Optimization Algorithm for Divisible Load Scheduling on Heterogeneous Star Networks

Xiaoli Wang, Yuping Wang* and Kun Meng
School of Computer Science and Technology, Xidian University, Xi’an, China
Email: wangxiaolibox@gmail.com, ywang@xidian.edu.cn

Abstract—Scheduling divisible loads on heterogeneous distributed computing systems is addressed in this paper. The platform considered here is more general and realistic, where processors are connected in star topology with arbitrary communication and computation speeds and non-zero start-up overheads. A new optimization algorithm, called WX-GA, is proposed to tackle the following four issues: (1) How many processors are needed in computation? (2) Finding the optimal distribution sequence among processors. (3) How much the load fraction should be assigned on each processor? (4) When workload is large enough, what is the sufficient and necessary condition for the minimum processing time? Finally, the experimental results indicate the efficiency and effectiveness of the proposed algorithm.

Index Terms—divisible loads, distributed computing, start-up overheads, optimal distribution sequence, weight-based crossover operator

I. INTRODUCTION

Divisible loads are parallel tasks which can be divided into arbitrary number of fractions [1]. There are no precedence relationships among fractions, thus they can be processed independently on the processors in parallel. Such workload model is useful in many real world applications, e.g., signal processing, image processing, experimental data processing and so on.

Divisible load theory has emerged as a powerful tool for modeling data-intensive computational problems, and a great amount of research on divisible load scheduling has been made in the last decades. Under certain conditions, closed-form expressions for the processing time and load fractions for processors involved in computation have been derived in both homogeneous systems and heterogeneous systems.

In the earlier studies, relatively simple models without start-up overheads have been proposed. For homogeneous networks in blocking mode of communication, a closed-form expression for the processing time of processors in linear topology was derived [2], and asymptotic performance analysis has been made for the cases of bus and tree topologies [3,4]. However, systems in reality are usually heterogeneous systems with arbitrary computation or communication speeds. For heterogeneous star networks, a closed-form expression for optimal processing time was derived by Ghose et al. [5]. It also has been proved that the sequence of load distribution should follow the order in which the communication speeds decrease in order to achieve the minimum processing time. In the case of heterogeneous tree networks, the effect of load distribution sequences on the processing time was analyzed by Kim et al. [6], and an algorithm which optimally determines the order of load distribution was developed. It was shown that the distribution order depends only on the communication speeds between processors but not on the computation speeds.

All the above models considered the blocking mode of communication. Kim [7] first introduced the nonblocking mode of communication in homogeneous systems with processors connected in star topology, and the results on the optimal sequencing and arrangement are presented by Mani et al. [8]. All the above works do not take start-up overheads into consideration. However, zero start-up overheads are quite not realistic for most distributed systems. In the case of constant start-up time on bus networks with blocking mode of communication, Suresh et al. [9], Bharadwaj et al. [10], Blazewicz [11] analyzed the influence of start-up overheads on the optimal processing time and studied the effect of changing the distribution sequence on the processing time. In particular, Bharadwaj et al. [10] gave a necessary and sufficient condition for the existence of the optimal processing time, and it was shown that the processing time is minimized when the load distribution sequence follows the decreasing order of the computation speeds. For heterogeneous star systems with non-blocking mode of communication, closed-form expressions for the processing time and load fractions have been derived by Shang [12].

Given that the start-up overheads and computation speeds of processors are with arbitrary values but that the communication speeds are all different, Beaumont et al. [13] proved that if load fractions were sent to processors according to the decreasing order of communication speeds, then when the processing time becomes large enough, the workload finished during time units is optimal among all possible orderings. For general cases with workload to be completed in arbitrary time, however, the optimal distribution sequence has not been addressed.

Based on the above study, many scholars have done a great deal of extended research [14-24], but the problem of deriving the optimal distribution sequence has not been solved.
In this paper, we propose a new optimization algorithm for divisible load scheduling on heterogeneous distributed systems in blocking mode of communication to address the following four issues:

- whether all processors are needed in computation. If not, how many and which processors should be selected;
- whether the distribution sequence has an effect on the processing time. If true, which order the load distribution should follow to achieve the minimum processing time;
- given a certain distribution sequence, how much the load fraction should be assigned on each processor;
- when workload is large enough, what is the sufficient and necessary condition for the minimum processing time.

The reminder of this paper is outlined as follows. Section II presents the optimization model for divisible load scheduling. Section III proves that when workload is large enough, the sufficient and necessary condition for the minimum processing time is that the distribution sequence follows the decreasing order of the communication speeds. A novel genetic algorithm for divisible load scheduling is proposed in Section IV.

### Figure 1. The time diagram of divisible load scheduling on heterogeneous distributed systems in blocking mode of communication.

When distributing workload fractions, $P_0$ sends data to only one processor at a time and slave processors start computing when they have finished receiving their load fractions, that is, slave processors are in blocking mode of communication and cannot communicate and compute simultaneously.

Since we focus on heterogeneous systems, slave processors are assumed with arbitrary computation and communication speeds. Communication and computation time is proportional to the amount of workload assigned to each processor. Let $w_i$ be the time of processor $P_i$ computing a unit workload, while $g_i$ be the time of link $L_i$ communicating a unit workload. In this paper, we assume communication speeds are much faster than computation speeds; otherwise, only one or two processors should be enough to involve in computation [7].

Besides arbitrary communication speeds and computation speeds, there exists a start-up overhead during each communication from $P_0$ to $P_i$, denoted as $o_i$. Similarly, a start-up overhead $s_i$ exists during each computation of processor $P_i$.

Experiments and their analysis appear in Section V. In the last section, conclusions are made.

### II. OPTIMIZATION MODEL FOR DIVISIBLE LOAD SCHEDULING

Before we give the scheduling model for divisible load, some relevant notations and assumptions should first be introduced. The platform considered in this paper is heterogeneous distributed systems. Processors are connected in a star topology, where $P_0$ is the master processor, while $\{P_1, P_2, \ldots, P_n\}$ are slave processors. $P_0$ connected with others by communication links $\{L_1, L_2, \ldots, L_m\}$. The entire workload, denoted as $W_{\text{total}}$, will be first partitioned into fractions. Then they will be distributed to slave processors in some order by the master processor $P_0$. Processor $P_0$ does not participate in computation itself but only takes the responsibility of assigning load to others.

$$\sum_{i=1}^{n} \alpha_i = W_{\text{total}}, \quad \alpha_i > 0, \quad i = 1, 2, \ldots, n, \quad n \leq m. \quad (1)$$

Bharadwaj, V., et al proved that for various network models the optimality criterion of scheduling divisible loads is that all processors have to finish computing at the same time. If all processors do not stop computing at the same time, certainly the load can be transferred from busy processors to idle processors to minimize the processing time [25]. Therefore, the following equation can be obtained intuitively.

$$s_{\sigma_i} + w_{\sigma_i} \alpha_{\sigma_i} = o_{\sigma_i} + g_{\sigma_i} \alpha_{\sigma_i} + s_{\sigma_{i-1}} + w_{\sigma_{i-1}} \alpha_{\sigma_{i-1}}, \quad i = 1, 2, \ldots, n,$$

which can be rewritten as

$$\alpha_{\sigma_i} = \frac{s_{\sigma_i} + (o_{\sigma_i} + s_{\sigma_{i-1}})}{g_{\sigma_i} + w_{\sigma_i}} + \frac{w_{\sigma_i}}{g_{\sigma_i} + w_{\sigma_i}} \alpha_{\sigma_{i-1}}. \quad (2)$$

Let

$$\phi_{\sigma_i} = \frac{s_{\sigma_i} + (o_{\sigma_i} + s_{\sigma_{i-1}})}{g_{\sigma_i} + w_{\sigma_i}}, \quad \delta_{\sigma_i} = \frac{w_{\sigma_i}}{g_{\sigma_i} + w_{\sigma_i}}. \quad (3)$$

Equation (2) can be written as

$$\alpha_{\sigma_i} = \phi_{\sigma_i} + \delta_{\sigma_i} \alpha_{\sigma_{i-1}}, \quad i = 1, 2, \ldots, n-1 \quad (4)$$

Expressing workload fraction $\alpha_{\sigma_i}$ in terms of $\alpha_{\sigma_1}$ as
where

\[ \alpha_i = \alpha_i \Gamma_i + \Phi_i, \quad i = 2, 3, ..., n, \]  

and

\[ \Gamma_i = \prod_{k=2}^{i} \delta_{ik}, \quad \Phi_i = \sum_{k=2}^{i} \left( \phi_i \prod_{k=1}^{i-k+1} \delta_{ik} \right), \quad i = 2, 3, ..., n. \]

Equation (5) can be set up as follows.

Thus, the closed-form expression of the processing time is given by

\[ T = a_0 + s_0 + \left( g_0 + w_0 \right) \alpha_0 \]

\[ = a_0 + s_0 + \left( g_0 + w_0 \right) W_{\text{total}} - \sum_{k=2}^{n} \Phi_{\alpha_k}. \]

The main objective of scheduling divisible load on heterogeneous distributed systems is to minimize the processing time \( T \). It can be seen from Eq. (7) that processing time \( T \) depends on parameters \( \Phi \) and \( \Gamma \), which are directly decided by the distribution sequence \( \left( P_{\alpha_0}, P_{\alpha_1}, ..., P_{\alpha_n} \right) \) of the processors participating in computation. Thus the divisible load scheduling problem on heterogeneous distributed systems can be modeled as determining the number \( n \) of processors required in computation and the optimal distribution sequence so that the processing time is minimized, that is, an optimization model for scheduling divisible load on heterogeneous distributed systems in blocking mode of communication can be set up as follows.

\[ \min T = \min \left[ a_0 + s_0 + \left( g_0 + w_0 \right) W_{\text{total}} - \sum_{k=2}^{n} \Phi_{\alpha_k} \right]. \]

Subject to:

1. \( n \leq m; \)
2. vector \( \left( \sigma_1, \sigma_2, ..., \sigma_n \right) \) is a sequence of \( n \) different numbers from 1 to \( m; \)
3. \( \Gamma_i = \prod_{k=2}^{i} \phi_k / \left( g_k + w_k \right), \quad i = 2, 3, ..., n; \)
4. \( \Phi_i = \sum_{k=2}^{i} \left[ s_{\alpha_i} - \left( g_{\alpha_i} + w_{\alpha_i} \right) \prod_{k=1}^{i-k+1} \left( g_{\alpha_k} + w_{\alpha_k} \right) \right], \quad i = 2, 3, ..., n; \)

5. \( \alpha_i = \frac{W_{\text{total}} - \sum_{k=2}^{n} \Phi_{\alpha_k}}{1 + \sum_{k=2}^{n} \Gamma_{\alpha_k}}; \)
6. \( \alpha_i = \alpha_i \Gamma_i + \Phi_i, \quad i = 2, 3, ..., n; \)
7. \( \alpha_i > 0, \quad i = 1, 2, ..., n. \)

The key issue of solving the above model is to find the optimal distribution sequence. Considering the practical needs, the easier to obtain the optimal distribution sequence, the better. Fortunately, when workload is large enough, such optimal distribution sequence can be obtained directly by theoretical analysis, which is presented in the following section. For other workload cases, the optimal distribution sequence can be derived efficiently by the proposed genetic algorithm introduced in Section IV.

III. SCHEDULING LARGE ENOUGH WORKLOAD

In this section we will prove that when workload is large enough, the sufficient and necessary condition for the minimum processing time is that the distribution sequence follows the decreasing order of the communication speeds. Before we prove this conclusion, the following lemmas are first introduced.

Lemma 3.1 ([13]) For heterogeneous systems in the blocking mode of communication with zero start-up overheads \( \left( a_i = s_i = 0 \right) \) and arbitrary communication and computation speeds, the distribution sequence should follow the decreasing order of communication speeds in order to achieve the minimum processing time.

Lemma 3.2 ([13]) For heterogeneous systems in the blocking mode of communication with start-up overheads, arbitrary computation speeds and all different communication speeds, if the distribution sequence follows the decreasing order of communication speeds, when the processing time \( T \) is large enough, the amount of processed workload during the time \( T \) is optimal among all possible orderings.

Theorem 3.3 For heterogeneous systems in the blocking mode of communication with startup overheads and arbitrary communication and computation speeds, when workload is large enough, the sufficient and necessary condition for the minimum processing time is that the distribution sequence follows the decreasing order of the communication speeds.

Proof:

(1) Now we prove that when distribution sequence follows the decreasing order of the communication speeds, the corresponding processing time is minimized.

Since the workload is large enough, then all processors should take participate in computation. Assume that there are \( n \) processors in the system, and sort them in the decreasing order of their communication speeds, that is, the increasing order of \( g_i \), where \( i \in \left\{ 1, 2, ..., n \right\} \). Assume the distribution sequence is denoted as \( \left( P_{\alpha_0}, P_{\alpha_1}, ..., P_{\alpha_n} \right) \)

where \( g_{\alpha_i} \leq g_{\alpha_{i+1}} \leq ... \leq g_{\alpha_n} \). Let \( T_{\text{opt}} \) be the processing time for the entire workload in that sequence. Since all processors stop processing at the same time, \( T_{\text{opt}} \) is also the finish time \( T_{\alpha_i} \) of each processor, where \( i = 1, 2, ..., n. \)

According to the model shown in Fig.1, we have

\[ T_{\alpha_i} = T_{\text{opt}} - s_{\alpha_i} + w_{\alpha_i} \alpha_i + \sum_{j=1}^{i} \left( a_{\alpha_j} + g_{\alpha_j} \alpha_j \right) \]

\[ = T_{\text{opt}} - s_{\alpha_i} + \sum_{j=1}^{i} \alpha_{\alpha_j}, \quad i = 1, 2, ..., n. \]

Now let \( a_{\alpha_i} = s_{\alpha_i} = 0 \) with \( i = 1, 2, ..., n \), which means that the start-up overheads are ignored, thus the finish time \( T_{\alpha_i} \) of processor \( P_{\alpha_i} \) will be advanced by

\[ T_{\alpha_i} = T_{\text{opt}} - s_{\alpha_i} + \sum_{j=1}^{i} \alpha_{\alpha_j}, \quad i = 1, 2, ..., n. \]
When \( i = n \), \( T_{\sigma_i} = T_{\sigma'} = \left( s_{\sigma_i} + \sum_{i=1}^{n} o_{\sigma_i} \right) \). That is to say, at the time \( T_{\sigma_i} \), processor \( P_{\sigma_i} \) just finished its workload fraction, while the other processors are still in computation because \( T_{\sigma_i} > T_{\sigma'} \) according to Eq. (8). As is proved in Lemma 3.1, in the case of zero start-up overheads, when the distribution sequence follows the decreasing order of communication speeds, the processing time is the minimum. Let \( T_{\sigma'} \) be the optimal processing time in this case, thus

\[
T_{\sigma'} = T_{\sigma'} - \left( s_{\sigma_i} + \sum_{i=1}^{n} o_{\sigma_i} \right) \leq T_{\sigma'}.
\]  

(10)

Meanwhile, since start-up overheads are ignored, so

\[
T_{\sigma''} \leq T_{\sigma'}.
\]  

(11)

From Eq. (10) and Eq. (11), one can get

\[
T_{\sigma'} \leq T_{\sigma'} + \left( s_{\sigma_i} + \sum_{i=1}^{n} o_{\sigma_i} \right).
\]  

(12)

Thus the workload is large enough, \( T_{\sigma'} \to \infty \). Thus

\[
1 \leq \frac{T_{\sigma'} - \left( s_{\sigma_i} + \sum_{i=1}^{n} o_{\sigma_i} \right)}{T_{\sigma'}} \leq 1 + \frac{T_{\sigma'}}{T_{\sigma'}} \to \infty.
\]

Thus (1) is proved.

(2) Next we prove that when the processing time is optimal, the distribution sequence must follow the decreasing order of the communication speeds. This will be proved by contradiction.

If the workload is large enough, then all processors should take part in computation. Without loss of generality, the optimal distribution sequence is denoted as \( \pi_{\text{opt}} \equiv \left( P_{\sigma_1}, P_{\sigma_2}, ..., P_{\sigma_n} \right) \). We will prove that \( g_{\sigma_i} \leq g_{\sigma_{i+1}} \) holds. If it is not true, one can interchange the sequences of \( P_{\sigma_i} \) and \( P_{\sigma_{i+1}} \). Since workload size is large enough, and then the processing time \( T \) becomes large enough. According to Lemma 3.2, the amount of processed workload in the new order of workload distribution is larger than that of the former sequence, which contradicts the assumption that \( p_{\pi_{\text{opt}}} \) is the optimal sequence.

Thus (2) is proved.

IV. A NEW GENETIC ALGORITHM FOR DIVISIBLE LOAD SCHEDULING: WX-GA

Task scheduling problems are among the well-known hardest combinatorial optimization problems. Here we choose genetic algorithms(GAs), invented by Holland [26], to solve the above model for the simple reason that genetic algorithms have been proven to be a promising technique for many application problems, for example, optimal design, control, and machine learning, etc. [27,28] and they are suitable to solve scheduling optimization problems [29].

A. Population Initialization

The key point of finding the optimal distribution sequence by using genetic algorithms is to develop an encoding scheme that allows genetic operators to generate “legitimate children” without any constraint violation. Applying GA to the scheduling model proposed in this paper has an intrinsic issue: each sequence must contain exactly one instance of a processor and any omission or duplication of a processor or processors leads to an illegal distribution sequence.

In this paper, a sequence with \( m \) processors is directly represented as a permutation \( S_{\pi_{\text{opt}}} \) of \( m \) elements from 1 to \( m \). Thus an individual \( I \) is denoted by \( I = \left( \pi_{\text{opt}}, S_{\pi_{\text{opt}}} \right) \), where \( n \) represents the number of processors taking part in computation.

After determining the encoding scheme, one can generate an initial population of \( N \) individuals by Algorithm 1.

Note that the first gene in each individual, which denotes the number of processors used in computation, will be initialized to \( m \), the total number of processors. It will be modified during the calculation of its cost value.

Algorithm 1 Generate initial population

\textbf{Ensure:} \( N \) initialized individuals \( \left( I_1, I_2, ..., I_N \right) \)

1: for \( i = 1, 2, ..., N \) do
2: generate individual \( I_i = \left( c_0, c_1, ..., c_m \right) \) as follows;
3: \quad let \( c_0 = m \);
4: \quad given an ordered list \( L = 1, 2, ..., m \);
5: \quad for \( j = 1, 2, ..., m \) do
6: \quad \quad randomly take an element from \( L \), assign it to \( c_j \) and then delete this element from \( L \).
7: \quad end for
8: end for

B. Cost Function

The cost function, also called fitness function, is defined over the genetic representation and measures the quality of the represented solution [30]. In general, a cost function is derived from the main objective of the problem and used in successive genetic operations. The main objective of scheduling divisible load on heterogeneous distributed systems is minimizing the processing time. How to map this objective to cost function is presented in Algorithm 2.

Algorithm 2 Cost function

\textbf{Require:} An individual \( I = \left( c_0, c_1, c_2, ..., c_m \right) \) and a workload \( W_{\text{total}} \)

\textbf{Ensure:} The cost function value \( f \left( I \right) \) of individual \( I \), the processing time \( T \), the number \( c_0 \) of processors required in computation and the workload fraction \( \alpha_i \) for each processor with \( i = 1, 2, ..., c_0 \).

1: for \( i = 1, 2, ..., m-1 \) do
2: calculate \( \varphi_{i+1} \) and \( \delta_{i+1} \) by Eq. (3);
3: end for
4: for \( i = 2, ..., m \) do
5: calculate \( \Gamma_i \) and \( \Phi_{i+1} \) by Eq. (5);
6: end for
7: compute $\alpha_i$ and the processing time $T$ by Eq. (6) and Eq. (7) respectively;
8: for $i = c_0, c_0 - 1, \ldots, 1$ do
9:  compute $\alpha_i$ according to Eq. (5);
10: if $\alpha_i \leq 0$ then
11:  let $c_0 = c_0 - 1$ and go to step 7;
12: endif
13: end for

C. Weight-based Crossover Operator

Depending on how the chromosome represents the solution, a direct swap such as N-point crossover may not be possible since the recombination of chromosome may violate the constraint of ordering and thus need to be repaired. In this paper, a novel crossover operator called weight-based crossover is designed to generate offspring.

The easiest way to explain the weight-based crossover is by giving an example. Assume that there exists an ordered list L = (10, 9, 8, 7, 6, 5, 4, 3, 2, 1), which serves as a reference sequence, and that the two parents are represented by

$$parent_1 = (8, 3, 1, 2, 5, 7, 8, 9, 6, 4, 10)$$ and $$parent_2 = (7, 3, 1, 8, 6, 5, 2, 7, 10, 9, 4),$$

where the number of processors used in $parent_1$ is 8 and that in $parent_2$ is 7.

The weight-based crossover operator should be executed on $parent_1$ and $parent_2$, as follows.

1. First according to the number of processors used in computation, assign weight values to each gene in $parent_1$ and $parent_2$ as follows. As is shown in Table I, for individual $parent_1 = \{c_0, c_1, \ldots, c_m\}$, since the number of processors needed in computation is $c_0 = 8$, then for the former 8 genes, from left to right, each of them takes the element of $L$ at the same position as their weight values. That is to say, the weight value of $c_j$ for $parent_1$ is assigned to $L_i$, where $i = 1, 2, \ldots, c_0$. For the remaining genes, which represent the processors that do not take part in computation, let their weight values be zeros. Finally, we have the weight list for $parent_1$: (10, 9, 8, 7, 6, 5, 4, 0, 0, 0, 0). Similarly, the weight list (10, 9, 8, 7, 6, 5, 4, 0, 0, 0, 0) for $parent_2$ can be obtained.

2. Based on the weight lists of $parent_1$ and $parent_2$, the weight value of each processor is calculated in the following way. For each processor, say $P_i$, since its weight in $parent_1$ is 9 and in $parent_2$ is also 9, take the sum of them $9 + 9 = 18$ as the final weight of processor $P_i$. Repeat the above process until the weights of all processors are calculated. Finally, for processors $(P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10})$, the weight list (18, 13, 20, 4, 13, 10, 13, 0, 0, 0) can be obtained.

3. Sort the processors in the descending order of their weights. Thus we have the sorted processor list $(P_3, P_2, P_1, P_5, P_6, P_7, P_8, P_9, P_{10})$ and its corresponding weight list (20, 18, 13, 13, 10, 10, 4, 0, 0, 0).

4. Adjust the order of the processors with the same weight values. For the processors with the same non-zero weight, such as $(P_3, P_5)$, choose the positions of them in their parents. In $parent_1$, $P_3$ is at position 5 and appears in front of $P_5$, while in $parent_2$, $P_5$ is at position 4 and appears in front of $P_3$. Thus we put $P_5$ with a smaller position indicator in front of $P_3$ in the final processor list. If processors are at the same position in their parents, such as $(P_3, P_5)$, then we can randomly select one of them in front of the other. For the processors with zero weight values, since they do not take part in computation, their orders do not matter.

5. Generate an offspring $O = \{c'_0, c'_1, c'_2, \ldots, c'_m\}$ according to the sorted processor list $(P_3, P_2, P_1, P_5, P_6, P_7, P_8, P_9, P_{10})$, where $c'_0$ is initialized to 0. Finally we can get $O = (10, 3, 1, 8, 2, 5, 6, 7, 4, 9, 10)$.

The proposed weight-based crossover operator has two advantages: first, it keeps the optimal subsequence of the parents into offspring, such as $(P_3, P_5)$; second, it takes the order of processors in the distribution sequence into consideration because the more front the position of a processor is in the distribution sequence, the much load fraction it will be assigned, and thus the more important the processor is.

D. Mutation Operator

If we randomly change one number in a chromosome, we are left with one integer duplicated and another missing. The simplest solution is to randomly choose a chromosome to mutate, and then randomly choose two positions within that chromosome to exchange until a local optimal is found. The process of the mutation operator is shown in Algorithm 3 by the pseudo-code.

Algorithm 3 Mutation operator

Require: An individual $I = (c_0, c_1, c_2, \ldots, c_m)$ and mutation probability $p_{mut}$.

Ensure: An offspring $h = (h_0, h_1, \ldots, h_m)$ generated by mutation.
1. Randomly generate a real number $r \in [0,1]$;
2. if $r \leq p_{mut}$ then
3.  let $h = I$;
4: evaluate the cost value $f(h)$ of individual $h$ by Algorithm 2;
5: let $count = 0$ and $f' = f(h)$;
6: randomly generate two integers $a,b \in [1,m]$;
7: swap $h_a$ and $h_b$;
8: evaluate the cost value $f(h)$ of individual $h$ by Algorithm 2;
9: if $f(h) \geq f'$ then
10: swap back $h_a$ and $h_b$ and let $count = count +1$;
11: if $count > m$ stop; otherwise go to step 6;
12: else
13: go to step 5;
14: endif
15: endif

E. A new GA for scheduling divisible loads: WX-GA

Once the encoding scheme and the cost function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover and selection operators. Algorithm 4 presents the process of the proposed genetic algorithm: WX-GA.

Algorithm 4 A new genetic algorithm for scheduling divisible load on heterogeneous systems

1: (Initialization) Choose population size $N$, proper crossover probability $p_{cro}$ and mutation probability $p_{mut}$ etc. Randomly generate initial population $P(0)$ by Algorithm 1. Let the generation number $t = 0$.
2: (Crossover) Choose the parents for crossover in $P(t)$ with probability $p_{cro}$. If the number of parents chosen is odd, then randomly choose an additional one from $P(t)$. Afterwards, randomly match every two parents as a pair and the specific-designed weight based crossover operator is used on each pair to generate one offspring. All the new offspring constitute a set denoted by $O_1$.
3: (Mutation) Select the parents for mutation from set $O_1$ with probability $p_{mut}$. For each chosen parent, the proposed mutation operator is used to generate a new offspring. These new offspring constitute a set by $O_2$.
4: (Selection) The best $E$ solutions from the set $P(t) \cup O_1 \cup O_2$ are maintained in the next generation so that the convergence is faster. The roulette-wheel selection is used to select $N - E$ individuals among the set $P(t) \cup O_1 \cup O_2$ as the next generation population $P(t+1)$.
5: Stopping Criteria: If termination conditions hold, then stop, and keep the best solution obtained as the global optimal solution of the problem; otherwise, go to step 2.

Several experiments are presented in this section to show the effectiveness and efficiency of the proposed algorithm. The parameters of the heterogeneous distributed system are shown in Table II [12]. In the proposed genetic algorithm, the following parameters are adopted: population size $N = 100$, crossover probability $p_{cro} = 0.6$, mutation probability $p_{mut} = 0.02$, elitist number $E = 5$ and stop criterion $t = 2000$.

<table>
<thead>
<tr>
<th>P</th>
<th>o</th>
<th>s</th>
<th>g</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>70.554750</td>
<td>57.951860</td>
<td>0.533424</td>
<td>2.895625</td>
</tr>
<tr>
<td>$p_2$</td>
<td>30.194800</td>
<td>1.4017640</td>
<td>0.774740</td>
<td>7.607236</td>
</tr>
<tr>
<td>$p_3$</td>
<td>81.449010</td>
<td>4.535275</td>
<td>0.709038</td>
<td>4.140327</td>
</tr>
<tr>
<td>$p_4$</td>
<td>86.261930</td>
<td>37.353620</td>
<td>0.790480</td>
<td>9.619532</td>
</tr>
<tr>
<td>$p_5$</td>
<td>87.145800</td>
<td>94.955670</td>
<td>0.056237</td>
<td>3.640187</td>
</tr>
<tr>
<td>$p_6$</td>
<td>52.486840</td>
<td>5.350452</td>
<td>0.767112</td>
<td>5.924582</td>
</tr>
<tr>
<td>$p_7$</td>
<td>46.870010</td>
<td>62.269670</td>
<td>0.298165</td>
<td>6.478212</td>
</tr>
<tr>
<td>$p_8$</td>
<td>26.379290</td>
<td>82.980160</td>
<td>0.279342</td>
<td>8.246021</td>
</tr>
<tr>
<td>$p_9$</td>
<td>58.916300</td>
<td>91.096430</td>
<td>0.980003</td>
<td>5.338731</td>
</tr>
<tr>
<td>$p_{10}$</td>
<td>69.511550</td>
<td>24.393140</td>
<td>0.980003</td>
<td>5.338731</td>
</tr>
<tr>
<td>$p_{11}$</td>
<td>10.636970</td>
<td>67.617590</td>
<td>0.999415</td>
<td>1.570390</td>
</tr>
<tr>
<td>$p_{12}$</td>
<td>57.518380</td>
<td>10.302260</td>
<td>0.100052</td>
<td>2.268660</td>
</tr>
<tr>
<td>$p_{13}$</td>
<td>28.448030</td>
<td>29.577290</td>
<td>0.045649</td>
<td>3.820107</td>
</tr>
<tr>
<td>$p_{14}$</td>
<td>30.090500</td>
<td>97.982940</td>
<td>0.948571</td>
<td>4.013743</td>
</tr>
<tr>
<td>$p_{15}$</td>
<td>27.828000</td>
<td>16.282160</td>
<td>0.160442</td>
<td>6.465871</td>
</tr>
</tbody>
</table>

In the first experiment, the workload size ranges from 100 to 10000. For convenience, let GA represents the algorithm WX-GA proposed in this paper, IG indicates the algorithms given by Beaumont et al. [13] and Shang [12], which schedules divisible load in the sequence of increasing value of $g$, while IW given by Bharadwaj et al. [10], which schedules workload in the sequence of increasing value of $w$. The processing time and used machine numbers for different workload by these three scheduling algorithms are recorded in Table III. It can be seen from Table III that the processing time used by the proposed algorithm is much less than that used by other two compared algorithms for all test cases.

Fig. 2(a) intuitively shows the processing time difference between algorithm IW and WX-GA, while Fig. 2(b) shows the time difference between IG and WX-GA. As is shown in Fig. 2(a), with the increasing size of workload, the difference of processing time is found in linear growth between the proposed algorithm IW and WX-GA. In other words, the proposed algorithm is much more effective in finding the optimal distribution sequence. Similarly, it can be seen from Fig. 2(b) that when workload size ranges from 100 to 9900, the processing time obtained by IG is larger than that by WX-GA. What’s more, we show the numbers of processors that are selected to participate in computation obtained by algorithm WX-GA and IG in Fig. 3.
Table III: Processing time for different workload from 100 to 9900.

<table>
<thead>
<tr>
<th>Alg</th>
<th>Size</th>
<th>No</th>
<th>Time</th>
<th>Size</th>
<th>No</th>
<th>Time</th>
<th>Size</th>
<th>No</th>
<th>Time</th>
<th>Size</th>
<th>No</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>100</td>
<td>5</td>
<td>209.255</td>
<td>300</td>
<td>7</td>
<td>411.123</td>
<td>500</td>
<td>9</td>
<td>572.316</td>
<td>700</td>
<td>10</td>
<td>700.423</td>
</tr>
<tr>
<td>IG</td>
<td>285.379</td>
<td>3</td>
<td>475.671</td>
<td>581.707</td>
<td>5</td>
<td>809.344</td>
<td>7</td>
<td>1004.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IW</td>
<td>1184.95</td>
<td>12</td>
<td>930.701</td>
<td>1300</td>
<td>10</td>
<td>1041.08</td>
<td>13</td>
<td>1149.98</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>1059.494</td>
<td>11</td>
<td>980.935</td>
<td>1096.19</td>
<td>15</td>
<td>1207.3</td>
<td>15</td>
<td>1274.87</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>1199.39</td>
<td>9</td>
<td>1386.48</td>
<td>1569.33</td>
<td>10</td>
<td>1748.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IW</td>
<td>1256.35</td>
<td>13</td>
<td>1468.04</td>
<td>1149.98</td>
<td>13</td>
<td>1207.3</td>
<td>13</td>
<td>1274.87</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>1318.42</td>
<td>15</td>
<td>1426.03</td>
<td>2100</td>
<td>15</td>
<td>1526.87</td>
<td>2300</td>
<td>15</td>
<td>1627.71</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>1928.07</td>
<td>10</td>
<td>2274.43</td>
<td>13444.43</td>
<td>15</td>
<td>1748.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IW</td>
<td>2613.90</td>
<td>5</td>
<td>2890.91</td>
<td>3100</td>
<td>15</td>
<td>2031.06</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>2131.89</td>
<td>15</td>
<td>2333.57</td>
<td>3700</td>
<td>15</td>
<td>2434.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>3286.37</td>
<td>15</td>
<td>3616.43</td>
<td>4700</td>
<td>15</td>
<td>3781.46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IW</td>
<td>2494.61</td>
<td>15</td>
<td>2697.19</td>
<td>4700</td>
<td>15</td>
<td>2837.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>2535.24</td>
<td>15</td>
<td>2736.91</td>
<td>4700</td>
<td>15</td>
<td>2837.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>3946.49</td>
<td>15</td>
<td>4276.55</td>
<td>5500</td>
<td>15</td>
<td>3241.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IW</td>
<td>2899.77</td>
<td>15</td>
<td>3203.63</td>
<td>5700</td>
<td>15</td>
<td>4441.58</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>2938.59</td>
<td>15</td>
<td>3102.35</td>
<td>5700</td>
<td>15</td>
<td>4441.58</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>4606.61</td>
<td>15</td>
<td>3543.61</td>
<td>6300</td>
<td>15</td>
<td>3644.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IW</td>
<td>3304.92</td>
<td>15</td>
<td>3507.5</td>
<td>6300</td>
<td>15</td>
<td>3644.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>3341.94</td>
<td>15</td>
<td>3543.61</td>
<td>6300</td>
<td>15</td>
<td>3644.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>5266.73</td>
<td>15</td>
<td>3596.79</td>
<td>6300</td>
<td>15</td>
<td>3644.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IW</td>
<td>3710.07</td>
<td>15</td>
<td>3596.79</td>
<td>6300</td>
<td>15</td>
<td>3644.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>3745.28</td>
<td>15</td>
<td>3912.54</td>
<td>6900</td>
<td>15</td>
<td>4013.77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>5926.85</td>
<td>15</td>
<td>3946.96</td>
<td>6900</td>
<td>15</td>
<td>4013.77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IW</td>
<td>4115.00</td>
<td>15</td>
<td>6256.91</td>
<td>7100</td>
<td>15</td>
<td>4013.77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>4148.63</td>
<td>15</td>
<td>4317.47</td>
<td>7100</td>
<td>15</td>
<td>4013.77</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>6586.97</td>
<td>15</td>
<td>4249.47</td>
<td>7900</td>
<td>15</td>
<td>4418.70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IW</td>
<td>4519.93</td>
<td>15</td>
<td>6917.03</td>
<td>7900</td>
<td>15</td>
<td>4418.70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>4551.98</td>
<td>15</td>
<td>4621.17</td>
<td>8700</td>
<td>15</td>
<td>4823.63</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>7247.09</td>
<td>15</td>
<td>4722.4</td>
<td>8700</td>
<td>15</td>
<td>4823.63</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IW</td>
<td>4924.86</td>
<td>15</td>
<td>5157.00</td>
<td>9500</td>
<td>15</td>
<td>5228.56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>4955.33</td>
<td>15</td>
<td>5026.10</td>
<td>9500</td>
<td>15</td>
<td>5228.56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>7907.2</td>
<td>15</td>
<td>8237.26</td>
<td>9500</td>
<td>15</td>
<td>5228.56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IW</td>
<td>5329.79</td>
<td>15</td>
<td>5431.03</td>
<td>9500</td>
<td>15</td>
<td>5228.56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>5358.67</td>
<td>15</td>
<td>5459.51</td>
<td>9500</td>
<td>15</td>
<td>5228.56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>8567.32</td>
<td>15</td>
<td>8732.35</td>
<td>9500</td>
<td>15</td>
<td>5228.56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From Fig. 2(b) and Fig. 3, we can come to the following conclusions: (1) For divisible load scheduling problems on heterogeneous distributed systems in blocking mode of communication, the proposed algorithm is much more effective than other compared algorithms because processors involved in computation after scheduling by WX-GA is fewer than that by IG and meanwhile the processing time obtained by WX-GA is smaller than that by other compared algorithms. (2) Distribution sequence plays a significant influence on the processing time, since algorithms with different distribution schemes lead to distinct experimental results. In the second experiment, the workload size ranges from $1 \times 10^4$ to $2 \times 10^5$. The processing time and used machine numbers for different workload by Algorithm WX-GA, IW and IG are recorded in Table IV.

It can be seen from Table IV that the processing time used by the proposed algorithm is less than or at least equal to that used by other two compared algorithms for all test cases. Fig. 4(a) shows the processing time difference between algorithm IW and WX-GA, while Fig. 4(b) shows the time difference between IG and WX-GA.
From Fig. 4(a), one can see that the larger the workload, the greater the difference of the processing time between IW and WX-GA. On the contrary, Fig. 4(b) shows that the larger the workload, the smaller the difference of the processing time between IG and WX-GA. Especially, when workload is large enough, the processing time obtained by WX-GA and IG are the same, because all processors are needed in computation and the two scheduling algorithms get the same distribution sequence, which follows the decreasing order of the communication speeds. The experimental results are consistent with the conclusion given by Theorem 3.3 in Section 3. That is to say, the experimental results reconfirmed the validity of the conclusion.

**Table IV.**

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Size</th>
<th>No.</th>
<th>Time</th>
<th>Size</th>
<th>No.</th>
<th>Time</th>
<th>Size</th>
<th>No.</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>$1\times10^4$</td>
<td>15</td>
<td>5481.64</td>
<td>$2\times10^4$</td>
<td>15</td>
<td>10553.0</td>
<td>$3\times10^4$</td>
<td>15</td>
<td>15579.5</td>
</tr>
<tr>
<td>IG</td>
<td></td>
<td>15</td>
<td>5509.93</td>
<td></td>
<td>15</td>
<td>10551.8</td>
<td></td>
<td>15</td>
<td>15594.6</td>
</tr>
<tr>
<td>IW</td>
<td></td>
<td>15</td>
<td>8814.87</td>
<td></td>
<td>15</td>
<td>17066.4</td>
<td></td>
<td>15</td>
<td>25317.8</td>
</tr>
<tr>
<td>GA</td>
<td>$4\times10^4$</td>
<td>15</td>
<td>20622.8</td>
<td>$5\times10^4$</td>
<td>15</td>
<td>25665.7</td>
<td>$6\times10^4$</td>
<td>15</td>
<td>30708.6</td>
</tr>
<tr>
<td>IG</td>
<td></td>
<td>15</td>
<td>20635.5</td>
<td></td>
<td>15</td>
<td>25677.3</td>
<td></td>
<td>15</td>
<td>30719.2</td>
</tr>
<tr>
<td>IW</td>
<td></td>
<td>15</td>
<td>33569.3</td>
<td></td>
<td>15</td>
<td>41820.8</td>
<td></td>
<td>15</td>
<td>50072.3</td>
</tr>
<tr>
<td>GA</td>
<td>$7\times10^4$</td>
<td>15</td>
<td>35751.5</td>
<td>$8\times10^4$</td>
<td>15</td>
<td>40794.5</td>
<td>$9\times10^4$</td>
<td>15</td>
<td>45837.4</td>
</tr>
<tr>
<td>IG</td>
<td></td>
<td>15</td>
<td>35761</td>
<td></td>
<td>15</td>
<td>40802.9</td>
<td></td>
<td>15</td>
<td>45844.7</td>
</tr>
<tr>
<td>IW</td>
<td></td>
<td>15</td>
<td>58323.8</td>
<td></td>
<td>15</td>
<td>66575.3</td>
<td></td>
<td>15</td>
<td>72429.2</td>
</tr>
<tr>
<td>GA</td>
<td>$1\times10^5$</td>
<td>15</td>
<td>50880.3</td>
<td>$1.1\times10^5$</td>
<td>15</td>
<td>55923.2</td>
<td>$1.2\times10^5$</td>
<td>15</td>
<td>60966.2</td>
</tr>
<tr>
<td>IG</td>
<td></td>
<td>15</td>
<td>50886.6</td>
<td></td>
<td>15</td>
<td>55928.4</td>
<td></td>
<td>15</td>
<td>60970.2</td>
</tr>
<tr>
<td>IW</td>
<td></td>
<td>15</td>
<td>83078.3</td>
<td></td>
<td>15</td>
<td>91329.7</td>
<td></td>
<td>15</td>
<td>99581.2</td>
</tr>
<tr>
<td>GA</td>
<td>$1.3\times10^5$</td>
<td>15</td>
<td>66009.1</td>
<td>$1.4\times10^5$</td>
<td>15</td>
<td>71052.0</td>
<td>$1.5\times10^5$</td>
<td>15</td>
<td>76094.9</td>
</tr>
<tr>
<td>IG</td>
<td></td>
<td>15</td>
<td>66012.1</td>
<td></td>
<td>15</td>
<td>71053.9</td>
<td></td>
<td>15</td>
<td>76095.8</td>
</tr>
<tr>
<td>IW</td>
<td></td>
<td>15</td>
<td>107833</td>
<td></td>
<td>15</td>
<td>116084</td>
<td></td>
<td>15</td>
<td>124336</td>
</tr>
<tr>
<td>GA</td>
<td>$1.6\times10^5$</td>
<td>15</td>
<td>81137.6</td>
<td>$1.7\times10^5$</td>
<td>15</td>
<td>86179.5</td>
<td>$1.8\times10^5$</td>
<td>15</td>
<td>91221.3</td>
</tr>
<tr>
<td>IG</td>
<td></td>
<td>15</td>
<td>81137.6</td>
<td></td>
<td>15</td>
<td>86179.5</td>
<td></td>
<td>15</td>
<td>91221.3</td>
</tr>
<tr>
<td>IW</td>
<td></td>
<td>15</td>
<td>132587</td>
<td></td>
<td>15</td>
<td>140839</td>
<td></td>
<td>15</td>
<td>140909</td>
</tr>
<tr>
<td>GA</td>
<td>$1.9\times10^5$</td>
<td>15</td>
<td>96263.2</td>
<td>$2\times10^5$</td>
<td>15</td>
<td>101305</td>
<td></td>
<td>15</td>
<td>101305</td>
</tr>
<tr>
<td>IG</td>
<td></td>
<td>15</td>
<td>96263.2</td>
<td></td>
<td>15</td>
<td>101305</td>
<td></td>
<td>15</td>
<td>101305</td>
</tr>
<tr>
<td>IW</td>
<td></td>
<td>15</td>
<td>157342</td>
<td></td>
<td>15</td>
<td>165593</td>
<td></td>
<td>15</td>
<td>165593</td>
</tr>
</tbody>
</table>
VI. CONCLUSIONS

The goal of this paper was to find an optimal scheduling for divisible load on heterogeneous distributed systems in blocking mode of communication. The goal was successfully achieved by designing a novel genetic algorithm WX-GA. In order to examine the performance of the proposed algorithm, a set of experiments were carried out. From the experimental results, the following conclusions can be drawn. First, distribution sequence plays a significant role in processing time. Second, the proposed algorithm greatly decreases the processing time. Third, when workload is large enough, in order to achieve minimum processing time, the optimal distribution sequence should follow the decreasing order of the communication speeds.

ACKNOWLEDGEMENT

This work was supported by National Natural Science Foundation of China (No.61272119).

REFERENCES


Xiaoli Wang and is currently a Ph.D. candidate at Xidian University, and major in computer science and technology.

She was born in ShanDong Province. She received her BS degree in software engineering from Xidian University in 2004. She is a member of the ACM. Her research interests include parallel and distributed systems, grid computing and cloud computing.

Yuping Wang is a professor with the School of Computer Science and Technology, Xidian University, Xi'an, China. He got Ph.D from the Department of Mathematics, Xi'an Jiaotong University, China in 1993. He is the Senior member of IEEE, and visited Chinese University of Hong Kong, City University of Hong Kong, and Hong Kong Baptist University as a research fellow many times from 1997 to 2010. He has authored and co-authored over 100 research papers in journals and conferences. His current research interests include evolutionary computation, non-bilevel programming, global optimization, and multi-objective programming.

Kun Meng is currently a postgraduate student at Xidian University, and major in computer science and technology, Xi'an, China. His current research interests include task scheduling and evolutionary algorithms.