Genomic repeats detection using Boyer-Moore algorithm on Apache Spark Streaming

Lala Septem Riza[1], Farhan Dhiyaa Pratama[2], Erna Piantari[3], Mahmoud Fahsi[4]
[1,2,3]Department of Computer Science Education, Universitas Pendidikan Indonesia, Indonesia
[4]EEDIS Laboratory, Djillali Liabes University, Algeria

Article Info

Article history:

Received Jul 24, 2019
Revised Jan 2, 2020
Accepted Feb 6, 2020

Keywords:

Apache Spark Streaming
DNA
Genomic repeats
Human genom
String matching

ABSTRACT

Genomic repeats, i.e., pattern searching in the string processing process to find repeated base pairs in the order of deoxyribonucleic acid (DNA), requires a long processing time. This research builds a big-data computational model to look for patterns in strings by modifying and implementing the Boyer-Moore algorithm on Apache Spark Streaming for human DNA sequences from the ensemble site. Moreover, we perform some experiments on cloud computing by varying different specifications of computer clusters with involving datasets of human DNA sequences. The results obtained show that the proposed computational model on Apache Spark Streaming is faster than standalone computing and parallel computing with multicore. Therefore, it can be stated that the main contribution in this research, which is to develop a computational model for reducing the computational costs, has been achieved.

Corresponding Author:

Lala Septem Riza,
Department of Computer Science Education,
Universitas Pendidikan Indonesia, Indonesia.
Email: lala.s.riza@upi.edu

1. INTRODUCTION

DNA repeats in the eukaryotic genome [1]. Repetition identification and classification are important fundamental annotation tasks for several reasons. First, repetition is believed to play an important role in the evolution of genomes and diseases [2]. Second, cellular elements (transposons and retrotransposons) may contain coding regions that are difficult to distinguish from other gene types. Finally, repetition often causes a lot of local alignments, complicated sequence assembly, comparison between genomes and large-scale duplication analysis and rearrangement [3]. In the past decade scientists have been doing laboratory research for 3 years to analyze DNA [4]. One of the cases of DNA analysis that requires time and energy on a large scale is to analyze diseases caused by repetitive genomic patterns or called genomic repeats [3] like three repeated base pairs that can cause diseases in the trinucleotide repeat disorders category [5].

A task of genomic repeats, which basically is an analysis of string matching or pattern matching, is carried out to look for a pattern in a large text. The basic algorithm for searching strings or patterns is by matching all the possibilities contained in the data from the first index in the text to the end of sequences. The Brute Force (Naïve) Algorithm has the worst possible complexity, which is O (mn), where it will be very time consuming if more and more text will be used as objects to search for strings or patterns [6]. So, the need to reduce the computational cost while performing string matchin on large datasets makes scientists to develop algorithms that are more efficient than brute force algorithms, such as the algoirithms of Knuth Morris Pratt (KMP) [7] and Boyer Moore (BM) [8].

The KMP algorithm is a string matching algorithm that works by utilizing the shift pattern of the text from the left to the right during matching strings in the text. The KMP algorithm was firstly developed by Donald E. Knuth in 1967 and then continued by James H. Morris with Vaughan R. Pratt in 1966. Then in 1977 the KMP algorithm was published. Then, the BM Algorithm, which is one of the most efficient algorithms compared to other string matching algorithms, was proposed by creating two tables known as the BM Bad Character (bmBc) table and the BM good-suffix (bmGs) table [9]. For each character in the alphabet set, bad character tables store shift values based on the appearance of characters in the pattern. This algorithm forms the basis for several pattern matching algorithms.

The KMP and BM algorithm can be used as tools for identical identical sequences in source sequences or repeated subsequence searches [10]. The BM type algorithm for matching compressed patterns in a system collage, and shows that an instance of algorithm search in BPE (byte per encoding) is compressed text 1.2~3.0 faster than the agrep software package (fastest pattern mathing tool) in the original text [11]. Moreover, nowadays, scientists or biologists have begun to face issues related to large datasets. So, challenges on handling, processing, and transferring information are appeared. It means that biologists need to change from traditional data processing to more towards big data analysis to investigate all biological problems. For example, modifying the KMP in parallel computing with multicore by using the R package pbdMPI has been introduced by Riza et al. for searching genomic repeats [12]. The R package pbdMPI was also utilized for parallel random projection in order to handle planted motif search [13].

In other side, some developments in open-source software, namely the Hadoop project, made discoveries to provide scalable storages (e.g., in petabytes of data) in hadoop distributed file systems (HDFS) that combines with the programming model, called MapReduce [14]. However, because of the Hadoop-based I/O access pattern, the intermediate calculation results are not cached. Therefore, Hadoop is only suitable for batch data processing, and shows poor performance for repetitive data processing [15]. To overcome this problem, Apache Spark was found, which is a faster platform designed to handle large amounts of data [16].

Apache Spark is an open-source cluster computing framework for large data processing. It has emerged as a next-generation large data processing engine, overtaking Hadoop MapReduce which helped revive the Big Data revolution. It maintains linear scalability and fault tolerance of MapReduce, but extends it in several important ways. Unlike Apache Hadoop as disk-based computing, Apache Spark does memory computing by introducing a powerful concept, i.e., resilient distributed dataset (RDD). Because it is possible to store results in memory, it is more efficient for repetitive operations. In terms of performance, Apache Spark can reach 100 times faster in terms of memory access than Apache Hadoop [16]. The gap between Apache Spark and Apache Hadoop is more than 10 times greater, even if we compare between the two based on disk performance [17, 18]. In terms of flexibility, Apache Spark provides a high-level application programming interface (API) in Java, Scala, Python, and R. In general terms, Apache Spark provides structured data processing, machine learning, graph computing, and flow computing capabilities by supporting several sophisticated components.

In contrast to batch-based large volume data processing, streaming processing takes a more advanced step towards data streaming. With exponential growth in continuous data streaming, it has gained a lot of popularity. Apache Spark Streaming is one of the open source frameworks for reliable streaming processing, high-throughput, and low latency streaming processing [19]. It is an extension of API of Apache Spark, which is intended to process streaming data streams. Although Apache Spark is a batch processing engine, with Spark Streaming it is able to process streaming data from various sources including from Twitter. Here the incoming data flow is divided into small groups which are then processed by the Spark engine. In Apache Spark Streaming, discretization flows (DStreams) which are RDD sequences, represent continuous data streams. The operations on DStreams are converted to basic RDD transformations which are then calculated by the Spark engine [20].

Moreover, Apache Spark Streaming, Apache Storm, and Yahoo! S4 [21] are three typical platforms that support the streaming calculation model for direct data processing. Apache Storm is a free and open source distributed real-time calculation system. Apache Storm makes it easy to process data flow without limits reliably. Unlike Apache Storm, Apache Spark Streaming takes a very different approach and processes events in batches. Most traditional flow processing systems are designed to process records one by one. This is known as a continuous operator model, a simple model that works very well on a small scale, but faces several challenges with large scale analysis and real time. To overcome these challenges, Apache Spark Streaming uses micro batch architecture [22-24] where the data stream is treated as a small batch of data and streaming computing is done through a series of continuous batch computations on this batch of data. Therefore, this research is aimed at building computational models and implementing the BM algorithm in finding string patterns in human chromosome genome data contained in ensemble pages. This study consisted of three stages, namely the stage of entering data into the system, the BM processing, and the stage of analyzing the results.

2. RESEARCH METHOD

The computational model built in this research can be seen in Figure 1. There are several stages in the computational model of detection of genomic repeats using the BM algorithm on Apache Spark Streaming. It starts with the preprocessing stage, then continues with the input data stage and uploads the data. Then, the data is processed by moving the uploaded data into the streaming folder in hadoop distributed file system (HDFS) which is then captured by the system from Apache Spark Streaming and processed by the BM algorithm. After processing is complete, the resulting data from processing can be downloaded into a personal computer.

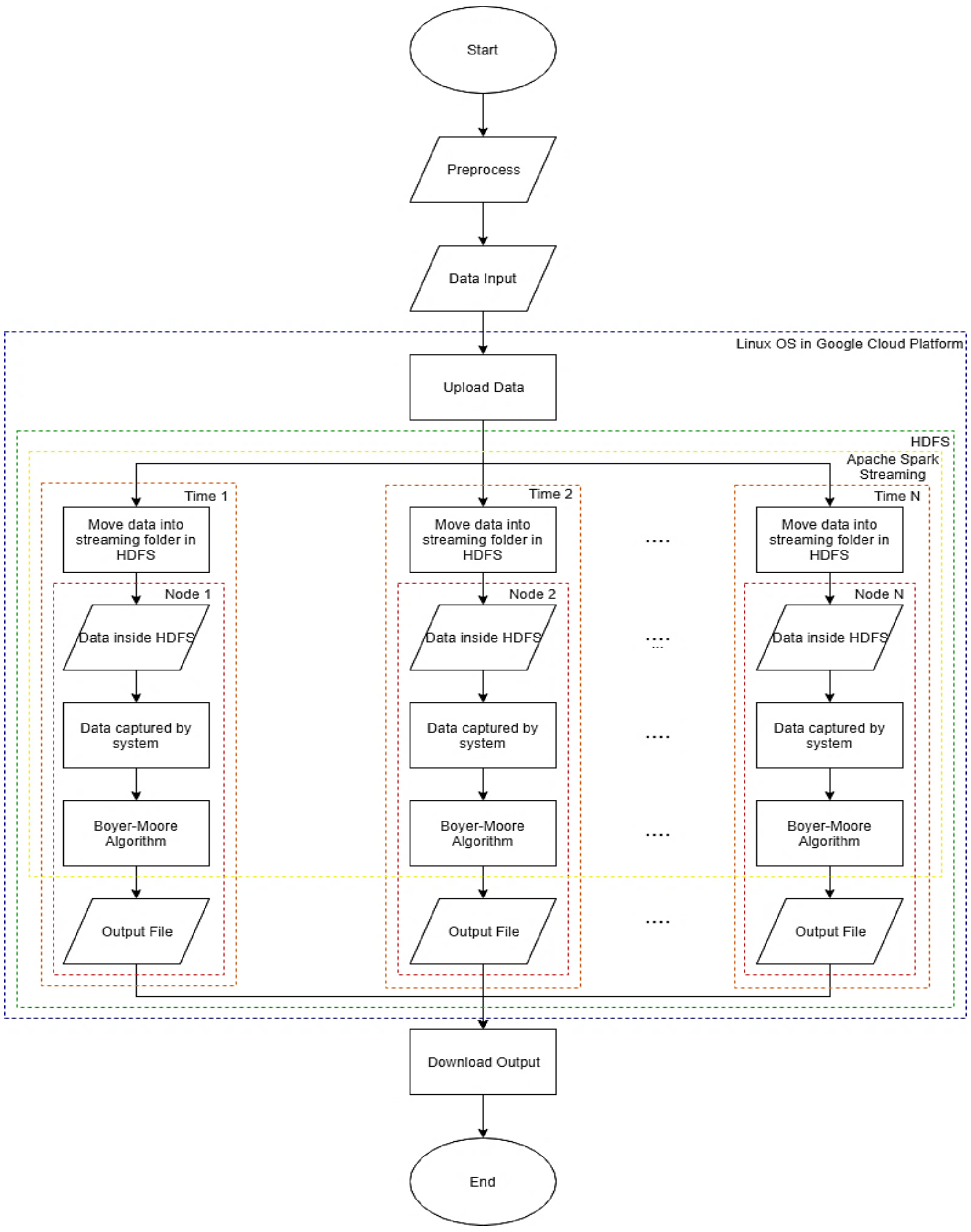

Figure 1. Research computational model

Based on the environment where the computational model works, we divide into four system environments, as follows:

– Working in personal computers: the model starts in the environment of the personal computer where the data is located, then goes into the preprocessing stage that is carried out in the personal computer environment. The environment changes when the process has entered the data upload stage where data is transferred from the environment of the personal computer into the virtual machine in the Google Cloud Project.

- Working on virtual machines in Google Cloud Project: Then the data should be uploaded into the cloud computing, which is Google Cloud Project. In this step, we set some certain parameters such as numbers of cores, numbers of worker nodes.
- Working on HDFS: The large datasets are then copied to HDFS streamingly for each block to determined folder in HDFS.
- Working with Apache Spark Streaming: After obtaining a block of data, the modification of the BM algorithm on the Apache Spark Streaming runs. This work is running until there is no a block in the HDFS folder. It should be noted that the tasks runs along with all worker nodes.

According to Figure 2, the first step in this algorithm is to call the package needed in program code such as SparkContext, SparkConf, and StreamingContext and initialize the package so that it can be used in the program code.

```
     Input: Jupyter Notebook Accessed
     Output:  Number of Genomic repeats in 1 data, Number of pattern repeats in 1
     data, Process time


     Algorithm:
1    Call packages needed in the system
2    Initializes spark configuration and spark context
3    Initializes the spark streaming context
4    Specifies the folder used as the streaming folder
5    Transforming from the incoming data as a form of Dstream data into RDD
6    Repartition the data entered into one partition and zipped with index
7    Call the pattern from the folder in hdfs
8    Take action against the pattern
9    Set the time for data processing to start
10   Transform the incoming data and pattern by entering it into the Boyer-Moore
         algorithm function
11   Transform the data that has been processed to be filtered if there are no results
         in one RDD
12   Transform the data that has been processed to be sorted by the number of genomic
         repeats in an ascending RDD
13   Reducing the start time of processing data with real time
14   Specifying the directory contained in hdfs to be used as a destination for
         storing results
15   Perform an action to save the results of the number of genomic repeats into hdfs
         and display the number of pattern repeats in one data and time or duration of
         the process
```

Figure 2. Pseudo codeo of the BM algorithm on apache spark streaming

Then, we set the directory in HDFS which will be used as a streaming folder, so that every block of data entered into the folder after the program starts will be processed immediately. To process the blocks of data, it is necessary to transform them as a Dstream data type in Apache Spark Streaming. It should be noted that Dstream data types cannot work with RDD data types so a transformation is needed, which is to change Dstream data types to RDD data types. The code shows the processes as illustrated in Figure 3.

```
def main() :
    #read input text file to RDD
    dataDstream = sc.textFileStream("hdfs://cluster:8020/user/data").repartition(1)

    #transform to saveResult function
    dataDstream_save = dataDstream.transform(saveResult)

    #save the result to hdfs
    dataDstream_save.saveAsTextFiles("hdfs://cluster6:8020/user/result/resultTTAGGG")

    #show total pattern and processing time
    dataDstream_print = dataDstream.foreachRDD(printResult)
```

Figure 3. Code for inputting the blocks of data in apache spark streaming

After transforming the streaming data, the next step is to repartition the incoming data and provide an index into each RDD in the data. Data entered into the streaming folder will be automatically partitioned by the Apache Spark Streaming so that if it is not partitioned the number of partitions in one data will be determined automatically by Apache Spark Streaming. Repartitioning will be useful in finding genomic repeats so that we could obtain the right number of genomic repeats.

In this model, a data pattern is selected which is converted into a text data type because the data pattern must have a data size smaller than the incoming chromosome data, so to specify the process time, the data pattern is selected which is converted into a data text type. When the search process for genomic repeats using the Boyer-Moore function has been completed, the next step is to transform the results of processing by removing the results from one RDD that does not find genomic repeats and sequencing the results of ascending genomic repeats or finding genomic repeats. numbering at least up to the most number. This is done to make it easier to see or record the number of genomic repeats that occur. At this stage the search process has ended so it is necessary to do the timing of the process by reducing the real time time with the start of the process that has been initialized in the previous stage. The code showing main processes and saving results can be seen in Figure 4.

```
def main() :
    #read input text file to RDD
    dataDstream = sc.textFileStream("hdfs://cluster:8020/user/data").repartition(1)

    #transform to saveResult function
    dataDstream_save = dataDstream.transform(saveResult)

    #save the result to hdfs
    dataDstream_save.saveAsTextFiles("hdfs://cluster6:8020/user/result/resultTTAGGG")

    #show total pattern and processing time
    dataDstream_print = dataDstream.foreachRDD(printResult)
```

Figure 4. Code to perform the main processes and saving the results in apache spark streaming

3. RESULTS AND DISCUSSION

In this section, it is explained the results of research and at the same time is given the comprehensive discussion.

3.1. Data collection

The data used in this study are human DNA sequences which can be downloaded freely on page ftp://ftp.ensembl.org/pub/release-95/fasta/homo_sapiens/dna/. These data are examples of human DNA sequences in publication number 95 provided on the ensembl file transfer protocol (FTP) website. On that page there are 24 chromosome DNA sequence files which can be seen in Table 1.

3.2. Experimental scenario

In this experiment, we performed an experimental scenario using several worker nodes with each node having 4 CPU cores. The data used in this study uses all the files mentioned in section 3.1 Data Collection multiplied by the number of experiments carried out with the number of 576 files and the number of siz 75360 MB. The pattern that will be used is 'CCG', a pattern which if found repeatedly as much as 200-900 times, it can be concluded that the human has Fagile XE Syndrome, which normally the pattern 'CGG' only repeats 4-39 times. Then, we also use the ‘CAG’ pattern, which is the cause of the disease including the polyglutamine category. The difference in the 'CCG' and 'CAG' patterns does not only lie in the difference of one character in the middle, but also has a difference in the prefix that is generated. The prefix for 'CCG' is '0 0', while the prefix for 'CAG' is '0 0 0'. We need to know the difference in speed caused by these prefix. Then, there is the 'TTAGGG' pattern, which is a telomere or the very tip of linear DNA. “The search for 'TTAGGG' is intended to see the effect of pattern length on differences in computational speed. In addition, the selection of 'TTAGGG' is also because it is confirmed to exist in every human DNA sequence [12]. Table 2 shows the experiment will use different worker nodes and cores on each of the same nodes on the Google Cloud Platform. In other experimental scenarios the master will not do computing.

Table 1. Data used in experiments

Files Name	File Size (KB)
Homo_sapiens.GRCh38.chromosome.1.fa	253.105
Homo_sapiens.GRCh38.chromosome.2.fa	246.230
Homo_sapiens.GRCh38.chromosome.3.fa	201.600
Homo_sapiens.GRCh38.chromosome.4.fa	193.384
Homo_sapiens.GRCh38.chromosome.5.fa	184.563
Homo_sapiens.GRCh38.chromosome.6.fa	173.652
Homo_sapiens.GRCh38.chromosome.7.fa	162.001
Homo_sapiens.GRCh38.chromosome.8.fa	147.557
Homo_sapiens.GRCh38.chromosome.9.fa	140.701
Homo_sapiens.GRCh38.chromosome.10.fa	136.027
Homo_sapiens.GRCh38.chromosome.11.fa	137.338
Homo_sapiens.GRCh38.chromosome.12.fa	135.496
Homo_sapiens.GRCh38.chromosome.13.fa	116.270
Homo_sapiens.GRCh38.chromosome.14.fa	108.827
Homo_sapiens.GRCh38.chromosome.15.fa	103.691
Homo_sapiens.GRCh38.chromosome.16.fa	91.884
Homo_sapiens.GRCh38.chromosome.17.fa	84.645
Homo_sapiens.GRCh38.chromosome.18.fa	81.712
Homo_sapiens.GRCh38.chromosome.19.fa	59.594
Homo_sapiens.GRCh38.chromosome.20.fa	65.518
Homo_sapiens.GRCh38.chromosome.21.fa	47.488
Homo_sapiens.GRCh38.chromosome.22.fa	51.665
Homo_sapiens.GRCh38.chromosome.X.fa	158.641
Homo_sapiens.GRCh38.chromosome.Y.fa	58.181
Total	**3.139.770**

Table 2. Experimental scenario

No	Master Nodes	Worker Nodes	Each Master Core	Each Worker Core
1	1	2	4	4
2	1	4	4	4
3	1	5	4	4
4	1	11	4	4

3.3. Result and analysis of experiments

Based on the scenario designed in the previous section, Table 3 shows the results of the experiment with the scenario. In Table 3, there are 7 columns, namely pattern, file, number of worker nodes, total pattern, genomic repeats, index pattern repeats, and time cost. Pattern column is the pattern that we take a look for in sequences. After we conducted the experiment, we made a comparison of the output of the system that we built using a number of nodes which were the variables of this study in influencing the speed of processing finding patterns. Apart from the number of nodes that we varied, we did not make changes to other variables such as the number of cores on the master or worker nodes and the number of nodes from the master of this experiment. In Figures 5 and 6, we present the comparison of the results of the experiment using 4 cores from each master and the worker nodes carried out in this experiment with CCG and CAG patterns on chromosome 1.

Although the number of worker nodes influences the speed of the computational process, this does not have a significant effect. Can be seen from the computational speed of 2, 4, and 5 Nodes where the difference in speed of data processing only experiences differences per few seconds. The most significant computational differences occur when the number of worker nodes is added by 11 Nodes. In Figure 6 it is increasingly evident that not only is the speed difference inconsistent, even with the increasing number of worker nodes making computing in the search for CAG patterns on chromosome 1 slow. This is because the CAG pattern is the most found pattern among the patterns carried out in the experiment so that the time cost obtained will be longer or higher when compared to other patterns. With the number of patterns obtained in the computational process, the effectiveness of the number of worker nodes is needed. It is evident from the histogram on the number of 4 worker nodes with the number of 5 worker nodes actually increasing that 4 worker nodes are more effective in performing search computing the CAG pattern is compared to 5 worker nodes. However, if the number of worker nodes is too far away, such as using 4 worker nodes with the use of 11 worker nodes, the time cost obtained decreases significantly.

Table 3. Experimental result

Pattern	File	Number of Worker Nodes	Total Pattern	Genomic repeats	Index Pattern Repeats	Time Cost (Seconds)
CCG	Chromosome 1	2	647.388	12	(127633684, 127633687, 127633690, 127633693, 127633696, 127633699, 127633702, 127633705, 127633708, 127633711, 127633714, 127633717)	79,50
CCG	Chromosome 2	2	571.882	13	(210171362, 210171365, 210171368, 210171371, 210171374, 210171377, 210171380, 210171383, 210171386, 210171389, 210171392, 210171395, 210171398)	72,24
CCG	Chromosome 3	2	423.073	9	(129605344, 129605347, 129605350, 129605353, 129605356, 129605359, 129605362, 129605365, 129605368)	59,87
CCG	Chromosome 4	2	374.004	7	(3075005, 3075008, 3075011, 3075014, 3075017, 3075020, 3075023)	56,62
					(2059458, 2059461, 2059464, 2059467, 2059470, 2059473, 2059476)	
					(576301, 576304, 576307, 576310, 576313, 576316, 576319)	
...	...	...	...	...	...	...
TTAGGG	Chromosome Y	11	4.518	3	(24009196, 24009202, 24009208)	9,54

Moreover, we also made a comparison with the previous research [12] as illustrated in Figure 7. It can be seen the proposed model involving data streaming on Apache Spark Streaming is faster than the computational model on parallel computing with multicore conducted in the previous research [12]. Even though we have been faster than the previous research [12], the proposed computational model has a drawback, i.e., the number of genomic repeats could not be accurate. It happens when the match patterns found are on the spliting locations. In term of the comparison in methods utilized via parallel computing, the previous research also conducted data streaming by using R package and Sequential K-Means for determining trending topics in Twitter [25]. Other following methods in machine learning can also be used in data streaming for genomic repeats are Naive Bayes [26], various intelligent classifiers [27], and bootstrap method [28].

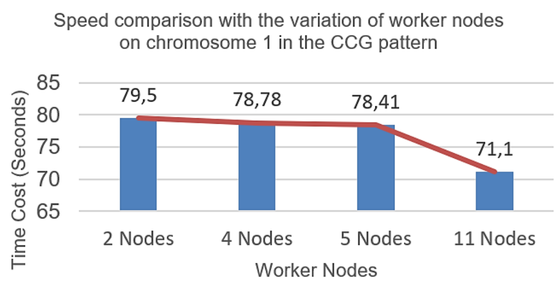

Figure 5. Speed comparison with the variation of worker nodes on chromosome 1 in the CCG pattern

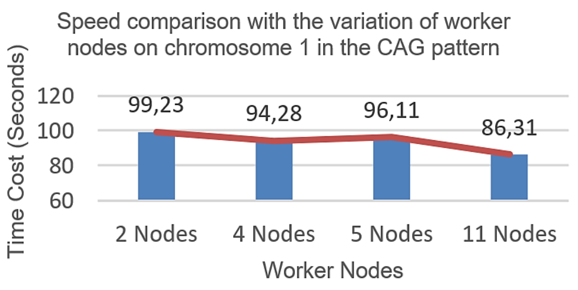

Figure 6. Speed comparison with the variation of worker nodes on chromosome 1 in the CAG pattern

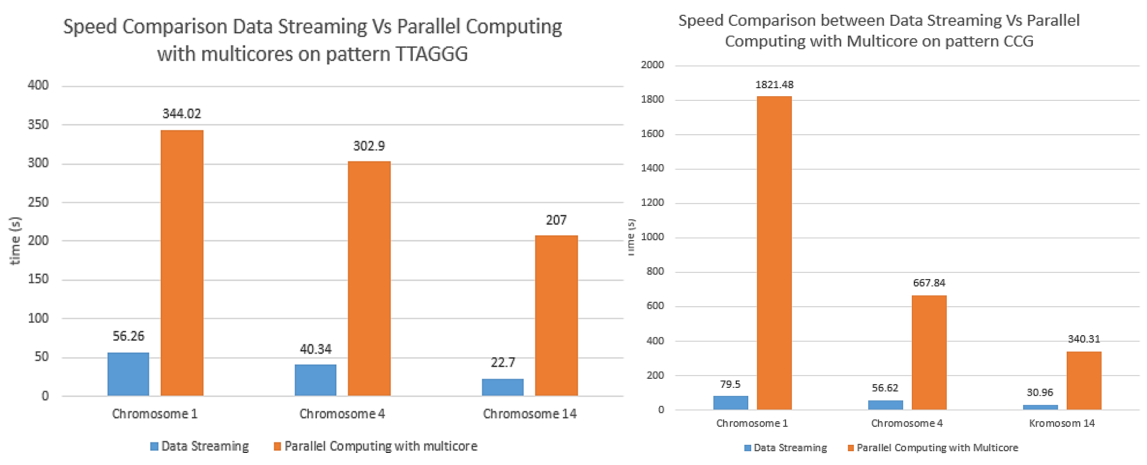

Figure 7. Speed comparison with the previous research [12]

4. CONCLUSION

The main contribution of this research is (i) to provide computational models for big data in detecting genomic repeats using the Boyer-Moore algorithm with Apache Spark Streaming. This model contains several stages, such as preprocess models, input data, Boyer-Moore algorithm systems, and download outputs; (ii) to conduct several experiments by varying the number of worker nodes used. Based on the results obtained, we can state that the proposed model can be used to obtain data in the form of the number of patterns found on one chromosome, the number of genomic repeats found on one chromosome and the location of genomic repeats. Moreover, the comparisons that have been done show that the proposed model is faster than the computing on standalone and parallel computing with multicore. In the future, we have plans to improve the development and use of knowledge in this model so that it can be developed into various science sectors or in various case studies that can utilize Big Data technology with streaming processing using Apache Spark Streaming.

REFERENCES

[1] Charlesworth. B, Sniegowski. and P, Stephan. W, "The Evolutionary Dynamics of Repetitive DNA in Eukaryotes," *Nature*, vol. 371, pp. 215-20, 1994.
[2] Buard J, and Jeffreys A. J, "Big, Bad Minisatellites," *Nature Genetics,* vol 15, pp. 327-328, 1997.
[3] Edgar. R. C, and Myers. E. W, "PILER: Identification and Classification of Genomic Repeats," *Bioinformatics*, vol. 21, pp. 152-158, 2005.

[4] Pahadia. M, Srivastava. A, Srivastava. D, and Patil. N, "Genome Data Analysis Using MapReduce Paradigm," *Second Int. Conf. Adv. Comput. Commun. Eng.,* pp. 556-559, 2015.

[5] Orr. H. T, Zoghbi. H. Y, "Trinucleotide Repeat Disorders," *Encycl Neurol Sci.,* vol. 30, pp. 525-533, 2014.

[6] Al Kindhi. B, Sardjono. T. "Pattern Matching Performance Comparisons as Big Data Analysis Recommendations for Hepatitis C Virus (HCV) Sequence DNA," *3rd Int. Conf. Artif. Intell. Model Simu.,* pp. 99-104, 2015.

[7] Donald. E. K, James. H. M. J and Vaughan. R. P, "Fast Pattern Matching in Strings," *SIAM J Comput.,* vol. 6, no. 2, pp. 323-350, 1977.

[8] Boyer. R. S, Moore. J. S, "A Fast String Searching Algorithm," *Commun. ACM*, vol. 20, pp. 762-72, 1977.

[9] Sheik. S. S, Aggarwal. S. K, Poddar. A, Balakrishnan. N, and Sekar. K, "A Fast Pattern Matching Algorithm," *J. Chem. Inf. Comput. Sci.,* vol. 44, pp. 1251-1256, 2004.

[10] Haponiuk. M, Pawelkowicz. M, Przybecki. Z, and Nowak. R. M, "CuGene as a Tool to view and Explore Genomic Data," *Photonics App in Astron, Commu, Industry, and High Energy Physics Experimen,* vol. 10445, pp. 1-8, 2017.

[11] Shibata. Y, *et al.,* "A Boyer–Moore Type Algorithm for Compressed Pattern Matching," *Annual Symposium on Combinatorial Pattern Matching,* pp. 181–194, 2000.

[12] Riza. L. S, Rachmat. A. B, Munir. T. H, and Nazir. S. "Genomic Repeat Detection Using the Knuth-Morris-Pratt Algorithm on R High-Performance-Computing Package," *Int. J. of Adv. in Soft Comp. and Its App.*, vol. 11, pp. 94-111, 2019.

[13] Riza. L. S, Dhiba. T. F, Setiawan. W, Hidayat. T, and Fahsi. M. "Parallel Random Projection Using R High Performance Computing for Planted Motif Search," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 17, no. 3, pp. 1352-1359, 2009.

[14] Taylor. R. C, "An Overview of the Hadoop/MapReduce/HBase Framework and its Current Applications in Bioinformatics," *BMC Bioinformatics,* vol. 11, pp. 1-6. 2010.

[15] Guo. R, Zhao. Y, Zou. Q, Fang. X, and Peng. S, "Bioinformatics Applications on Apache Spark," *GigaScience,* vol. 7, pp. 1-10, 2018.

[16] Zaharia. M, *et al.,* "Spark: Cluster Computing with Working Sets," *HotCloud,* vol. 10, pp. 1-7, 2010.

[17] Shanahan. J. G, Street. H, Street. H, and Francisco. S. "Large Scale Distributed Data Science using Apache Spark," *Proc 21th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.,* pp. 2323-2324, 2015.

[18] Han. Z, Sql. A. S. "Spark: A Big Data Processing Platform Based On Memory Computing," *Seventh Int. Symp. Parallel Archit. Algorithms Program,* pp. 172-176, 2016.

[19] Liao. X, Gao. Z, Ji. W, and Wang. Y. "An Enforcement of Real Time Scheduling in Spark Streaming," *2015 Sixth International Green and Sustainable Computing Conference (IGSC).* pp. 1-6, 2015.

[20] Lekha. R. N, Sujala D. S, Siddhanth D. S. "Applying Spark Based Machine Learning Model on Streaming Big Data for Health Status Prediction," *Comput Electr Eng*, vol. 65, pp. 393-399, 2018.

[21] Leonardo. N, Bruce. R, Anish. N, and Anand. K, "S4: Distributed Stream Computing Platform," *Proc-IEEE Int Conf Data Mining, ICDM,* pp. 170-177, 2010.

[22] Karau H, Andy. K. Patrick. W. and Matei. Z. "Learning Spark: Lightning-Fast Big Data Analysis," *O'Reilly Media,* Inc., 2015.

[23] Chintapalli S, Dagit D, Evans B, Farivar R, Graves T, Holderbaugh M, Liu Z, Nusbaum K, Patil K, Peng BJ, Poulosky P. Benchmarking streaming computation engines: Storm, flink and spark streaming. *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, 1789-1792, 2016.

[24] Kroß J., Krcmar H., "Modeling and simulating Apache Spark streaming applications," *Softwaretechnik-Trends*, vol. 36, no. 4, pp. 1-3, 2016.

[25] Mediayani M., Wibisono Y., Riza L. S., Pérez A. R., "Determining Trending Topics in Twitter with a Data-Streaming Method in R," *Indonesian Journal of Science and Technology*, vol. 4, no. 1, pp. 148-157, 2019.

[26] Mulyani Y., Rahman E. F., Riza L. S., "A new approach on prediction of fever disease by using a combination of Dempster Shafer and Naïve bayes," *2016 2nd International Conference on Science in Information Technology (ICSITech),*pp. 367-371, 2016.

[27] Alasker H., Alharkan S., Alharkan W., Zaki A., Riza L. S., "Detection of kidney disease using various intelligent classifiers," *2017 3rd International Conference on Science in Information Technology (ICSITech)*, pp. 681-684, 2017.

[28] Riza L. S., Utama J. A., Putra S. M., Simatupang F. M., Nugroho E. P., "Parallel Exponential Smoothing Using the Bootstrap Method in R for Forecasting Asteroid's Orbital Elements," *Pertanika Journal of Science & Technology*, vol. 26, no. 1, pp. 441-462, 1 Jan 2018.