Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 126 (2018) 596–605

Procedia Computer Science

www.elsevier.com/locate/procedia

22nd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Spark-based data analytics of sequence motifs in large omics data

Oluwafemi A. Sarumi[a,b], Carson K. Leung[b,*], Adebayo O. Adetunmbi[a]

[a] *The Federal University of Technology – Akure (FUTA), Akure, PMB 704, Nigeria*
[b] *University of Manitoba, Winnipeg, MB, R3T 2N2, Canada*

Abstract

Data explosion in bioinformatics in recent years has led to new challenges for researchers to develop novel techniques to discover new knowledge from the avalanche of omics data (e.g., genomics, proteomics, transcriptomics). These data are embedded with a wealth of information including frequently repeated patterns (i.e., sequence motifs). In genomics, *deoxyribonucleic acid* (*DNA*) *sequence motifs* are short repeated contiguous frequent subsequences located in the prompter region. Due to the high volume and various degrees of veracity of these DNA datasets generated by the next-generation sequencing techniques, sequence motif mining from DNA sequences poised a major challenge in bioinformatics. In this article, we present a distributed sequential algorithm—which uses the MapReduce programming model on a cluster of homogeneous distributed-memory system running on an Apache Spark computing framework—for DNA sequence motif mining. Experimental results show the effectiveness of our algorithm in Spark-based data analytics of sequence motifs in large omics data.

Keywords: Bioinformatics; Spark; MapReduce; deoxyribonucleic acid (DNA); genomics; sequence motifs

1. Introduction

Advances in technology has led to the recent data explosion in bioinformatics and many other application domains. (e.g., biomedicine, smart world, social media, social networks, sensor and stream systems [1-4]). In bioinformatics, petabytes of omics data (e.g., genomics, proteomics, transcriptomics) are currently available to

* Corresponding author.
E-mail address: kleung@cs.umanitoba.ca

10.1016/j.procs.2018.07.294

researchers in many public repositories such as US National Centre for Biotechnology Information (NCBI)†, European Molecular Biology Laboratory - European Bioinformatics Institute (EMBL-EBI)‡, DNA Data Bank of Japan (DDBJ)§, and FlyBase**. In genomics, petabytes of DNA sequences are currently available with the advent of the next-generation sequencing (NGS) technologies [5]. Each DNA sequence is made of four chemical units called *nucleotide bases* [6], which are also known as *genetic alphabets*. These nucleotide bases are as follows:

- adenine (A),
- thymine (T),
- guanine (G), and
- cytosine (C).

Bases on the complementary base pairing in DNA, (i) adenine (A) is the complementary base of thymine (T), and (ii) guanine (G) is the complementary base of cytosine (C). The order of these nucleotide bases determines the meaning of the information encoded in that part of the DNA molecule. Embedded in these large volumes of DNA sequences are frequently recurring patterns called *sequence motifs* [7], which are also known as *transcription factor binding sites* (*TFBS*) [8].

These high volumes of a wide variety of valuable data with various degrees of veracity accentuate the need for data-intensive algorithms capable of handling such big data that usually do not fit into the main memory of a single machine. In developing these forms of data-intensive algorithms, the *MapReduce programming model* [9-11] has attracted a lot of attention from many researchers. MapReduce processes high volumes of big data by using parallel and distributed computing on large clusters of grids, nodes, or in clouds [12], which consist of a master node and multiple worker nodes. MapReduce is based on the functional programming concept [13] that consists of the map, reduce and combine functions. The operational style of MapReduce is also conceptually similar to the scatter and reduce functions in the Message Passing Interface (MPI) paradigm [14].

Apache Hadoop†† and Apache Spark‡‡ are examples of parallel frameworks that provide platforms for the implementation of MapReduce programming model. As Hadoop MapReduce [15] runs on the disk, it is usually slow and expensive due to the high I/O operations. In contrast, as Spark MapReduce [16] implementation is performed in memory of computing nodes on the cluster, it provides a faster and more cost effective way of implementing scalable parallel and distributed algorithms for big data analytics. In this article, we present an efficient distributed Spark-based algorithm for mining sequence motifs from large DNA sequence repositories. To evaluate the effectiveness of our algorithm for big data analytics, we apply our algorithm to the human genome and bacteria datasets. Our ***key contribution*** of this article is our scalable distributed frequent sequence mining algorithm that mimics the MapReduce programming model on an Apache Spark framework to mine sequence motifs from big DNA sequences in a timely manner.

The remainder of this article is organized as follows. The next section discusses related works, and Section 3 provides background information (e.g., MapReduce programming model, Apache Spark framework, DNA sequence motifs). We then describe our Spark memory-based distributed algorithm in Section 4. Experimental results and conclusions are given in Sections 5 and 6, respectively.

2. Related works

Several sequential pattern mining algorithms [17-21] have been proposed over two decades. For example, as AprioriAll and AprioriSome [17], as well as the Generalized Sequential Pattern (GSP) mining algorithm [22], are all extensions of the Apriori [23] algorithm (which was designed for frequent itemset mining), they rely on the candidate generation-and-test approach. Among them, AproriAll returns all the frequent sequences from a database,

† https://www.ncbi.nlm.nih.gov/
‡ https://www.ebi.ac.uk/
§ https://www.ddbj.nig.ac.jp/index-e.html
** http://flybase.org/
†† http://hadoop.apache.org/
‡‡ https://spark.apache.org/

whereas AprioriSome returns only maximal frequent sequences. GSP [22] incorporates taxonomy in sequential pattern mining. These three algorithms find frequent sequences horizontally. On the other hand, both the Sequential PAttern Discovery using Equivalence classes (SPADE) algorithm [24] and the Sequential PAttern Mining (SPAM) algorithm [25] find frequent sequences vertically. Between them, SPADE uses a breadth-first search methodology by utilizing some combinatorial properties to decompose the original problem into smaller sub-problems that can be independently solved in main-memory. In contrast, SPAM [25] uses a depth-first transversal approach with bitmaps to scan the search space with effective pruning mechanism. Observing that Apriori-based algorithms usually incur a huge set of candidate generation and multiple scan of database, tree-based algorithms mine frequent sequences from database without candidate generation by using trees to capture the content of the transaction databases. For example, FreeSpan [26] applies Frequent Pattern growth (FP-growth) algorithm [27] (which was designed for frequent itemset mining) to mine sequential patterns from sequential databases. It used of a projected sequence database to confine the search and growth of subsequence fragments. PrefixSpan [28] further reduces the size of the projected database generation and improves the performance of FreeSpan.

To speed up the mining process for frequent sequential patterns, parallel and distributed sequential pattern mining algorithms [29, 30] have also been proposed. Examples include the Hash Partitioned Sequential Pattern Mining (HPSPM) algorithm [31], which partitions the candidate sequences among the nodes on a shared-nothing parallel architecture using a hash function. To avoid processor idling and selective sampling, the Parallel CloSpan with Sampling (Par-CSP) algorithm [32] uses dynamic scheduling to address the load imbalance problem of the parallel framework. The Parallel SPADE (pSPADE) algorithm [33] discovers frequent sequences from large databases in parallel by decomposing the original search space into smaller suffix-based classes. Other parallel and distributed sequential pattern mining algorithms include the Parallel PrefixSpan with Sampling (Par-ASP) algorithm [34].

In this era of big data, researchers have proposed algorithms for mining sequential patterns from large genomic sequence databases. For instance, PSmile [35] extracts binding-site consensus from genomic sequences using a heuristic partitioning approach, but it suffers from highly imbalanced workload and parallel overhead. A Combinatorial Method for Extracting motifs (ACME) [36] uses a parallel combinatorial approach to extracts motifs from a single very long sequence, but it suffers huge communication cost due to message passing, data sharing and I/O operations. The Distributed GSP (DGSP) algorithm [37] was introduced as extension of GSP [22] for a MapReduce distributed environment. However, DGSP does not perform well due to repeated scans of the input data. The Multiple Expectation maximum for Motif Elicitation (MEME) algorithm [38] was proposed as a deterministic optimization algorithm for the discovery of motifs in DNA or protein sequences by using Gibbs sampling and expectation maximization (EM) techniques. A gap-constraint algorithm with MapReduce called MG-FSM [39] minds the gap with large-scale frequent sequence mining. MG-FSM partitions the input database in a way that each partition is allowed to be mined independently by using any existing frequent sequence mining algorithm. The Sequential PAttern Mining algorithm based on MapReduce model on the Cloud (SPAMC) [40] was proposed as a MapReduce extension of SPAM [25]. The IMRSPM algorithm [41] was proposed as an iterative MapReduce-based framework for uncertain sequential pattern mining. All the aforementioned MapReduce-based algorithms were implemented on Apache Hadoop framework. However, Apache Hadoop disk-based MapReduce algorithms may suffer from high disk I/Os. Hence, we propose in this article an Apache Spark in-memory-based algorithm for mining sequential patterns (i.e., sequence motifs) from DNA sequences.

3. Background

In this section, we provide some background information on (1) MapReduce programming model, (2) Apache Spark framework, and (3) DNA sequence motifs.

3.1. MapReduce programming model

Over the past decade, researchers have used a high-level programming model called *MapReduce* to process high volumes of big data by using parallel and distributed computing on large clusters or grids of nodes (i.e., commodity machines) or clouds, which consist of a master node and multiple worker nodes. The MapReduce programming model usually uses two key functions:

1. the map function, and
2. the reduce function.

With these functions, the input data are read and divided into several partitions (sub-problems), which are then assigned to different processors. Each processor executes the map function on each partition (i.e., each processor executes its sub-problem). The map function takes a pair of $\langle key_1, value_1 \rangle$ and returns a list of $\langle key_2, value_2 \rangle$ pairs as an intermediate result:

$$map: \langle key_1, value_1 \rangle \rightarrow \langle key_2, value_2 \rangle$$

where (i) key_1 and key_2 are keys in the same or different domains and (ii) $value_1$ and $value_2$ are the corresponding values in some domains. Afterwards, these pairs are shuffled and sorted. Each processor then executes the reduce function on both (i) a single key_2 from this intermediate result and (ii) the list of all values associated with key_2 in the intermediate result. In other words, each processor executes the reduce function on $\langle key_2, \text{list of } values_2 \rangle$ to "reduce"—by combining, aggregating, summarizing, filtering, or transforming—the list of values associated with a given key_2 (for all keys). As a result, a single (aggregated or summarized) $value_3$ is returned:

$$reduce: \langle key_2, \text{list of } value_2 \rangle \rightarrow value_3$$

where (i) key_2 is a key in some domains and (ii) $value_2$ and $value_3$ are the corresponding values in some domains.

An advantage of using the MapReduce model is that users only need to focus on specifying these map and reduce functions—without worrying about implementation details for (i) partitioning the input data, (ii) scheduling and executing the program across multiple machines, (iii) handling machine failures, (iv) straggler mitigation, or (v) managing inter-machine communication.

3.2. Apache Spark framework

Apache Spark recently becomes a popular parallel framework for processing high volumes of big data on a distributed system. In Spark, data-parallel programming is executed on a cluster of commodity computers that automatically provide locality-aware scheduling, fault tolerance, and load balancing. As the main abstraction in Spark, *resilient distributed dataset* (*RDD*) represents a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Users can explicitly cache an RDD in memory across machines and reuse it in multiple parallel operations. RDD achieves fault tolerance through a notion of lineage, i.e., if a partition of an RDD is lost, the RDD has sufficient information to rebuild the lost partition from other RDDs. In Spark, all jobs are expressed as either (i) creating a new RDD, (ii) transforming existing ones, or (iii) calling action operations on RDD to compute the result.

Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in the driver program on the master node. The driver program connects to one or more worker nodes through the cluster manager. The driver program defines various *transformations* and invokes consequence *actions* on worker nodes. The action either returns a value to the driver program or writes data to the external storage. After transforming the input data to RDD and assigning them to different memory partitions on the worker nodes, Spark then performs lazy evaluation on all the transformations applied to the dataset and evaluates them in each partition on the worker nodes when an action is invoked. Spark actions are executed through a set of stages, separated by distributed operations. These are made possible by the use of broadcast variables that allows programmers to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.

3.3. DNA sequence motifs

A *DNA sequence motif* [7, 42] is specifically a short distinctive sequence pattern shared by a number of related sequences. They are usually found in upstream of transcription sites of genes. The distinctiveness of a motif is mainly reflected in the over-representation of the motif pattern at certain locations in the related sequences. DNA motifs are also referred to as transcribing factor binding sites (TFBS), which are specific sites on a DNA sequence that are conserved by their short repeated frequent subsequences of about 6 to 20 nucleotides in length located in the promoter region of the sequence [43]. They are conjectured to be useful for drug design, genes expression control, understanding of genes functions, and classification of genes.

Given a set of DNA sequences, the motif finding problem seeks to identify the frequent contiguous sequences that are good candidates for being transcription factor binding sites. It is assumed that co-expression of genes frequently arises from transcriptional co-regulation. As co-regulated genes [43] are known to share some similarities in their regulatory mechanism (possibly at transcriptional level), their promoter regions might also contain some common motifs that are binding sites for transcriptional regulators.

4. Our Spark memory-based distributed algorithm

In this section, we present our Spark memory-based distributed algorithm for analytics of sequence motifs from large DNA sequence dataset. Given that $\sum$ = {A, C, G, T} is a set of finite DNA alphabets, a *DNA sequence* (denoted by **S**) over an alphabet is an ordered and finite list of alphabets from $\sum$. A *sequence motif* (denoted by **M**) is a contiguous subsequence of sequence **S** that satisfies the following user-specified parameters:

1. a subsequence length [L_{min}, L_{max}];
2. an uncertainty factor D, which is a measure of Hamming distance among the subsequences; and
3. a minimum support threshold δ.

4.1. Overview of our algorithm

Our algorithm first applies a *map function* to the input DNA sequence on the master node to (i) flatten the sequence, (ii) convert it to RDD, and (iii) split it into memory partitions across all the worker nodes on the cluster. The algorithm uses the subsequence length [L_{min}, L_{max}] and the uncertainty factor D to generate support counts for subsequences.

Then, our algorithm applies a *reduce function* to all the worker nodes in the cluster. Subsequences that have its associated support count greater than the user defined minimum support (δ) are retained on each worker nodes.

Afterwards, our algorithm applies a *combine function* to collect all the subsequences from each of the worker nodes and outputs them as a single motif on the driver node (i.e., master node).

4.2. The map function for our algorithm

To start the data analytic process of mining sequence motifs from DNA sequence, our algorithm first applies a *map function* to the input DNA sequence **S** on the master node to (i) flatten the sequence, (ii) convert it to RDD, and (iii) split it into memory partitions across all the worker nodes on the cluster:

$$\text{map: } \langle \text{DNA sequence } \mathbf{S} \rangle \rightarrow \langle \text{subsequence, support count} \rangle$$

The algorithm uses the subsequence length L and the uncertainty factor D to generate support counts for subsequences. A pseudocode for this map function is given in Algorithm 1.

Algorithm 1. Map function
Input: DNA sequence **S**, L_{min}, L_{max}, D
Output: Set of subsequence RDDs, support count
1: **set** $ST \leftarrow \emptyset$
2: **for each** subsequence α **do**
3: compute count(α, **S**) as its support count
4: **if** ($L_{min} \leq |\alpha| \leq L_{max}$) and (HammingDistance($s_1$, s_2) ≤ D) **then**
5: add ⟨α, count(α, **S**)⟩ to ST

Let us consider illustrative example in mining sequence motif from the DNA sequence as shown in Fig.2. Given the following input parameters:

1. a subsequence length $L_{min} = L_{max} = 4$,
2. an uncertainty distance $D \leq 2$, and
3. a minimum support threshold δ =3.

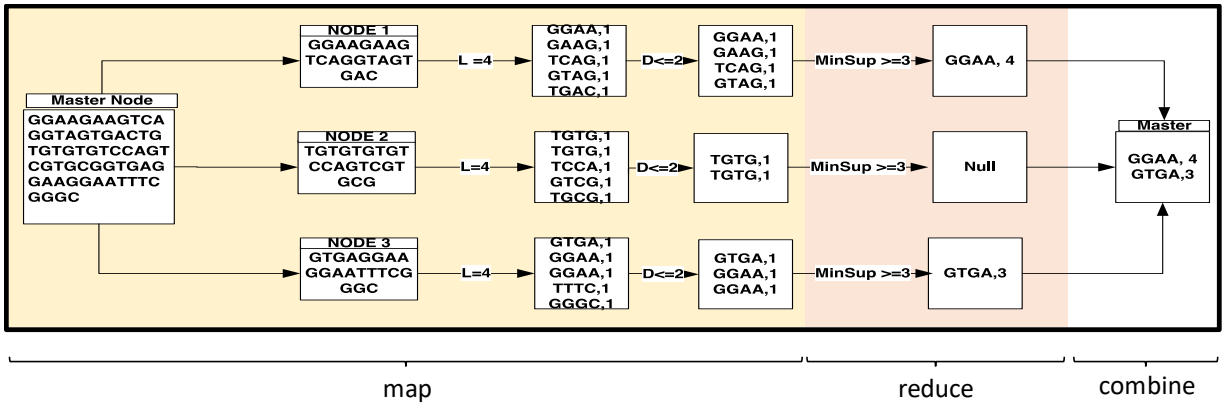

Fig. 1. Our data analytic process with the (a) map, (b) reduce, and (c) combine functions.

GGAAGAAGTCAGGTAGTGAC TGTGTGTGTCCAGTCGTGCG GTGAGGAAGGAATTTCGGGC

Fig. 2. A sample DNA input sequence.

Our algorithm applies the *map function* to the input DNA sequence **S** shown in Fig. 2 on the master node to (i) flatten the sequence, (ii) convert it to RDD, and (iii) it into memory partitions across all $N_p = 3$ worker nodes on the cluster. We make use of one master node and $N_p = 3$ worker nodes to illustrate this example as shown in Fig. 1. In real-life applications, the DNA sequence is usually longer, and the number N_p of workers is usually higher. In this example, Worker Node 1 obtains the partition GGAAGAAGTCAGGTAGTGAC, Worker Node 2 obtains TGTGTGTGTCCAGTCGTGCG, and Worker Node 3 obtains GTGAGGAAGGAATTTCGGGC. By applying the input parameters a subsequence length $L_{min} = L_{max} = 4$ and an uncertainty distance $D = 2$, Worker Node 1 then computes five subsequences GGAA, GAAG, TCAG, GTAG and TGAC, each of length $L_{min} = L_{max} = 4$. Among them, the worker selects four subsequences GGAA, GAAG, TCAG and GTAG, each satisfying a Hamming distance $D \leq 2$. Moreover, the worker also computes support counts for subsequences. Consequently, Worker Node 1 returns four subsequences GGAA:1, GAAG:1, TCAG:1 and GTAG:1. Worker Node 2 applies a similar procedure to generate two (identical) subsequences TGTG:1 and TGTG:1; Worker Node 3 applies a similar procedure to generate three subsequences GTGA:1, GGAA:1 and GGAA:1.

4.3. The reduce function for our algorithm

Once the subsequences are generated and their support counts are computed by each worker node, our algorithm then applies a reduce function to all the worker nodes in the cluster:

reduce: ⟨subsequence, support count⟩ → list of ⟨frequent subsequence, support count⟩

Subsequences that have its associated support count greater than the user defined minimum support δ are retained as shown in Fig. 1. Pseudocode for this reduce function is given in Algorithm 2.

Algorithm 2. Reduce function
1: Input: Subsequences and the support count, minsup δ
2: Output: All subsequences ≥ δ on each worker node
3: **for each** subsequence ∈ ⟨subsequence list⟩ emitted by map **do**
4: **set** counter[subsequence] ← 0; **set** list[subsequence] ← Ø
5: **for each** subsequence ∈ ⟨subsequence, 1⟩ emitted by map **do**
6: counter[subsequence] ← counter[subsequence] + 1; list[subsequence] ← list[subsequence] ∪ {subsequence}
7: **if** counter[subsequence] ≥ user-specified minsup δ **then**
8: **emit** ⟨subsequence, counter [subsequence]⟩

Let us continue with our illustrative example in mining sequence motif from the DNA sequence as shown in Fig.2. Recall that, with the map function, Worker Node 1 returns the following four subsequences (of length $L_{min} = L_{max} = 4$ each and satisfying Hamming distance $D \leq 2$) with their local support counts: GGAA:1, GAAG:1, TCAG:1 and GTAG:1. Similarly, Worker Node 2 returns TGTG:1 and TGTG:1; Worker Node 3 returns three subsequences GTGA:1, GGAA:1 and GGAA:1. These subsequences are then shuffled and sorted (e.g., {GAAG:1, GGAA:1, GGAA:1, GGAA:1, GTAG:1, GTGA:1, TCAG:1, TGTG:1, TGTG:1}). Afterwards, the reduce function groups similar subsequences and redistributes to N_p = 3worker nodes who compute the global support counts of similar subsequences by summing their local support counts. For instance, from {GAAG:1, GGAA:1, GGAA:1, GGAA:1}, Worker Node 1 computes frequent subsequence GGAA:4. Similarly, from {TCAG:1, TGTG:1, TGTG:1}), Worker Node 3 computes frequent subsequence TGTG:3. In contrast, from {GTAG:1, GTGA:1}), Worker Node 2 cannot find any frequent subsequence.

4.4. The combine function for our algorithm

Once the frequent subsequences are generated and their support counts are computed by each worker node, our algorithm then applies a combine function to (i) collect all these frequent subsequences from each of the worker nodes and (ii) output them as a single motif on the driver node (i.e., master node) as shown in Fig 2. Pseudocode for this combine function is given in Algorithm 3.

Algorithm 3. Combine function
1: Input: the set of emerging subsequences
2: Output: the set of motifs
3: **set** RS ← Ø
4: collect(⟨subsequence, support count⟩)
5: **for each** worker node on the cluster CL do
6: K ← align subsequence from CL_i
7: add subsequences in K to RS
8: **emit** RS

Let us continue with our illustrative example in mining sequence motif from the DNA sequence as shown in Fig.2. Recall that, with the reduce function, Worker Node 1 returns the following *frequent* subsequence (of length $L_{min} = L_{max} = 4$ each and satisfying Hamming distance $D \leq 2$) with its support counts: GGAA:4. Similarly, Worker Node 3 returns frequent subsequence TGTG:3. Our algorithm then applies a combine function to (i) collect all these two frequent subsequences from the worker nodes and (ii) return GGAA:4 TGTG:3 as sequence motifs to the user by the driver node (i.e., master node).

5. Experimental results

To evaluate our algorithm in mining sequence motifs from large DNA sequences, we used various datasets including the following:

1. bacteria DNA sequence datasets from the NCBI data repository (www.ncbi.org), which is of size 16 KB; and
2. human genome datasets (Homo Sapiens GRCh38.p7 DNA Chromosome 1, which is of size 240.8 MB; and

All the experiments were run using a cluster distributed-memory system. The cluster was configured using the standalone architecture of Apache Spark version 1.6.1 and made up of five homogeneous distributed-memory systems running Ubuntu-15.0 and 64-bit Linux operating system connected through a low-latency infini-band and gigabit Ethernet network. Each computer has Opteron core processor of 2.6 GHz, 8 dual core processor with Hyper-Threading, and 32 gigabytes of DDR3 memory. We implemented our algorithm in Scala programming language.

The results shown in Tables 1 and 2 are based on the standard performance metrics for distributed algorithms. These metrics include runtime, speedup, efficiency and cost. The distributed run time is the time that elapses from the moment that a distributed computation starts to the moment that the last processor finishes execution. The

speedup is the ratio of the serial runtime of the algorithm for solving a problem to the time taken by the distributed algorithm to solve the same problem on N_p processors as given in Equation (1):

$$S = T_s / T_p \tag{1}$$

where T_s and T_p are algorithm serial and distributed runtimes respectively.

The efficiency of a distributed algorithm is the ratio of speedup to the number of processors on the cluster as given in Equation (2). Efficiency measures the fraction of time for which a processor is usefully utilized:

$$\text{Efficiency} = (S / N_p) = (T_s / N_p T_p) \tag{2}$$

where N_p is the number of processors in the cluster.

The cost (C) of solving a problem on a distributed system is given as the product of (i) the runtime (T_p) and (ii) the number of processors (N_p) on the cluster as given in Equation (3):

$$C = T_p \times N_p \tag{3}$$

Table 1. Speedup, efficiency and cost our algorithm using the bacteria DNA sequence dataset.

Motif length (L)	Runtime on Single Node (T_s)	Runtime on Spark Cluster (T_p)	Speedup (S)	Efficiency	Cost (C)
2	222.0s	32.0s	6.936	1.387	160.0
3	306.0s	35.2s	8.693	1.739	176.0
4	340.7s	40.1s	8.496	1.699	200.5
5	432.0s	44.3s	9.751	1.950	221.5
6	510.0s	48.1s	10.625	2.125	240.5

Table 2. Speedup, efficiency and cost our algorithm using the human genome DNA dataset.

Motif length (L)	Runtime on Single Node (T_s)	Runtime on Spark Cluster (T_p)	Speedup (S)	Efficiency	Cost (C)
2	22,900.0s	4,500.1s	5.089	1.017	22,500.5
3	31,504.1s	6,300.8s	5.000	1.000	31,504.0
4	37,001.5	7,200.3s	5.139	1.027	36,001.5
5	49,700.2s	99,00.5s	5.019	1.004	49,502.5
6	55,100.2s	10,800.2s	5.101	1.020	54,001.0

Results from Table 1 shows that the execution time on single machine increases significantly with a growing trend with the increase of the difficulty of the problem. Thus, the execution time increases significantly as the size of the dataset increases. However, for the cluster, small size of dataset does not have significant change on the execution time. Tables 2 shows that, for a large input dataset, the speedup on the cluster is approximately five times the speed on the single node with varying motif lengths, thus we can significantly reduce the time required by our algorithm to extract motifs from the datasets by increasing the nodes on the cluster.

Furthermore, Fig. 3 shows the asymptotic worst case scenario time complexities analysis of our algorithm when it runs on a single node and on three nodes. It shows that the growth rate on a single node is $O(n^2)$ which follows quadratic growth rate function, while the growth rate on $N_P = 3$ three nodes is $O(\log_3 n)$ which follows a logarithmic order of computation. Hence, the time requirement for our algorithm increases slowly as the dataset size increases.

6. Conclusions

In the current epoch of big data, high volumes of a wide variety of valuable data can be easily collected and generated from a broad range of data sources of different veracities at a high velocity. In bioinformatics, terabytes of deoxyribonucleic acid (DNA) sequences can now be generated within a few hours with the use of next generation sequencing (NGS) technologies. These high volumes of data are beyond the ability traditionally used algorithms to

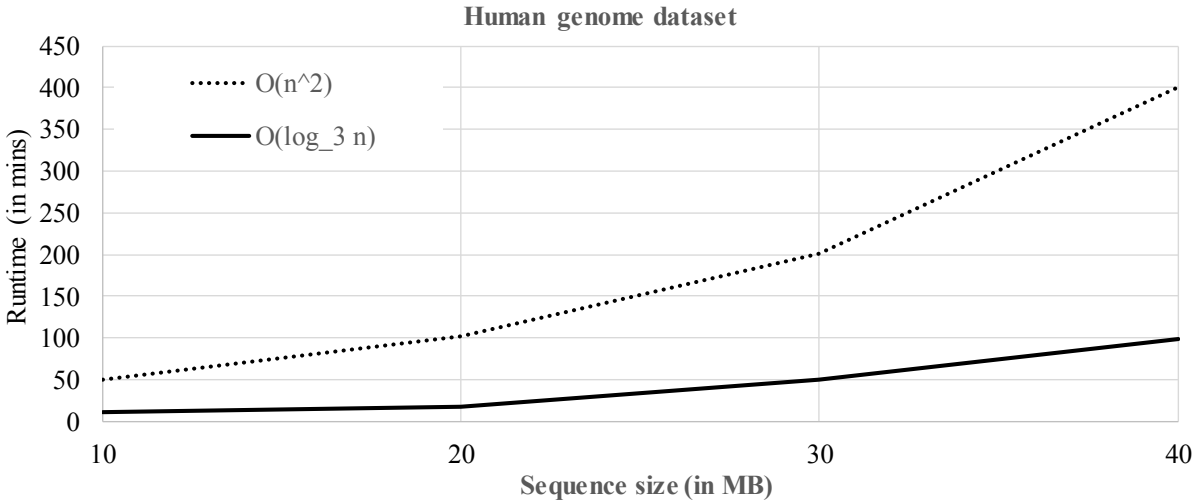

Fig. 3. Worst-case time complexity analysis with one and three nodes.

manage, query, and process within a tolerable elapsed time. In this article, we presented a big data analytics model for mining motifs from large volumes of DNA sequences. Specifically, we developed an Apache Spark-based algorithm for data analytics of sequence motifs (i.e., frequent sequences) in large omics data. This distributed algorithm mimics the MapReduce programing model in an Apache Spark parallel computing framework. Evaluation results show the speedup, efficiency and cost of our algorithm in mining motifs from large volumes of DNA datasets within a short time frame.

As ongoing and future work, we are exploring ways to further enhance our algorithm. Moreover, we are conducting more extensive evaluation.

Acknowledgements

This project is partially supported by (i) NSERC (Canada), (ii) TETFund (Nigeria) and (iii) University of Manitoba.

References

[1] Braun P, Cuzzocrea A, Doan LMV, Kim S, Leung CK, Matundan JFA, Singh RR. Enhanced prediction of user-preferred YouTube videos based on cleaned viewing pattern history. Procedia Computer Science 2017; 112: 2230-2239.

[2] Braun P, Cuzzocrea A, Keding TD, Leung CK, Pazdor AGM, Sayson D. Game data mining: clustering and visualization of online game data in cyber-physical worlds. Procedia Computer Science 2017; 112: 2259-2268.

[3] Braun P, Cuzzocrea A, Leung CK, Pazdor AGM, Tran K. Knowledge discovery from social graph data. Procedia Computer Science 2016; 96: 682-691.

[4] Leung CK, Tanbeer SK, Cuzzocrea A, Braun P, MacKinnon RK. Interactive mining of diverse social entities. KES Journal 2016; 20(2): 97-111.

[5] Koboldt DC, Steinberg KM, Larson DE, Wilson RK, Mardis E. The next-generation sequencing revolution and its impact on genomics. Cell. 2013;155(1): 27-38.

[6] Chaffey N, Alberts B, Johnson A, Lewis J, Raff M, Roberts K, Walter P. Molecular biology of the cell, 4th ed. Garland Science; 2002.

[7] D'haeseleer P. What are DNA motifs? Nature Biotechnology 2006; 4(24): 423-425.

[8] Stormo GD. DNA binding sites: representation and discovery. Bioinformatics 2000; 16(1): 16–23.

[9] Dean J, Ghemawat S. MapReduce: a flexible data processing tool. CACM 2010; 53(1): 72-77.

[10] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. CACM 2008; 51(1): 107–113.

[11] Lammel R. Google's MapReduce programming model revisited. Science of Computer Programming 2008; 70(1): 1–30.

[12] Sadashiv N, Kumar SM. Cluster grid and cloud computing: a detailed comparison. In: ICCSE 2011, pp. 477-482.

[13] Hughes J. Why functional programming matters. The Computer Journal 1989; 32(2):98–107.

[14] Snir M, Otto SW, Huss-Lederman S, Walker DW, Dongarra J. MPI - The Complete Reference, 2nd ed. MIT Press; 1998.

[15] White T. Hadoop: The Definitive Guide, 4th ed. O'Reilly; 2015.

[16] Zaharia M, Chowdhury M, Frankli M, Shenker S, Stoica I. Spark: cluster computing with working sets. In: USENIX HotCloud 2010.

[17] Agrawal R, Srikant R. Mining sequential patterns. In: IEEE ICDE 1995, pp. 3-14.

[18] Bernhard SD, Leung CK, Reimer VJ, Westlake J. Clickstream prediction using sequential stream mining techniques with Markov chains. In: IDEAS 2016, pp. 24-33.

[19] Jiang F, Leung CK, Sarumi OA, Zhang CY. Mining sequential patterns from uncertain big DNA data in the Spark framework. In: IEEE BIBM 2016, pp. 874-881.

[20] Rahman MM, Ahmed CF, Leung CK, Pazdor AGM. Frequent sequence mining with weight constraints in uncertain databases. In: ACM IMCOM 2018, art. 48.

[21] Shen W, Wang J, Han J. Sequential pattern mining. In: Frequent Pattern Mining, 2014; pp. 261-282.

[22] Srikant R, Agrawal R. Mining sequential patterns: generalizations and performance improvements. In: EDBT 1996, pp. 3–17.

[23] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases. In: VLDB 1994, pp. 487–499.

[24] Zaki MJ. SPADE: an efficient algorithm for mining frequent sequences. Machine Learning 2001; 42(1-2): 31-60.

[25] Ayres J, Flannick J, Gehrke J, Yiu T. Sequential pattern mining using a bitmap representation. In: ACM KDD 2002, pp. 429–435.

[26] Han J, Pei J, Mortazavi-Asl B, Chen Q, Dayal U, Hsu M. FreeSpan: Frequent pattern-projected sequential pattern mining. In: ACM KDD 2000, pp. 355–359.

[27] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: ACM SIGMOD 2000, pp. 1– 12.

[28] Pei J, Han J, Mortazavi-Asl B, Pinto H, Chen Q, Dayal U, Hsu M. PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In: IEEE ICDE 2001, pp. 215–224.

[29] Abastasiu DC, Iverson J, Smith S, Karypis G. Big data frequent pattern mining. In: Frequent Pattern Mining, 2014; pp. 225-259.

[30] Zaki MJ. Parallel and distributed association mining: a survey. IEEE Concurrency 1999; 7(4): 14-25.

[31] Shintani T, Kitsuregaw M. Mining algorithms for sequential patterns in parallel: hash based approach. In: PAKDD 1998, pp. 283–294.

[32] Cong S, Han J, Padua D. Parallel mining of closed sequential patterns. In: ACM KDD 2005, pp. 562–567.

[33] Zaki MJ. Parallel sequence mining on shared-memory machines. JPDC 2001; 61(3): 401–426.

[34] Cong S, Han J, Hoeflinger J, Padua D. A sampling-based framework for parallel data mining. In: ACM SIGPLAN PPoPP 2005, pp. 255–265.

[35] Carvalho M, Oliveira AL, Freitas AT, Sagot MF. A parallel algorithm for the extraction of structured motifs. In: ACM SAC 2004, pp. 147–153.

[36] Sahli M, Mansour E, Kalnis P. Parallel motif extraction from very long sequences. In: ACM CIKM 2013, pp. 549–558.

[37] Qiao S, Tang C, Dai S, Zhu M, Peng J, Li H, Ku Y. PartSpan: parallel sequence mining of trajectory patterns. In: FSKD 2008, pp. 363– 36.

[38] Bailey TL, Boden M, Whitington T, Machanick P. The value of position-specific priors in motif discovery using MEME. BMC Bioinformatics 2010; 11: art. 179.

[39] Miliaraki K, Berberich R, Gemulla R, Zoupanos S. Mind the gap: large-scale frequent sequence mining. In: ACM SIGMOD 2013, pp. 797–808.

[40] Chen MC, Tseng C. Highly scalable sequential pattern mining based on MapReduce model on the cloud. In: IEEE BigData Congress 2013, pp. 310–317.

[41] Ge J, Xia Y, Wang J. Mining uncertain sequential patterns in iterative MapReduce. In: PAKDD 2015, Part II, pp. 243–254.

[42] Sami A, Nagatomi R. A new definition and look at DNA motif. In: Data Mining in Medical and biological research, 2008; pp. 227-236.

[43] Das MK, Dai H. A survey of DNA motif finding algorithms. BMC Bioinformatics 2007; 8(Suppl 7): S21.