Clustering of Bigdata Using Genetic Algorithm in Hadoop MapReduce

Chandra Shekhar Gautam [1], Mr. Laxmi Narayan Soni [2], Dr. Prabhat Pandey [3]

[1, 2] AKS University Satna (M.P.), India

[3] APS University Rewa (M.P.), India

Email: [1] shekharg84@gmail.com, [2] lnsoni205@gmail.com, [3] pandeyprabhat51@gmail.com

Abstract

The clustering of Bigdata is a common task in data mining and machine learning. The goal is to group similar data points to identify patterns and relationships in the data. However, clustering large datasets can be computationally expensive and time-consuming. This is where Hadoop MapReduce comes in. Hadoop is a sophisticated framework that facilitates the distributed processing of voluminous datasets across multiple clusters of computers. MapReduce is a programming model that simplifies the processing of large datasets by breaking them down into smaller chunks and processing them in parallel across the cluster. One approach to clustering Bigdata using Hadoop MapReduce is to use a genetic algorithm. A genetic algorithm is an optimization technique that is inspired by the process of natural selection. It works by iteratively generating and evaluating candidate solutions and using the best solutions as a basis for generating the next generation of candidates. This paper introduces a technique to parallelize GA-based clustering by extending Hadoop MapReduce. An analysis of the proposed approach to evaluate performance gains to a sequential algorithm is presented. The analysis is predicated upon a substantial real-world dataset.

Index Terms— Big Data, Clustering, Davies-Bouldin Index, Parallel Genetic Algorithm, Distributed processing, Hadoop MapReduce.

1. Introduction

Clustering is a popular technique for grouping similar data points, and it has a wide range of applications in pattern recognition, data mining, and other fields. parallelization [3] is often used to speed up the process. By distributing the data and computation across multiple processors or machines, parallelization can significantly reduce the time required to perform clustering on large data sets. Hadoop MapReduce is one popular framework for the parallel processing of big data sets, and it can be used to implement clustering algorithms that are scalable to large data sets. Other parallel computing frameworks like Apache Spark and MPI (Message Passing Interface) can also be used for parallel clustering.

Hadoop MapReduce [4] is a parallel programming technique built on the frameworks of the Google app engine MapReduce. it is utilized for the purpose of processing vast amounts of data within a distributed environment. It boasts a high degree of scalability and can be constructed using readily available hardware. The Hadoop MapReduce framework partitions the input data into chunks of specific sizes and concurrently processes these chunks across the cluster. This approach effectively diminishes the time complexity required to solve the problem by distributing the processing load among the nodes within the cluster. In this paper,

we propose a technique to implement clustering using a genetic algorithm in a parallel fashion using Hadoop MapReduce. In order to accomplish this task, we expand upon the coarse-grained parallel model of genetic algorithms and execute a two-phase clustering procedure on the given data set. This two-phase clustering approach is realized by exploiting the Hadoop MapReduce architecture. The subsequent sections of the document are structured as follows: Section 2 provides a comprehensive outline of genetic algorithms. Section 3 explains the MapReduce model and discusses the Hadoop MapReduce. Section 4 presents the technique we devised to parallelize genetic algorithm-based clustering by extending Hadoop MapReduce. Sections 5 and 6 describe the criteria we used for deploying parallelized GA on Hadoop MapReduce as well as the results of experimentation.

2. Genetic Algorithms

Genetic algorithms (GAs) are a type of optimization algorithm that is inspired by the principles of natural selection and genetics. GAs is used to solve complex optimization problems by simulating the process of evolution. In other words, GAs applies the principles of survival of the fittest and genetic crossover and mutation to generate and evolve solutions to a problem. It is a nature-inspired heuristic approach used for solving search-based and optimization problems. It belongs to a class of evolutionary algorithms [10], [11]. In GAs, we evolve a population of candidate solutions toward an optimal solution. The genetic algorithm (GA) employs nature-inspired mechanisms such as crossover, mutation, selection, and inheritance to attain an optimal solution. Within the framework of GA, the principle of survival of the fittest is implemented to optimize the candidate solutions. The GA technique proceeds in a sequential manner as follows:

- ***Initialization***: The algorithm creates an initial population of candidate solutions to the problem being solved. The population is typically generated randomly.
- ***Fitness Evaluation***: Each candidate solution in the population is evaluated for its fitness, which is a measure of how well it solves the problem being considered.
- ***Selection****:* A subset of the population is selected for reproduction based on their fitness. This step simulates the survival of the fittest principle.
- ***Crossover***: The selected individuals are combined to create new candidate solutions through a genetic crossover. This step simulates the creation of offspring through the mixing of genetic material from parents.
- ***Mutation****:* Some of the newly created candidate solutions are randomly mutated to introduce further genetic diversity.
- ***Evaluation***: The fitness of the new population is evaluated, and the process is repeated until a stopping criterion is met. The stopping criterion could be a certain number of generations or a satisfactory level of fitness achieved.

Genetic algorithms [3] have been applied to a wide range of optimization problems, including function optimization, clustering, scheduling, and data mining. One advantage of genetic algorithms is that they can handle non-linear, non-differentiable, and discontinuous optimization problems. However, they can be computationally expensive and may not always converge to the global optimum. Nonetheless, they remain a popular optimization technique in various fields due to their ability to generate good solutions to complex optimization problems.

2.1 Sequential Genetic Algorithm (SGA)

There are several possible versions of GAs execution flows. The parallel adaptations are constructed upon the foundation of the subsequent SGA implementation, which comprises a series of genetic operators that are iteratively applied across generations, as delineated in the following.

Algorithm 1 Sequential Genetic Algorithm (SGA)

1: population ← INITIALIZATION (populationSize)
2: for i ← 1, n do
3: if i = 1 then
4: for individual ∈ population do
5: FITNESSEVALUATION (individual)
6: elitists ← ELITISM (population)
7: population ← population – elitists
8: selectedCouples ← PARENTSSELECTION (population)
9: for ($parent_1$, $parent_2$) ∈ selectedCouples do
10: ($child_1$, $child_2$) ← CROSSOVER ($parent_1$, $parent_2$)
11: offspring ← offspring ∪ {child1} ∪ {child2}
12: for individual ∈ offspring do
13: MUTATION (individual)
14: for individual ∈ offspring do
15: FITNESSEVALUATION (individual)
16: population ← SURVIVALSELECTION (population, offspring)
17: population ← population ∪ elitists

The execution flow starts with an initial population initialised with the INITIALIZATION function (1), which can be either a random function or a specifically defined one based on other criteria. Then, at the first generation, the genetic operator applied is the FITNESSEVALUATION (3−5), which evaluates and assign a fitness value to each individual, letting them be comparable. The ELITISM operator (5−6) allows to add some individuals directly to the next generation (17). The PARENTSSELECTION operator (8) selects the couples of parents for the CROSSOVER phase based on the fitness values. The mixing of parent couples produces the offspring population (9−11), which is submitted to the MUTATION phase (12−13) in which the genes may be altered. The SURVIVAL SELECTION process involves selecting individuals from both parents and offspring (16) to determine which individuals will participate in the subsequent generations.

3. MapReduce Paradigm

MapReduce [6] is a programming paradigm used for processing large data sets. It was developed by Google to handle the large-scale distributed data processing in a parallel and fault-tolerant manner. MapReduce divides a large data set into smaller parts and processes them in parallel across multiple computers in a cluster.

The MapReduce paradigm consists of two main phases: the map phase and the reduces phase. In the map phase, the input data is processed and converted into intermediate key-value pairs. The map function takes input data and produces a set of intermediate key-value pairs. The key-value pairs are then shuffled and sorted based on the keys, and sent to the reduce phase.

In the reduce phase, the intermediate key-value pairs are combined based on their keys to produce the final output. The reduce function takes a set of intermediate key-value pairs with the same key and produces a set of output key-value pairs.

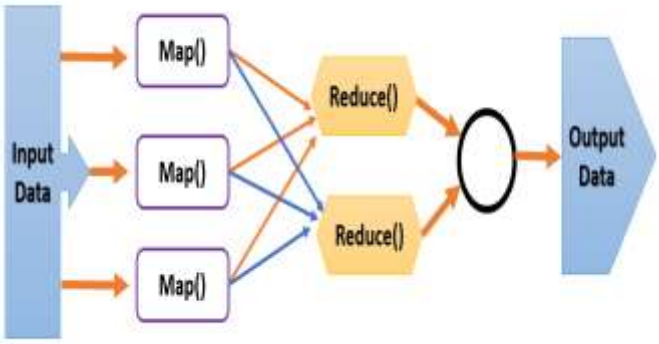

Figure 1 MapReduce Paradigm

The MapReduce [8] paradigm also includes a partitioning function that determines how the intermediate key-value pairs are distributed across the reducers. The partitioning function ensures that all key-value pairs with the same key are sent to the same reducer.

The advantages of MapReduce include scalability, fault tolerance, and ease of programming. MapReduce can handle large-scale data processing across multiple machines in a cluster, and it automatically handles failures by re-executing failed tasks on other machines. MapReduce also provides a simple programming model for developers to write distributed data processing applications.

MapReduce [12] is commonly used in big data processing frameworks such as Apache Hadoop, Apache Spark, and Apache Flink. These frameworks provide high-level APIs for developers to write MapReduce programs, and handle the details of distributed data processing and fault tolerance.

4. Hadoop MapReduce

Apache Hadoop [14] MapReduce is a distributed data processing framework that is designed to handle large data sets in a distributed and parallel manner. It is a key component of the Apache Hadoop ecosystem and allows for the distributed processing of data across a large number of nodes. MapReduce framework consists of two main components: Map and Reduce. The Map component takes input data and transforms it into a set of intermediate key-value pairs. The Reduce component takes the output of the Map component, processes it, and produces the final output.

In Hadoop MapReduce [14], data is stored in a distributed file system called Hadoop Distributed File System (HDFS). The HDFS is designed to handle large data sets and provides a fault-tolerant storage layer for Hadoop MapReduce jobs.

The Hadoop MapReduce [17][18] programming model is based on the concept of functional programming. The Map and Reduce components are functions that operate on the input data and produce output data. The Map function is executed in parallel on multiple nodes, with each node processing a subset of the input data. The Reduce function is also executed in parallel on multiple nodes, with each node processing a subset of the intermediate key-value pairs produced by the Map function.

One of the key advantages of Hadoop MapReduce is its scalability. It can be used to process large amounts of data by distributing the workload across multiple nodes in a cluster. This makes it possible to process large data sets that cannot be processed on a single machine. It also provides fault tolerance, meaning that if a node fails during the processing of a job, the framework will automatically re-run the job on another node to ensure that it is completed successfully. It has become a popular framework for large-scale data processing, and it has been used in a wide range of applications, including web search, image processing, and machine learning.

5. Parallel Genetic Algorithms

Parallel Genetic Algorithms [19][21] (PGAs) are a variant of the Genetic Algorithm (GA) that uses multiple processors to speed up the search process. PGAs are a form of parallel computing that takes advantage of the fact that GAs are well-suited for parallelization because of their population-based search approach.

In a PGA, the population [9][11] of candidate solutions is divided among multiple processors, and each processor evaluates a subset of the solutions. The fitness values of these solutions are then transmitted to a central location, where the selection, crossover, and mutation operators are applied to generate a new population. This new population is then divided among the processors for evaluation in the next generation.

The primary advantage of PGAs is their ability to speed up the search process by exploiting parallelism. This is particularly useful for problems that require a large number of fitness evaluations, such as those found in optimization problems.

PGAs can be implemented in various ways, depending on the architecture of the computer system being used. For example, they can be implemented using shared-memory systems, distributed-memory systems, or a combination of both.

One of the challenges of using PGAs is maintaining the diversity of the population across multiple processors. In a traditional GA, diversity is maintained through selection, crossover, and mutation operators. In a PGA, it is important to ensure that these operators are properly synchronized to avoid losing diversity as the search progresses.

Overall, PGAs are a powerful tool for solving complex optimization problems in a parallel and distributed environment. They are widely used in various fields, including engineering, computer science, and finance.

6. Customized Parallel Implementation for Clustring using Hadoop MapReduce

Customized parallel implementation [25] for clustering using Hadoop MapReduce involves developing a MapReduce program that partitions the data set into smaller subsets, performs clustering on each subset in parallel, and then combines the results to produce the final clustering.

The following is a general outline of the steps involved in implementing clustering using Hadoop MapReduce:

- ***Data Partitioning***: The first step is to partition the input data set into smaller subsets that can be processed in parallel. This can be done using a custom Map function that reads the input data and partitions it based on some criteria such as data range, data distribution, or some other factor.
- ***Clustering***: Once the data is partitioned, each subset can be processed in parallel by a separate Map task. Each Map task performs clustering on its assigned subset using a

clustering algorithm such as K-means, hierarchical clustering, or DBSCAN. The output of each Map task is a set of intermediate key-value pairs, where the key represents the cluster ID and the value is the data point that belongs to that cluster.

- ***Reducing****:* The intermediate results generated by the Map tasks are then combined in the Reduce phase to produce the final clustering. The Reduce function receives the intermediate key-value pairs and aggregates them to produce the final clusters.
- ***Output:*** The final clustering result can be written to the output file system or database.
- The architecture of the proposed model is depicted in Figure 2

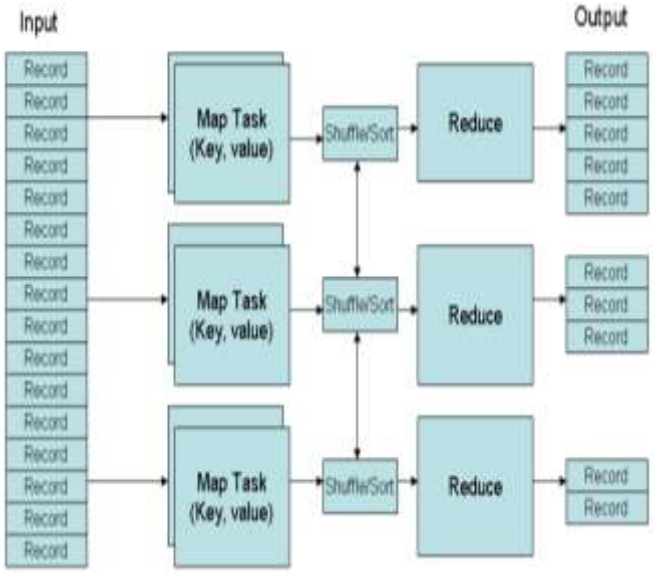

Figure 2. The architecture of the parallel MapReduce Model

To optimize the performance of the clustering program, various techniques such as data compression, speculative execution, and task pipelining can be employed.

6.1 First Phase Clustering

- Population initialization: After receiving the input split each mapper forms the initial population of individuals. Every segment of the chromosome is a centroid. The centroids are selected at random from the split data received. Clustering is carried out for each data point in every chromosome. For this data point, the received data set is assigned to the cluster of the closest centroid.
- ***Fitness evaluation***
- For evaluating fitness, we are computing the Davies-Bouldin [7] index of each individual. Davies-Bouldin index is the ratio of inter-cluster scatter to intra-cluster separation.

The inter-cluster scatter of a cluster Ci is computed as:

$$Si = 1/Ti \sum_{J=1}^{Ti} ||Xj - Ai||p \quad (1)$$

Here, Ai is the centroid point, Xj is the cluster point, Ti is the cluster size, and p is 2 as we are calculating the Euclidian distance. The intra-cluster separation of two centroids Ai and Aj is computed as:

$$M_{ij} = ((\sum_{k=1}^{n} | ak,i - ak,j | p)1/p) \quad (2)$$

Here, k is the number of dimensions of the data point and the value of p is 2. Now the Davies-Bouldin index is

$$DB = 1/N \sum_{i=1}^{N} Di \quad (3)$$

Where Di:

$$Di = max\ j{:}i{\neq}j\ \frac{si+sj}{Mij} \quad (4)$$

- ***Mating & Selection:*** For mating, we are using cross-over and mutation techniques. For cross-over, we are using arithmetic cross-over with 0.7% probability. This generates one offspring from two parents. The centroid of the offspring is the arithmetic average of the corresponding centroid of the parents. For mutation swap mutation is applied with 0.02%. Under swap mutation, we take 9's complement of the data points. The offspring from the older population are selected to populate a new population. For selection, we are using the Tournament selection procedure. Under tournament selection, the individual is selected by performing a tournament based on fitness evaluation among several individuals chosen at random from the population.
- ***Termination:*** A new population as generated replaces the older population. This population would again form a newer population using mating and selection procedures. This whole procedure would be repeated again and again until the termination condition is met. The proposed methodology entails accomplishing the designated number of iterations to attain the desired outcome. The most adept individual from the ultimate population of each mapper is subsequently transmitted to the reducer as the conclusive result. The reducer then performs Second phase clustering on the mapping results of all mappers.

6.2 Second Phase Clustering

- Reducer forms a new chromosome by joining the chromosomes received from each mapper.
- This newly created chromosome is analyzed. For those centroids for which intra-cluster separation is less than the threshold, their respective clusters are merged. For two clusters the threshold is computed as a sum of 20% of the intra-cluster separation and the maximum of the largest distance of a cluster point from the centroid among the two clusters. The Centroid of this newly created cluster is the arithmetic mean of centroids of original clusters. The threshold computation:

$T = (0.2 \times M_{i,j}) +(max\ D_i , D_j)$

Here T is the threshold, $M_{i,\ j}$ is the intra-cluster separation of the clusters C_i and C_j, $D_i\ and\ D_j$ is the distance of farthest Est points of the clusters C_i and C_j from their respective centroids:

- Above stated process is repeated until all centroids of the chromosome have an inter-cluster separation greater than a threshold value.
- The final chromosome contains the location of the centroid of optimal clusters.

7. Evaluation Critariya and Result

We compared the performance of the proposed algorithm with a sequential algorithm. Both the algorithms were executed for a total of 500 iterations with a cross-over probability of 6% and mutation probability of 0.25%. The algorithm under consideration was implemented on a cluster comprising five nodes, each of which was equipped with Hadoop v1.2.1 and operated on an Ubuntu 13.0 platform within a VMware virtual machine. The nodes were allocated a random-access memory (RAM) of 2 GB, a hard disk of 250 GB, and two processing cores each. The hardware configuration of the cluster is shown in Table 1. The sequential algorithm

was executed on a single node with the configuration shown in Table 2. To evaluate performance, we measured the accuracy achieved and total execution time. Execution time was measured using a system clock. The data set used for this experiment represents the differential coordinates of the Europe map. It consists of 169308 instances and 2 dimensions.

Table 1: Hardware Configurations

Nodes	CPU	RAM	Hard Disk
Node1	Intel Core i3	4GB DDR3	520 GB
Node2	Intel Core i3	4GB DDR3	520 GB
Node3	Intel Core i5	6GB DDR3	640 GB
Node4	Intel Core i5	6GB DDR3	1TB
Node5	Intel Core i7	4GB DDR3	1 TB

Table 2: Hardware Specifications

CPU	Intel core i3
RAM	4GB DDR3
Hard Disk	640

Figures 3 and 4 show the result of the total execution time and accuracy achieved for the proposed algorithm and a sequential algorithm. The total execution time is highly reduced by using a parallel genetic algorithm. A speed-up of 80% was observed for PGA with a clustering accuracy of 92%. This shows that the proposed algorithm considerably speeds up the clustering process for big datasets without compromising the accuracy to large extent.

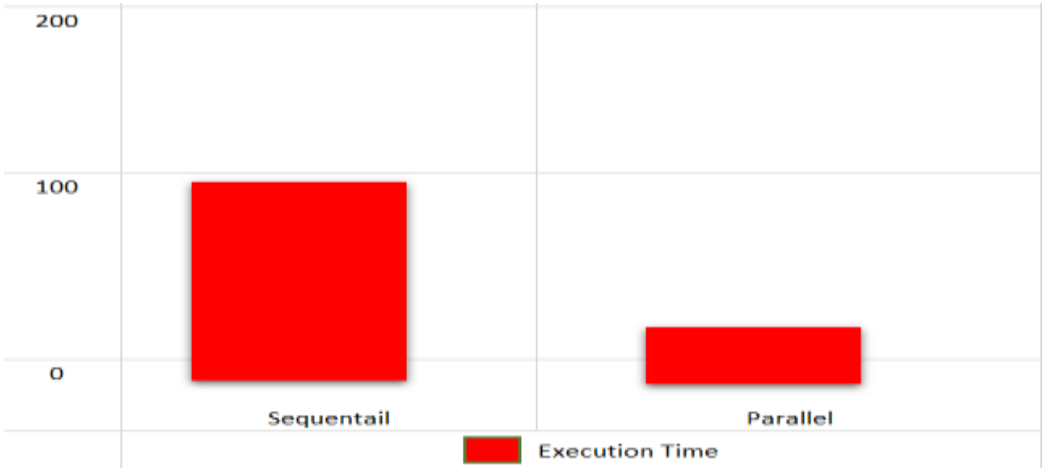

Figure 3 Total Execution Time

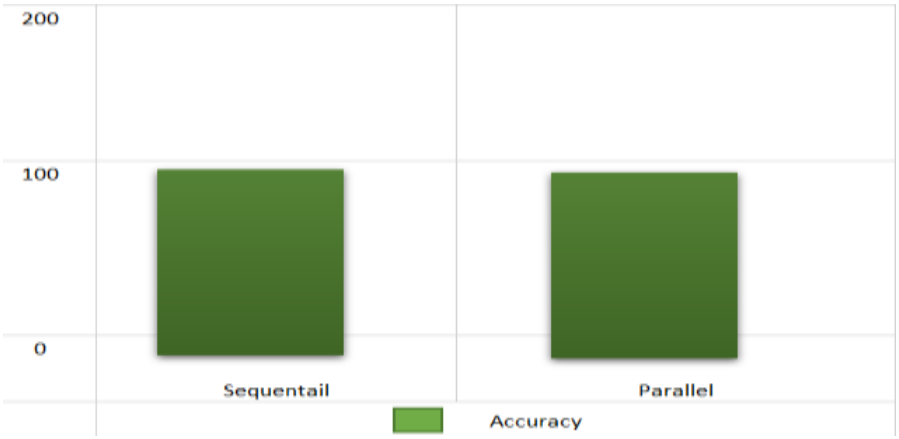

Figure 4 Accuracy Achieved

8. Conclusion

Genetic algorithms are a type of optimization algorithm that mimics the process of natural selection to find the optimal solution to a problem. The combination of clustering and genetic algorithms can be used to partition big data into groups that have similar characteristics and to optimize the clustering process. Hadoop MapReduce is a popular framework for processing large datasets in a distributed environment. By using MapReduce, the clustering of big data can be parallelized and distributed across multiple nodes, enabling faster and more efficient processing. Overall, the use of genetic algorithms in the clustering of big data in Hadoop MapReduce can improve the accuracy and efficiency of the clustering process, leading to better insights and decision-making. However, the effectiveness of this approach will depend on the specific dataset and problem being analyzed.

This Paper Introduces a Novel Technique to Parallelize GA-based clustering. For this, we have Customized Hadoop MapReduce by Implementing Dual Phase Clustering. The speed-up based on Evaluation is presented. In the Future, we Hope to Improve Upon the Accuracy and Enhance the Speed Gains.

Reference

[1]. Elham Azhir, Mehdi Hosseinzadeh, Faheem Khan, and Amir Mosavi: Performance Evaluation of Query Plan Recommendation with Apache Hadoop and Apache Spark. *10*(19), 3517(2022)

[2]. Deepak Kumar, Vijay Kumar Jha: An improved query optimization process in big data using ACO-GA algorithm and HDFS map reduce technique. Springer Science + Business Media, LLC, part of Springer Nature (2020)

[3]. Song, J., Ma, Z., Thomas, R., Ge, Yu.: Energy efficiency optimization in big data processing platform by improving resources utilization. Sustainable Computing: Informatics and Systems 21, 80–89 (2019)

[4]. Panahi, V.; Navimipour, N.J. Join query optimization in the distributed database system using an artificial bee colony algorithm and genetic operators. *Concurr. Comput. Pract. Exp.* 2019, *31*, e5218.

[5]. Pasquale Salza, Filomena Ferrucci. Speed up genetic algorithms in the cloud using software containers. (2019)

[6]. Jain, Anil K., M. Narasimha Murty, and Patrick J. Flynn. "Data clustering: a review." ACM computing surveys (CSUR) 31, no. 3 (1999): 264-323.

[7]. White, Tom. Hadoop: the definitive guide: the definitive guide. " O'Reilly Media, Inc.", 2009.

[8]. Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51, no. 1 (2008): 107-113.

[9]. Mackey, Grant, Saba Sehrish, and Jun Wang. "Improving metadata management for small files in HDFS." In Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on, pp. 1-4. IEEE, 2009.

[10]. Jin, Chao, Christian Vecchiola, and Rajkumar Buyya. "Mrpga: an extension of MapReduce for parallelizing genetic algorithms." In eScience, 2008. eScience'08. IEEE Fourth International Conference on, p

[11]. Song, J., Ma, Z., Thomas, R., Ge, Yu.: Energy efficiency optimization in big data processing platform by improving resources utilization. Sustainable Computing: Informatics and Systems 21, 80–89 (2019)

[12]. Panahi, V.; Navimipour, N.J. Join query optimization in the distributed database system using an artificial bee colony algorithm and genetic operators. Concurr. Comput. Pract. Exp. 2019, 31, e5218.

[13]. Pasquale Salza a,∗ , Filomena Ferrucci. Speed up genetic algorithms in the cloud using software containers. (2019)

[14]. Mahajan, D., Blakeney, C., Zong, Z.: Improving the energy efficiency of relational and NoSQL databases via query optimizations. Sustainable Computing: Informatics and Systems 22, 120–133 (2019).

[15]. Bao, C., Cao, M.: Query optimization of massive social network data based on hbase. In: Proceedings of the IEEE 4th International Conference on Big Data Analytics (ICBDA), pp. 94–97 (2019).

[16]. Sahal, R., Nihad, M., Khafagy, M.H., Omara, F.A.: iHOME: index-based join query optimization for limited big data storage. J. Grid Comput. 16(2), 345–380 (2018)

[17]. Rawat, J.S., Kishor, S., Kumari, M.: A survey on query optimization in cloud computing. Int J Adv Technol Eng Sci 4(10), 2348 (2016)

[18]. Kiranjit Pattnaik, Bhabani Shankar Prasad Mishra: A Review on Parallel Genetic Algorithm Models for Map Reduce in Big Data. International Journal of Engineering Research & Technology (IJERT) ISSN: 2278-0181 IJERTV5IS080400 Vol. 5 Issue 08, August-2016

[19]. Panahi, V.; Navimipour, N.J. Join query optimization in the distributed database system using an artificial bee colony algorithm and genetic operators. Concurr. Comput. Pract. Exp. 2019, 31, e5218.

[20]. Rani, S.; Rama, B. MapReduce with Hadoop for Simplified Analysis of Big Data, International Journal of Advanced Research in Computer Science, May-June 2017, Volume 8, No. 5, ISSN No. 0976-5697, pp. 853-856.

[21]. Joseph, C.W.; Pushpalatha, B., A Survey on Big Data and Hadoop, International Journal of Innovative Research in Computer and Communication Engineering, ISSN(Online): 2320-9801, March 2017, Vol. 5, Issue 3, pp. 5525-5530.

[22]. Ferrucci, F., Salza, P., and Sarro, F. (2016). Using Hadoop MapReduce for Parallel Genetic Algorithms: A Comparison of the Global, Grid and Island Models - Appendix. https: //doi.org/10.6084/m9.figshare.5091898.

[23]. Fu, W., Menzies, T., and Shen, X. (2016). Tuning for Software Analytics: Is It Really Necessary? Information and Software Technology, 76:135–146.

[24]. Salza, P., Ferrucci, F., and Sarro, F. (2016a). Develop, Deploy and Execute Parallel Genetic Algorithms in the Cloud. In Genetic and Evolutionary Computation Conference (GECCO), pages 121–122.

[25]. Salza, P., Ferrucci, F., and Sarro, F. (2016b). Elephant56: Design and Implementation of a Parallel Genetic Algorithms Framework on Hadoop MapReduce. In Genetic and Evolutionary Computation Conference (GECCO), pages 1315–1322.

[26]. Chandra Shekhar Gautam1, Pandey* (2019)" A REVIEW ON GENETIC ALGORITHM MODELS FOR HADOOP MAPREDUCE IN BIG DATA" IJSRS ISSN:0976-3031, Vol.13, Issue-03(E), Pageno771-775, June 2022.