", Advanced Communication Technology (ICACT),The 12th International Conference, vol. 1,2010,pp.197-210.

[3] Kuksa and Pavlovic: Efficient motif finding algorithms for large-alphabet inputs. BMC Bioinformatics 2010 (Suppl 8):S1.

[4] D. S. a. D. I. Ratne, "Use of a Probabilistic Motif Search to Identify Histidine Phosphotransfer Domain-Containing Proteins," PLOS one, pp. 1-18, 2016.

[5] H. M. Faheem, B. Koenig-Riez, Mahmoud Fayez, Iyad Katib, and N.AlJohani. "Solving the Motif Finding Problem on a Heterogeneous Cluster using CPUs, GPUs, and MIC Architectures", 2015. Mathematics and Computers in Sciences and Industry, CPS Published, pp. 226-232.

[6] H. M. Faheem and B.König-Ries, A New Scheduling Strategy for Solving Motif Finding Problem on Heterogeneous Architectures. International Journal of Computer Application 101 (5), 27-31

[7] S. Oehmen, "ScalaBLAST 2.0: rapid and robust BLAST calculations on multiprocessor systems," Oxford.

[8] N. Henrik, B. Karan, W. Kai and W. Zhong, "BioPig: a Hadoop-based analytic toolkit for large-scale sequence data," oxford, September 10, 2013.

[9] S. Mingming, Z. Xuehai and Y. Feng, "Bwasw-Cloud: Efficient sequence alignment algorithm for two big data with MapReduce," in Applications of Digital Information and Web Technologies (ICADIWT), 2014 Fifth International Conference, 2014.

[10] M.H.Padgavankar ,Dr.S.R.Gupta, Big Data Storage and Challenges, M.H.Padgavankar, (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (2) , 2014, 2218-2223

[11] Venketesh Palanisamy, Ramkumar Thirunavukarasu, Implications of big data analytics in developing healthcare frameworks – A review, Journal of King Saud University - Computer and Information Sciences, 2017.

[12] Ambika P R, Dr. K.N. Narasimha Murthy, Sowmya Naik PT, Aparna J S, Big Data: Towards Next Generation Analytics, International Journal of Innovative Research in Computer and Communication Engineering. Vol.3, Special Issue 5, May 2015.

[13] Sabia, Sheetal Kalra, Applications of big Data: Current Status and Future Scope, International Journal on Advanced Computer Theory and Engineering (IJACTE), , Volume -3, Issue -5, 2014, ISSN 2319-2526

[14] H. Karloff, S. Suri, and S. Vassilvitskii. A Model of Computation for MapReduce. In SODA '10: Symposium on Discrete Algorithms. ACM, January 2010.

[15] Ralf Lammel. Google's MapReduce programming model – Revisited. Science of Computer Programming, 70(1):1–30, January 2008.

[16] D. Rajasekar, C. Dhanamani, S. K. Sandhya, A Survey on Big Data Concepts and Tools

[17] R. Chaiken, B. Jenkins, P.˚A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: Easy and efficient parallel processing of massive

data sets. In Proceedings of the VLDB Endowment, volume 1, pages 1265–1276. VLDB Endowment, August 2008.

[18] Apache Software Foundation. Hadoop: A framework for running applications on large clusters built of commodity hardware, 2006

[19] A. Oussous, F.-Z. Benjelloun, A. A. Lahcen, S. Belfkih, "Big data technologies: A survey", Journal of King Saud University-Computer and Information Sciences, 2017.

[20] S. Chen and S.W. Schlosser. Map-Reduce Meets Wider Varieties of Applications. Technical Report IRP-TR-08-05, Intel Research, Pittsburgh, May 2008.

[21] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-Reduce for Machine Learning on Multicore. In Advances in Neural Information Processing Systems, volume 19, pages 281–288. MIT Press, 2007.

[22] S. Maitrey and C. K. Jha, “MapReduce: Simplified Data Analysis of Big Data,” Procedia Comput. Sci., vol. 57, pp. 563–571, Jan. 2015.

[23] “CodHoop: A system for optimizing big data processing: https://www.researchgate.net/publication/283123951_CodHoop_A_system_for_optimizing_big_data_processing.

[24] Suresh Lakavath, Ramlal Naik L, A Big Data Hadoop Architecture for Online Analysis, International Journal of Computer Science and Information Technology & Security (IJCSITS), Vol. 4, No.6, December 2014, ISSN: 2249-9555.

[25] M. U. Ali, S. Ahmad and J. Ferzund, "Harnessing the Potential of Machine Learning for Bioinformatics using Big Data Tools," International Journal of Computer Science and Information Security (IJCSIS), vol. 14, no. 10, pp. 668-675, 2016.

[26] M. M. Al-Qutt, H. Khaled, Rania ElGohary, H. M. Faheem, Iyad Katib, Nayif Al-Johani: Accelerating Motif Finding Problem Using Skip Brute-Force on CPUs and GPU’s Architectures. Int'l Conf. Par. and Dist. Proc. Tech. and Appl. 2017.