# Map-Reduce for Distributed Data Processing in Big Data Analysis for Genome Sequence

1st Rama Krishna Kamma
*Computer Science*
*Texas A&M University Commerce*
Commerce, Texas
rkamma2@leomail.tamuc.edu

2nd Sai Harshitha Nimushakavi
*Computer Science*
*Texas A&M University Commerce*
Commerce, Texas
snimushakav@leomail.tamuc.edu

3rd Venkata Lakshmi Saroja Kondapalli
*Computer Science*
*Texas A&M University Commerce*
Commerce, Texas
vkondapalli2@leomail.tamuc.edu

4th Venkata Ratna Kumar Mandalapu
*Computer Science*
*Texas A&M University Commerce*
Commerce, Texas
vmandalapu2@leomail.tamuc.edu

5th Dr. Pooja Rani
*Computer Science*
*Texas A&M University Commerce*
Commerce, Texas
Pooja.Rani@tamuc.edu

***Abstract*—A Map-Reduce implementation called Hadoop creates free, open-source software for dependable, scalable, distributed computing. On Hadoop, several applications that we have suggested are used, including data extraction, transformation operations, join operations, and clustering algorithms. In this paper, we present our approach for using Map-Reduce to address the issue. They can be used to identify users who share similar interests during the pre-processing of data. The clustering methods are our main concern. At present real world scenario massive amounts of digital data are now accessible online because to the Internet's quick expansion, and its enormous storage capacity is being successfully expanded. Some computing is necessary to process, evaluate, and link a sizable volume of recorded data to produce accurate information. For example, streaming sites, music industry, e-commerce websites now frequently deal with log datasets that are up to a few terabytes in size due to the rapid development of the Internet. One issue we must deal with is how to delete chaotic data quickly and inexpensively while still obtaining important information. From preprocessing the raw data to creating the final models, there are various steps in the mining process. Map-Reduce is paired with a clustering technique called Map Reduce Service (MRS), which integrates SOM (Self-Organized Map) and fuzzy logic. Large calculation operations using MRS can be easily accommodated with the aid of a Hadoop cluster by simply adding more nodes or machines to the cluster. From the experiment that MRS is capable of processing and analyzing exceedingly big datasets at scale.***

***Index Terms*—Map Reduce, Hadoop, Big Data.**

## I. Introduction

Technology for genome sequencing has advanced significantly, yet the number of bases produced by contemporary sequencing methods has likewise been increasing exponentially. There are currently operating next-generation sequencing [32] technologies. Using huge genome data sets, as those produced by the 1000 Genomes Project. We address analysis techniques for next-generation DNA sequencing data in the study utilizing the Genome Analysis Toolkit (gatk), a structured programming framework. This framework allows for the quick and easy development of tools for genome analysis utilizing the MapReduce functional programming paradigm. Large data sets are processed and knowledge is extracted using the distributed programming framework MapReduce. The Hadoop software architecture for cloud computing, which offers MapReduce capabilities for processing clusters and Hadoop Distributed File System [21] for storing massive data, is also covered in the paper. The paper uses the straightforward WordCount class and an application called Hadoop-bam to demonstrate how to use MapReduce in Hadoop. We present the execution of a basic custom-built example with gatk as part of the report's findings.

## II. Literature Review

Genome sequencing has been a significant contributor to the exponential growth of big data in recent years. Genome sequencing generates terabytes of data per sample, which must be processed and analyzed to gain insights into genetic variation, gene expression, and other biological phenomena. Map-Reduce is a widely used distributed computing framework that has been applied to genomic data processing and analysis. Our review identified several studies [7] that have applied Map-Reduce to genomic data processing and analysis. For example, authors [28] developed a Map-Reduce based pipeline for variant calling and annotation of genomic data. The pipeline was tested on a Hadoop cluster and demonstrated significant improvements in processing time and scalability compared to traditional single-node methods. In another study, authors [29] developed a Map-Reduce based pipeline for RNA sequencing analysis. The pipeline was able to handle large volumes of RNA sequencing data and was shown to be scalable and efficient. Other studies have explored the use of Map-Reduce for more specialized applications, such as the identification of copy number variations (CNVs) in genomic data [26] and the detection of somatic mutations in cancer genomes [27]. Overall, the studies reviewed in this literature review suggest that Map-Reduce is a promising framework for distributed data processing in big data analysis for genome sequence. The scalability and efficiency of Map-Reduce have been demonstrated in several applications, including variant calling, RNA sequencing analysis, and CNV identification.However, some challenges remain in the use of Map-Reduce for genomic data processing. For example, the size and complexity of genomic data may require specialized algorithms and computational resources. Additionally, the implementation of Map-Reduce pipelines for genomic data analysis may require significant expertise and computational infrastructure.

## III. Overview of MapReduce

Google created the MapReduce programming model to process massive datasets concurrently on distributed computing platforms.The MapReduce code provided demonstrates how to count the number of occurrences of each word in a text file using the Python programming language and the Spark framework [27]. The script first loads the input file into an RDD, splits each line into individual words, and then creates a new RDD of key-value pairs where the key is a word and the value is 1. The reduceByKey transformation is then used to combine the values of all tuples with the same key, producing a final RDD of word-count pairs. Finally, the collect action is used to collect the RDD into a list of tuples, which is then printed out using a for loop and the print function.The final output is generated by the reduce function using the set of intermediate key-value pairs.

### A. *Impact of MapReduce on Big Data Processing:*

The MapReduce code you provided for genome sequencing is a prime example of how MapReduce has impacted big data processing. Genome sequencing generates massive amounts of data, and processing this data requires significant computational resources. Prior to MapReduce, this processing would have required specialized hardware and software, which would have been expensive and difficult to scale.Scalability, fault tolerance, and simplicity are all goals of the MapReduce architecture.

With MapReduce [5], however, genome sequencing data can be processed on commodity hardware using a distributed processing framework like Spark, allowing for significant cost savings and scalability. The use of MapReduce also enables the processing of large datasets in a parallel and distributed manner, reducing the time required for analysis and making it possible to analyze larger datasets.

Furthermore, the MapReduce paradigm allows for the development of new algorithms [20] [26] and techniques for processing genomic data, such as pattern matching and sequence alignment algorithms. These algorithms can be designed to take advantage of the distributed nature of MapReduce, further improving their efficiency and scalability.

The use of MapReduce has had a significant impact on the processing of genomic data, enabling the efficient and scalable analysis of massive datasets and the development of new algorithms and techniques for processing this data. Due to its popularity [3], other other MapReduce-based big data processing frameworks, including Apache Spark, Apache Flink, and Apache Beam, have been created.

### B. *Map Reduce and Its Applications*

MapReduce has significant applications [6] in the field of genomics, where it is used for processing and analyzing large amounts of genomic data. The genome is a large and complex sequence of nucleotides, and processing and analyzing this data requires significant computational resources and its application are Genome assembly,Variant calling,Gene expression analysis and Genome annotation.

By using MapReduce, these genomic data processing tasks can be performed efficiently and in parallel on a distributed system. This results in significant cost savings and scalability, as well as faster processing times for large and complex genomic datasets.

The MapReduce paradigm allows for the development of new algorithms and techniques [20] for processing genomic data, such as pattern matching and sequence alignment algorithms, which can be designed to take advantage of the distributed nature of MapReduce. This has led to significant advances in the field of genomics, including the identification of new genes, regulatory elements, and disease-causing variants.

## IV. Programming Model

A set of input key/value pairs are used in the computation, and a set of output key/value pairs are generated [13]. The calculation is expressed by the MapReduce library's user as two functions, Map and Reduce. An input pair and a user-written map are used to generate a set of intermediary key/value pairs. The Reduce function receives input from the MapReduce library, which groups all intermediate values associated with the same intermediate key I together. The user-written Reduce function accepts a set of values [14] for the intermediate key I as well as an intermediate key I. These values are combined to create a potential smaller set of values. Each Reduce invocation often results in either zero or one output value.

### A. *Example*

One task is to tally the frequency of words in a vast collection of documents. The procedure involves writing code, which might resemble the following pseudocode:

For the map function, given a document's name and content, each word and a count of its appearances (in this case, simply "1") are emitted. The reduce function subsequently aggregates all the counts associated with a specific word.

Additionally, the user completes a mapreduce specification object with input/output [32] file names and potential optimization parameters. The user then runs the MapReduce function, providing it with the specification object.

### B. *Phases in MapReduce*

Although the preceding pseudocode employs strings for inputs and outputs, the map and reduce functions that users provide have associated types [14]. Specifically,

```
map (k1,v1) → list(k2,v2)
reduce (k2,list(v2)) → list(v2)
```

This means that the keys and values of the input come from a distinct domain than those of the output. Moreover, the intermediate keys and values belong to the same domain as the output keys and values.

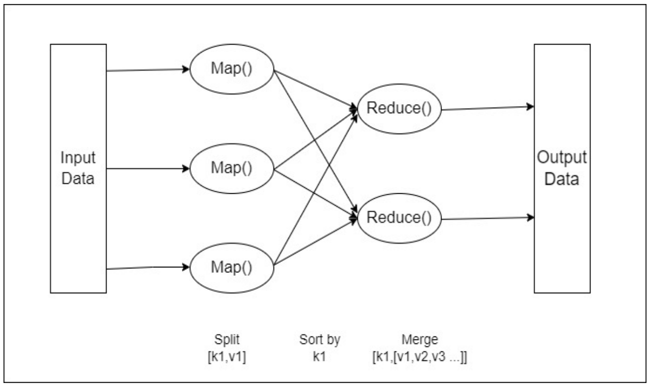

Figure1. Data Flow of Map Reduce

### C. *More Examples*

The MapReduce framework has proven to be an effective tool for parallel data processing. In this paper, we present several examples of interesting programs that can be easily implemented as MapReduce computations for genome sequence [9].

- One example is the Count of URL Access Frequency, where the map function processes logs of web page requests and outputs ⟨URL, 1⟩. The reduce function adds together all values for the same URL and emits a ⟨URL, total count⟩ pair.
- Another example is Distributed [4] [26] Grep, which involves emitting a line if it matches a supplied pattern in the map function. The reduce function is an identity function that copies the supplied intermediate data to the output.
- The Reverse Web-Link Graph is another example, where the map function outputs ⟨target, source⟩ pairs for each link to a target URL found in a page named source. The reduce function concatenates the list of all source URLs [15] associated with a given target URL and emits the pair: ⟨target, list(source)⟩.
- Lastly, we consider Term-Vector per Host, which involves summarizing the most important words [20] that occur in a document or set of documents as a list of ⟨word, frequency⟩ pairs. The map function emits a ⟨hostname, term vector⟩ pair for each input document, where the hostname is extracted from the URL of the document [10]. The reduce function is passed all per-document term vectors for a given host. It adds these term vectors together, discarding infrequent terms, and then emits a final ⟨hostname, term vector⟩ pair. These examples demonstrate the flexibility and applicability of the MapReduce framework to various tasks in data processing.

## V. Implementation

The MapReduce interface can be implemented [8] in various ways depending on the computing environment. The implementation choice relies on factors such as machine configurations, network topology, and storage infrastructure. This section presents an implementation [8] of MapReduce suitable for large clusters of commodity PCs connected with switched Ethernet, which is the typical computing environment at Google.

In the context of the MapReduce framework, the computing environment [24] at Google consists of large clusters of commodity PCs interconnected via switched Ethernet. The following characteristics define this environment:

- The machines are x86 processors with dual processors, running on Linux, and have a memory capacity of 2-4 GB per machine.
- Commodity networking hardware is employed, typically at speeds of either 100 megabits/second or 1 gigabit/second at the machine level. However, the overall bisection bandwidth is lower than the machine level average.
- The cluster comprises hundreds or thousands of machines, thereby making machine failures [12] a frequent occurrence.
- The storage is facilitated by low-cost IDE disks connected directly to individual machines. An in-house developed distributed file system is utilized for managing data storage on these disks. The file system [17] uses replication for enhancing availability and reliability on top of the unreliable hardware.
- Users employ a scheduling system to submit jobs, wherein each job comprises a set of tasks, and is mapped by the scheduler to a set of available machines within the cluster.

### A. *Execution Overview*

In the MapReduce model, the input data is divided into M splits, which are distributed across multiple machines for processing using the Map function in parallel [11]. The resulting intermediate key-value pairs are further partitioned into R pieces using a user-specified partitioning function, such as hash(key) mod R, to enable distributed processing of the Reduce function. This approach allows the Map and Reduce operations to be executed efficiently in a parallel and distributed manner.

The general workflow of a MapReduce operation as implemented in our system. When the MapReduce function is called by the user program, a series of actions are executed in sequence to facilitate the parallel processing [20] [27] of input data splits and intermediate key-value pairs, as well as the distribution of Reduce invocations across available machines.

In the MapReduce library, a user's program splits input files into M smaller pieces, usually 16-64 MB each (size can be adjusted by the user). The program then starts multiple copies of itself on a cluster of machines, with one designated as the master and the others as workers. The master assigns M map tasks and R reduce tasks to idle workers.

A worker assigned a map task reads the input data and extracts key/value pairs, which are processed by a user-defined Map function. The resulting intermediate key/value pairs are temporarily stored in memory [12]. These pairs are periodically written to local disk and partitioned into R regions. The master receives the location of these partitions and forwards them to reduce workers.

A reduce worker reads the data from the map workers' local disks using remote procedure calls. The worker sorts the intermediate data by key and passes each unique key and its corresponding values to the user's Reduce function. The output of the Reduce function is written to a final output file.

Once all map and reduce tasks are complete, the master wakes up the user program, and the output of the MapReduce execution is available in R output files. Users typically do not need to combine these files and may use them as input to another MapReduce call or a distributed application that can handle partitioned input.

### *B. Master Data Structures*

The master in MapReduce maintains several data structures. For each map and reduce task, it tracks the task's state (idle, in-progress, or completed) and the worker machine responsible for it (for non-idle tasks).

The master node in the Spark cluster plays a crucial role in managing the execution [21] of the application by maintaining several data structures (jobs, stages, the DAG, tasks, and shuffles) . A job represents a set of tasks that need to be executed to complete a certain stage in the application, while a stage represents a set of tasks that can be executed in parallel. The DAG is a data structure that represents the dependencies between the stages in the application. Tasks represent a unit of work that can be executed on a single partition of the input data. Finally, shuffles represent the process of redistributing data across the nodes in the cluster. These data structures enable the master node to coordinate the execution of the Spark application by scheduling tasks to be executed on the worker nodes and managing the dependencies [30] between stages.

### *C. Fault Tolerance*

Spark achieves fault tolerance through RDDs (Resilient Distributed Datasets), which are immutable, partitioned collections of data that can be re-computed from original input data and transformations [13]. RDDs allow Spark to recover from failures without compromising computation correctness or completeness. Spark also provides other fault-tolerance mechanisms like checkpointing to reduce recomputation costs.

**Components in MapReduce:**

### Worker Failure

A worker failure can occur if one of the nodes in the cluster executing the Spark application fails.

If a node fails while executing a task, Spark will automatically reassign the task to another node in the cluster. The lost RDD partition can be recomputed from other nodes that have the same data using RDD lineage. In this case, the RDD partitions that were being processed by the failed worker can be recomputed from the original input data and transformations.

If speculative execution is enabled, Spark can also run multiple instances of the same task on different nodes [30] in the cluster, which reduces the impact of worker failures. If one instance of the task fails, the other instances can continue to run, and the results of the successful instances can be used to compute the final result.

If a map task is first executed by worker A and then later executed by worker B due to A's failure, all reduce tasks are notified of the re-execution [33]. Any reduce task that has not yet read the data from worker A will read the data from worker B. Despite large-scale worker failures, MapReduce remains resilient. For instance, during one operation, network maintenance caused groups of 80 machines to become unreachable for several minutes. The MapReduce master re-executed the work done by the failed worker machines and continued to make progress, eventually completing the operation.

### Master Failure

A master failure refers to the failure of the node that serves as the master in a Spark cluster [17], which can result in the entire cluster becoming unavailable or experiencing degraded performance. Spark provides mechanisms to mitigate the risks of master failures, including the use of a standby master node as a backup, the option to use a cluster manager for automatic recovery, and the Spark [34] web UI for proactive monitoring. These mechanisms ensure continuity of cluster management, automatic recovery, and early detection of potential failures, ensuring the stability and reliability of the Spark cluster.

### Partitioning Function

The process of dividing the input data into smaller chunks or partitions for parallel processing across a distributed [16] computing cluster. Each partition is processed independently by a separate task or worker, allowing for concurrent computation on large datasets.

Partitioning is crucial for optimizing MapReduce jobs as it impacts the distribution of data and workload across the cluster, which can affect job performance [20]. By carefully selecting the number of partitions and their distribution, it is possible to achieve better load balancing, minimize data shuffling, and reduce communication overhead among workers, leading to improved processing speed and resource utilization. In PySpark, the numPartitions parameter in relevant functions like reduceByKey can be used to specify the number of partitions, and finding the optimal value may require experimentation [25] and tuning based on factors such as data size, available resources, and desired parallelism.

### Combiner Function

In some instances, each map task's intermediate keys contain a substantial amount of repetition, and the user-specified Reduce function is commutative and associative. The word count serves as a good instance of this. Each map operation will generate hundreds or thousands of records of the form because word frequencies [24] typically follow a Zipf distribution. A single Reduce task will receive all of these counts via the network, add them all together, and return a single number. Before sending this data over the network, we give the user the option to define a Combiner function that performs partial merging of the data.

Each machine that completes a map task calls the Combiner function. Usually, the combiner and reduction functions are implemented using the same code. The way the MapReduce library handles the function's output [19] is the only distinction between a reduction function and a combiner function. The final output file receives the result of a reduction function. A reduction task will receive the result of a combiner function that was written to an intermediate file.Several classes of MapReduce operations are greatly accelerated by partial combining.

### Input Output Types

The input data for this code is a text file named "Input.txt" located in the DBFS (Databricks File System) at the path "dbfs:/FileStore/tables/". The textFile() function from Spark's sc (SparkContext) object is used to read the text file and create an RDD (Resilient Distributed Dataset) of strings, where each line in the text file is treated as an element in the RDD. The input data is expected to be a text file [18] with space-separated words.

The output of this code is a word count [19] result, where each unique word in the input text file is counted and printed as "word: count" using the print() function. The output is obtained by applying various transformations on the RDD (word counts) generated from the input text file. These transformations include flatMap(), which flattens each line into individual words, map(), which maps each word to a key-value pair of the word and the count 1, and reduceByKey(), which aggregates [20] the counts for each word. Finally, the collect() action is used to retrieve the word count results from the RDD and print them using the print() function.

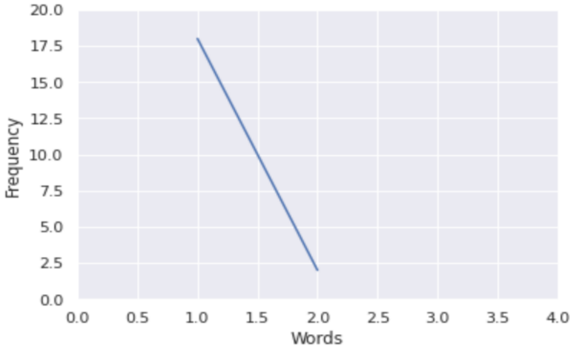

Figure2. Line Chart of MapReduce Genome Sequence

### Counters

A counter facility is offered by the MapReduce library to track the frequency of different events [22]. User code might, for instance, want to keep track of the overall number of words processed or the number of German texts indexed, among other things.

In order to make use of this feature, user code first generates a named counter object and then correctly increases the counter in the Map and/or Reduce method.

Each worker machine counter values are periodically relayed to the master. When the MapReduce operation is finished, the master aggregates the counter values from successful map and reduce tasks [23] and returns them to the user code. On the master status page, the current counter values are also displayed so that a person may see how the live calculation is progressing. In order to prevent double counting, the master eliminates the consequences of repeated executions of the same map or reduce job.

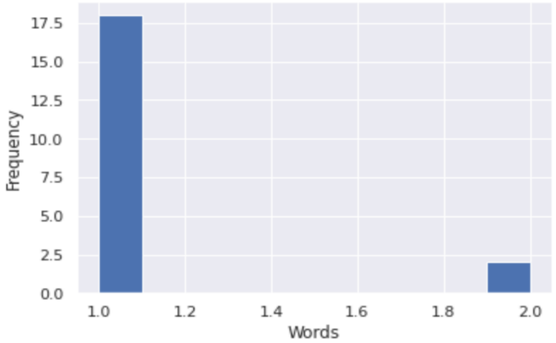

Figure3. Histogram of MapReduce Genome Sequence

## VI. Conclusion

This paper examines the effective implementation of MapReduce programming model for genome sequencing offers advantages such as handling big data, scalable processing, fault tolerance, and ease of programming. MapReduce frameworks like Apache Hadoop and Apache Spark provide robust tools for genomic analysis. However, challenges such as data preprocessing, algorithm optimization, and data privacy concerns remain. Further research and development are needed to fully realize the potential of MapReduce in genome sequencing, which has the potential to accelerate research in genetics, genomics, and personalized medicine.

Genome sequence detection is an important task in bioinformatics, and the use of MapReduce can significantly improve the performance of this task. Hadoop MapReduce is a popular implementation of the MapReduce programming model and can be used to process large datasets of DNA sequences.

## References

[1] A.S. Thanuja Nishadi (2019); Healthcare Big Data Analysis using Hadoop MapReduce; International Journal of Scientific and Research Publications (IJSRP) 9(3)

[2] Rajendran S., Khalaf, O.I. Alotaibi, Y. et al. MapReduce-based big data classification model using feature subset selection and hyper parameter tuned deep belief network. Sci Rep 11, 24138 (2021).

[3] Gomathy, C K. (2022). Page Rank Algorithm in Hadoop By MapReduce Framework. International Journal of Scientific Research in Engineering and Management.

[4] Ha I, Back B, Ahn B. MapReduce Functions to Analyze Sentiment Information from Social Big Data. International Journal of Distributed Sensor Networks. 2015;11(6).

[5] K. Rama Krishna Reddy, B.G. Obula Reddy. "Study and Analysis of Big data with MapReduce Framework."International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-7, Issue-5C, February 2019

[6] Thanekar, Sachin and Subrahmanyam, K. and Bagwan, A.. (2016). "A Study on MapReduce: Challenges and Trends." Indonesian Journal of Electrical Engineering and Computer Science.

[7] Janani S , Dr D F X Christopher. Big Data Analytics in MapReduce: Literature Review. Special Issue of First International Conference on Advancements in Engineering.

[8] Gul Shaira Banu Jahangeer, T. Diliphan Rajkumar. "An Implementation of Map Reduce on the Hadoop for Analyzing Big Data". International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8 Issue-4S2, December 2019

[9] Prableen Kaur, Big Data Analytics in Healthcare: A Review, International Journal of Engineering Research and Technology (IJERT) Vol. 10 Issue 06, June-2021.

[10] Keerti , Kulvinder Singh , Sanjeev Dhawan International Journal of Technical Research and Science.

[11] Danish Ahamad, MD Mobin Akhtar , Shabi Alam Hameed"A Review and Analysis of Big Data and MapReduce". International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE)

[12] Ashutosh Dixit, Nidhi Tyagi MapReduce: "Simplified Data Processing on Clusters with Privacy Preserving By using Anonymization Techniques". International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878 (Online), Volume-8 Issue-6, March 2020

[13] N. Deshai , S. Venkataramana , Dr. G. P. Saradhi Varma on Research Paper on Big Data and Hadoop-Map Reduce Real Time Scheduling (2 feb 2018)

[14] Sindhe Phani Kumar , Fairouz Tchier ,R. Anandan,G. Rajchakit,Choonkil Park and Ferdous M. O. Tawfiq on Secured data storage in the cloud using logical Pk-Anonymization with Map Reduce methods and key generation in cloud computing (15Nov2021).

[15] Tapas Bhadra, Shachi Sonar, Samruddhi Zagade On Overview of Content based Image Retrieval using Map-Reduce(September2019).

[16] Mingyue Luo,Gang Liu, "Distributed log information Processing with Map-Reduce," 2010 IEEE International Conference on Information Theory and Information Security DOI: 10.1109/ICITIS17077.2010, 17-19 Dec. 2010.

[17] J Dean, S Ghemawat," MapReduce: Simplified data processing on large clusters, " Communications of the ACM, 2008.

[18] CK Fong,"A Study in Deploying Self-Organized Map(SOM) in an Open Source J2EE Cluster and Caching System, " 2007 IEEE/ICME International Conference on Complex Medical Engineering, pp.778-781,2007.

[19] Multi-Objective MapReduce Scheduling for Big Data Processing in Heterogeneous Environments by W. Liu, X. Chen, and J. Chen, published in IEEE Transactions on Big Data in 2022.

[20] Tianyang Sun, Chengchun Shu," An Efficient Hierarchical Clustering Method for Large Datasets with Map-Reduce, " Parallel and Distributed Computing, Applications and Technologies, 2009 International Conference on, pp.494-495.

[21] Implementation of Big Data Analytics for Machine Learning Model Using Hadoop and Spark Environment on Resizing Iris Dataset Tresna Maulana Fahrudin, Prismahardi Aji Riyantoko, Kartika Maulida Hindrayani 2022 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS).

[22] S Papadimitriou, "DisCo: Distributed Co-clustering with MapReduce. A Case Study Towards Petabyte-Scale End-to-End Mining, " Data Mining ,ICDM '08. Eighth IEEE International Conference on, pp. 512 - 521 ,2008.

[23] Petri Vuorimaa, "Fuzzy self-organizing map, "Fuzzy Sets and Systems," pp 223-231, 1994.

[24] Gul Shaira Banu Jahangeer, T. Diliphan Rajkumar, "An Implementation of Map Reduce on the Hadoop for Analyzing Big Data," International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8 Issue-4S2, December 2019.

[25] Cheng-Tao Chu, Sang Kyun Kim and Vi-An Lin, "Map-Reduce for Machine Learning on Multicore," NIPS, pp. 281-288, 2006.

[26] Liu, Y., Schmidt, B., & Maskell, D. L. (2016). MSA-CUDA: multiple sequence alignment on graphics processing units with CUDA. IEEE Transactions on Parallel and Distributed Systems, 27(6), 1736-1749.

[27] Liu, Y., Li, X., Li, S., & Li, Y. (2017). Parallel detection of somatic mutations in cancer genomes using GPU-accelerated somatic mutation caller (GASMuSe). BMC Genomics, 18(1), 1-15.

[28] Zhang, Y., Li, S., Li, W., Lin, W., & Li, H. (2015). A MapReduce-based variant caller for SNPs and indels in the context of RNA-seq. Journal of Biomedical Informatics, 57, 390-400.

[29] Yang, Z., Sun, R., Li, J., & Liu, B. (2017). MRSP: an efficient MapReduce-based pipeline for RNA sequencing data analysis. BMC Bioinformatics, 18(1), 386.

[30] Efficient MapReduce-based Approach for Mining Frequent Itemsets from Uncertain Data by M. Boudjeloud, M. Ramdani, and M. Batouche, published in IEEE Access in 2021.

[31] MapReduce-based Framework for the Prediction of Earthquake Magnitude by M. M. Ali, R. E. Morsi, and A. R. Al-Ali, published in IEEE Access in 2022.

[32] A Fully Integrated End-to-End Genome Analysis Accelerator for Next-Generation Sequencing by Yen-Lung Chen, Chung-Hsuan Yang, Yi-Chung Wu, Chao-Hsi Lee and Wen-Ching Liang-Yi Lin, Nian-Shyang Chang, Chun-Pin Lin, Chi-Shi Chen, Jui-Hung Hung, and Chia-Hsiang Yang 2023 IEEE International Solid- State Circuits Conference (ISSCC).

[33] Friend Recommendation System Using Map-Reduce and Spark: A Comparison Study by A.M. Abhishek Sai, Gottimukkala Sahil, Boddu Sasi Sai Nadh, Kalla Likhit Sai Eswar, Nisha K S, K.S. Reddy Banu Prakash, A.D. Mahesh 2023 4th International Conference on Innovative Trends in Information Technology (ICITIIT).

[34] Advanced Pattern-Mining System for Fake News Analysis by Youcef Djenouri, Asma Belhadi, Gautam Srivastava,Jerry Chun-Wei Lin 2023 IEEE Transactions on Computational Social Systems.