A Hybrid Mechanism of Particle Swarm Optimization and Differential Evolution Algorithms based on Spark

Debin Fan[1, 2] and Jaewan Lee[2*]
[1] School of Information Science and Technology, Jiujiang University
Jiujiang, China
[e-mail: dbfan@kunsan.ac.kr]
[2] Department of Information and Communication Engineering, Kunsan National University
Kunsan, South Korea
[e-mail: jwlee@kunsan.ac.kr]
*Corresponding author: Jaewan Lee

Received November 22, 2018; revised June 11, 2019; accepted October 12, 2019; published December 31, 2019

Abstract

With the onset of the big data age, data is growing exponentially, and the issue of how to optimize large-scale data processing is especially significant. Large-scale global optimization (LSGO) is a research topic with great interest in academia and industry. Spark is a popular cloud computing framework that can cluster large-scale data, and it can effectively support the functions of iterative calculation through resilient distributed datasets (RDD). In this paper, we propose a hybrid mechanism of particle swarm optimization (PSO) and differential evolution (DE) algorithms based on Spark (SparkPSODE). The SparkPSODE algorithm is a parallel algorithm, in which the RDD and island models are employed. The island model is used to divide the global population into several subpopulations, which are applied to reduce the computational time by corresponding to RDD's partitions. To preserve population diversity and avoid premature convergence, the evolutionary strategy of DE is integrated into SparkPSODE. Finally, SparkPSODE is conducted on a set of benchmark problems on LSGO and show that, in comparison with several algorithms, the proposed SparkPSODE algorithm obtains better optimization performance through experimental results.

Keywords: Particle Swarm Optimization, Differential Evolution, Large-scale optimization, Cloud Computing, Resilient Distributed Datasets

http://doi.org/10.3837/tiis.2019.12.010 **ISSN : 1976-7277**

1. Introduction

With the fast development of technology and society, large volumes of data are generated in many real-world applications, such as deep neural network optimization, large-scale cluster resource scheduling, and urban intelligent transportation systems. And so many optimization problems become more and more complicated. Large-scale global optimization (LSGO) problems attract much attention from the researchers in academia and industrial fields.

Recently, evolutionary algorithms (EAs), such as the genetic algorithm (GA), differential evolution (DE), and particle swarm optimization (PSO), have been successfully applied in many fields due to their versatility, reliability, and stability [1][2]. The PSO algorithm, proposed in [3], is a stochastic search algorithm for simulating the foraging behavior of birds. PSO is simple since fewer parameters need to be adjusted. As a result, it is extensively used in function optimization, pattern recognition, and other practical application fields of EAs, and significant results were achieved [4]. The premature convergence problem might easily occur in the PSO algorithm because of its singularity, adversely affecting the performance of the algorithm. Therefore, a hybrid algorithm that utilizes the global search ability of other algorithms is widely used to overcome the shortcomings of PSO [5][6]. Zhang et al. [7] presented a hybrid PSO with DE operator algorithm, which can provide the hell-shaped mutations while preserving the particle swarm dynamics. Das et al. [8] explored several schemes to control the convergence behaviors of PSO and DE by selecting the parameters reasonably. The above-stated studies perform well in solving low-dimensional problems; however, those studies might reduce the precision of the solution and deteriorate the efficiency when facing LSGO problems.

The cloud computing framework is one of the most feasible and reliable ways to solve LSGO problems, as it parallels a huge amount of computing to achieve better operational efficiency and scalability. Two popular cloud computing frameworks are Hadoop MapReduce and Apache Spark [9]. McNabb et al. [10] demonstrated that PSO could be naturally adapted to the MapReduce programming model. For large data sets, Aljarah et al. [11] introduced a parallel version of PSO that is in MapReduce-based to get over the inefficiency of PSO clustering. Wang et al. [12] presented a cooperative PSO algorithm using the MapReduce model, which has better performance and a significant advantage in terms of time. There are also some other EAs based on MapReduce [13]-[17]. However, MapReduce is a general batch processing computing model, which needs to read and write files frequently and lacks an effective mechanism of parallel computing. Spark has a new resilient distributed datasets (RDD) model [18]. RDD is based on memory calculation, which can effectively support iterative calculation. Deng et al. [19] proposed a parallel DE based on RDD, which can decrease the computational time of the objective function. Teijeiro et al. [20] presented two different parallel schemes based on Spark for DE algorithm. For LSGO problems, Peng et al. [21] designed a Spark-based DE with commensal learning and uniform local search. Cheng et al. [22] presented a distributed RDD-based PSO algorithm, which has high precision and acceleration. RDD greatly speeds up program processing, allowing Spark to be used in a variety of large-scale processing scenarios.

In this paper, we introduce a hybrid mechanism of PSO and DE algorithms based on Spark (SparkPSODE) for LSGO problems. The proposed SparkPSODE begins with PSO. Then SparkPSODE uses the evolution strategy of the DE algorithm to enhance population diversity and avoid the local convergence. SparkPSODE realizes its parallelization using the RDD and

island models for improving the convergence speed. Finally, by testing the benchmark functions on LSGO, in comparison with the other algorithms, experiments prove the effectiveness of SparkPSODE. Our proposal is applicable for large-scale data clustering, and also offers a novel solution to solve optimization problems with big data.

The remaining of this article is structured as follows. We introduce the classical PSO and DE in Section 2. Section 3 shows the details of our proposed SparkPSODE algorithm. Numerical experimental tests are presented in Section 4. Finally, we conclude the paper in the last section.

2. The classical PSO and DE algorithms

2.1 Particle swarm optimization

In PSO [23], each bird is one particle that denotes one point in the *D*-dimensional search space. The population consists of *n* particles, each of which has one current position and velocity, shown in Eqs. (1) and (2).

$$X_i = [x_{i1,} x_{i2,} ..., x_{id}] \tag{1}$$

$$V_i = [v_{i1,} v_{i2,} ..., v_{id}] \tag{2}$$

where $i = 1,2,..., N$. Each particle flies in search space, and the optimal solution is found by iterations.

$$P_i = [p_{i1,} p_{i2,} ..., p_{id}] \tag{3}$$

$$P_g = [p_{g1,} p_{g2,} ..., p_{gd}] \tag{4}$$

Eq. (3) represents the position of the personal best of particle, and Eq. (4) denotes the position of the global best particle.

During each iteration, the position *x* and velocity *v* of the particle is updated using Eqs. (5) and (6), respectively.

$$v_{id}(k+1) = \omega \cdot v_{id}(k) + c_1 \cdot r_1 \cdot (p_{id}(k) - x_{id}(k)) + c_2 \cdot r_2 \cdot (p_{gd} - x_{id}(k)) \tag{5}$$

$$x_{id}(k+1) = x_{id}(k) + v_{id}(k+1) \tag{6}$$

where *k* denotes the number of iterations, r_1 and r_2 are random values in the interval [0,1], which can make groups be diversity. c_1 and c_2 denote two acceleration factors, in which the particles have the ability to self-summarize and learn the excellent individual in the group, thus approaching the particle's optimal solution and the group global optimal solution. In the iterative process, adjusting these two parameters properly can reduce the disturbance of the local convergence and speed up the convergence. ω is the inertia factor that influences the exploration and development abilities of the particle. In the standard PSO algorithm, ω uses the same value, resulting in particle diversity is greatly reduced. In this article, the inertia factor of the linear decrement weight strategy [24] is used to improve particle diversity.

2.2 Differential evolution

DE [25] is a type of swarm intelligent algorithm that adopts the real coding method. In DE, the mutation operation uses the mutation strategy; an individual is disturbed by the mutant vector between the individuals in the population, and individual mutation is realized. Crossover can

be considered as a supplement to mutation. Moreover, the selection strategy is usually a tournament choice rule.

The main steps of DE are as follows.

Step 1. Population initialization.

DE organizes a population of *NP* individuals in the *D*-dimensional search space, and then individuals are initialized by Eq. (7).

$$x_{i,j}(0) = x_{i,j}^{L} + rand(0,1) \cdot (x_{i,j}^{U} - x_{i,j}^{L}) \tag{7}$$

where $i = 1,2,\ldots, NP$, $j = 1,2,\ldots, D$, $x_{i,j}^{U}$ and $x_{i,j}^{L}$ are the upper and lower constraints, and $rand \in (0,1)$ represents a random number uniformly distributed among the numbers [0,1].

Step 2. Mutation.

DE usually achieves individual mutation through the mutant vector between individuals in the population. The common mutation strategy randomly selects two different individuals, and then the mutant vector is scaled, and the vector is synthesized, as shown in Eq. (8).

$$v_i(g+1) = x_{r_1}(g) + F \cdot (x_{r_2}(g) - x_{r_3}(g)) \tag{8}$$

where F is the scaling factor lying between 0 and 1, and r_1, r_2, r_3, and i are random numbers uniformly distributed among the numbers [1, *NP*] and $r_1 \neq r_2 \neq r_3 \neq i$.

Step 3. Crossover.

After completing the previous step, the DE algorithm crosses the population $\{x_i(g)\}$, and its mutation intermediates $\{v_i(g+1)\}$ by the crossover probability, as shown in Eq. (9).

$$u_{i,j}(g+1) = \begin{cases} v_{i,j}(g+1), & if\ rand(0,1) \le CR\ or\ j = j_{rand} \\ x_{i,j}(g), & otherwise \end{cases} \tag{9}$$

where j_{rand} is a randomly generated integer among the numbers [1, *D*], and $CR \in [0,1]$ is the crossover probability.

Step 4. Selection.

DE mainly utilizes the greedy strategy to choose a better solution for the next generation.

$$x_i(g+1) = \begin{cases} u_i(g+1), & if\ f(u_i(g+1)) \le f(x_i(g)) \\ x_i(g), & otherwise \end{cases} \tag{10}$$

Step 5. Termination.

By performing the above operations, the DE algorithm stops searching and outputs the optimal value when the cycle algebra exceeds the maximum evolutionary algebra or when solution precision is required.

3. Proposed hybrid mechanism of PSO and DE algorithms based on Spark

3.1 The Spark Cloud Platform

Apache Spark is an efficient and stretchable clustering computing system, which inherits MapReduce's linear scalability and fault tolerance on the Hadoop platform. However, Spark

extends the MapReduce model in many ways and utilizes the RDD model for computing large-scale data in parallel [26].

The RDD model is the core component of Spark. In essence, RDD is the element set of a distributed cluster, which runs on different nodes of a cluster. In Spark, all the data operations are used to create RDD, transform existing RDD, and invoke RDD operations. Each RDD corresponds to one partition.

Users can create RDD in two ways: one is to read an external dataset, and the other is to generate RDD in-memory calculations by functions, such as *join* and *map*. After RDD is created, two kinds of operations can be performed: transformations and actions. Transformation operations mainly include such as *map*, *filter*, *flatMap*. Action operations mainly include such as *count*, *collect*, *reduce*, *save*. Transformation operations generate a new RDD from an existing one. Action operations compute a result for the RDD, and the result is returned to the driver program or stored in Hadoop Distributed File System (HDFS).

The difference between transformation operations and action operations is the method of calculating RDD on Spark. In addition to transformation and action operations, RDD can also be operated upon using the cache operation. The implementation mechanism of the RDD model is based on the iterator, which makes data access more efficient. The RDD computing model in Spark is shown in **Fig. 1**.

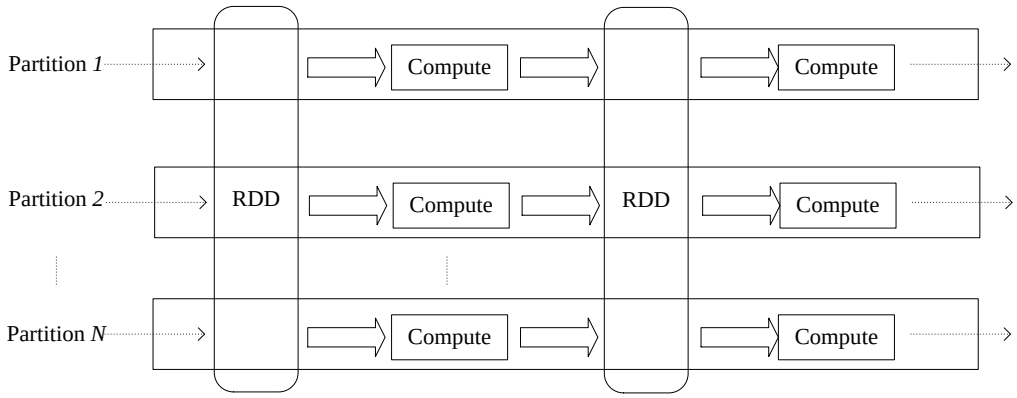

Fig. 1. RDD computing model

3.2 Island Model

In this paper, an island model [27] is utilized to realize the parallelization of the algorithm. The island model is coarse-grained and shown in **Fig. 2**. In the parallelization process, the population is divided into serval subpopulations, then each of which is evolved independently in the iteration cycle. The implementation of the island model is mainly based on five parameters as follows:

- The number of islands is the number of subpopulations. This value affects the algorithm's parallel efficiency and population diversity, so it should not be either too high or too low.
- The migration topology refers to the logical model of individual migration. Common migration topologies are ring, chain, and cascading topologies. In the SparkPSODE

algorithm, a ring+1+2 topology is utilized, in which each island is only connected with two adjacent islands.

- The migration strategy is the strategy of replacing individuals in target subpopulation with that from source subpopulation. A common strategy is replacing the worst individuals in target subpopulation with the best individuals from source subpopulation, which is termed "best-worst" strategy. Another possible strategy is "best+random-worst", where the worst individuals in target subpopulation are replaced with the best and random individuals from source subpopulation. A random replacement strategy is a "random-random" strategy.
- The number of migration individuals determines the degree of communication between the subpopulations, and should not to be too large or too small.
- The migration frequency is the algebra of migration interval. If it is too high, the solution might fall into local optimal solution; if it is too low, the information might not be fully shared among the subpopulations.

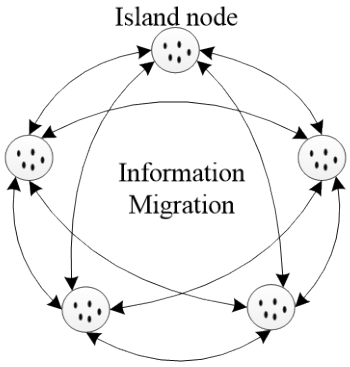

Fig. 2. Island Model

3.3 The proposed SparkPSODE algorithm

The SparkPSODE algorithm employs DE's evolution strategy in the framework of PSO, which is parallel and implemented based on Spark. Concretely, we firstly utilize DE/rand/1 mutation operator of DE. Subsequently, we apply the RDD and island models to realize the parallel computing of SparkPSODE.

The details of our proposal can be presented as follows:

1) Dividing the global population into independent subpopulations by using *parallelize* in Spark, and each subpopulation corresponds to a partition of the RDD model.

2) Calculating each particle's fitness value, and then comparing and updating the position of the personal best of particle and the position of the global best particle.

3) Using Eqs. (5) and (6) to change each particle's position and velocity.

4) Using Eqs. (8), (9) and (10) to implement three operations of the DE algorithm toward the updated position of each particle.

5) Repeat step 2 until the termination condition is satisfied.

6) Using *collect* to combine each partition for generating a new population and finding the global optimum by *reduce*.

Algorithm 1 demonstrates the pseudocode of SparkPSODE. The flowchart of SparkPSODE is given in **Fig. 3**. For clarity, the method of data storage utilized in **Fig. 3** is described below: the data is stored in key-value pairs, namely [key_i , $value_i$], where i = 1,2,..., m, m represents the population number, key_i is an integer that is the index of the subpopulation of i, and $value_i$ is a list containing all the individuals in the subpopulation.

Algorithm 1. The pseudocode of SparkPSODE

Input:
NP: the number of population;
popsize: the size of subpopulation;
pbest: the best position of particle;
gbest: the optimal location of population;
migrationInterval: the migration frequency;
related parameters.

Output:
The global optimum.

```
Initialize the parameters: NP, popsize and t=0;
Randomly initialize the population;
Calculate each particle's fitness function f(x_i);
Map the subpopulations to RDD partitions (islands);
while termination criterion is not met do
    while t < migrationInterval do
        for i=1 to popsize do
            if (f(x_i) < f(pbest_i)) then
                pbest_i = x_i;
            end if
            if (f(x_i) < f(gbest)) then
                gbest = x_i;
            end if
            Change the velocity and position of each particle by Eqs. (5) and (6);
            Update the position of each particle by Eqs. (8) , (9) and (10);
            Evaluate each particle's fitness function f(x_i);
        end for
        t++;
    end while
    Migrate individuals;
    Collect the subpopulations.
end while
```

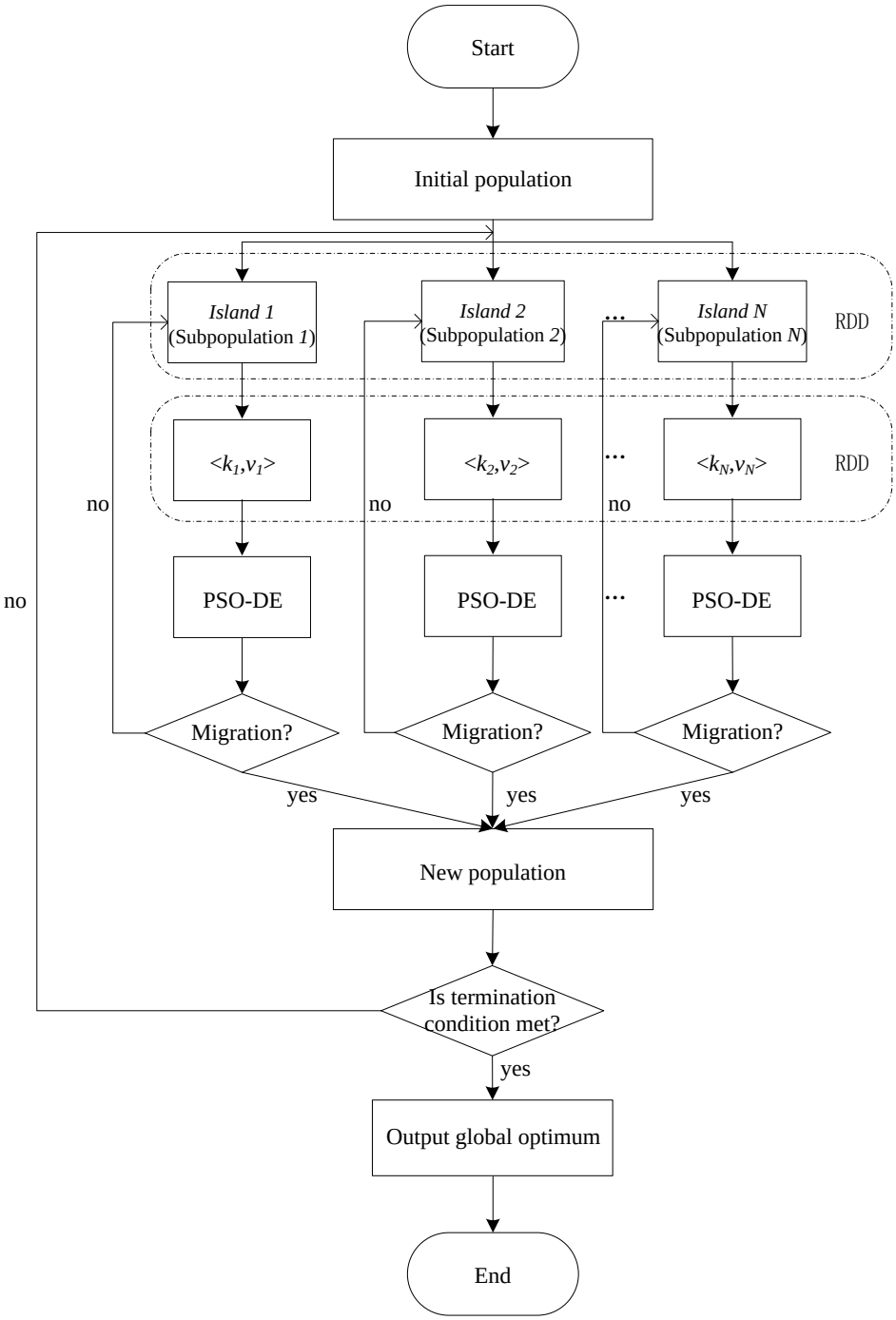

Fig. 3. The flowchart of SparkPSODE

4. Numerical experiments

4.1 Benchmark problems

Aiming to testify the optimization capability of SparkPSODE in solving LSGO problems, eleven widely used benchmark problems [28] with size up to 1000 dimensions were selected for analysis. The descriptions of these problems are presented in **Table 1**.

Table 1. Benchmark problems

Name	Expression	Value range	Optimum
Sphere Model	$f_1(x) = \sum_{i=1}^{D} x_i^2$	[-100,100]	0
Schwefel's Problem 2.22	$f_2(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	[-10,10]	0
Schwefel's Problem 1.2	$f_3(x) = \sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2$	[-100,100]	0
Schwefel's Problem 2.21	$f_4(x) = \max_i \{\lvert x_i \rvert, 1 \le i \le D\}$	[-100,100]	0
Generalized Rosenbrock's Function	$f_5(x) = \sum_{i=1}^{D-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	[-30,30]	0
Step Function	$f_6(x) = \sum_{i=1}^{D} (\lvert x_i + 0.5 \rvert)^2$	[-100,100]	0
Quartic with Noise	$f_7(x) = \sum_{i=1}^{D} i x_i^4 + random[0,1)$	[-1.28,1.28]	0
Generalized Schwefel's Problem 2.26	$f_8(x) = \sum_{i=1}^{D} -x_i \sin(\sqrt{\lvert x_i \rvert})$	[-500,500]	-418.9829*D
Generalized Rastrigin's Function	$f_9(x) = \sum_{i=1}^{D} (x_i^2 - 10\cos(2\pi x_i) + 10)$	[-5.12,5.12]	0
Ackley's Function	$f_{10}(x) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D} \cos 2\pi x_i) + 20 + e$	[-32,32]	0
Generalized Griewank Function	$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}}) + 1$	[-600,600]	0

4.2 Experimental settings

In this study, we conducted the experiments on two PCs: one PC is with Intel Core i7-8700 3.20 GHz CPUs and 16 GB memory, and the other PC is with Intel Core i3-2120 3.30 GHz CPUs and 4 GB memory. The operating systems of the two PCs are Ubuntu 16.04. Hadoop

2.2.0 and Spark 2.3.3 were installed in the two PCs. The proposed algorithm is implemented by Scala and Java languages in IntelliJ IDEA 14.1.2. The main parameters and settings for SparkPSODE are described in **Table 2**.

Table 2. The parameters of SparkPSODE

Parameters		Definitions	Value
NP		The number of the population	100
D		The size of the dimension	1000
Max_FEs		The maximum number of fitness evaluations	5,000,000
PSO	c_1	Acceleration factor	2
	c_2	Acceleration factor	2
	ω	Inertia weight	0.4-0.9
DE	*F*	The scaling factor	0.5
	CR	The crossover probability	0.9
islands		The number of islands	4
topology		The migration topology	ring+1+2
migrationIndividuals		The number of migration individuals	15
migrationStrategy		The migration strategy	best-worst
migrationInterval		The migration frequency	1000

4.3 Experimental results and analysis

Table 3 lists the statistical results of PSO, DECCG [29], SparkDECC [30], and SparkPSODE. For a fair comparison, *Max_FEs* is set to 5,000,000, and all algorithms independently run 25 times for each benchmark problem. We analyze the experimental results by Wilcoxon's rank-sum test. In **Table 3**, "-", "+", and "≈" represent the statistical results of the compared algorithms being worse than, better than, and similar to that of SparkPSODE, respectively.

Table 3. Experimental results of PSO, DECCG, SparkDECC, and SparkPSODE

Fun	PSO (mean±std)	DECCG (mean±std)	SparkDECC (mean±std)	SparkPSODE (mean±std)
f_1	2.30E+06±2.79E+04 −	9.61E-29±3.11E-29 −	5.85E-13±1.62E-13 −	7.36E-65 ± 3.60E-64
f_2	7.16E+04±3.30E+04 −	1.70E-14±1.77E-14 −	6.60E-07±1.00E-07 −	2.59E-35 ± 1.18E-34
f_3	8.68E+08±6.51E+07 −	1.20E-03±6.70-04 −	5.31E+07±7.20E+06 −	8.02E-53 ± 3.93E-52
f_4	4.01E+02±7.98E+00 −	3.19E-02±4.72E-03 −	9.76E+01±2.22E-01 −	0.00E+00±0.00E+00
f_5	9.23E+12±1.29E+12 −	9.86E+02±4.11E-01 +	1.62E+03±1.62E+02 −	9.93E+02 ± 3.38E+00
f_6	2.30E+06±1.29E+05 −	0.00E+00±0.00E+00 ≈	1.60E-01±4.73E-01 −	0.00E+00±0.00E+00
f_7	3.48E+13±3.87E+12 −	2.62E-03±6.68E-04 +	3.62E+00±1.67E-01 −	7.90E-03 ± 6.65E-03
f_8	-1.34E+05±4.51E+03 −	-4.19E+05±9.28E-11 +	-6.11E+04±1.18E+03 −	-2.47E+06 ±3.48E+05
f_9	2.33E+06±1.35E+05 −	1.25E-14±7.49E-15 +	1.10E+04±3.93E+01 −	1.56E+03±3.57E+03
f_{10}	2.15E+01±3.77E-02 −	1.27E-13±7.11E-15 +	4.55E-08±8.16E-09 +	8.65E+00±1.06E+01
f_{11}	5.75E+02±4.15E+01 −	9.50E-16±1.44E-16 −	3.54E-14±1.03E-14 −	0.00E+00±0.00E+00
-/+/≈	11/0/0	5/5/1	10/1/0	

From **Table 3**, it can be observed that SparkPSODE consistently performs better than PSO in all eleven benchmark functions. SparkPSODE is better than DECCG in functions f_1, f_2, f_3, f_4, and f_{11}. Especially, in functions f_4, f_6, and f_{11}, SparkPSODE can even converge to 0. The proposed SparkPSODE algorithm is superior to SparkDECC in ten benchmark functions. The above analysis indicates that the SparkPSODE algorithm is effective.

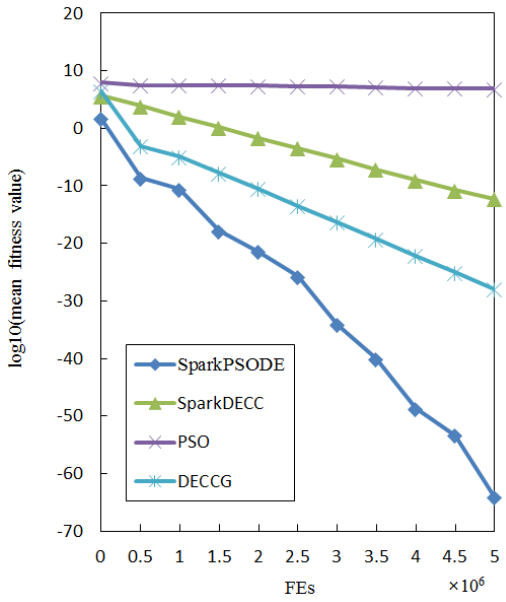

Fig. 4. Convergence figure of f_1

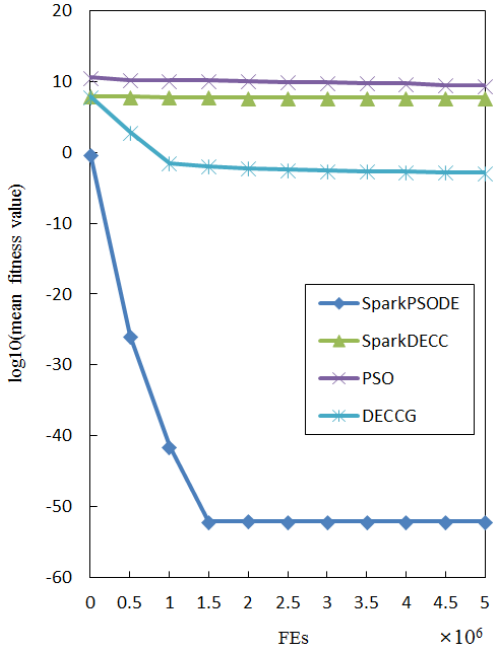

Fig. 5. Convergence figure of f_3

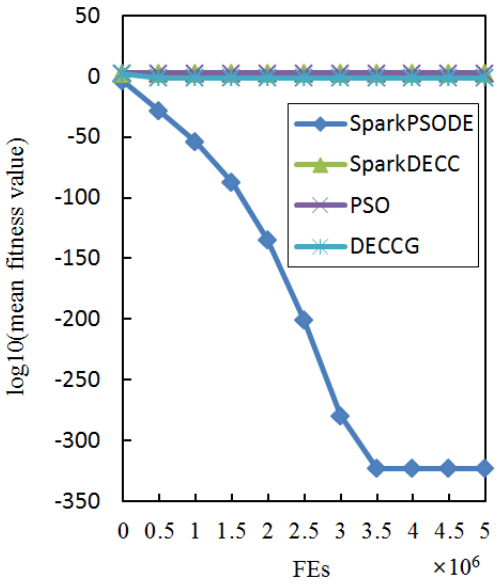

Fig. 6. Convergence figure of f_4

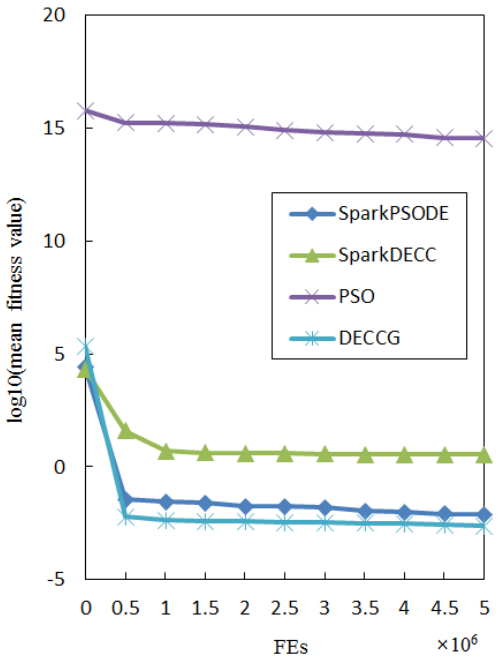

Fig. 7. Convergence figure of f_7

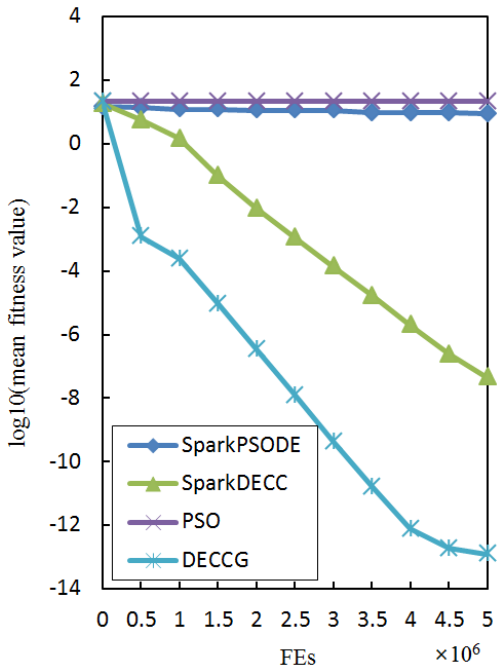

Fig. 8. Convergence figure of f_{10}

Fig. 4 to **Fig. 8** show the convergence process of four algorithms on functions $f_1, f_3, f_4, f_7,$ and f_{10}, respectively. For $f_1, f_3,$ and f_4, SparkPSODE can obtain better convergence speed than other algorithms.

Above all, we observe that the proposed SparkPSODE algorithm outperforms in terms of solution accuracy and convergence speed as compared to the other algorithms.

4.4 The influence of the number of subpopulations

To analyze the effectiveness of the number of subpopulations on the performance of SparkPSODE, different numbers of subpopulations, such as 1, 2, 4, and 5, were selected for comparison experiments. In different subpopulations, the proposed algorithm was independently tested 20 times. **Table 4** and **Table 5** record the averaged optimal values and the averaged computational time of four different subpopulations, respectively. The results of **Table 4** show that the convergence accuracy of functions $f_1, f_2, f_3, f_5, f_6, f_7, f_9,$ and f_{11} improves following the increase of subpopulation number, while the convergence accuracy of functions $f_4, f_8,$ and f_{10} are not enhanced. According to **Table 5**, we can see that the interaction time between the subpopulations raises following the increase of subpopulation numbers on all the eleven test functions. Overall, the appropriate number of subpopulations should be set for achieving satisfying performance in the two aspects of time and accuracy.

Table 4. Experimental results of SparkPSODE with different numbers of subpopulations

F/S	1 (mean±std)	2(mean±std)	4 (mean±std)	5(mean±std)
f_1	7.78E+01±1.34E+02	3.23E+01 ± 9.41E+01	9.20E-65±4.01E-64	2.18E-44±9.51E-44
f_2	3.91E+00± 6.57E+00	1.48E-04 ± 5.45E-04	6.20E-12±1.63E-11	6.77E-17±1.42E-16
f_3	8.09E+00± 2.45E+01	2.03E-07 ± 8.82E-07	8.10E-29±3.53E-28	1.20E-52±5.22E-52
f_4	2.70E-01±4.25E-01	1.50E-01±3.55E-01	3.82E-01±4.84E-01	1.24E-01±3.29E-01
f_5	7.13E+03±8.31E+03	7.18E+03±8.44E+03	9.93E+02±3.28E+00	2.76E+03±5.29E+03
f_6	6.89E+01±1.64E+02	4.62E+01±1.38E+02	2.24E+01±9.76E+01	0.00E+00±0.00E+00
f_7	1.80E+04±3.59E+04	1.34E+04±3.20E+04	4.55E-03±1.01E-02	5.74E-03±8.96E-03
f_8	-2.67E+06±3.56E+05	-2.47E+06±4.09E+05	-2.41E+06±4.17E+05	-2.38E+06±2.17E+05
f_9	1.22E+03±2.95E+03	2.85E+03±4.36E+03	4.90E+02±2.14E+03	4.55E+02±1.99E+03
f_{10}	1.52E+01±9.92E+00	1.95E+01±6.49E+00	1.41E+01±1.03E+01	1.84E+01±7.73E+00
f_{11}	1.34E-01±2.63E-01	1.02E-01±2.43E-01	1.67E-01±2.90E-01	0.00E+00±0.00E+00

Table 5. The averaged computational time of SparkPSODE with different numbers of subpopulations (ms)

F/S	1	2	4	5
f_1	3.68E+04	6.81E+04	1.12E+05	1.36E+05
f_2	2.85E+04	7.32E+04	1.21E+05	1.41E+05
f_3	1.88E+05	2.09E+05	2.47E+05	2.69E+05
f_4	4.13E+04	7.34E+04	1.14E+05	1.26E+05
f_5	4.27E+04	8.10E+04	1.18E+05	1.29E+05
f_6	4.20E+04	7.63E+04	1.23E+05	1.26E+05
f_7	3.38E+04	8.03E+04	1.04E+05	1.43E+05
f_8	5.67E+04	1.06E+05	1.53E+05	1.83E+05
f_9	5.57E+04	9.79E+04	1.32E+05	1.53E+05
f_{10}	5.94E+04	1.03E+05	1.33E+05	1.69E+05
f_{11}	5.49E+04	9.41E+04	1.32E+05	1.52E+05

4.5 The influence of the number of migrating individuals

To analyze the effectiveness of the number of migrating individuals on the performance of SparkPSODE, 5, 10, 15, and 20 migrating individuals are selected for comparison experiments. For different numbers of migrating individuals, the proposed algorithm was independently tested 20 times. **Table 6** and **Table 7** record the averaged optimal values and the averaged computational time of our proposed algorithm in the cases of four different numbers of migrating individuals, respectively. It can be seen from **Table 6** that with the number of migrating individuals increasing, the convergence accuracy of functions f_4, f_6, f_{10}, and f_{11} improving. The solutions of functions f_2, f_9, and f_{10} are optimal when the number of migrating individuals is 15, and increasing the number of migrating individuals does not improve solution accuracy. **Table 7** shows that the interaction time between migrating individuals raises following the increase of the number of migrating individuals on ten test functions. In general, the number of migrating individuals should be appropriately set to achieve a satisfying performance.

Table 6. Experimental results of SparkPSODE with different numbers of migrating individuals

F/M	5(mean±std)	10 (mean±std)	15(mean±std)	20 (mean±std)
f_1	2.80E-89±1.22E-88	9.78E-27±4.26E-26	8.87E-62±3.85E-61	1.10E-68±4.78E-68
f_2	2.99E-12±1.01E-11	8.47E-13±1.42E-12	2.00E-13±7.97E-13	3.73E-12±1.03E-11
f_3	1.92E-43±8.22E-43	3.69E-46±1.61E-45	2.63E-31±1.15E-30	5.90E-28±2.35E-27
f_4	2.21E-01±4.13E-01	3.11E-01±4.61E-01	2.84E-01±4.49E-01	0.00E+00±0.00E+00
f_5	9.94E+02±4.06E+00	9.94E+02±3.72E+00	9.94E+02±3.49E+00	9.96E+02±3.84E+00
f_6	7.02E+01±1.67E+02	0.00E+00±0.00E+00	0.00E+00±0.00E+00	0.00E+00±0.00E+00
f_7	4.28E-03±6.22E-03	6.10E-03±8.69E-03	4.17E-03±3.79E-03	3.29E-03±3.87E-03
f_8	-2.48E+06±3.27E+05	-2.53E+06±4.36E+05	-2.44E+06±3.48E+05	-2.37E+06±2.90E+05
f_9	3.37E+03±4.60E+03	1.89E+03±3.79E+03	1.48E+03±3.52E+03	1.92E+03±3.84E+03
f_{10}	1.41E+01±1.03E+01	1.62E+01±9.37E+00	2.16E+00±6.49E+00	8.66E+00±1.06E+01
f_{11}	3.28E-02±1.43E-01	1.01E-01±2.40E-01	0.00E+00±0.00E+00	0.00E+00±0.00E+00

Table 7. The averaged computational time of SparkPSODE with different numbers of migrating individuals (ms)

F/M	5	10	15	20
f_1	6.39E+04	8.88E+04	1.12E+05	1.45E+05
f_2	6.45E+04	9.01E+04	1.13E+05	1.44E+05
f_3	1.98E+05	2.16E+05	2.36E+05	2.67E+05
f_4	6.28E+04	8.71E+04	1.06E+05	1.28E+05
f_5	6.25E+04	8.61E+04	1.07E+05	1.58E+05
f_6	6.39E+04	8.60E+04	1.19E+05	1.43E+05
f_7	6.34E+04	9.43E+04	1.12E+05	1.43E+05
f_8	9.06E+04	1.22E+05	1.54E+05	1.84E+05
f_9	7.86E+04	9.32E+04	1.34E+05	1.69E+05
f_{10}	8.05E+04	1.08E+05	1.26E+05	1.43E+05
f_{11}	8.06E+04	1.09E+05	1.34E+05	1.29E+05

5. Conclusion

This paper introduces a hybrid mechanism of PSO and DE algorithms based on Spark for LSGO problems. The proposed algorithm relies on Spark cloud computing platform and uses the RDD and island models to realize its parallelization. The proposed algorithm divides the global population into several subpopulations, and each subpopulation evolves independently. In order to realize the communication between subpopulations, migrating individuals are carried out at specific generation intervals. The results demonstrate that our proposal is a fast algorithm that has high acceleration performance and scalability. Using more nodes to solve massive or real-world optimization problems, and further improving the performance and the applicability of the algorithm will be considered in a future study.

Acknowledgment

This research is partially supported by Institute of Information and Telecommunication Technology of Kunsan National University, South Korea, and the National Natural Science Foundation of China (No. 61763019, 61662038), and the Science and Technology Plan Projects of Jiangxi Provincial Education Department, China (No. GJJ180891, GJJ161072).

References

[1] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Proc. of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, vol. 2, pp. 1980-1987, September 2004. Article (CrossRef Link)

[2] M. Črepinšek, S.H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Computing Surveys (CSUR)*, vol. 45, no. 3, pp. 35, June 2013. Article (CrossRef Link)

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. of ICNN'95 - International Conference on Neural Networks*, pp. 1942-1948, August 2002. Article (CrossRef Link)

[4] M. Meissner, M. Schmuker, and G. Schneider, "Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training," *BMC Bioinformatics*, vol. 7, no.1, pp. 125, March 2006. Article (CrossRef Link)

[5] H. Liu, Z. Cai, and Y. Wang, "Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization," *Applied Soft Computing*, vol. 10, no. 2, pp. 629-640, March 2010. Article (CrossRef Link)

[6] T. Zhang and X. Gao, "Hybridizing particle swarm optimization with differential evolution solving constrained problems," *Microcomputer & Its Applications*, vol. 33, no. 17, pp. 83-87, September 2014. Article (CrossRef Link)

[7] W.J. Zhang and X.F. Xie, "DEPSO: hybrid particle swarm with differential evolution operator," in *Proc. of 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483)*, vol. 4, pp. 3816-3821, November 2003. Article (CrossRef Link)

[8] S. Das, A. Abraham, and A. Konar, "Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives," *Advances of computational intelligence in industrial systems, Springer, Berlin, Heidelberg*, pp. 1-38, 2008. Article (CrossRef Link)

[9] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. of the 2nd USENIX conference on Hot topics in cloud computing*, pp. 10-10, June 2010. Article (CrossRef Link)

[10] A.W. McNabb, C.K. Monson, and K.D. Seppi, "Parallel PSO using MapReduce," in *Proc. of* 2007 *IEEE Congress on Evolutionary Computation*, pp. 7-14, January 2008. Article (CrossRef Link)

[11] I. Aljarah and S.A. Ludwig, "Parallel particle swarm optimization clustering algorithm based on MapReduce methodology," in *Proc. of 2012 Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pp. 104-111, January 2013. Article (CrossRef Link)

[12] Y. Wang, Y. Li, Z. Chen, and Y. Xue, "Cooperative particle swarm optimization using MapReduce," *Soft Computing*, vol. 21, no. 22, pp. 6593-6603, November 2017. Article (CrossRef Link)

[13] G.S. Sadasivam and D. Selvaraj, "A novel parallel hybrid PSO-GA using MapReduce to schedule jobs in Hadoop data grids," in *Proc. of 2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pp. 377-382, February 2011. Article (CrossRef Link)

[14] J. Cao, H. Cui, H. Shi, and L. Jiao, "Big Data: A Parallel Particle Swarm Optimization-Back-Propagation Neural Network Algorithm Based on MapReduce," *PloS one*, vol. 11, no. 6, pp. 1-17, June 2016. Article (CrossRef Link)

[15] A. Sinha and P.K. Jana, "A hybrid MapReduce-based k-means clustering using the genetic algorithm for distributed datasets," *The Journal of Supercomputing*, vol. 74, no. 4, pp. 1562-1579, April 2018. Article (CrossRef Link)

[16] N. Al-Madi, I. Aljarah, and S.A. Ludwig, "Parallel glowworm swarm optimization clustering algorithm based on MapReduce," in *Proc. of 2014 IEEE Symposium on Swarm Intelligence*, pp. 1-8, January 2015. Article (CrossRef Link)

[17] S. Yuan, C. Deng, X. Dong, D. Fan, and C. Yin, "Cloud differential evolution algorithm with multi-strategy for high dimensional optimization problems," *Computer Engineering and Design*, vol. 39, no. 9, pp. 2792-2799, September 2018. Article (CrossRef Link)
[18] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, et al., "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 2-2, April 2012. Article (CrossRef Link)
[19] C. Deng, X. Tan, X. Dong, and Y. Tan, "A Parallel Version of Differential Evolution Based on Resilient Distributed Datasets Model," in *Proc. of Bio-Inspired Computing-Theories and Applications, Springer, Berlin, Heidelberg*, pp. 84-93, December 2015. Article (CrossRef Link)
[20] D. Teijeiro, X.C. Pardo, P. González, J.R. Banga, and R. Doallo, "Implementing parallel differential evolution on Spark," in *Proc. of European Conference on the Applications of Evolutionary Computation*, pp. 75-90, Springer, Cham, April 2016. Article (CrossRef Link)
[21] H. Peng, X. Tan, C. Deng, and S. Peng, "SparkCUDE: a spark-based differential evolution for large-scale global optimisation," *international Journal of High Performance Systems Architecture*, vol. 7, no. 4, pp. 211-222, June 2018. Article (CrossRef Link)
[22] L. Cheng, Z. Wu, H. Peng, S. Wu, C. Deng et al., "Distributed Particle Swarm Optimization Algorithm Based on Resilient Distributed Datasets," *Journal of Chinese Computer Systems*, vol. 37, no. 11, pp. 2542-2546, November 2016.
[23] J. Kennedy, "Particle swarm optimization," *Encyclopedia of Machine Learning, Springer, Boston, MA*, pp. 760-766, 2011. Article (CrossRef Link)
[24] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. of IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pp. 69-73, August 2002. Article (CrossRef Link)
[25] S. Das and P.N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE transactions on evolutionary computation*, vol. 15, no. 1, pp. 4-31, February 2011. Article (CrossRef Link)
[26] R. Jin, G. Chen, A.K.H. Tung, L. Shou, and B.C. Ooi, "An Optimized Iterative Semantic Compression Algorithm And Parallel Processing for Large Scale Data," *KSII Transactions on Internet & Information Systems*, vol. 12, no. 6, pp. 2761-2781, June 2018. Article (CrossRef Link)
[27] Y.J. Gong, W.N. Chen, Z.H. Zhan, J. Zhang, Y. Li et al., "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art," *Applied Soft Computing*, vol. 34, pp. 286-300, September 2015. Article (CrossRef Link)
[28] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions On Evolutionary Computation*, vol. 3, no. 2, pp. 82-102, July 1999. Article (CrossRef Link)
[29] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *information Sciences*, vol. 178, no. 15, pp. 2985-2999, August 2008. Article (CrossRef Link)
[30] X. Tan, C. Deng, Z. Wu, H. Peng, and Q. Zhu, "Cooperative differential evolution in cloud computing for solving large-scale optimization problems," *CAAI Transactions On Intelligent Systems*, vol. 13, no. 2, pp. 243-253, April 2018. Article (CrossRef Link)

Debin Fan is currently pursuing his Ph.D. degree in the Department of Information and Communication Engineering, Kunsan National University, Kunsan, South Korea, and received the M.Eng. in Computer technology from Huazhong University of Science and Technology, Wuhan, China. He has been an associate professor in the School of Information Science and Technology, Jiujiang University, since 2013. His research interests include cloud computing, evolutionary computation, swarm intelligence, and large-scale optimization.

Jaewan Lee received his B.S., M.S., and Ph.D. degrees in Computer Engineering from Chung-Ang University in 1984, 1987, and 1992, respectively. Currently, he is a professor at the Department of Information and Communication Engineering, Kunsan National University, Kunsan, South Korea. His research interests include distributed systems, cloud computing, data mining, and computer networks.