A Parallel Genetic K-Means Algorithm based on the Island Model

Xikang Wang, Tongxi Wang, Hua Xiang, Lan Huang

Abstract—**The K-Means clustering algorithm is widely employed in cluster analysis but is known for its sensitivity to initial center selection and its tendency to become trapped in local optima. These limitations have prompted researchers to explore optimization techniques. The genetic K-Means algorithm (GKA) leverages the optimization capabilities of genetic algorithms to enhance the clustering performance of K-Means. However, this improvement comes at the cost of increased computational complexity, rendering the algorithm less efficient for large-scale datasets. To address these issues, we propose a parallel genetic K-Means algorithm (PGKA) based on the island model. In PGKA, the overall population is partitioned into multiple sub-populations of equal size, each evolving independently on different nodes. The evolutionary process is divided into several generations, with sub-populations exchanging information between generations to preserve diversity. We employ multi-threaded computation to maximize the CPU utilization for the most computationally intensive part of the algorithm, the fitness computation. Additionally, we have modified certain evolutionary operators to better suit the optimization of the K-Means algorithm. Experimental results demonstrate that the proposed algorithm achieves superior clustering accuracy compared to other recent proposed GKA variants. It also significantly enhances computational efficiency relative to the serial GKA and the non-multi-threaded PGKA. Specifically, with a configuration of 16 nodes, PGKA is 12.4 times faster than the serial version when tested on the largest dataset in our experiments. Furthermore, the speedup can be further improved with clusters having more CPU cores.**

Index Terms—**Parallel Computing, MapReduce, Genetic K-Means Algorithm, Clustering**

I. Introduction

The K-Means clustering algorithm [1] is widely used in cluster analysis, which adopts a partitioning method and can customize similarity methods for data features with unsupervised learning and fast execution. However, the K-Means algorithm has the disadvantage of being sensitive to the initial clustering center and the data input sequence, which makes it easy to fall into the local optimum [2].

Manuscript received December 12, 2023; revised June 28, 2024.

This work was supported in part by the National Natural Science Foundation of China (62276032)

Xikang Wang is a postgraduate student of Yangtze University, Jingzhou, Hubei, 434020, China. (e-mail: 2021710547@yangtzeu.edu.cn)

Tongxi Wang is an associate professor of Yangtze University, Jingzhou, Hubei, 434020, China. (corresponding author to provide e-mail: txwang@yangtzeu.edu.cn).

Hua Xiang is a lecturer of Yangtze University, Jingzhou, Hubei, 434020, China. (e-mail: xianghua@yangtzeu.edu.cn).

Lan Huang is an associate professor of Yangtze University, Jingzhou, Hubei, 434020, China. (e-mail: lanhuang@yangtzeu.edu.cn).

To enhance the clustering performance of the K-Means algorithm, researchers have proposed various modifications, such as the K-Means++ algorithm [3] and the kernel K-Means algorithm [4]. Despite these improvements, these algorithms still produce unstable clustering results and remain sensitive to noise. To address these issues, K. Krishna et al. introduced the Genetic Algorithm based K-Means Algorithm (GKA) [5], which offers more stable and accurate clustering outcomes [6]-[7]. GKA has found applications in numerous fields. For instance, Reza Ghezelbash et al. developed a GKA to detect multi-element geochemical anomalies [8], while Qingguo Liu et al. used GKA in an optimized non-dominated sorting genetic algorithm [9].

While GKA can significantly enhance the clustering accuracy of the K-Means algorithm, it also introduces a substantial computational burden. This increased computational demand results in the algorithm's inefficiency when handling large-scale datasets, a notable drawback in the current era of big data. There are fewer recent studies on the parallel model of the GKA, and the recent studies mainly focus on applying the GKA in various fields [10]-[11]. However, the main body of the GKA is the evolutionary operation of the genetic algorithm, in which the K-Means algorithm is mainly used to evaluate chromosome fitness, so the research of parallel GKA can be based on the research of parallel genetic algorithms. Hao-Chun Lu et al. highly integrated the genetic algorithm into Hadoop, used the traveling salesman problem to test it, and achieved good optimization results [12]. Matheus F. Torquato et al. [13] proposed an FPGA algorithm to achieve the full parallel implementation of the genetic algorithm. Regarding the distributed parallel genetic algorithm, E Alba discussed applying parallel computing technology to meta-heuristic algorithms [14]. Carolina Salto et al. compared the efficiency of parallel genetic algorithms on distributed computing frameworks such as Hadoop and Spark [15]. Tomohiro Harada et al. fully summarized and prospected the research status of parallel genetic algorithms [16]. These studies have shown that implementing genetic algorithms in a distributed computation framework can effectively improve computational efficiency. In the study conducted by Filomena Ferrucci et al. [17], the computational efficiency of three different parallel model of genetic algorithms was compared: the global, grid, and island models. Among these models, the island model demonstrated the highest computational efficiency.

Based on the above problems and research, we propose a Parallel Genetic K-Means Algorithm (PGKA), which improves the evolutionary operator to make it more suitable for optimizing K-Means, implements its parallel execution based on MapReduce and the island model, and parallelizes

its fitness computation part (which contains the K-Means operations) using a multithreaded approach. The PGKA shows high computational efficiency and scalability when facing large-scale datasets, offering a novel method for implementing the GKA to efficiently process large-scale datasets.

II. Background

A. K-Means Algorithm

Algorithm 1 K-Means Algorithm

Input: Dataset $\{x_1, x_2, \ldots, x_n\}$, Number of clusters k, max iterations T

Output: Clustering result formed by k clusters

1: Initialize: Select k random datapoints as centers $M = \{m_1, m_2, \ldots, m_k\}$
2: $t \leftarrow 0$
3: **while** assignments change or $t < T$ **do**
4: Assign x_i to nearest C_j: $\text{argmin}_j \|x_i - m_j\|^2$, where $j \in$ [1,k]
5: Update centers: $m_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$
6: $t \leftarrow t + 1$
7: **end while**

Fig. 1. Process of the K-Means algorithm.

The K-Means algorithm iteratively assigns data points to the closest clusters and updates the cluster centroids until a stopping condition is reached, resulting in a dataset partitioned into K distinct clusters.

The goal of the classical K-Means algorithm is to minimize the sum of squares of the distance between the data points in each cluster and the center point, that is, minimize (1):

$$\sum_{j=1}^{k} \sum_{x_i \in C_j} \left\| x_i - m_j \right\|^2 \tag{1}$$

The process of the K-Means algorithm is shown in Fig. 1.

B. Genetic Algorithm

Algorithm 2 Genetic Algorithm

Input: Population size n, max iterations T, crossover probability p_c, mutation probability p_m

Output: Best individual(solution) x_{best}

1: Initialize population: Generate n random individuals x_i
2: $t \leftarrow 0$
3: **while** $t < T$ **do**
4: fitness evaluation: Calculate fitness $f(x_i)$ for each individual x_i
5: selection: Calculate the selection probability of each chromosome $p_i = \frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)}$, then randomly select according to these probabilities.
6: crossover: randomly select 2 chromosomes x_i and x_j, Whether to crossover is determined according to p_c, randomly select a crossover point k to combine the first k elements of x_i with the last l-k elements of x_j to form a new chromosome, where l is the chromosome length.
7: mutation: For each gene in each chromosome, Whether to crossover is determined according to p_m, the mutated gene is set as a random number
8: Update population with offspring
9: $t \leftarrow t + 1$
10: **end while**

Fig. 2. Process of the Genetic algorithm.

The genetic algorithm is an optimization method inspired by natural selection, where solutions evolve over generations to solve complex problems. It utilizes techniques such as mutation, crossover, and selection to generate increasingly better solutions to a given problem [18]. Genetic algorithms are commonly used to address combinatorial optimization problems, such as the traveling salesman problem and function optimization [19].

The performance of genetic algorithms depends on parameter settings, such as population size, crossover probability, and mutation probability. The process of the genetic algorithm is shown in Fig. 2.

C. Genetic K-Means Algorithms

To address the sensitivity of the K-Means algorithm to initial centroids and its tendency to fall into local optima, researchers have developed various GKAs. D. Mustafi et al. applied a hybrid optimization method combining differential evolution and genetic algorithm for text clustering, optimizing both the number of clusters and the initial centroids [20]. Dharmendra K Roy et al. proposed a GKA that effectively handles mixed numeric and categorical data by redefining the cluster center [21]. Xiao et al. introduced KMQGA, a quantum-inspired genetic algorithm for K-Means clustering [22].

In the research on the GKAs, researchers have proposed various chromosome encoding forms, such as real-number encoding [7] and index encoding [6]. Each encoding form has its advantages and disadvantages in different scenarios. Due to the differences in chromosome encoding, their evolution methods also vary. However, the common goal of the GKAs is to find a set of optimal centroids for the K-Means algorithm in the solution space and use a clustering evaluation metric to assess the capability of the chromosomes in the GKAs to solve the problems.

D. Hadoop MapReduce

Hadoop MapReduce is a distributed computing framework within the Hadoop ecosystem, widely used for processing large-scale datasets. The core concept of this framework is to break down a task into smaller parts and execute these parts in parallel across different nodes. Inspired by Google's MapReduce research [23], the Hadoop MapReduce framework adopts Google's approach to processing large-scale datasets.

The Hadoop MapReduce process is as follows:

Splitting: The input data is divided into multiple equal-sized blocks (default size is 64 MB). Each 'Input Split' object represents a block.

Map Tasks: Each input split is assigned to nodes in the cluster for processing. The map task reads data from the input split and transforms it into key-value pairs. Each key-value pair then executes a map function, producing a new key-value pair as output.

Shuffle: Upon completion of all map tasks, the output data is sorted by key and directed to the corresponding reduce task for each key.

Reduce Tasks: Reduce tasks receive key-value pairs from the map task outputs and execute the reduce function. This function aggregates each key with its associated values, generating an output key-value pair. The results from the reduce tasks are then written to the Hadoop Distributed File System (HDFS).

In summary, Hadoop MapReduce decomposes a massive

task into multiple smaller tasks, executing them in parallel across multiple nodes, thus making the processing of large-scale datasets efficient, fast, and reliable.

E. Fork Join

In modern computing, where multicore processors are prevalent and the demand for large-scale data processing is escalating, the significance of efficient parallel computing frameworks is paramount. Among these, the Fork/Join framework offers a powerful solution for tackling parallel computing challenges in Java programming. Conceived by Doug Lea and introduced in Java 7 [24], with subsequent enhancements in Java 8, this framework embodies a parallel model grounded on task decomposition and work-stealing principles. Such a model is adept at harnessing the computational might of multicore processors, thereby elevating both the performance and scalability of applications.

At its core, the Fork/Join framework is inspired by the divide-and-conquer strategy. It commences by breaking down large tasks into manageable subtasks. These subtasks are then executed in parallel, following which their results are amalgamated to form the ultimate outcome. This method of breaking tasks down facilitates a more efficient use of multicore processors' computing resources and significantly boosts program execution speed.

Central to the Fork/Join framework are the operations known as 'fork' and 'join.' The 'fork' operation splits a sizeable task into smaller, more manageable subtasks, dispatching these subtasks to a work queue for execution. Upon completing a task, a thread may engage in 'work-stealing' by appropriating tasks from the queues of other threads, thus maintaining load balance and maximizing the utilization of the processor's computing capabilities.

Moreover, the Fork/Join framework extends various methods for task execution control and results in retrieval, such as 'invoke()' and 'join().' These provisions empower developers to swiftly craft parallel computing code while minimizing the intricacies associated with thread management and synchronization, thereby streamlining the development process.

III. Proposed PGKA

A. Evolutionary operators

In the GKA proposed by Mahnaz Mardi et.al [6], a chromosome encodes each feature vector of the initial center as a single gene bit. That is, assume the dataset is $D=\{d_1,d_2,...,d_n\}$ where $d_i=\{v_1,v_2,...,v_m\}$, m is the feature dimension, for the center point set $C=\{d_1,d_2,...,d_k\}$,where k is the number of clusters, there is a chromosome $I=\{v_1,v_2,...,v_{m*k}\}$. The GKA proposed by Shruti Kapil et al. [7] uses 1/0 to encode chromosomes; that is, for the dataset D the encoding form of the chromosomes is $I=\{g_1,g_2,...,g_n\}$, which represents a solution. The chromosome length is equal to the length of the dataset, where g_i has only two possible values: 0 or 1, and when g_i is 1, d_i in the dataset is one of the initial centers. Both genetic K-Means algorithms adopt single-point crossover.

Considering the IO overhead on Hadoop MapReduce, our model draws on the chromosome coding method of the research by Shruti Kapil et al. [7] and improves some evolutionary operators based on it. The evolutionary operators in our PGKA are as follows.

1. Initialization: For each chromosome $I=\{g_1,g_2,...,g_n\}$, randomly select K data points as the initial centers., g_i = 1 indicates that d_i in the dataset is one of the initial centers.

2. Crossover: The proposed algorithm performs a double-point crossover. Compared with single-point crossover, double-point crossover can better retain the information of excellent individuals. Single-point crossover can only exchange chromosome fragments at one crossover point, which easily destroys excellent individuals' overall structure and information. In contrast, double-point crossover can exchange information at two points, which can retain the important information of excellent individuals and better avoid premature convergence [25]. The process involves selecting two parent chromosomes and determining whether to perform the crossover operation based on the crossover probability, if yes, the selected two parental chromosomes are divided into four fragments, and the four chromosome fragments are spliced according to the selected two points to form two new chromosomes, which are added to the next generation population as offspring chromosomes.

The number of offspring chromosomes generated in the crossover stage is 50 % of the number of chromosomes in the parent generation. Moreover, because the number of '1' in the offspring chromosome generated after the crossover operation may be different from the parent chromosome, the mutation operation at the gene level is performed for those offspring chromosomes where the number of '1' is not the same as the parent chromosomes. That is, if the number of '1' is less than the parent chromosome. In these single/multiple genes, the value is '0' are randomly selected for inversion to '1', and vice versa, until the number of '1' is the same as the parent chromosome. The crossover operator is as shown in Fig. 3.

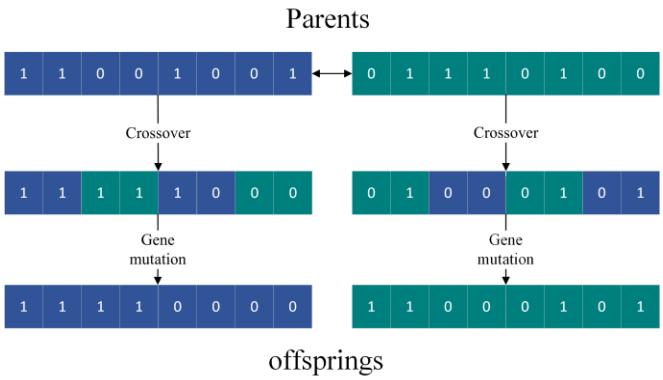


Fig. 3. Crossover operator in the proposed PGKA

3. Mutation: Unlike the gene-level mutation in crossover, the proposed algorithm uses a chromosome-level mutation operator. The traditional mutation method described by Shruti Kapil et al. [7] relies on the mutation probability of each gene, where the operation flips the gene's value. However, in this encoding form, the number of '1's corresponds to the K value in the K-Means algorithm. If the mutation operation is performed separately for each gene, the number of '1' needs to be kept unchanged, which increases the complexity of the algorithm and causes additional

inversion operations that do not follow the mutation probability: flip some positions to keep the K value unchanged just like the mutation operation in Crossover. Therefore, in the proposed GKA, the chromosome selected according to the mutation probability will be randomly reset to a new chromosome. The chromosomes from parents and offspring are all involve in mutation. The mutation operator is as shown in Fig. 4.

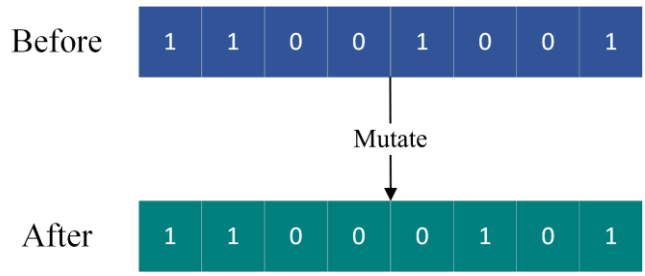


Fig. 4. Mutation operator in the proposed PGKA.

4.Fitness evaluation: a small amount of K-Means operation is performed after decoding the chromosome, and compute the Calinski-Harabasz Index (CHI) [26] for the clustering result, using it as the chromosome's fitness. The formula is shown in (2).

$$CHI = \frac{Tr(B)}{Tr(W)} * \frac{N-k}{k-1} \tag{2}$$

Among them, $Tr(B)$ is the covariance matrix between the clusters, $Tr(W)$ is the covariance matrix within the cluster, N is the number of instances in the dataset, k is the number of the clusters, the larger the CHI, the better the result.

5.Selection: All chromosomes, including parents and children, are involved in the selection. The elitism operator [27] is performed in the selection. A small number of excellent individuals in the population are retained and directly entered into the next generation, and the remaining individuals are selected using the roulette selection. The process of the selection is as follows: First, calculate the probability of each chromosome entering the next generation as (3).

$$P(x_i) = \frac{f(x_i)}{\sum_{j=1}^{N}(x_j)} \tag{3}$$

Among them, $f(x_i)$ is the fitness of the chromosome x_i, N is the current population size. Then, calculate the cumulative probability sequence $Q = \{q_1, q_2, ..., q_N\}$, where q_i is as (4).

$$q_i = \sum_{j=1}^{i} P(x_j) \tag{4}$$

Then, generate a random number $r \in [0,1]$. traverse sequence Q, until q_i is found that fits $q_{i-1} < r < q_i$. The selection restores the population size that becomes larger after the crossover to the initial population size to keep the number of populations in each iteration unchanged.

The process of the serial version of the proposed PGKA(SGKA) is as Fig. 5.

Algorithm 3 SGKA

Input: Number of clusters k, population size n, max iterations T, crossover probability p_c, mutation probability p_m, dataset D, Elite rate E

Output: clustering result

1: Initialize population: Generate n random chromosomes x_i according to D.
2: $t \leftarrow 0$
3: **while** $t < T$ **do**
4: Crossover: Perform crossover according to p_c
5: Mutation: Perform mutation according to p_m
6: Fitness evaluation: Calculate fitness $f(x_i)$ of each chromosome using K-Means algorithm and CHI
7: Elitism: Select $n * E$ chromosomes with the highest fitness to enter the next iteration.
8: Selection: Calculate selection probabilities p_i for each chromosomes, then randomly select based on these probabilities
9: $t \leftarrow t + 1$
10: **end while**
11: Get the chromosome with the highest fitness then decode it into a set of initital centers
12: Run K-Means algorithm using the initital centers
13: Output the clustering result

Fig. 5. Process of the SGKA.

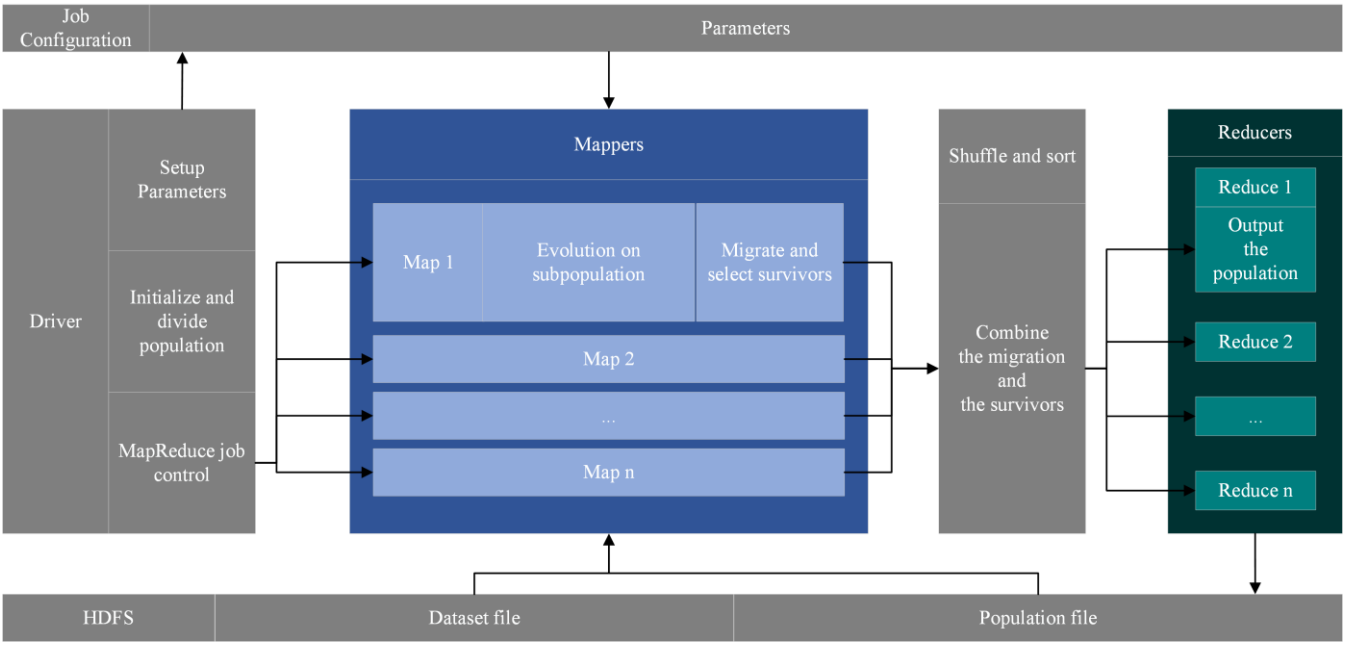


Fig. 6. A generation in the PGKA, the number of generations depends on the parameter.

B. PGKA on Hadoop MapReduce

In the study by Filomena Ferrucci et al. [17], various parallel Genetic Algorithm models are compared using Hadoop MapReduce. Among these models, the Genetic Algorithm based on the island model demonstrates clear advantages in computational efficiency compared to global and grid models. Building upon this insight, we propose a Parallel Genetic Algorithm (PGKA) implemented on Hadoop MapReduce, leveraging the island model as its foundation.

In the island model, each subpopulation evolves independently within its own environment, thereby ensuring diversity within the overall population. This autonomy allows each subpopulation to potentially explore different evolutionary paths [28]. However, to prevent subpopulations from becoming trapped in local optima, it is crucial to facilitate information exchange between them. In the island model, evolutionary tasks are organized into multiple generations, during which subpopulations exchange information at regular intervals.

To maximize the computational resources of the clusters, the Fork/Join framework is employed in fitness evaluation. This framework optimizes CPU utilization across nodes, thereby enhancing computational efficiency.

The process of a generation in PGKA is shown in Fig. 6.

1) Driver

The Driver begins by configuring the parameters and initializing the population, then submitting MapReduce tasks to the processing framework. In scenarios involving multiple generations, the Driver adopts an iterative approach, submitting multiple MapReduce tasks. Process of the Driver is illustrated in Fig. 7.

Algorithm 4 Driver of the PGKA

Input: Dataset file path, number of clusters k, population size n, number of migrations c, max iterations T, crossover probability p_c, mutation probability p_m, Elite rate E
Output: clustering result

Setup parameters: number of nodes in the distributed cluster a, iterations in each generation $T_s \leftarrow \frac{T}{c}$, subpopulation size $p \leftarrow \frac{n}{a}$, Flag F
Initialize population: Randomly generate n chromosomes, divide them into a subpopulations, then write them and their index I into HDFS
Load Dataset file to the distributed cache
$t \leftarrow 0$
$F \leftarrow False$
while $t < c$ **do**
    if $t == c$ **then**
        set F = true
    end if
    Submit PGKA MapReduce job
    $t \leftarrow t + 1$
end while
Read final population and their fitness from HDFS
Decode the best chromosome into a set of initial centers
Run the K-Means algorithm using the centers

Fig. 7. The process of the Driver of the PGKA

To optimize clustering operations within the Map phase, the dataset file is pre-loaded into the distributed cache to leverage data locality, thereby enhancing efficiency. Upon the completion of all MapReduce jobs, the global best chromosome is selected from the best chromosomes of various subpopulations. This selected chromosome is then decoded into a series of initial center points for the K-Means algorithm, which is subsequently executed using these points.

2) Map

In the Maps, evolutionary operators are applied to each subpopulation. Following the division of the population by the Driver, each Map processes a sub-population, executing the evolutionary operators on its designated subpopulation. The process of the Maps is as shown in Fig. 8.

Algorithm 5 Map of the PGKA

Input: $< K1, V1 >$, $K1$ is the text offset and $V1$ is the subpopulation
Output: migrants and survivors, or the best chromosome

Read the parameters : k, T_s, p_c, p_m, p, E, a, F
Set the subpopulation index I and subpopulation information according to $V1$
evaluate fitness using Fork-Join framework
$t \leftarrow 0$
while $t < T_s$ **do**
    Perform crossover on subpopulation according to p_c
    Perform mutation on subpopulation according to p_m
    Evaluate fitness using Fork-Join framework according to k
    Perform selection on subpopulation according to E
    $t \leftarrow t + 1$
end while
if F **then**
    select $E * p$ chromosomes as the migrants
    $t \leftarrow 0$
    for $t < a$ **do**
        if t!=I **then**
            Output $< K2, V2 >$, $K2$ is the index of other subpopulations and $V2$ is the migrants
        end if
    end for
    Drop $(a-1) * p * E$ chromosomes with the lowest fitness
    Output $< K2, V2 >$, $K2$ is the index of this subpopulation and $V2$ is the survivors
else
    Output $< K2, V2 >$, $K2$ is a fixed value, $V2$ is the best chromosome and its fitness
end if

Fig. 8. The process of the Map of the PGKA, the variables here are the same as those in Driver

After completing the evolutionary task for a generation, if F equals false, this indicates that additional generations require processing. Each Map then duplicates and dispatches migrants to other subpopulations, subsequently removing the chromosomes of the lowest fitness to make room for incoming migrants from other subpopulations before finally outputting the survivors. Conversely, if F equals true, signaling the final generation, each subpopulation outputs solely the best chromosome and its fitness. Consequently, $K2$ is set to a fixed value, ensuring a singular reducer in the subsequent reduce phase to minimize overhead.

The fitness evaluation is the most computationally intensive part of the algorithm; we use a multi-thread framework to improve the computational efficiency. The method is similar to MapReduce. Both of them split large computing tasks into multiple small computing tasks for processing, which can efficiently use the computing power of the CPUs in the distributed clusters, the process of fitness evaluation in PGKA is as shown in Fig. 9.

3) Reduce

Reduce receives V2 values with the same K2 value as an

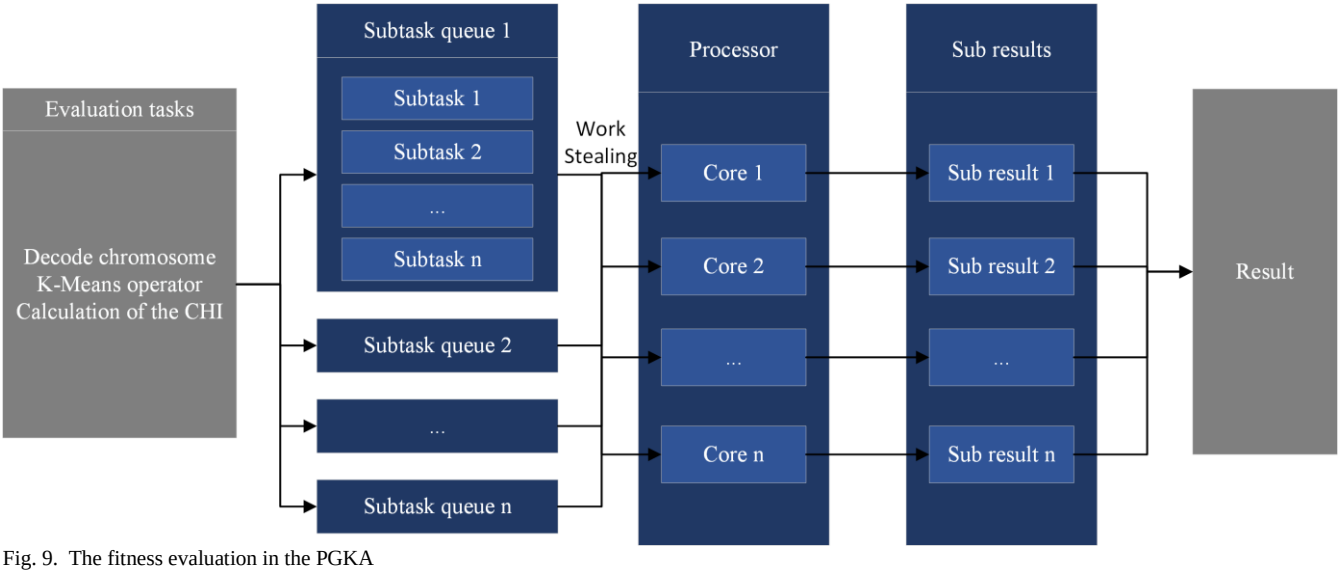


Fig. 9. The fitness evaluation in the PGKA

iterator. There are multiple Reducers processing tasks in the MapReduce job in the non-last generation, and only one in the last generation. Reducer's task is to write the V2 as the input format of the Map to HDFS for the next generation, or output the global best chromosome for the Driver, the process is as shown in Fig. 10.

Algorithm 6 Reduce of the PGKA

Input: $< K2, V2 >$, $K2$ is the subpopulation index ,or a fixed value; $V2$ is the surviours and migrants ,or the best chromosomes of the subpopulations.

Output: chromosomes formatted as the input format of the Map

1: **for** c in $V2$ **do**
2: Output c in the same format as $V1$
3: **end for**

Fig. 10. The process of the Reduce of the PGKA

IV. Experiments

A. Datasets

The datasets utilized in this study are obtained from the UCI Machine Learning Repository, with different datasets chosen for two separate experiments: one focusing on clustering accuracy and the other on computational efficiency. This differentiation is based on the premise that smaller datasets facilitate numerous executions, yielding more precise outcomes. Conversely, larger datasets, which require more processing time, are better suited for assessing the computational efficiency of the algorithms.

TABLE I
DATASETS USED IN THE EXPERIMENTS

Name	Instances	Features
Seeds	345	6
Wine	440	8
Wholesale customers	178	13
Toxicity	171	1203
Har70	2259597	6
Cover type	581010	54

As shown in Table I, the Seeds, Wine, and Wholesale customers datasets are employed for the clustering accuracy experiment, while the remaining datasets are reserved for evaluating computational efficiency. Additionally, two datasets from the experiments are chosen to examine the sensitivity of the algorithms to parameter changes.

B. Distributed clusters

The experiment is conducted on a distributed cluster comprising up to 16 nodes, with the specifics of the software and hardware detailed in Table II.

TABLE II
SOFTWARE AND HARDWARE OF THE DISTRIBUTED CLUSTERS

Architecture	64bit
CPU cores	4
Ram	4GB
Storage	20GB
Network	LAN 1Gbps
OS	CentOS 7
Java	1.8.0
Hadoop	3.1.3

To clarify the identification of cluster configurations varying by node count within the experiment, the naming conventions for these configurations, corresponding to the different numbers of nodes, are shown in Table III.

TABLE III
CONFIGURATIONS OF DISTRIBUTED CLUSTERS WITH VARYING NUMBERS OF COMPUTER NODES

Configurations	Master	Worker(s)	Total
C1	1	0	1
C2	1	1	2
C4	1	3	4
C8	1	7	8
C16	1	15	16

C. Clustering accuracy

1) Evaluation metrics

The results are evaluated using four different clustering evaluation indices: Davies-Bouldin Index (DBI) [29], Silhouette Coefficient (SC) [30], Calinski-Harabasz Index (CHI), and SSE.

The formula for DBI is shown in (5).

$$DBI = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left(\frac{S_i + S_j}{M_{ij}}\right) \quad (5)$$

Among them, k is the K value in K-Means algorithm, S_i is the average distance between the data points in the cluster and the center of the cluster, M_{ij} is the distance between the two cluster centers, a smaller DBI indicates a better result..

The formula for SC is shown as (6) and (7).

$$SC = \frac{1}{N}\sum_{i=1}^{N} s_i \quad (6)$$

$$s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}} \quad (7)$$

Among them, a_i represents the average distance of a data point to other data points within the cluster, while b_i represents the average distance from the data point to the nearest other cluster. A larger SC indicates a better result.

The formula for SSE is as (8).

$$SSE = \sum_{j=1}^{k}\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\right) \quad (8)$$

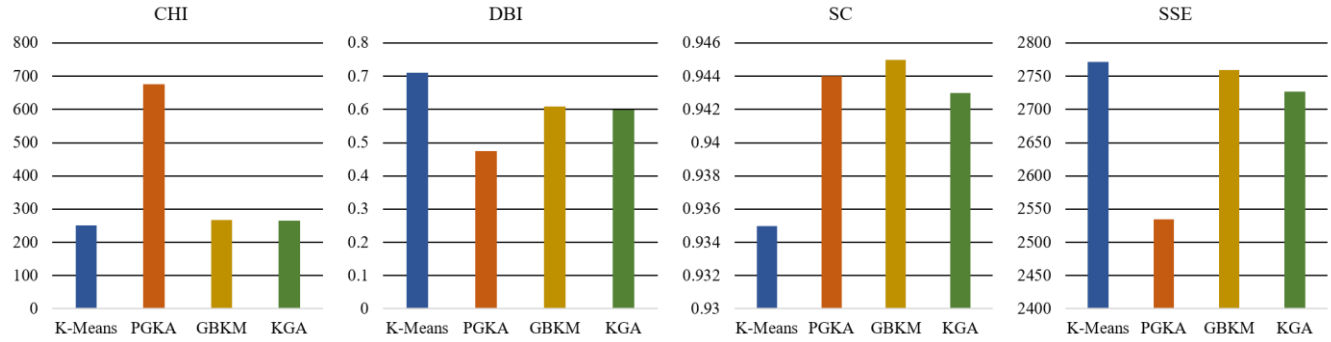
Among them, n is the number of datapoints in the cluster, $\bar{x}$ is the center of the cluster, k is the number of clusters. The smaller the SSE, the better the result.

2) Clustering accuracy

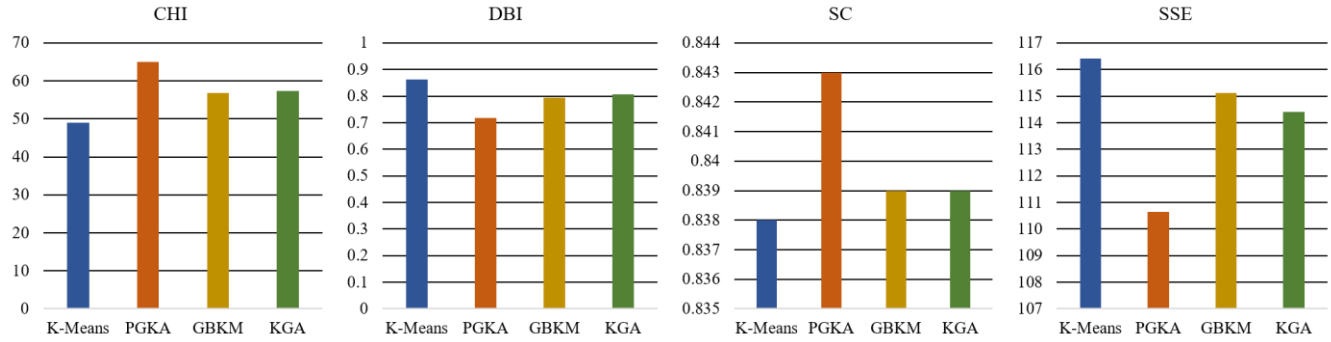
This experiment compares the clustering accuracy of K-Means, KGA [7], GBKM [6], and PGKA. The

TABLE IV
PARAMETERS OF THE ALGORITHMS USED IN THE CLUSTERING ACCURACY EXPERIMENT

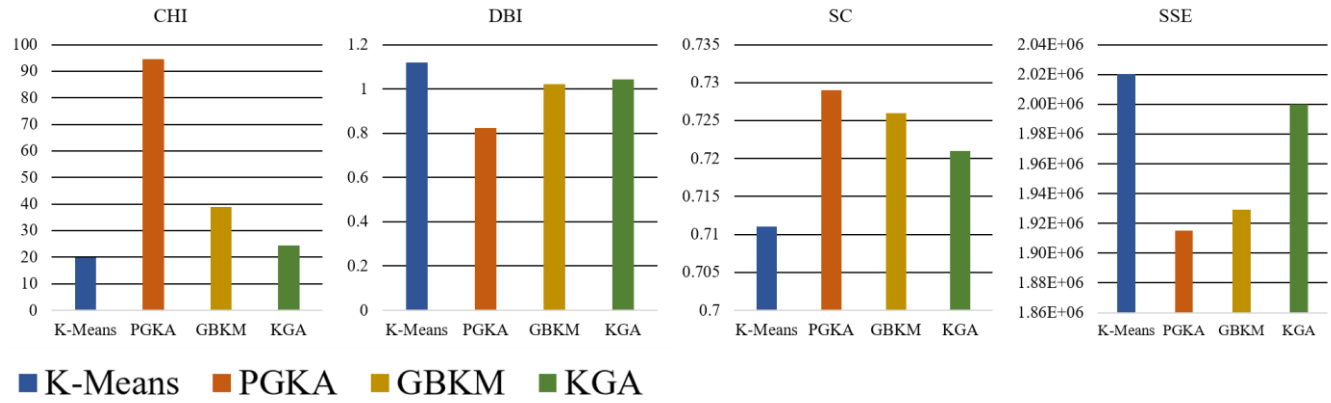
Algorithms	K-Means	KGA	GBKM	PGKA
K	30	30	30	30
Iterations	100	100	100	100
Population	\	200	200	200
Crossover prob.	\	0.5	0.5	0.5
Mutation prob.	\	0.05	0.05	0.05
Elite rate	\	\	\	0.05
Number of migrations	\	\	\	0



(a) The results on wine dataset



(b) The results on seeds dataset



(c) The results on wholesale customers dataset

Fig. 11. Clustering metrics of the algorithms on different datasets

experimental parameters are shown in Table IV. Given that population splitting can influence clustering accuracy, the PGKA algorithm was executed on a single node to ensure the reliability of the experimental results. Additionally, the parameters for PGKA are aligned with those of KGA and GBKM in terms of the common parameters.

Employing a larger K value in the clustering accuracy experiment serves to broaden the search space and elevate the optimization challenge, thereby more effectively highlighting the performance disparities between the algorithms.

All algorithms were executed 30 times under each configuration, and the value of each metric is recorded as the outcome and takes average as the result.

The results, shown in Fig. 11, reveal that PGKA surpasses both KGA and GBKM across all metrics and datasets. Additionally, all GKAs outperform K-Means on every evaluated metric.

D. Computational efficiency

In this experiment, we compare the execution times of SGKA (the serial version of PGKA), nm-PGKA (the PGKA algorithm without the multi-threaded framework), and PGKA. The parameters used in the computational efficiency experiment are shown in Table V.

TABLE V
PARAMETERS OF THE ALGORITHMS USED IN THE COMPUTATIONAL EFFICIENCY EXPERIMENT

Algorithm	K-Means	KGA	GBKM	PGKA
K value	5	5	5	5
Iterations	100	100	100	100
Population size	\	200	200	200
Crossover prob.	\	0.5	0.5	0.5
Mutation prob.	\	0.05	0.05	0.05
Elite rate	\	\	\	0.05
Number of migrations	\	\	\	3

PGKA, which is built upon the MapReduce framework, follows the same procedure for executing evolutionary operators on a single node as SGKA. However, the implementation of the MapReduce framework introduces additional overheads. Therefore, to ensure a more accurate and fair comparison, the SGKA algorithm is utilized in place of PGKA with a single node for the experiment.

1) Execution time

The nm-PGKA and PGKA algorithms were executed 10 times across all datasets and configurations. SGKA was also executed 10 times on all datasets using the master node, with averages computed for the results. These results are presented in Fig. 12.

The results show that nm-PGKA's execution time exceeds that of SGKA on the Toxicity dataset under the C2 configuration. In contrast, for other scenarios, the execution times for the MapReduce algorithms (nm-PGKA and PGKA) are shorter than that of SGKA, with PGKA consistently outperforming nm-PGKA across all cases.

The results indicate that multi-thread computing significantly enhances PGKA's performance. The execution time of nm-PGKA is longer than that of SGKA on the Toxicity dataset and C2 configuration because the overhead of the MapReduce framework negates the computational advantage of parallel computing on the two nodes. On larger data sets, time is mainly consumed in computing; multi-node parallel computing brings more advantages in computing, so the execution time of nm-PGKA is less than that of the SGKA. The speedup rate of the two MapReduce algorithms on the SGKA algorithm is shown in the Fig. 13.

With the increase in the number of nodes, the speedup rate of the two MapReduce algorithms increases, showing good scalability. The speedup rate is as (9).

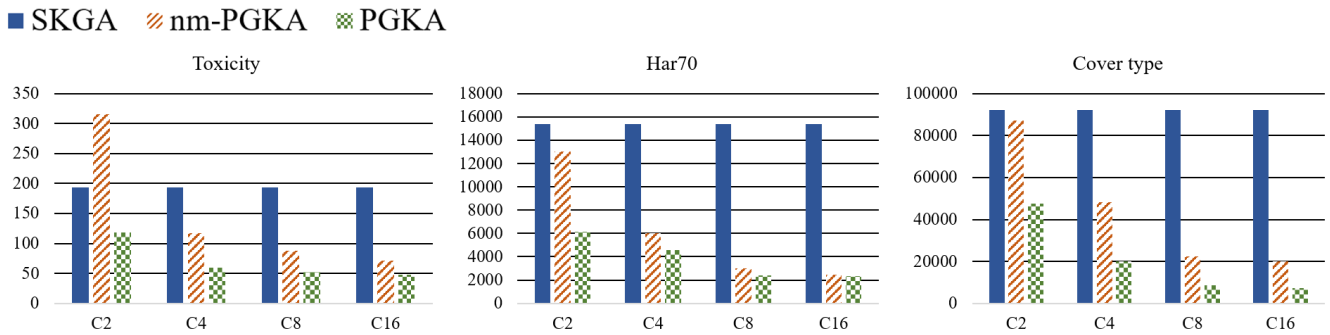
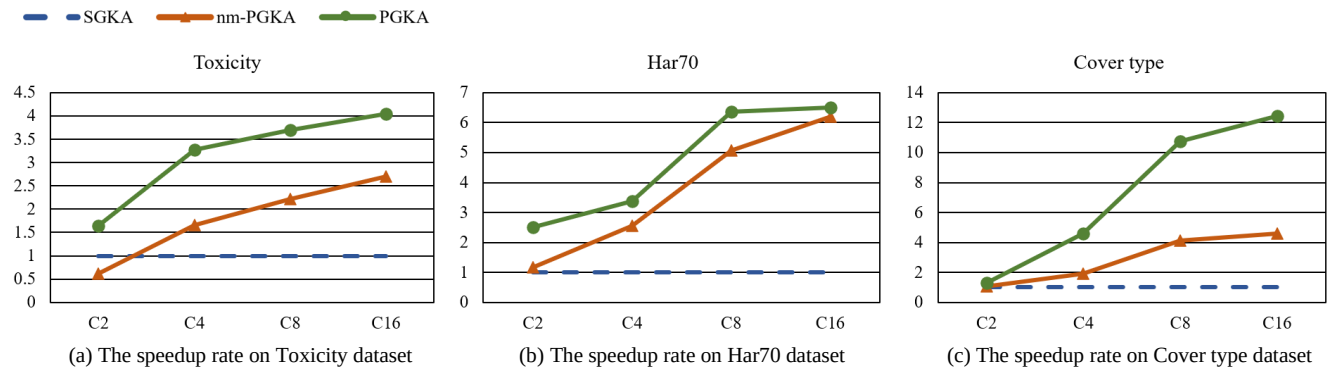


Fig. 12. Execution time of the algorithms on different datasets



(a) The speedup rate on Toxicity dataset (b) The speedup rate on Har70 dataset (c) The speedup rate on Cover type dataset

Fig. 13. Speedup rate of the 2 MapReduce algorithms on the SGKA

$$Speedup = \frac{T_s}{T_p} \quad (9)$$

Among them, T_s represents the execution time of the serial algorithm, while T_p denotes the execution time of the parallel algorithms.

2) Efficiency under different CPU cores

In this experiment, we compare the execution times of SGKA, nm-PGKA, and PGKA across various CPU configurations within same clusters. This comparison aims to determine the sensitivity of each algorithm to CPU configuration variations.

The experiment was conducted on virtual machines hosted by VMware Workstation, which allowing easy modification of the number of CPU cores. Despite the difference in CPU models compared to the previous experiments, all other configurations remained consistent with those shown in Table II. The different CPU core configurations are shown in Table VI, with the Toxicity dataset selected for this evaluation.

TABLE VI
CONFIGURATIONS WITH DIFFERENT NUMBER OF CPU CORES

Configuration	CPU cores	Total nodes
V1	1	4
V2	2	4
V3	3	4
V4	4	4

Each algorithm was executed 10 times under all configurations, with the average results presented in Fig. 14.

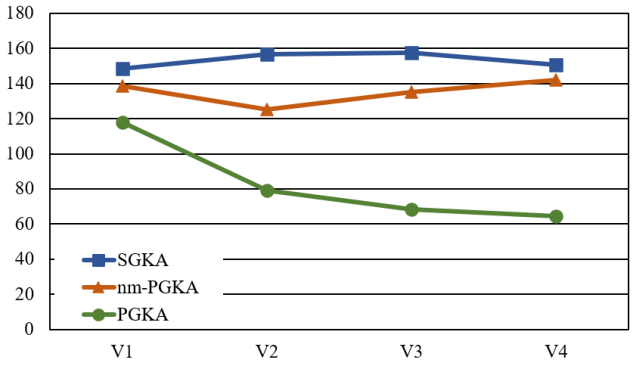


Fig. 14. Execution time on the configurations with different CPU cores of the algorithms

The results indicate that both SGKA and nm-PGKA exhibit low sensitivity to changes in the number of CPU cores. Conversely, the execution time of PGKA decreases as the number of CPU cores increases, demonstrating PGKA's robust scalability across both distributed clusters and CPU configurations. This shows PGKA's ability to leverage additional computational resources to enhance performance effectively.

E. Parameter sensitivity

In this experiment, we examine how PGKA's performance is influenced by specific parameters, notably the impact of population splitting on clustering accuracy and the effect of migration numbers on both clustering accuracy and computational efficiency.

1) Number of subpopulations

In PGKA, the number of subpopulations matches the number of nodes in the cluster. Therefore, PGKA was executed on clusters with varying node counts to assess the impact of population division on clustering accuracy.

The parameters used in this experiment are the same as those in Table IV. The environment is consistent with Table II and Table III, and the seeds dataset was used. PGKA was executed 30 times under all configurations, with the average CHI taken as the result. The results are shown in Fig. 15.

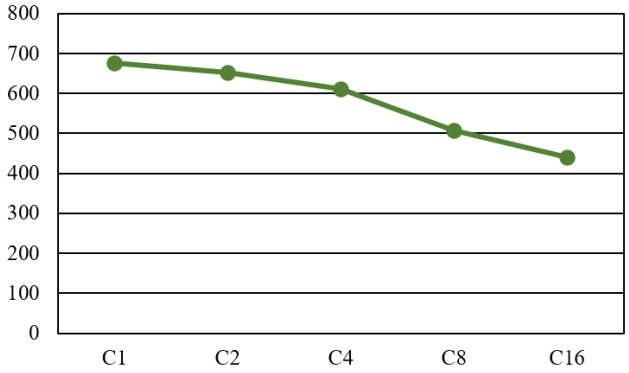
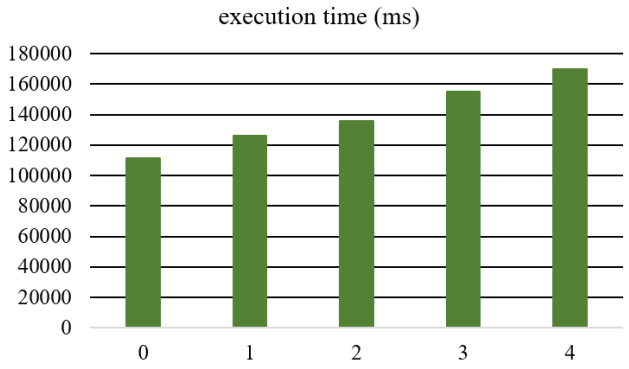


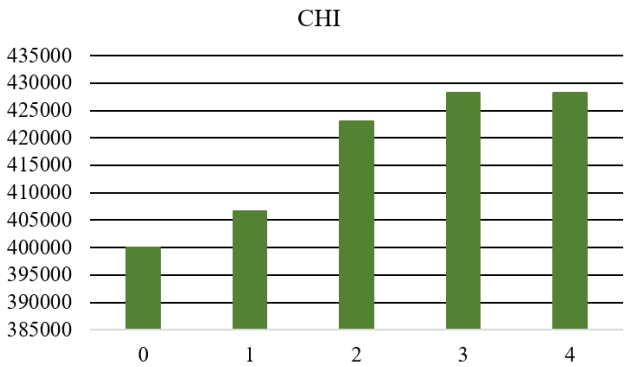
Fig. 15. The clustering accuracy of the PGKA on different number of nodes on the seeds dataset

The results show that clustering accuracy gradually decreases as the number of nodes increases. This indicates that to achieve better clustering results with more nodes, the population size or the number of migrations should be appropriately increased.

2) Number of migrations



(a) The execution time of the PGKA with different number of migrations



(b) The CHI of the PGKA with different number of migrations

Fig. 16. The CHI and the execution time of the PGKA with different number of migrations

TABLE VII
RESULTS OF THE WILCOXON TESTS

Dataset	Conf.	SGKA & nm-PGKA		nm-PGKA & PGKA		SGKA & PGKA	
		Statistic	P	Statistic	P	Statistic	P
Toxicity	C2	0	0.0019	0	0.0019	0	0.0019
	C4	0	0.0019	0	0.0019	0	0.0019
	C8	0	0.0019	0	0.0019	0	0.0019
	C16	0	0.0019	0	0.0019	0	0.0019
Har70	C2	0	0.0019	0	0.0019	0	0.0019
	C4	0	0.0019	0	0.0019	0	0.0019
	C8	0	0.0019	0	0.0019	0	0.0019
	C16	0	0.0019	0	0.0019	11.0	0.1054
Cover type	C2	9.0	0.0644	0	0.0019	0	0.0019
	C4	0	0.0019	0	0.0019	0	0.0019
	C8	0	0.0019	0	0.0019	0	0.0019
	C16	0	0.0019	0	0.0019	0	0.0019

The number of migrations affects both clustering accuracy and computational efficiency. In this part of the experiment, PGKA is tested on the C4 configuration and the Toxicity dataset. Except for the number of migrations, all other parameters are the same as those in Table IV. PGKA with all parameters is executed 10 times, and the average is taken as the result. The results are shown in Fig. 16.

The results indicate that as the number of migrations increases, both the execution time and clustering accuracy of PGKA increase, suggesting that this parameter needs to be adjusted according to different situations.

F. Further analysis on the PGKA

1) Wilcoxon test

To ensure the reliability of the experimental results, a Wilcoxon test was conducted on the outcomes of the computational efficiency experiment. The Wilcoxon test results are detailed in Table VII.

A Wilcoxon statistic of 0 implies that all observed differences consistently favor one algorithm over another, indicating that one algorithm invariably executes faster. The p-value, on the other hand, measures the likelihood of observing the current or more extreme results under the assumption that the null hypothesis is true. In this context, the null hypothesis posits that there is no significant difference in the efficiency of the two algorithms being compared.

Most of the p-values obtained were 0.0019, which is well below the commonly accepted significance thresholds of 0.05 or 0.01 in research. This strongly suggests rejecting the null hypothesis in most cases, providing substantial statistical evidence of a significant efficiency difference between the algorithms.

Analyzing the execution times from the results of computational efficiency experiments, it is evident that PGKA consistently outperforms both SGKA and nm-PGKA in terms of computational efficiency. However, the results for the C16 configuration with the Har70 dataset do not show a significant difference between nm-PGKA and PGKA. This lack of significant difference is likely due to the diminishing returns of multithreaded computation when the number of nodes is sufficiently large or the dataset size is relatively small. Conversely, the benefits of multithreaded computation were evident in experiments with larger datasets, such as those involving the cover type dataset.

2) Analysis on MRPGKA

A Wilcoxon statistic of 0 suggests that one algorithm consistently outperforms another in execution time. The p-value, in contrast, measures the likelihood of obtaining the observed results under the assumption that the null hypothesis is true. Here, the null hypothesis states that there is no efficiency difference between the two algorithms.

To gain a more comprehensive understanding of PGKA's computational performance, we apply Gustafson's Law to its execution steps. This principle helps evaluate the algorithm's scalability and efficiency in parallel computing contexts, offering broader insights into its performance.

The formula for Gustafson's Law is as follows:

$$\frac{t_s}{t_p} = \alpha + P*(1-\alpha) \tag{10}$$

In this context, t_s represents the execution time of the serial algorithm, t_p denotes the execution time of the parallel algorithm, a is the proportion of the non-parallelizable part of the program, and P is the number of units involved in parallel computation.

The execution process of SGKA, nm-PGKA, and PGKA can be broken down into multiple steps, as detailed in Table VIII.

TABLE VIII
NOTATIONS OF THE STEPS IN THE DIFFERENT ALGORITHMS

Notation	Description
T_i	Execution time of initialization.
T_c	Execution time of crossover operator.
T_m	Execution time of mutation operator.
T_{sl}	Execution time of selection operator.
T_e	Execution time of evaluating the chromosomes
T_{mr}	Time overhead introduced by the MapReduce framework.
T_{fj}	Time overhead caused by the Fork/Join framework.
T_{im}	Time consumption in migrating and resubmitting MapReduce task once.

Assuming the distributed cluster has n nodes, and each with c cores,the number of migrations is m.

For the SGKA, the execution time is:

$$T_{SGKA} = T_i + T_c + T_m + T_{sl} + T_e \tag{11}$$

The execution time of nm-PGKA can be calculated as follows:

$$T_{nm-PGKA} = \frac{T_i + T_c + T_m + T_{sl} + T_e}{T_i + n*(T_c + T_m + T_{sl} + T_e)} + T_{mr} \quad (12)$$

The execution time of nm-PGKA can be calculated as follows:

$$T_{PGKA} = \frac{T_i + T_c + T_m + T_{sl} + T_e}{T_i + n*(T_c + T_m + T_{sl} + \frac{T_e}{n*c*T_e} + T_{fj})} + T_{mr} + m*T_{im} \quad (13)$$

Ideally, the computational efficiency of PGKA increases significantly as the *n* and *c* increase. However, real-world applications must take other factors into account, especially since the execution time of a MapReduce task often depends on the performance of the least computationally efficient node in the cluster.

These analyses establish that PGKA's computational efficiency substantially surpasses that of both SGKA and nm-PGKA, offering a theoretical foundation for its application in various settings.

V. Conclusion

In this research, we proposed the PGKA, which implements distributed parallel computing based on the MapReduce framework. It uses the multi-threaded computing framework Fork/Join to improve its computational efficiency further. The distributed model of the PGKA is based on the island model. In the most time-consuming fitness evaluation part in the PGKA, multi-threaded computing is used to fully use the computing power of multi-core CPU in the distributed clusters.

We conducted multiple experiments on PGKA, focusing on clustering accuracy, computational efficiency, and parameter sensitivity. All datasets were sourced from the UCI Machine Learning Repository.

In the clustering accuracy experiment, PGKA demonstrated superior clustering accuracy compared to the two GKA algorithms proposed in other studies and the K-Means algorithm across four clustering evaluation indices.

In the computational efficiency experiment, we compared the execution time of PGKA with its serial version, SGKA, and the nm-PGKA algorithm, which does not use multi-threaded computing. PGKA consistently had shorter execution times than both SGKA and nm-PGKA across all datasets and configurations. Notably, nm-PGKA also outperformed SGKA except in a few specific cases. For the largest dataset and given parameters, PGKA's speedup rates compared to SGKA were 1.3, 4.6, 10.7, and 12.4 times faster on 2, 4, 8, and 16 nodes, respectively.

We also tested the algorithms on different CPU configurations. The results showed that SGKA and nm-PGKA were not sensitive to the number of CPU cores, while PGKA's execution time decreased as the number of CPU cores increased.

In the parameter sensitivity experiment, we examined the effects of the number of subpopulations and the number of migrations on PGKA's performance. The results indicated that as the number of subpopulations increased, PGKA's clustering accuracy decreased, suggesting the need to increase the population size or the number of migrations appropriately. Additionally, while an increase in the number of migrations improved clustering accuracy, it also resulted in longer execution times. It shows that the parameters must be adjusted in different situations to achieve better results. Finally, we examined the experimental results on computational efficiency using the Wilcoxon test, verified the conclusion that PGKA is more computationally efficient, and analyzed PGKA using Gustafson's law.

There are also some other problems. For example, Hadoop MapReduce does not support saving intermediate results but reading and writing in HDFS. Therefore, iterative algorithms such as the PGKA have a large IO overhead. The PGKA's computational efficiency on Spark will be better [31]. Compared with Hadoop MapReduce, iterative algorithms there are more suitable platforms [32] to implement distributed parallel computing.

In the future, we will test the performance of the PGKA on different platforms or investigate distributed computing solutions to the problem of optimizing K-Means with more advanced optimization algorithms [33].

References

[1] MacQuuen, J. B. "Some methods for classification and analysis of multivariate observation." Proceedings of the 5th Berkley Symposium on Mathematical Statistics and Probability. 1967.

[2] Jain, Anil K. "Data clustering: 50 years beyond K-means." Pattern Recognition Letters 31.8 (2010): 651-666.

[3] Arthur, David, and Sergei Vassilvitskii. "K-means++ the advantages of careful seeding." Proceedings of the eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms. 2007.

[4] Dhillon, Inderjit S., Yuqiang Guan, and Brian Kulis. "Kernel k-means: Spectral Clustering and Normalized Cuts." Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. 2004.

[5] Krishna, K., and M. Narasimha Murty. "Genetic K-means algorithm." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 29.3 (1999): 433-439.

[6] Mardi, Mahnaz, and Mohammad Reza Keyvanpour. "GBKM: A New Genetic Based K-Means Clustering Algorithm." 2021 7th International Conference on Web Research (ICWR). IEEE, 2021.

[7] Kapil, Shruti, Meenu Chawla, and Mohd Dilshad Ansari. "On K-means data clustering algorithm with genetic algorithm." 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC). IEEE, 2016.

[8] Ghezelbash, Reza, Abbas Maghsoudi, and Emmanuel John M. Carranza. "Optimization of geochemical anomaly detection using a novel genetic K-means clustering (GKMC) algorithm." Computers & Geosciences 134 (2020): 104335.

[9] Liu, Qingguo, Liu Xinyue, Wu Jian and Li Yaxiong. "An improved NSGA-III algorithm using genetic K-means clustering algorithm." IEEE Access 7 (2019): 185239-185249.

[10] Sukumar, JV Anand, et al. "Network intrusion detection using improved genetic k-means algorithm." 2018 international conference on advances in computing, communications and informatics (ICACCI). IEEE, 2018.

[11] Ghezelbash, Reza, Abbas Maghsoudi, and Emmanuel John M. Carranza. "Optimization of geochemical anomaly detection using a novel genetic K-means clustering (GKMC) algorithm." Computers & Geosciences 134 (2020): 104335.

[12] Lu, Hao-Chun, F. J. Hwang, and Yao-Huei Huang. "Parallel and distributed architecture of genetic algorithm on Apache Hadoop and Spark." Applied Soft Computing 95 (2020): 106497.

[13] Torquato, Matheus F., and Marcelo AC Fernandes. "High-performance parallel implementation of genetic algorithm on fpga." Circuits, Systems, and Signal Processing 38.9 (2019): 4014-4039.

[14] Alba, Enrique. Parallel metaheuristics: a new class of algorithms. John Wiley & Sons, 2005.

[15] Salto, Carolina, Gabriela Minetti, Enrique Alba and Gabriel Luque. "Big optimization with genetic algorithms: Hadoop, Spark, and MPI." Soft Computing (2023): 1-16.

[16] Harada, Tomohiro, and Enrique Alba. "Parallel genetic algorithms: a useful survey." ACM Computing Surveys (CSUR) 53.4 (2020): 1-39.

[17] Ferrucci, Filomena, Pasquale Salza, and Federica Sarro. "Using hadoop mapreduce for parallel genetic algorithms: A comparison of the global, grid and island models." Evolutionary Computation 26.4 (2018): 535-567

[18] Lambora, Annu, Kunal Gupta, and Kriti Chopra. "Genetic algorithm-A literature review." 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon). IEEE, 2019.
[19] Katoch, Sourabh, Sumit Singh Chauhan, and Vijay Kumar. "A review on genetic algorithm: past, present, and future." Multimedia Tools and Applications 80 (2021): 8091-8126.
[20] Mustafi, Debjani, and Gadadhar Sahoo. "A hybrid approach using genetic algorithm and the differential evolution heuristic for enhanced initialization of the k-means algorithm with applications in text clustering." Soft Computing 23 (2019): 6361-6378.
[21] Roy, Dharmendra K., and Lokesh K. Sharma. "Genetic k-means clustering algorithm for mixed numeric and categorical data sets." International Journal of Artificial Intelligence & Applications 1.2 (2010): 23-28.
[22] Xiao, Jing, et al. "A quantum-inspired genetic algorithm for k-means clustering." Expert Systems with Applications 37.7 (2010): 4966-4973.
[23] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.
[24] Lea, Doug. "A java fork/join framework." Proceedings of the ACM 2000 conference on Java Grande. 2000.
[25] Beasley, David, David R. Bull, and Ralph Robert Martin. "An overview of genetic algorithms: Part 2, research topics." University computing 15.4 (1993): 170-181.
[26] Caliński, Tadeusz, and Jerzy Harabasz. "A dendrite method for cluster analysis." Communications in Statistics-theory and Methods 3.1 (1974): 1-27.
[27] Sampson, Jeffrey R. "Adaptation in natural and artificial systems (John H. Holland)." (1976): 529.
[28] Gong, Yue-Jiao, Wei-Neng Chen, Zhihui Zhan et al. "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art." Applied Soft Computing 34 (2015): 286-300.
[29] Davies, David L., and Donald W. Bouldin. "A cluster separation measure." IEEE Transactions on Pattern Analysis and Machine Intelligence 2 (1979): 224-227.
[30] Rousseeuw, Peter J. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis." Journal of Computational and Applied Mathematics 20 (1987): 53-65.
[31] Wang, Bowen, Jun Yin, Qi Hua, Zhiang Wu, Jie Cao. "Parallelizing k-means-based clustering on spark." 2016 International Conference on Advanced Cloud and Big Data (CBD). IEEE, 2016.
[32] Khezr, Seyed Nima, and Nima Jafari Navimipour. "MapReduce and its applications, challenges, and architecture: a comprehensive review and directions for future research." Journal of Grid Computing 15 (2017): 295-321.
[33] Jiayin Wang, and Yukun Wang, "An Efficient Improved Whale Optimization Algorithm for Optimization Tasks," Engineering Letters, vol. 32, no. 2, pp392-411, 2024