Hindawi
Scientific Programming
Volume 2020, Article ID 8810215, 10 pages
https://doi.org/10.1155/2020/8810215

Research Article

Heuristic Algorithms for MapReduce Scheduling Problem with Open-Map Task and Series-Reduce Tasks

Feifeng Zheng,[1] Zhaojie Wang,[1] Yinfeng Xu,[1] and Ming Liu [2]

[1] *Glorious Sun School of Business & Management, Donghua University, Shanghai, China*
[2] *School of Economics & Management, Tongji University, Shanghai, China*

Correspondence should be addressed to Ming Liu; mingliu@tongji.edu.cn

Received 20 May 2020; Revised 14 June 2020; Accepted 23 June 2020; Published 15 July 2020

Academic Editor: Lu Zhen

Copyright © 2020 Feifeng Zheng et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Based on the classical MapReduce concept, we propose an extended MapReduce scheduling model. In the extended MapReduce scheduling problem, we assumed that each job contains an *open-map task* (the *map task* can be divided into multiple unparallel operations) and *series-reduce tasks* (each *reduce task* consists of only one operation). Different from the classical MapReduce scheduling problem, we also assume that all the operations cannot be processed in parallel, and the machine settings are unrelated machines. For solving the extended MapReduce scheduling problem, we establish a mixed-integer programming model with the minimum makespan as the objective function. We then propose a genetic algorithm, a simulated annealing algorithm, and an *L-F* algorithm to solve this problem. Numerical experiments show that *L-F* algorithm has better performance in solving this problem.

1. Introduction

For meeting the unpredictable demand, manufacturing companies often acquire more manufacturing equipment. Scheduling jobs to be processed on these manufacturing equipment increase the complexity of business operations [1]. Job scheduling is mainly used to assist decision makers in deciding the processing sequence of jobs and the allocation of machines. Many scholars have studied the classical scheduling problems, such as single-machine scheduling problem and parallel-machine scheduling problem. It is worth noting that the combination of MapReduce job and machine scheduling has attracted more and more scholars' attention in recent years. In fact, MapReduce system is a popular method of big data processing [2]. MapReduce generally consists of one map task and one reduce task, and the reduce task is strictly behind the map task. Normally, the map task can be arbitrarily split into multiple map operations, and all the map operations can be processed in parallel on multiple parallel machines, while the reduce tasks are unsplittable; thus, the reduce tasks can only be processed by one machine. In the classical MapReduce scheduling problem, each job consists of the map tasks and the reduce tasks, and the reduce task is strictly behind the map task [3]. Normally, the reduce task contains only one operation.

Many current studies on MapReduce scheduling problem assume that the available machines are identical parallel machines, i.e., for the same operation, the processing speed of machines is the same. In actual production and scheduling environment, production models of MapReduce-like systems are widely used. For example, the manufacturing and manual assembly of parts and components of some valuable watches can be regarded as the production mode of MapReduce. Another example, after finishing the basic production of a high-grade cheongsam, it often requires manual embroidery by multiple processors. In the process of embroidery, many embroidered patterns are often not processed in sequence. After the embroidery is finished, it needs to be ironed by one of the processors and then be packed. From embroidery to packaging, this cheongsam can be regarded as the extended MapReduce-like job. Undoubtedly, MapReduce scheduling has a strong practical significance.

However, in the actual situation, there is a kind of jobs similar to MapReduce job. It may be easier for us to describe their features in terms of MapReduce terminology. Thus, we will mention such jobs as *extended MapReduce jobs*. Specifically, each extended MapReduce job consists of one *open-map task* and several *series-reduce tasks*. Different from the classical MapReduce job, the extended MapReduce job cannot be processed in parallel. The operations of open-map task of the extended MapReduce job can be processed by any machine without strict sequential processing sequence (like operations of an open shop job), but cannot be processed in parallel. This is similar to open shop, so we call it *open-map task*. We say each open-map task of a job can be divided into multiple map operations. The expanded MapReduce job also consists of several *series-reduce tasks*. In this paper, we mainly study the expanded MapReduce scheduling problem.

We consider an extended MapReduce scheduling problem with the open-map task and series-reduce tasks, motivated by their practical fixed partition applications. There are three main differences between our research and the classical MapReduce scheduling problem:

(1) We assume that the parallel machines are unrelated, while the machines are identical in the classical MapReduce model

(2) In the extended MapReduce scheduling problem, map operations (i.e., a partition of the open-map task) cannot be processed in parallel. In the extended MapReduce scheduling problem, the division of an open-map task, i.e., map operations, is known a priori

(3) We extend the classical MapReduce scheduling problem by assuming that there are several series-reduce tasks during processing.

In the extended MapReduce scheduling problem, each job consists of two kinds of tasks: one open-map task and several series-reduce tasks. There is no strict requirement of processing sequence for the map operations (like operations of one job in open shop), but there is strict requirement of processing sequence for the reduce operations (each operation corresponds to one series-reduce task). Moreover, it is obvious that the extended MapReduce scheduling problem is an NP-hard problem, because it is an extension of the parallel machine makespan minimization problem.

To address the extended MapReduce scheduling problem, we establish a mixed-integer programming model with the minimum makespan as the objective function. We then solve it by CPLEX, a genetic algorithm, and a simulated annealing algorithm, respectively. Besides, we design a heuristic algorithm which we called *L-F* algorithm to improve the solution efficiency. We compare *L-F* algorithm with the genetic algorithm and the simulated annealing algorithm to verify its performance.

The studies of MapReduce scheduling problem often assume that all the jobs are processed on identical parallel machines [4, 5]. First, in the case of online study based on this assumption, [6] considers a two-stage classical flexible flow shop problem based on MapReduce system, and give an online 6-approximation and an online $(1+\epsilon)$-speed $O(1/\epsilon^4)$-competitive algorithm. Assuming that the reduce task is unknown before the map task is completed, [7] considers the preemptive and nonpreemptive processing scenarios and prove that no matter what the processing scenario is the competition ratio of any algorithm is at least $2-1/n$. For improving system utilization for batch jobs, [8] proposes a modified MapReduce architecture that allows data to be pipelined between operators. Jiang et al. [9] propose an online algorithm which can optimize their MapReduce problem, where the map operations are fractional and the reduce operations are preemptive. The paper [10] studies an online MapReduce scheduling problem on two uniform machines and discusses the competitive ratio with preemption or nonpreemption, respectively.

Second, in the online scenario, [11] solves an online job scheduling problem with flexible deadline bounds via a receding horizon control algorithm. The study in [12] expands the MapReduce tasks by allowing user specify a job's deadline and try to make the job finished before the deadline. Li et al. [13] study a MapReduce periodical batch scheduling problem with makespan minimization and propose three heuristic algorithms to solve the problem. Fotakis et al. [14] propose an approximation algorithm for the nonpreemptive MapReduce scheduling. They also verify the performance of the algorithm through numerical experiments. Most studies in the field of MapReduce scheduling have only focused on the improvement of algorithms for the classical MapReduce scheduling problem. However, few researches expand the classical MapReduce scheduling problem according to its practical applications.

In this work, we consider an extended MapReduce scheduling problem with the open-map task and several series-reduce tasks. In this problem, we assume that the open-map task of each job is splittable (with fixed partition pattern), and all the series-reduce tasks are unsplittable in processing. Specifically, the operations of the open-map task of one job are given beforehand. These operations must be performed in sequence, but without precedence relation. We establish a mixed-integer programming model with the minimum makespan as the objective function then adopt CPLEX to solve it. We design a heuristic algorithm called *L-F* algorithm for this problem and compare it with the baseline heuristic algorithms. Finally, we found that *L-F* algorithm has advantages through numerical experiments.

The main contributions of this work are threefold:

(1) Different from the classical MapReduce scheduling problem, we consider series-reduce tasks (in this work, we are mainly considering two series-reduce tasks). The map operations cannot be processed in parallel.

(2) We consider the processing scenario in which each job is processed on a set of unrelated parallel machines with different processing speeds.

(3) We establish a mixed-integer programming model; then, we adopt CPLEX and two classical heuristic algorithms to solve this problem. Besides, we design an *L-F* algorithm to solve this problem more efficiently.

The remainder of this paper is organized as follows. Section 2 mathematically models the problem and solves the problem via CPLEX. In order to solve this problem in large scale, Section 3 designs heuristic algorithms (a genetic algorithm, a simulated annealing algorithm, and an *L-F* algorithm). Section 4 compares the designed *L-F* algorithm with genetic algorithm and simulated annealing algorithm. Section 5 concludes the paper and gives future research directions.

2. Problem Statement and Mathematical Model

There are n jobs to be processed on the m unrelated machines. Each job contains k operations. There are one open-map task and several series-reduce tasks. The operations 1 to $(k-r)$ represent the map operations of the open-map task of each job. The operations $(k-r+1)$ to k represent reduce operations of the series-reduce tasks of the job. The specific characteristics of a job with two series-reduce tasks are shown in Figure 1. The processing time of operation u $(1 \leq u \leq k)$ for job j by machine i is denoted by P_i^{ju}. In addition, the processing of any two (no matter map or reduce) operations of a job cannot be overlapped. That is to say, any job cannot be processed on more than one machine at the same time. The other processing rules are as follows. We aim to produce a processing schedule so as to minimize the makespan.

(1) In the open-map task of any job, there is no strict requirement for the processing sequence of the given map operations.

(2) For any job, only after completing map operations, the reduce operations can start to be processed.

(3) The reduce operations can be processed by any available machine, and the reduce operations of a job are processed in a strict sequence. After the previous reduce operation is finished, the next one can be processed.

The problem under consideration is based on the assumptions that setup time of machine and transportation time of job between operations are not considered. In this work, we propose scheduling methods to minimize the makespan for this problem. Without loss of generality, we assume that in a reasonable schedule, there may be idle time between the processing times of any two consecutive operations on a machine. Below, we propose a mixed-integer linear programming model for this problem. For convenience, we first explain the basic symbols and decision variables and then build the model.

2.1. Basic Notations

2.1.1. Indices

i: index of machines

j: index of jobs

u: index of all operations of a job

h: index of positions on a machine

v: index of reduce operations of a job

s: index of processing orders of operations of a job

2.1.2. Input Parameters

$\mathscr{M}$: set of machines, and $\mathscr{M} = \{1, 2, \ldots, m\}$

J: set of jobs, and $J = \{1, 2, \ldots, n\}$

r: the number of reduce operations

U: set of operations, and $U = \{1, 2, \ldots, k-r, k-r+1, \ldots, k\}$

H: set of positions on a machine, and $H = \{1, 2, \ldots, p\}$

M: a sufficiently large positive integer

P_i^{ju}: the processing time of operation u of job j when processed by machine i

2.1.3. Decision Variables

x_{ih}^{ju}: a binary variable equal to 1 if operation u of job j begins to be processed by machine i at position h, 0 otherwise.

z_i^{ju}: a binary variable equal to 1 if operation u of job j is processed by machine i, 0 otherwise.

y_s^{ju}: a binary variable equal to 1 if operation u in job j is processed in the order of s, 0 otherwise. Note that if $u \in [r+1, k]$, s must be equal to u, because they correspond to series-reduce tasks.

C^{ju}: the completion time of the operation u of job j.

$C_{\max}$: the maximum completion time of operation u of job j.

S^{ju}: the time when operation u of job j began to be processed.

P^{ju}: the processing time of operation u of job j.

2.2. Mathematical Model.

The objective is to minimize the makespan, as shown in the following formula:

$$\min f = C_{\max}, \tag{1}$$

subject to

$$\sum_{j \in J} \sum_{u \in U} x_{ih}^{ju} = 1, \quad i \in \mathscr{M}, h \in H, \tag{2}$$

$$\sum_{h \in H} x_{ih}^{ju} = z_i^{ju} 1, \quad i \in \mathscr{M}, j \in J, u \in U, \tag{3}$$

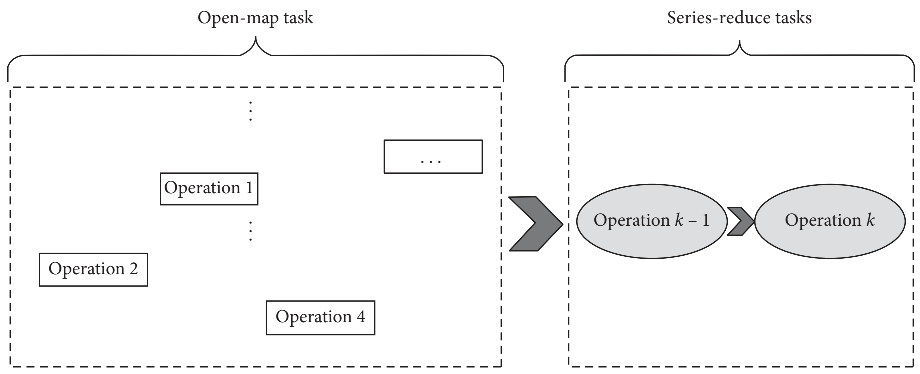

Figure 1: The specific process characteristics of the extended MapReduce job (the case that $r = 2$).

$$\sum_{i\in\mathscr{M}} z_i^{ju} = 1, \quad j \in J,\ u \in U, \tag{4}$$

$$\sum_{s\in U} y_s^{ju} = 1, \quad j \in J,\ u \in U, \tag{5}$$

$$\sum_{u\in U} y_s^{ju} = 1, \quad j \in J,\ u \in U, \tag{6}$$

$$S^{ju} - C^{ju'} \geq 0 - M * \left(2 - y_s^{ju} - y_{s'}^{ju'}\right), \quad j \in J,\ u, u', s, s' \in U, \text{ and } u' \neq u, s \geq s', \tag{7}$$

$$P^{ju} = \sum_{i\in\mathscr{M}} P_i^{ju} * z_i^{ju}, \quad j \in J,\ u \in U, \tag{8}$$

$$S^{ju} \geq 0 - M * \left(1 - x_1^{ju}\right), \quad j \in J,\ u \in U, \tag{9}$$

$$\begin{aligned} S^{ju} - C^{j'u'} \geq{} & 0 - M \\ & * \left(2 - \sum_{i\in M} x_{ih}^{ju} - \sum_{i\in M} x_{ih}^{j'u'}\right), \quad j, j' \in J,\ u, u' \in U, \text{ and } j \neq j' \text{ or } u \neq u', h, h' \in H \text{ and } h > h, \end{aligned} \tag{10}$$

$$\begin{aligned} S^{ju} \geq{} & \sum_{j'\in J} \sum_{u'\in U} \sum_{h'\in[1,h-1]} P_i^{j'u'} * x_{ih}^{j'u'} - M * \left(1 - x_{ih}^{ju}\right), \\ & j \in J,\ u \in U,\ h \in H \text{ and } j \neq j' \text{ or } u \neq u', \end{aligned} \tag{11}$$

$$C^{ju} = S^{ju} + P^{ju}, \quad j \in J,\ u \in U, \tag{12}$$

$$C_{\max} \geq C^{jk}, \quad j \in J, \tag{13}$$

$$S^{jv} \geq C^{jv'}, \quad j \in J,\ v \in \{k - r + 1, k - r + 2, \ldots, k - 1\},\ v' \in \{v + 1, v + 2, \ldots, k\}, \tag{14}$$

$$S^{jv} \geq C^{ju}, \quad j \in J,\ u \in \{1, 2, \ldots, k - r\},\ v \in \{k - r + 1, k - r + 2, \ldots, k\}, \tag{15}$$

$$x_{ih}^{ju}, z_i^{ju}, y_s^{ju} \in \{0, 1\}, \quad i \in \mathscr{M},\ j \in J,\ u, s \in U,\ h \in H, \tag{16}$$

$$C^{ju}, C_{\max}, P^{ju}, S^{ju}, \quad j \in J,\ u \in U. \tag{17}$$

Constraints (2)–(4) guarantee that each operation can only be processed by at most one machine at a time, and any operation must be completed. Constraints (5)–(7) guarantee that any operation cannot be processed in parallel. Constraint (8) calculates the actual processing time of operation u of job j. Constraints (9)–(11) calculate the start processing time of any operation and describe the relationship between the start processing time of two operations on the same machine. Constraints (12)-(13) calculate the completion time of any operation and the makespan. Constraints (14)-(15) calculate the start processing time of the reduce operation. Constraints (16)-(17) limit the ranges of the variables.

3. Heuristic Algorithms

As the problem is NP-hard, we propose a genetic algorithm and a simulated annealing algorithm to solve it. Since the job with two reduce tasks are common in practice, we solve the extended MapReduce scheduling problem with two reduce tasks via CPLEX and heuristic algorithms. Otherwise, a new heuristic algorithm, i.e., *L-F* algorithm, is proposed to solve this problem more efficiently.

3.1. Genetic Algorithm. Genetic algorithms are computational models to simulate the natural selection and genetic mechanism of Darwin's biological evolution theory. It is a method to search the optimal solution by simulating the natural evolution process. Genetic algorithms have been widely used in scheduling problems [15, 16] and other NP-complete problems [17]. In genetic algorithms, each chromosome represents a feasible solution to the problem studied. In this section, for solving this problem, we generate a certain number of initial feasible solutions, i.e., initial population, and then through cyclic selection of the parents, we get offspring through the process of crossover and mutation of the parents' chromosomes, constantly optimizing to find the optimal solution.

3.1.1. Chromosome Representation. In this study, each chromosome can be divided into two parts on average. All the elements of the first part represent all the operations of all the jobs, which are called operation genes. All the elements of the second part represent the number of machines, which are called machine genes. For example, in Figure 2, the first element of the chromosome is 21, representing operation 1 of job 2, and the tenth element (the first machine gene) is 1, representing machine 1. The operation genes correspond to the machine genes one by one. For example, in Figure 2, the first operation gene is 21, and the first machine gene is 1, which means that operation 1 of job 2 is processed by machine 1.

In addition, on each chromosome, if both operations belong to a job, the start processing time in the subsequent operation is later than the completion time of the preceding operation. Similarly, on each chromosome, if two operations are assigned to the same machine, the start time of the subsequent operation is later than the completion time of the previous operation.

3.1.2. Initialization of Chromosomes. We randomly generate chromosomes according to the number of jobs, the number of machines, and the processing rules. For example, in Figure 2, in the first part of chromosomes, operation 2 of job 2 (22) and operation 1 of job 1 (11) correspond to machine 2. However, operation 2 of job 2 is ahead of operation 1 of job 1. That is to say, the starting time of operation 1 of job 1 must be after the completion time of operation 2 of job 2.

3.1.3. Fitness Function. In the genetic algorithm, fitness determines the probability that a chromosome is selected as a parent chromosome. We assume that NIND is the number of individuals in the population, j is the jth chromosome in the population, and obj (j) is the corresponding objective value of chromosome j in the population. Since this study aims to minimize the objective function, we define the fitness value of chromosome j as $1/\text{obj}(j)$.

3.1.4. Crossover and Mutation. In genetic algorithm, after mutation and crossover, the offspring cannot only retain the high-quality genes stained by the parent chromosomes, but also make the new chromosome population more diverse. However, the processes of crossover and mutation are easy to generate infeasible solution. For example, in this study, if we cross or mutate the operation genes of the chromosomes, this will easily lead to some operations being reprocessed and some operations not being processed, which is obviously inconsistent with our intention. Adopting a correcting algorithm to modify unfeasible offspring is very time-consuming; hence, it is preferable to design operators such that precedence constraints are not violated [18].

In this problem, any operation can be processed by any machine. Therefore, in order to avoid infeasible solutions, we only cross the machine genes. Since there is no crossover of the operations genes in the chromosomes, the order of operation genes in the chromosome is still feasible. It is obvious that each of the obtained offspring still represents a feasible solution. From the aspect of chromosome mutation, we randomly select a chromosome in a population. When the generated random number is less than the mutation probability, based on empirical data (the following six steps are executed with a probability of 0.1 to 1, respectively; we find that the effect is better when the probability is 0.3), we divide the genetic algorithm into the following six steps to carry out chromosome mutation based on empirical data:

Step 1. Randomly select two machine genes on the chromosome with the probability of 0.3, and then generate them again.

Step 2. Randomly select two jobs with the probability of 0.3, and then exchange all the operation genes of the two jobs in chromosome. The genes of the corresponding machine parts are then randomly generated.

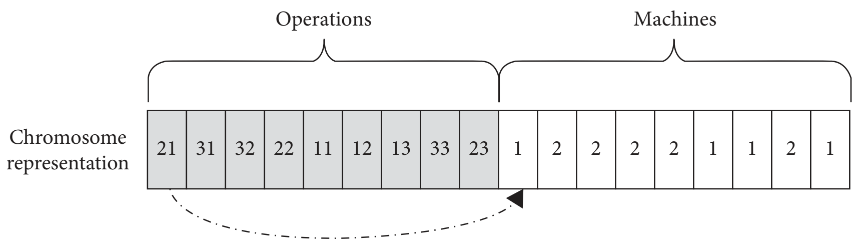

Figure 2: Chromosome representation.

Step 3. Randomly select a job with the probability of 0.3, and then exchange the operation genes in chromosome that represent the map operations of the job.

Step 4. Randomly select two jobs with the probability of 0.3. Then, randomly select one map operation from each job, under the condition that no infeasible solution is generated. Then, the map operation genes of the two jobs are changed.

Step 5. Randomly select two jobs with the probability of 0.3, and then change the first reduce operation genes of the two jobs.

Step 6. Randomly select two jobs with the probability of 0.3, and then change the second reduce operation genes of the two jobs.

3.1.5. Termination Conditions. When the following conditions are satisfied, the genetic algorithm runs out. (1) The number of iterations reaches the maximum preset number; (2) the fitness value of the best chromosome in the population remains unchanged in the last 100 consecutive generations.

3.2. Simulated Annealing Algorithm. Simulated annealing algorithms (SA) are local search methods [19]. In recent years, the algorithms have been more and more widely used in scheduling problems [20]. The earliest idea of the algorithm is proposed by [21]. Simulated annealing algorithms are derived from the principle of solid annealing [22]. The solid is heated to a sufficiently high temperature and then allowed to cool slowly. When heating, the internal particles of the solid become disordered with temperature rise, and the internal energy increases, while the particles gradually decrease during cooling. It tends to be in an orderly state, reaching an equilibrium state at a temperature, and finally reaching the ground state at normal temperature. The internal energy is minimized. In this section, we solve the above problem via a simulated annealing algorithm.

3.2.1. Initial Feasible Solution. We encode the initial feasible solution with $2^*\eta$ (η is the sum of the number of operations of all jobs) elements. In the initial feasible solution, the first η elements represent the operations, which are called operation elements. The last η elements represent the machines, which are called machine elements. The initial feasible decoding method is the same as that of chromosome coding in genetic algorithm. However, it is worth noting that in simulated annealing algorithm, we only generate an initial feasible solution. The diversity of feasible solutions is achieved by continuously generating new flexible solutions.

3.2.2. New Feasible Solution. To avoid generating infeasible solutions, we transform the initial feasible solutions in the following steps based on empirical data (the following six steps are executed with a probability of 0.1 to 1, respectively; we find that the effect is better when the probability is 0.3):

Step 1. Randomly select two machines elements from the old solution with a probability of 0.3, and then generate them randomly again.

Step 2. With a probability of 0.3, the operation elements in the new solution are mutated by randomly selecting two jobs. All the operation elements of the two jobs are exchanged with each other in the solution, and then the elements of the corresponding machine parts are randomly generated.

Step 3. Randomly select a job with a probability of 0.3, and then exchange the position of the operation elements in the solution of the open-map task of the job.

Step 4. Randomly select two jobs with a probability of 0.3, and then change the elements in the solution representing the map operations of the two jobs.

Step 5. Randomly select two jobs with a probability of 0.3, and then exchange the elements in the solution representing the first reduce operations of the two jobs.

Step 6. Randomly select two jobs with a probability of 0.3, and then exchange the elements in the solution representing the second reduce operations of the two jobs.

Each initial feasible solution is called new feasible solution after the above steps of transformation.

3.2.3. Selection. We use $R1$, $R2$, and T_0 to represent the objective values of the initial feasible solution, the objective values of the new feasible solution, and current temperature, respectively. We judge whether to accept the new feasible solution according to the Metropolis criterion. That is, if the objective value of the generated new feasible solution is better than the objective value of the initial feasible solution, then accept the new feasible solution. Otherwise, it is necessary to further judge whether $\exp(R1 - R2/T_0)$ is greater than or equal to the number randomly generated from 0 to 1.

If so, accept the new feasible solution; otherwise, refuse to accept the new feasible solution.

3.3. L-F Algorithm. For solving the extended MapReduce scheduling problem, an algorithm based on Least Processing Time first (LPT) rule and First Come, First Served (FCFS) rule is proposed, which is called an *L-F* algorithm. We divide all the extended MapReduce jobs' processing process into three stages. The first stage is dedicated to the open-map task, the second stage is for the first series-reduce task, and the third stage is for the second series-reduce.

When an operation is processed by a machine which takes the longest processing time to process this operation in the machine set, we call it processed in the worst case. For example, if the processing time of an operation is 10 on machine 1 and 5 on machine 2, then the processing time in the worst case is 10. In order that a job with a longer processing time in the worst case can be processed on a faster machine, in the first stage, all the map operations of all jobs are processed in the sequence of processing time under the worst case from the longest to the shortest. When starting to process each operation, we choose the machine with the minimum completion time for the operation. For the operations which belong to the second and third stages of the process, we sort them according to the principle of first come, first served. Among all the jobs, open-map task of a job is the first to be completed, and the first series-reduce task of the job is the first to start being processed. Similarly, the first series-reduce task of a job is the first to be completed, and the second series-reduce task of the job is the first to start being processed.

The specific steps of *L-F* algorithm are as follows:

Step 1. According to the processing time in the worst case, the map operations are sorted in a nonincreasing order. Each operation is processed on the machine that can complete it in the shortest completion time.

Step 2. After all the map operations of a job is completed, the first reduce operation 10 of the job starts to be processed on the available machine that can complete it in the shortest completion time.

Step 3. After the first reduce operation of a job is completed, the second reduce operation starts to be processed on the available machine that can complete it in the shortest completion time.

4. Numerical Experiments

In this section, in order to compare the performance of the *L-F* algorithm, the genetic algorithm, and the simulated annealing algorithm, we carry out three numerical experiments in different scales by MATLAB R2014b on a PC with Intel Core i5. This paper focuses on solving the extended MapReduce scheduling problem with two reduce tasks; thus, in numerical experiments, only two reduce tasks are included in each job.

In small-scale instances, we adopt CPLEX to solve the mathematical model. CPLEX is one of the important tools for solving mixed linear programming problems [23]. In the genetic algorithm, based on parameter of experiments, we assume that the initial population number is 100, the number of iterations is 200, the crossover rate is 0.3, and the mutation rate is 0.7. In the simulated annealing algorithm, we assume that the initial temperature is 1000, the minimum temperature is 0.001, the cooling rate is 0.1, and the number of iterations at each temperature is 100. In addition, we assume that the solution value obtained by one heuristic algorithm is obj, and the optimal solution obtained by CPLEX is obj^*; then, there is $\text{gap} = (\text{obj} - \text{obj}^*/\text{obj}^*)$.

4.1. Small-Scale Instances. We first carry out numerical experiments on small-scale instances. In each instance, we first randomly generate 10 sets of processing time data for operations of all jobs from 1 to 5. We then calculate the average of the objective values, the gap value, and the running time of CPLEX, the genetic algorithm, the simulated annealing algorithm, and *L-F* algorithm; the results are shown in Table 1. Besides, we also randomly generate 10 sets of processing time data for operations of all jobs from 1 to 10 for each instance. Similarly, we then calculate the average of the objective values, the gap values, and the running times of CPLEX and the three heuristic algorithms; the results are shown in Table 2.

From the small-scale instances, it can be seen that when the range of the processing time of operations is enlarged from (1, 5) to (1, 10), the running time of CPLEX increases significantly. While the running time of the genetic algorithm, the simulated annealing algorithm, and *L-F* algorithm does not change significantly. From the aspect of gap value, when the range of the processing time of operations is enlarged, the gap values of the genetic algorithm and simulated annealing algorithm decrease slightly. However, the gap value of *L-F* algorithm increases slightly with the increase of processing time of all the operations.

Besides, Tables 1 and 2 report that the running times of all the three heuristic algorithms are significantly less than CPLEX. *L-F* algorithm is slightly better than the genetic algorithm and the simulated annealing algorithm in solution quality and computational time. In addition, the genetic algorithm is better than the simulated annealing algorithm in terms of running time. The gap value of the genetic algorithm is slightly smaller than that of the simulated annealing algorithm. Specifically, when the number of machines is 2, the genetic algorithm and the simulated annealing algorithm are better than *L-F* algorithm in solution quality. In general, gap values of the genetic algorithm and the simulated annealing algorithm are proportional to the number of machines and jobs. However, with the increase of the number of machines and jobs, the gap value of *L-F* algorithm decreases in general.

4.2. Large-Scale Instances. In this section, for comparing the performance of the genetic algorithm, the simulated annealing algorithm, and *L-F* algorithm, we carry out numerical experiments on large scale. In this numerical experiment, we also randomly generate 10 sets of data in a range of 1 to 10 of processing time of operations of all jobs under the different scale of machines and jobs. Due to the fact that

Table 1: Experimental results for small-scale instances of the processing time of operations range from 1 to 5.

(m, n, k)	CPLEX		GA			SA			*L-F*		
	obj*	Time	obj	Time	gap	obj	Time	gap	obj	Time	gap
(2, 3, 3)	10.90	0.24	11.40	1.75	0.05	11.50	2.75	0.06	12.40	0.00	0.14
(2, 3, 4)	14.40	5.24	15.20	2.18	0.06	15.30	3.17	0.06	17.00	0.00	0.18
(2, 3, 5)	16.00	45.95	16.90	2.53	0.06	17.60	3.63	0.10	18.40	0.00	0.15
(2, 4, 3)	13.50	7.14	13.80	2.07	0.02	13.90	3.12	0.03	15.00	0.00	0.11
(2, 4, 4)	16.70	545.07	18.10	2.61	0.08	18.40	3.82	0.10	19.70	0.00	0.18
(2, 5, 3)	16.70	32.14	17.30	2.42	0.04	17.10	3.67	0.02	17.70	0.00	0.06
(3, 4, 3)	8.00	1.07	9.50	2.07	0.19	9.90	3.22	0.24	9.40	0.00	0.18
(3, 4, 4)	10.60	112.79	12.80	2.61	0.21	13.60	3.80	0.28	13.30	0.00	0.25
(3, 5, 3)	9.40	25.75	11.30	2.41	0.20	11.30	3.61	0.20	11.10	0.00	0.18
(3, 6, 3)	11.60	7231.70	13.30	2.74	0.15	13.90	4.02	0.20	13.90	0.00	0.20
(4, 5, 3)	7.30	1.35	9.40	2.43	0.29	9.90	3.58	0.36	8.20	0.00	0.12
(4, 5, 4)	9.30	243.26	13.40	3.13	0.44	14.10	4.46	0.52	12.60	0.00	0.35
(4, 6, 3)	8.30	49.19	10.90	2.82	0.31	11.80	4.04	0.42	9.50	0.00	0.14
(5, 5, 4)	7.50	12.23	11.20	3.13	0.49	12.60	4.40	0.68	9.40	0.00	0.25
(5, 6, 3)	6.20	2.52	8.90	2.78	0.44	10.50	3.99	0.69	7.60	0.00	0.23
(5, 6, 4)	8.20	177.18	12.80	3.51	0.56	14.60	4.95	0.78	10.60	0.00	0.29
(5, 7, 3)	6.90	251.84	11.10	3.16	0.61	11.80	4.19	0.71	8.40	0.00	0.22
(8, 10, 3)	5.70	304.38	11.00	4.16	0.93	12.00	5.71	1.11	6.80	0.00	0.19
Average	10.40	502.83	12.68	2.70	0.28	13.32	3.90	0.36	12.28	0.00	0.19

Note. gap = (obj − obj*/obj*).

Table 2: Experimental results for small-scale instances of the processing time of operations range from 1 to 10.

(m, n, k)	CPLEX		GA			SA			*L-F*		
	obj*	Time	obj	Time	gap	obj	Time	gap	obj	Time	gap
(2, 3, 3)	19.20	0.24	20.30	1.76	0.06	20.70	2.74	0.08	21.80	0.00	0.14
(2, 3, 4)	24.00	5.93	26.40	2.19	0.10	25.90	3.16	0.12	27.40	0.00	0.14
(2, 3, 5)	34.40	553.70	36.50	2.55	0.06	37.80	3.74	0.10	39.80	0.00	0.16
(2, 4, 3)	21.50	7.93	22.80	2.09	0.06	22.90	3.08	0.07	24.80	0.00	0.15
(2, 4, 4)	30.90	8047.90	32.60	2.63	0.06	34.40	3.77	0.11	37.20	0.00	0.20
(2, 5, 3)	30.90	2910.40	32.00	2.44	0.04	32.00	3.60	0.04	36.30	0.00	0.17
(3, 4, 3)	15.20	1.20	28.20	2.11	0.20	18.60	3.15	0.22	17.30	0.00	0.14
(3, 4, 4)	19.00	32.20	23.70	2.65	0.25	25.10	3.80	0.32	23.30	0.00	0.23
(3, 5, 3)	16.60	251.24	19.70	2.45	0.19	20.80	3.60	0.25	20.00	0.00	0.20
(3, 6, 3)	19.30	2181.60	22.80	2.80	0.18	24.00	4.15	0.24	22.80	0.00	0.18
(4, 5, 3)	13.10	1.12	15.50	2.44	0.18	17.10	3.60	0.31	13.70	0.00	0.05
(4, 5, 4)	15.10	209.78	22.40	3.10	0.48	24.70	4.12	0.64	18.90	0.00	0.25
(4, 6, 3)	14.10	35.06	18.00	2.80	0.28	19.50	4.07	0.38	16.40	0.00	0.23
(5, 5, 4)	12.00	9.94	19.40	3.09	0.62	21.90	4.48	0.83	14.50	0.00	0.21
(5, 6, 3)	11.60	3.03	16.00	2.76	0.38	17.90	4.03	0.54	12.90	0.00	0.11
(5, 6, 4)	11.80	22.69	18.20	3.12	0.54	20.20	4.43	0.71	13.70	0.00	0.16
(5, 7, 3)	10.60	77.17	17.20	3.13	0.62	20.50	4.52	0.93	13.10	0.00	0.24
(8, 10, 3)	9.10	101.94	18.70	4.13	1.05	21.40	5.81	1.35	10.70	0.00	0.18
Average	18.11	802.95	22.24	2.68	0.30	23.69	3.89	0.40	21.37	0.00	0.17

Note. gap = (obj − obj*/obj*).

CPLEX takes a lot of time to solve the problem, we calculate gap value by replacing the exact objective value with the minimum objective value obtained by the three heuristic algorithms. In Table 3, we further expand the scale of jobs and machines to compare the three heuristic algorithms.

From Table 3, we can see that *L-F* algorithm has better performance than the genetic algorithm and the simulated annealing algorithm in large-scale instances. The gap value of the genetic algorithm and the simulated annealing algorithm is proportional to the number of operations. Otherwise, when the number of jobs and operations is constant, the gap values of the genetic algorithm and the simulated annealing algorithm are proportional to the number of machines.

Table 3 also reports that when the number of machines increases form 4 to 6, the gap values of the genetic algorithm and the simulated annealing algorithm increase significantly. However, when the number of operations of each job and machines is constant, with the increase of the number of jobs, the gap value of *L-F* algorithm decreases, while the gap

Table 3: Experimental results for large-scale instances.

(m, n, k)	GA			SA			*L-F*		
	obj	Time	gap	obj	Time	gap	obj	Time	gap
(2, 30, 3)	178.70*	12.17	0.00	183.90	15.96	0.03	191.50	0.00	0.07
(2, 30, 5)	323.90*	21.53	0.00	332.10	27.28	0.03	333.50	0.01	0.03
(2, 30, 10)	683.40	55.08	0.12	696.80	65.45	0.14	611.60*	0.01	0.00
(2, 50, 3)	312.80*	21.03	0.05	318.30	27.29	0.02	327.00	0.01	0.05
(2, 50, 5)	563.20*	47.70	0.00	572.80	50.83	0.02	565.40	0.01	0.00
(2, 50, 10)	1204.10	117.87	0.17	1193.80	138.27	0.16	1029.90*	0.01	0.00
(2, 100, 3)	671.30	51.13	0.02	662.70	64.93	0.01	658.00*	0.01	0.00
(2, 100, 5)	1181.30	111.44	0.07	1168.30	135.79	0.06	1103.70*	0.01	0.00
(2, 100, 10)	2536.50	376.14	0.21	2480.60	435.82	0.18	2102.60*	0.03	0.00
(4, 30, 3)	87.80	11.85	0.19	95.10	15.66	0.29	73.70*	0.00	0.00
(4, 30, 5)	164.80	21.72	0.28	173.60	27.71	0.35	128.50*	0.01	0.00
(4, 30, 10)	368.70	55.72	0.49	378.10	66.39	0.53	247.20*	0.01	0.00
(4, 50, 3)	151.30	21.28	0.31	157.80	27.58	0.37	115.20*	0.01	0.00
(4, 50, 5)	280.60	41.30	0.39	291.90	51.65	0.45	201.50*	0.01	0.00
(4, 50, 10)	652.60	116.99	0.57	638.80	137.93	0.60	398.90*	0.02	0.00
(4, 100, 3)	338.70	53.83	0.47	333.50	68.64	0.44	231.10*	0.01	0.00
(4, 100, 5)	606.30	116.50	0.47	604.50	141.69	0.47	411.80*	0.02	0.00
(4, 100, 10)	1329.60	376.39	0.58	1301.20	433.62	0.62	821.20*	0.05	0.00
(6, 30, 3)	62.30	21.29	0.64	68.60	16.10	0.81	37.90*	0.01	0.00
(6, 30, 5)	114.50	22.41	0.77	125.70	28.28	0.94	64.70*	0.01	0.00
(6, 30, 10)	265.60	57.08	0.71	274.70	67.99	0.77	155.30*	0.01	0.00
(6, 50, 3)	109.80	21.16	0.70	113.80	27.39	0.76	64.50*	0.01	0.00
(6, 50, 5)	198.90	42.64	0.81	207.60	53.06	0.89	109.70*	0.01	0.00
(6, 50, 10)	447.00	118.65	0.94	457.50	139.43	0.99	230.40*	0.03	0.00
(6, 100, 3)	232.80	53.58	0.89	233.60	68.20	0.89	123.40*	0.01	0.00
(6, 100, 5)	416.30	114.81	0.88	421.50	139.36	0.90	221.80*	0.03	0.00
(6, 100, 10)	923.20	363.43	1.01	907.60	417.32	0.98	458.20*	0.06	0.00
Average	532.56	89.89	0.43	533.16	107.02	0.47	408.08	0.02	0.01

Note. gap = $(\text{obj} - \text{obj}^*/\text{obj}^*)$.

value of the genetic algorithm and the simulated annealing algorithm does not change significantly.

5. Conclusion

In this paper, we study an extended MapReduce scheduling model with one open-map task and several series-reduce tasks. Different from the classical MapReduce scheduling problem, we assume that the map operations cannot be processed in parallel and the available machines are unrelated machines. We then propose a genetic algorithm and a simulated annealing algorithm and design *L-F* algorithm to solve the problem. Finally, compared with the genetic algorithm and the simulated annealing algorithm, *L-F* algorithm has better performance in large-scale instances.

We propose two future research directions for studying this problem. First, when different operations or jobs are processed on the same machine, there may be setup time. Future research on this problem can take setup time into account. Second, considering preemptive processing, this may be more realistic in some actual production scenarios.

Data Availability

The parameter setting data involved are available, and the data involved in the numerical experiments are all obtained from the solver and also included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant nos. 71771048, 71832001, 71531011, and 71571134) and the Fundamental Research Funds for the Central Universities (Grant no. 2232018H-07).

References

[1] A. Vital-Soto, A. Azab, and M. F. Baki, "Mathematical modeling and a hybridized bacterial foraging optimization algorithm for the flexible job-shop scheduling problem with sequencing flexibility," *Journal of Manufacturing Systems*, vol. 54, pp. 74–93, 2020.

[2] J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[3] Y. Ji, L. Tong, T. He, K.-W. Lee, and L. Zhang, "Improving multi-job MapReduce scheduling in an opportunistic environment," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, pp. 9–16, Santa Clara, CA, USA, June 2013.

[4] H. Chang, M. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee, "Scheduling in MapReduce-like

systems for fast completion time," in *Proceedings of the IEEE INFOCOM*, pp. 3074–3082, Shanghai, China, April 2011.

[5] J. Huang, F. Zheng, Y. Xu, and M. Liu, "Online MapReduce processing on two identical parallel machines," *Journal of Combinatorial Optimization*, vol. 35, no. 1, pp. 216–223, 2017.

[6] B. Moseley, A. Dasgupta, R. Kumar, and T. SarlÓs, "On scheduling in map-reduce and flow-shops," in *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures—SPAA '11*, pp. 289–298, San Jose, CA, USA, June 2011.

[7] T. Luo, Y. Zhu, W. Wu, Y. Xu, and D.-Z. Du, "Online makespan minimization in MapReduce-like systems with complex reduce tasks," *Optimization Letters*, vol. 11, no. 2, pp. 271–277, 2015.

[8] T. Condie, N. Conway, P. Alvaro, and J. M. Hellerstein, "MapReduce online," in *Proceedings of the USENIX Conference on Networked Systems Design & Implementation*, San Jose, CA, USA, April 2010.

[9] Y. Jiang, W. Zhou, and P. Zhou, "An optimal preemptive algorithm for online MapReduce scheduling on two parallel machines," *Asia-Pacific Journal of Operational Research*, vol. 35, no. 3, pp. 1–11, 2018.

[10] Y. Jiang, P. Zhou, T. C. E. Cheng, and M. Ji, "Optimal online algorithms for MapReduce scheduling on two uniform machines," *Optimization Letters*, vol. 13, no. 7, pp. 1663–1676, 2019.

[11] D. Cheng, X. Zhou, Y. Xu, L. Liu, and C. Jiang, "Deadline-aware MapReduce job scheduling with dynamic resource availability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 814–826, 2019.

[12] Z. Tang, J. Zhou, K. Li, and R. Li, "MTSD: a task scheduling algorithm for MapReduce base on deadline constraints," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & Ph. D. Forum*, Shanghai, China, May 2012.

[13] X. Li, T. Jiang, and R. Ruiz, "Heuristics for periodical batch job scheduling in a MapReduce computing framework," *Information Sciences*, vol. 326, no. 1, pp. 119–133, 2016.

[14] D. Fotakis, I. Milis, O. Papadigenopoulos, E. Zampetakis, G. Zois, and Z. S. Georgios, "Scheduling MapReduce jobs and data shuffle on unrelated processors," *Experimental Algorithms*, vol. 9125, pp. 137–150, 2015.

[15] U. Aickelin and K. A. Dowsland, "An indirect Genetic Algorithm for a nurse-scheduling problem," *Computers & Operations Research*, vol. 31, no. 5, pp. 761–778, 2004.

[16] Z. Zakaria and S. Petrovic, "Genetic algorithms for match-up rescheduling of the flexible manufacturing systems," *Computers & Industrial Engineering*, vol. 62, no. 2, pp. 670–686, 2012.

[17] A. Hassani and J. Treijs, "An overview of standard and parallel genetic algorithms," in *Proceedings of the IDT Workshop on Interesting Results in Computer Science and Engineering*, pp. 1–7, Västerås, Sweden, 1975.

[18] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 10, pp. 3202–3212, 2008.

[19] J. Ji, M. Mika, R. Rafał, W. Grzegorz, and W. Jan, "Simulated annealing for multi-mode resource-constrained project scheduling," *Annals of Operations Research*, vol. 102, no. 1–4, pp. 137–155, 2001.

[20] M. Kolonko, "Some new results on simulated annealing applied to the job shop scheduling problem," *European Journal of Operational Research*, vol. 113, no. 1, pp. 123–136, 1999.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[22] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical Science*, vol. 8, no. 1, pp. 10–15, 1993.

[23] Y. Shinano and T. Fujie, "ParaLEX: a parallel extension for the CPLEX mixed integer optimizer," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp. 97–106, Springer, Berlin, Germany, 2007.