Article

Unavailable Time Aware Scheduling of Hybrid Task on Heterogeneous Distributed System

Hejun Xuan [1,*], Shiwei Wei [2], Yanling Li[1], Ran Li[1] and Wuning Tong[3]

[1] School of Computer and Information Technology, Xinyang Normal University, Xinyang 464000, China;liyanling@xynu.edu.cn(Y.L.);liran@xynu.edu.cn(R.L.);
[2] School of Computer and Technology, Guilin University of Aerospace Technology, Guilin 541000 China; swwei_2001@163.com(S.W.);
[3] Shaanxi University of Chinese Medicine, Xianyang 712000, China; successful77@163.com(W.T.);
* Correspondence: xuanhejun0896@126.com; Tel.: +86-187-4043-7786

Abstract: The resource allocation for tasks in heterogeneous distributed system is a well known NP-hard problem. For the sake of making the makespan is minimized, it is hard to distribute the tasks to proper processors. The problem is even more complex and challenging when the processors have unavailable time and the tasks type are various. This paper investigates a resource allocation problem for hybrid tasks comprising both divisible and bag-of-tasks(BoT) in heterogeneous distributed system when the processors has unavailable time. First, the mathematical model, which minimizes the makespan of the hybrid tasks when the processors have unavailable time, is established. Second, we propose a scheduling algorithm referred to as bag-of-tasks allocate-pull and divisible task allocation (BoTAPDTA) algorithm for handling hybrid tasks on heterogeneous distributed systems. In addition, to solving the optimization model efficiently, a generic algorithm(GA) is proposed. For the sake of reducing the search space and solving the optimization model effectively, a two step scheduling algorithm(TSGA), which first allocate bag-of-tasks(BoT) using generic algorithm and then assign divisible task to processors like BoTAPDTA, is designed. Finally, numerical simulation experiments are conducted, and experimental results indicate the effectiveness of the proposed model and algorithm.

Keywords: Distributed computing; Task scheduling; Hybrid Tasks; Generic algorithm;

1. Introduction

Heterogeneous distributed system has emerged as commonly systems for handling large scale scientific and commercial applications in various fields, such as image processing, signal processing, pattern matching in text, and many scientific computation problems [1,2]. For the sake of improving the performance of the system, many task scheduling algorithms for heterogeneous or homogeneous distributed system have been proposed in the past decades[3,4]. Wang[3] proposed a multi-objective bi-level programming model for energy and locality aware multi-job scheduling in heterogeneous system. Literature [4] proposed two scheduling algorithms to schedule a BoT (bag-of-tasks, BoT) on heterogeneous system so as to minimize the makespan and the energy consumption. In Literature [5], the reliability cost, which is defined as the product of failure rate of processors and task processing time, is incorporated into scheduling algorithm for the tasks with precedence constraints on heterogeneous system. Lee and Zomaya classified the tasks into computation-intensive and data-intensive BoT task in literature[6] and presented two task scheduling algorithms in Grid computing system respectively. Anglano et al.[7] evaluated the performance of five knowledge-free task scheduling algorithms for scheduling multiple BoT in a desktop Grids computing system. In addition, the performance of several BoT scheduling solutions in large-scale distributed systems also have been studied in literature [8]. For BoT, there are some other studies that aim to maximize throughput by establishing linear programs or nonlinear programs[9,10], and the works focused on steady-state optimization problems and concentrated on numerous bag-of-tasks including independent and similar tasks. Legrand

 doi:10.20944/preprints201809.0331.v1

et al.proposed a centralized and decentralized scheduling algorithm in literature[9]. To schedule concurrent bag-of-tasks, the online and off-line scheduling algorithms are presented by Benoit et al.[10]. In literature[11], a decentralized scheduling algorithm, which minimizes the maximum stretch among user-submitted tasks, is designed. Yang Y et al.[12] take the constraints of time, cost, and security into consideration, a scheduling algorithm for data-intensive tasks is designed. Literature [13] investigated both two problems: optimizing the makespan of the tasks under the constraints of energy, or minimizing energy consumption subject to makespan. However, this paper studied the static resource allocation to optimize makespan and energy robust stochastic for bag-of-tasks(BoT) on a heterogeneous computing system. A multi-objective optimization model, which minimizes makespan and resource cost, is established in literature[14]. To solve the optimization model, a scheduling algorithm based on the ordinal optimization method is designed. However, the scheduling algorithm is inefficient when the task number or processing node number is large.

In scheduling theory, the fundamental assumption is that all processors which take participate in processing tasks are always available for processing tasks[15]. However, it might be unreasonable. If some certain maintenance requirements, breakdowns, or other constraints exits, they will make the processers unavailable for executing the tasks. In literature [16], availability is defined. For a processor, availability is defined as the ratio of the total available time to the total time during a given interval. In previous work, Some work has investigated task scheduling algorithm with processor availability constraints[17]. Adiri et al[18]. investigates the scheduling problem with availability constraints in a single machine system. For minimizing maximum lateness of the n jobs, literature [19] studied the problem on homogeneous machines under machine availability and eligibility constraints. A branch-and-bound method is proposed in literature[20] to solve the single-machine scheduling problem with machine availability constraints. For minimizing the total flow time, literature [21] investigated the non-permutation flow shop scheduling issue with the learning effects and machine availability constraints. To minimize the makespan, the two-machine permutation flowshop scheduling problem with an availability constraint is investigate in literature [22]. However, the basic assumption of this work is that the availability constraint imposed only on the first machine. Literature[23] developed a Hybrid Heuristic-Ant Colony Optimization (H2ACO) for multiclass tasks on heterogeneous distributed systems with availability constraint. H2ACO algorithm can make a good trade-off between availability and makespans of the tasks. An availability-aware scheduling model is investigated in literature [24], and an optimization algorithm to increase the availability and to minimize the makespan of tasks in heterogeneous systems is proposed. Literature [25] proposed a quantum-behaved particle swarm optimization algorithm to optimize the availability-aware task Scheduling on heterogeneous systems. A novel distributed availability-aware adaptive rate-allocation scheduling algorithm for multimedia tasks in heterogeneous wireless networks is proposed in literature [26].

Divisible task has been studied extensively in the last several decades, resulting in a cohesive theory called Divisible Load Theory (DLT)[27]. The main objective of the divisible load scheduling problem in distributed computing systems is minimizing the processing time, also called the makespan[28–30]. In our work, we investigates a resource allocation problem for hybrid tasks comprising both divisible and bag-of-tasks in heterogeneous distributed system when the processors has unavailable time. The major contributions of this study are summarized as follows:

- To minimize the makespan of the hybrid tasks, a mathematical optimization model, which takes unavailable time constraint of processors into consideration, is established.
- We propose a algorithm referred to as Bag-of-Tasks Allocate-Pull and Divisible Task Allocation (BoTAPDTA) algorithm for the hybrid tasks scheduling problem.
- To solving the optimization model effectively, a generic algorithm(GA) is proposed.
- For the sake of reducing the search space and solving the optimization model effectively, a two step scheduling algorithm(TSGA), which first allocate bag-of-tasks using generic algorithm and then assign divisible task to processors like BoTAPDTA, is designed.
- An analysis on the effectiveness of our proposed algorithm on two different size systems that vary in both number of processors and tasks.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

The rest of this paper is organized as follows. Section 2 gives the system and task description, and the mathematical model is established. The scheduling algorithm referred to as bag-of-tasks allocate-pull allocate-pull scheduling (BoTAP) algorithm is described in section 3. Section 4 proposed a generic algorithm to solving the optimization model effectively. The two step hybrid tasks scheduling algorithm is explained in section 5. Section 6 presents simulation results to evaluate the algorithms. The paper is concluded with a summary and a future work in Section 7.

2. Problem Formulation

2.1. System and Task Description

In our work, the heterogeneous distributed system has $N+1$ processors, which includes a master processor and N slave processors. P_0 denotes the master processor, and the slave processors denoted by $\{P_1, P_2, \cdots, P_N\}$. Each slave processor $P_i(i = 1, 2, \cdots, N)$ is associated with a speed index w_i, which is the time taken to process a unit workload on processor P_i. Slave processor is the most basic processing unit in our research. Since some reasons, such as shutdown or maintenance requirements, slave processors have some unavailable time. $[a_i^j \;\; b_i^j](i = 1, 2, \cdots, N; j = 1, 2, \cdots, n_i)$ denote the $j^{th}(j = 1, 2, \cdots, n_i)$ unavailable segment of processor $P_i(i = 1, 2, \cdots, N)$, and n_i is the number of unavailable segment for processor $P_i(i = 1, 2, \cdots, N)$. For the convenience, the $j^{th}(j = 1, 2, \cdots, n_i)$ available segment of processor $P_i(i = 1, 2, \cdots, N)$ denoted by $[c_i^j \;\; d_i^j](i = 1, 2, \cdots, N; j = 1, 2, \cdots, m_i)$, and m_i is the number of available segment for processor $P_i(i = 1, 2, \cdots, N)$. To understand easily, the system model is shown in Fig.1

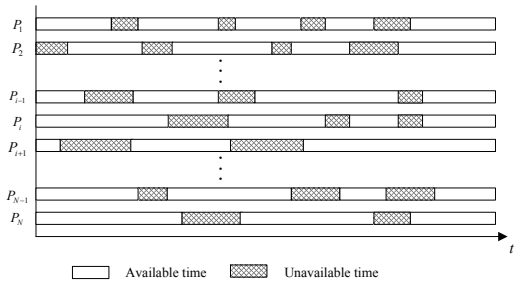


Figure 1. Available and unavailable time of the processors in the heterogeneous distributed system.

In our study, the hybrid tasks comprising both divisible and bag-of-tasks is investigated. Bag-of-tasks(BoT) are a representatively type of tasks including of numerous independent tasks and can be processed parallelly without communication. Divisible task(DT) can be partitioned into a large number of load fractions and can be processed independently on the processors in parallel since there are no precedence relationships among these tasks. That is to say, all the tasks our work investigated are independent. The workload includes $N_\tau + 1$ independent tasks and the $i^{th}(0 \leq i \leq N_\tau)$ task is denoted by τ_i, where task τ_0 is a divisible task, and $\tau_i(1 \leq i \leq N_\tau)$ are bag-of-tasks. Following the previous studies[31,32], we assume that the size of load $\tau_i^\sigma(0 \leq i \leq N_\tau)$ is known after a task arrives according to the prediction mechanisms such as code profiling and statistical prediction. As the previous work[33], these bag-of-tasks have different computing requirements, and we assume that each task can only be processed by some specific processors. Ω_i is a set of the processor's that τ_i can allocated to. In our work, we assume that the divisible τ_0 can be processed on all processors in the heterogeneous distributed system. Similarly, we assume that the bag-of-tasks are computation-intensive as prior works[17,34]. That is to say, the time consuming of input data transmission does not influence much the completion time and hence it can be negligible. In our work, the transmission of divisible task is negligible too.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

2.2. Mathematical Modeling

The task scheduling problem investigated in this paper is to schedule all the $N_\tau + 1$ tasks to the N processors in the heterogeneous distributed system with the purpose of minimizing the makespan of the tasks. Then, we will give the mathematical modeling of the optimization problem.

2.2.1. Objective Function

Generally speaking, makespan is the latest finish processing time of the processors. If T_i denote the finish processing time of processor P_i, the makespan T of the tasks can be denoted by Eq.(1).

$$T = \max_{1 \le i \le N} \{T_i\} \tag{1}$$

In our work, the purpose is minimize the makespan of the hybrid tasks in the heterogeneous distributed system with unavailable time considered. So, the objective function can be described as Eq.(2).

$$\min T = \min \left\{ \max_{1 \le i \le N} \{T_i\} \right\} \tag{2}$$

As shown in Eq.(1), we can see that the finish processing time of each processor should be calculated. For a specific processor P_i, its processing time diagram is shown in Fig.2. In the processing

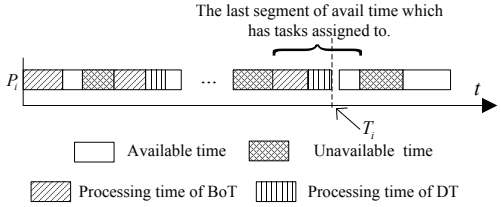


Figure 2. The processing time of processor P_i.

time diagram of processor P_i, it shows that the finish time of processor P_i determined by the last segment of available time which has tasks assigned to. If $l (1 \le l \le m_i)$ and δ_i^l denote the last segment of the tasks assigned to processor P_i and the set of BoT tasks assigned to segment lst respectively. α_i^l is the ratio of the fraction size assigned to the l segment on processor P_i to the workload τ_0^σ. So, the finish processing time of processor P_i can be calculated by Eq.(3).

$$T_i = c_i^l + w_i \left(\alpha_i^l \tau_0^\sigma + \sum_{q \in \delta_i^l} \tau_q^\sigma \right) \tag{3}$$

Then, we can rewrite Eq.(2) as Eq.(4)

$$\begin{aligned} \min T &= \min \left\{ \max_{1 \le i \le N} \{T_i\} \right\} \\ &= \min \left\{ \max_{1 \le i \le N} \left\{ c_i^l + w_i \left(\alpha_i^l \tau_0^\sigma + \sum_{q \in \delta_i^l} \tau_q^\sigma \right) \right\} \right\} \end{aligned} \tag{4}$$

2.2.2. constraint conditions

(a)All the BoT must be assigned to the processors that can satisfy BoT requirement: $\Theta = (\theta_{ij}^k)_{N_\tau \times N \times m_j}$ is a binary matrix, where $\theta_{ij}^k = 1$ if only and if task τ_i is assigned to the $k^{th} (1 \le k \le m_j)$ segment available time of processor $P_j (1 \le j \le N)$, otherwise $\theta_{ij}^k = 0$. So, we can obtain a conclusion:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

If $\theta_{ij}^{k} = 1$, the processor P_j of task $\tau_i (1 \leq i \leq N_\tau)$ assigned to must in the set Ω_i. That is to say, Eq.(5) is satisfied.

$$\theta_{ij}^{k} = 1 \Rightarrow P_j \in \Omega_i \tag{5}$$

(b)All the BoT must be allocated to assigned to the available time segments on processors: A crucial principle of bag-of-tasks scheduling problem is that all the tasks should be allocated to the suitable processors. According the definition of the binary matrix Θ, we can obtain Eq.(6) when all the tasks are allocated.

$$\sum_{i=1}^{N_\tau} \sum_{j=1}^{N} \sum_{k=1}^{m_j} \theta_{ij}^{k} = N_\tau \tag{6}$$

(c)All the workload of divisible task must be assigned to suitable processors: α_i^j denotes the ratio of the fraction size assigned to $j^{th} (1 \leq j \leq m_i)$ of processor $P_i (1 \leq i \leq N)$ to the entire workload of the divisible task τ_0. If the divisible task τ_0 is assigned Completely, Eq.(7) can be obtained.

$$\sum_{i=1}^{N} \sum_{j=1}^{m_i} \alpha_i^j = 1 \tag{7}$$

(d)The execution time of the tasks assigned to a available segment should not be greater than the available time of the segment: Since each processor $P_j (1 \leq j \leq N)$ has unavailable time, the processing time of tasks assigned to k^{th} segment should not be greater than the available time. The workload of assigned to $k^{th} (1 \leq k \leq m_j)$ segment available time on processor $P_j (1 \leq j \leq N)$ denoted by σ_j^k, the Eq.(8) should be satisfied as shown below.

$$w_j \sigma_j^k \leq d_j^k - c_j^k \tag{8}$$

There are BoT and divisible task fraction allocate to $k^{th} (1 \leq k \leq m_j)$ available time segment of processor $P_j (1 \leq j \leq N)$. According to the definition of $\theta_{ij}^k (1 \leq i \leq N_\tau)$ and α_j^k, we can can calculate the workload σ_j^k of assigned to k^{th} segment available time on processor P_j by Eq.(9).

$$\sigma_j^k = \alpha_j^k \tau_0^\sigma + \sum_{i=1}^{N_\tau} \theta_{ij}^k \tau_i^\sigma \tag{9}$$

Then, we can rewrite Eq.(8) as Eq.(10)

$$w_j \left(\alpha_j^k \tau_0^\sigma + \sum_{i=1}^{N_\tau} \theta_{ij}^k \tau_i^\sigma \right) \leq d_j^k - c_j^k \tag{10}$$

2.2.3. Mathematical Modeling

The task scheduling problem investigated in this paper is to schedule all the $N_\tau + 1$ tasks to the N processors in the heterogeneous distributed system with the purpose of minimizing the makespan of the tasks. In section 2.2.1 and section 2.2.2. we give the mathematical formulation of the objective

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

function and constraints respectively. Then, the mathematical optimization model with constraints are presented in Eq.(11).

$$\begin{cases} \min T = \\ \quad \min \left\{ \max\limits_{1\le j\le N} \left\{ c_j^l + w_i \left(\alpha_j^l \tau_0^\sigma + \sum\limits_{q\in\delta_j^l} \tau_q^\sigma \right) \right\} \right\} \\ s.t. \\ \quad (a)\theta_{ij}^k = 1 \Rightarrow P_j \in \Omega_i; \\ \quad (b) \sum\limits_{i=1}^{N_\tau} \sum\limits_{j=1}^{N} \sum\limits_{k=1}^{m_j} \theta_{ij}^k = N_\tau; \\ \quad (c) \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{m_i} \alpha_i^j = 1; \\ \quad (d) w_j \left(\alpha_j^k \tau_0^\sigma + \sum\limits_{i=1}^{N_\tau} \theta_{ij}^k \tau_i^\sigma \right) \le d_j^k - c_j^k; \\ \quad (e) 1 \le i \le N_\tau, 1 \le j \le N, 1 \le k \le m_j; \end{cases} \tag{11}$$

In this optimization model, constraints (a)-(d) has description in section 2.2.2, and constraints (e) gives the scope of parameters i, j, k. To solve this global optimization model, an algorithm referred to as bag-of-tasks allocate-pull and divisible task allocation (BoTAPDTA) algorithm and generic algorithm with a local research strategy are proposed. The algorithm of BoTAPDTA will be described in section 3. The proposed generic algorithm and two step scheduling algorithm(TSGA) will be given in section 4 and section 5 respectively.

3. Proposed BoTAPDTA Algorithm

In our work, hybrid tasks comprising bag-of-tasks(BoT) and divisible task scheduling problem is investigated. In addition, unavailable time is taken into account. Since divisible can be divided into arbitrary fractions and proposed on any processor in heterogeneous system, it has a great flexibility. We first allocate BoT to suitable processor, and then the divisible task assigned to the time slot before T^{BoT} and available time after T^{BoT} as shown in Fig.3.

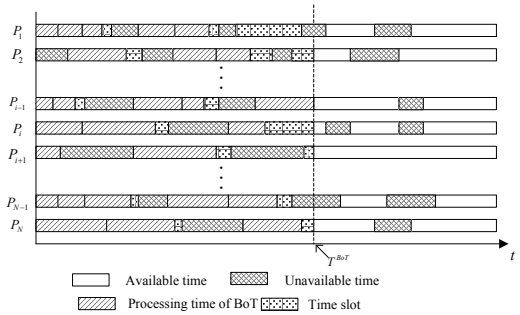


Figure 3. The processing time of processor P_i.

Based on analyzing, the scheduling algorithm can be divide into two procedures: (a)Allocation the BoT to suitable processors; (2)Allocation the divisible task to suitable time slot and available time segment. For the sake of understanding the algorithm macroscopically, the framework of the scheduling algorithm is presented before presenting the detailed steps of the algorithm. The bag-of-tasks allocate-pull and divisible task allocation(BoTAPDTA) algorithm is shown in Algorithm 1. Step 1 is sort all the tasks $\tau_i (i = 1, \cdots, N_\tau)$ in an descending order according to the workload τ_i^σ of tasks; Step 3 to step 8 is allocate the tasks to available time segments on processors. For minimizing the makespan, step 10 to step 25 is pull the tasks to available time segment before the segment current them allocated to. Step 28 to step 33 is allocate the divisible task to the processors.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

Algorithm 1: The algorithm framework of BoTAP

Input: Tasks $\tau_i(i = 1, \cdots, N_\tau)$, speed index w_i and available time $[c_i^j, d_i^j](j = 1, \cdots, m_i)$ of processor $P_i(i = 1, \cdots, N)$;
Output: a schedule scheme;
Put all the tasks into a task queue TQ, and sort them in an descending order according to the workload τ_i^σ of tasks $\tau_i(i = 1, \cdots, N_\tau)$;
Allocation:
while *TQ is not empty* **do**
    Take out the first task in TQ, and denote it as τ_{head};
    Select a processor P_{head} in Ω_{head} to make the current finish processing time is minimum;
    Select a available time segment $[c_{head}^s, d_{head}^s]$ on processor P_{head} and assign task τ_{head} to it.
    Update available time, $c_{head}^s = c_{head}^s + \tau_{head}^\sigma w_{head}$;
end
Pull:
for $i = 1$ *to* N **do**
    while $l > 1$ **do**
        %l is the last segment of available time which has tasks allocated to.
        Put all the tasks in l^{th} segment into a task queue $subTQ_l$, and sort them in an descending order according to the workload;
        while *$subTQ_i$ is not empty* **do**
            Take out the first task and denote it as $\tau_{subhead}, flag = 0, k = 1$;
            while $k < l$ & $flag = 0$ **do**
                if $\tau_{subhead}^\sigma w_i \le d_i^k - c_i^k$ **then**
                    Update available time, $c_i^k = c_i^k + \tau_{subhead}^\sigma w_i$;
                    $flag = 1$;
                end
                $k = k + 1$;
            end
        end
        $l = l - 1$;
    end
end
Allocation DT:
if $\sum_{j=1}^{N} w_j \left(\sum_{k=1}^{\mu_j} \phi_j^k \right) \ge \tau_0^\sigma$ **then**
    %ϕ_j^k and μ_j are the k^{th} time slot and number of time slot before T^{BoT} on processor P_j;
    assign the divisible task on the time slot of the processors;
else
    assign the divisible task on the time slot and the available time segment after T^{BoT};
end

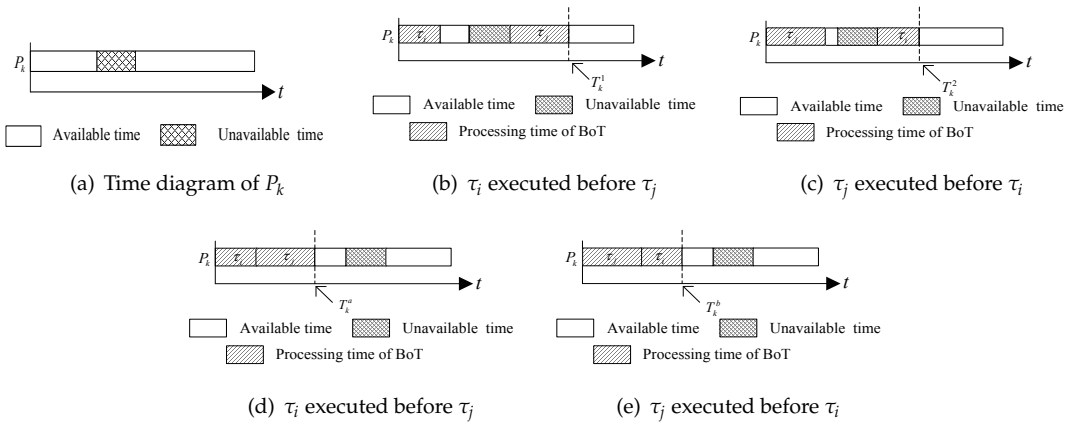
Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

3.1. Descending Order

In the algorithm of BoTAP, we first sort the tasks in descending order according to the workload of tasks. We will give the reason that we choose descending order as follow:

Case 1: As shown in Fig.4, suppose two tasks τ_i and τ_j are all allocated to processor P_k, and the workload of tasks τ_i and τ_j satisfy $\tau_i^\sigma < \tau_j^\sigma$. In addition, tasks τ_i and τ_j can executed in the first available time segment on processor P_k respectively, but τ_i and τ_j can not executed in the first available time segment simultaneous. As shown in Fig.4(b), if τ_i executed before τ_j, we should allocate τ_i to the first available time segment and allocate the τ_j to the second segment. So, the makespan of the two tasks is $T_k^1 = c_k^2 + \tau_j^\sigma w_k$. However, if τ_j executed before τ_i, the makespan of the two tasks is $T_k^2 = c_k^2 + \tau_i^\sigma w_k$. We have $\tau_i^\sigma < \tau_j^\sigma$, so $T_k^1 > T_k^2$ is obtained.

Case 2: As shown in Fig.4(d) and Fig.4(e), tasks τ_i and τ_j can executed in the first available time segment on processor P_k simultaneous. If τ_i executed before τ_j as shown in Fig.4(d), the makespan of the two tasks is $T_k^a = c_k^1 + (\tau_i^\sigma + \tau_j^\sigma)w_k$. Similarly, the makespan of the two tasks is $T_k^b = c_k^1 + (\tau_i^\sigma + \tau_j^\sigma)w_k$ when τ_j executed before τ_i as shown in Fig.4(e). So, we can obtain $T_k^a = T_k^b$. From the above, we can know that allocation order will effect the makespan of the tasks. If larger workload task allocated first, the makespan of tasks will equal to or shorter than that obtained by smaller workload task allocated first.



(a) Time diagram of P_k (b) τ_i executed before τ_j (c) τ_j executed before τ_i

(d) τ_i executed before τ_j (e) τ_j executed before τ_i

Figure 4. Influence of execute order on Makespan.

3.2. Processor Selection in Allocation

When the task queue TQ is not empty, the first task is taken out and allocated to the processor. Since the objective is minimize the makespan of the tasks, so we must allocate the task on a processor that can be make the makespan is minimum. In our work, Eq.(12) and Eq.(13) is used to determine the processor that task τ_i should be allocated to.

$$\Phi_i = \left\{ P_j | P_j \in \Omega_i, \exists 1 \le k \le m_j \Rightarrow \tau_i^\sigma w_j \le d_j^k - c_j^k \right\} \tag{12}$$

$$np_{proper} = \arg\min_{np} \left\{ \max_{np \in \Phi_i} \{T_{np}\} \right\} \tag{13}$$

where P_{np} is the processor in processors set Φ_i that the task τ_i can be allocated to, and np_{proper} is the proper processor determined. Eq.(12) is to find the set Φ_i that the task τ_i can be allocated to in Ω_i. The processors in Φ_i should satisfy two conditions: (1) the processors should in the set Ω_i. (2) At least a available time segment, which can execute the task τ_i punctually, exists.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

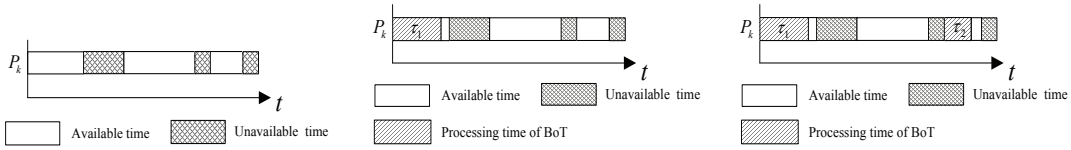
3.3. Segment Selection in Allocation

After a proper processor determined, we should allocate task τ_i to a optimal available time segment on processor P_{np}. A excellent strategy that allocate task to a available time segment will help to minimize makespan of the tasks. we use Eq.(14) and Eq.(15) to determine which segment should task τ_i allocate to.

$$NS = \left\{ ns | 1 \leq ns \leq m_{np}, d_{np}^{ns} - c_{np}^{ns} \geq \tau_i^{\sigma} w_{np} \right\} \tag{14}$$

$$ns_{proper} = \arg\min_{ns} \left\{ \left(d_{np}^{ns} - c_{np}^{ns} \right) | ns \in NS \right\} \tag{15}$$

Eq.(14) is used to find some available time segments that can complete task τ_i in time on processor P_{np}. For the sake of decreasing time debris which can not complete any task in time, the shortest time segment in NS is selected. There are two tasks τ_1 and τ_2 allocated on processor P_k, and $\tau_1^{\sigma} > \tau_2^{\sigma}$. τ_1 can completed in segment 1 and 2, and τ_1 can completed in segment 2 and 3. Because the tasks in task queue TQ are sorted in descending order according workload, τ_1 is allocated first before τ_2. For τ_1, since $d_k^1 - c_k^1 < d_k^2 - c_k^2$, so τ_1 is allocate to segment 1 according to Eq.(15) as shown in Fig.5(b). Though segment 2 can complete τ_2 in time and segment 2 is before segment 3, we can see that τ_2 is allocated to segment 3 from Fig.5(c). This strategy can help to decrease time debris and increase the utilization of the available time segment.

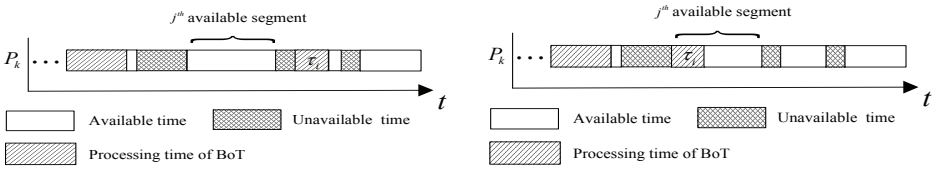


(a) Time diagram of P_k (b) Allocate τ_1 to segment 1 (c) Allocate τ_2 to segment 3

Figure 5. Segment determined.

3.4. The Strategy of Pull

After the process of allocate, all the tasks are allocated to the available time segments on processors. However, some available time segments are exit because the strategy that described in section 3.3. As shown in Fig.6(a), τ_i is allocated to $(j+1)^{th}$ available time segment processor P_k, so the processing finish time of P_k is $T_k = c_k^{j+1} + \tau_i^{\sigma} w_k$. Since the j^{th} available time segment can complete task τ_i in time, we can pull task τ_i from $(j+1)^{th}$ available time segment to j^{th} available time segment as shown in Fig.6(b). So the processing finish time of P_k can be denoted as $T_k' = c_k^j + \tau_i^{\sigma} w_k$. $T_k' < T_k$ can be obtained intuitively. So, the strategy of pull tasks to another available time segment can decrease the processing finish time of processors. For the sake of decreasing the processing time of the processors as much as possible, we should solve following two problems:(1)which task should be pull to the objective segment? (2) which segment should be selected as the objective segment? These two issues will be tackled in section 3.4.1 and section 3.4.2.



(a) Time diagram of P_k after tasks are allocated (b) Pull τ_i to j^{th} available time segment

Figure 6. Pull tasks to another available time segment.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

3.4.1. Selection of Objective Task

In this paper, we investigate the bag-of-tasks scheduling problem, and the objective of scheduling algorithm is minimize makespan of thed tasks. The makepan of the tasks is determined by the processing finish time of all processors in the heterogeneous computing system. From Eq.(3), we can see that the processing finish time $T_i(1 \leq i \leq N)$ is depended on the tasks completed time in last available time segment l which allocate tasks on processer P_i. Suppose $SubQ_l$ is the tasks set which allocate to the last available time segment l. Eq.(16) is used to determine the task which should be pull to another available time segment.

$$nt_{pro} = \arg\min_{nt} \{\tau_i^{\sigma} | \tau_i \in SubQ_l\} \tag{16}$$

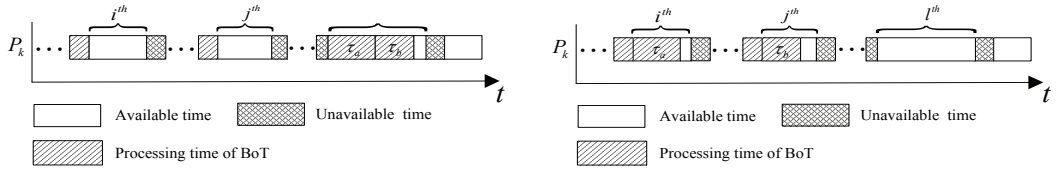
3.4.2. Selection of Objective Segment

To decrease the processing finish time of processors, the strategy, which pulls a task to another segment, is designed. In this strategy, two problems should be solved. The problem of objective task determined has been tackled in section 3.4.1. In this section, we will solve the other question. For the sake of guaranteeing the task τ_{nt} completed in time, the segment ns_{pro} is selected according to Eq.(17).

$$ns_{pro} = \arg\min_{ns} \{ns | d_k^{ns} - c_k^{ns} > \tau_{pro} w_k, 1 \leq ns \leq l\} \tag{17}$$

The strategy of pull task to another segment can decrease processing finish time as much as possible. First, the task $\tau_{nt_{pro}}$ with largest workload is selected according to Eq.(16), and a available time segment ns_{pro} is selected according to Eq.(17). If $ns_{pro} = \emptyset$, let $SubQ_l = SubQ_l \setminus \{\tau_{nt_{pro}}\}$, and then another task $\tau_{nt_{pro}}$ is selected according to Eq.(16). If $ns_{pro} \neq \emptyset$, We pull the task $\tau_{nt_{pro}}$ from segment l to the segment ns_{prop}.

A example is presented in Fig.7. Task τ_a and τ_b are allocated to the l^{th} available time segment, and $\tau_a^{\sigma} > \tau_b^{\sigma}$. First, τ_a and τ_b are put into $subQ_l$. According to Eq.(16), task τ_a is selected as the objective task. The i^{th} segment is selected as the objective segment according to Eq.(17). Then, we pull τ_a to i^{th} segment and update $c_k^i = c_k^i + \tau_a^{\sigma} w_k$. Since $ns_{pro} \neq \emptyset$, we can select task τ_b and j^{th} segment as the objective task and objective segment respectively. Then, task τ_b is pulled to j^{th} segment and update $c_k^j = c_k^j + \tau_b^{\sigma} w_k$. Let $l = l - 1$, a new round of pulling is conducted until the objective segment can not found.



(a) Time diagram of P_k after tasks are allocated.

(b) Pull τ_a and τ_b to i^{th}, j^{th} available time segment respectively.

Figure 7. An example of pulling tasks to another available time segment.

3.5. Allocation Divisible Task

After allocate the BoT to the processors, we should allocate the divisible task to the time slot and available time segment to make the makespan of the hybrid tasks minimized. We can see that scheduling of the divisible task can be divided into two situations from algorithm 1. If Eq.(18) is satisfy, the divisible task is called large divisible task, otherwise, the divisible task is called small divisible

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

task. Then we will scheduling the small or large divisible task according to different strategy which described in section 3.5.1 and section 3.5.2.

$$\sum_{j=1}^{N}\left(w_j\sum_{k=1}^{\mu_j}\phi_j^k\right)\geq\tau_0^\sigma \tag{18}$$

where ϕ_j^k and μ_j are the k^{th} time slot and number of time slot before T^{BoT} on processor P_j.

3.5.1. Small Divisible Task Scheduling

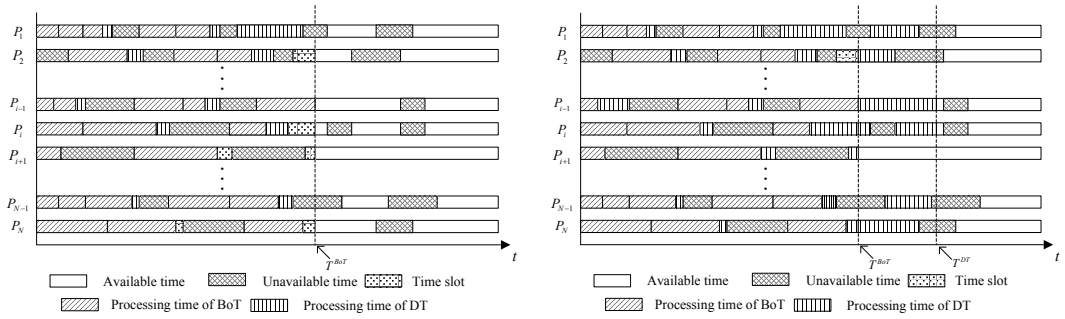
If the divisible task is regarded as a small task, it means that the divisible task τ_0 can be completed in the time slot before T^{BoT}. So, we can allocate the divisible task in the time slot as shown in Fig.8(a). In this case, we should solve two problems: (1)which processor should the divisible task allocate to? (2) which time slots in the processors should the divisible task allocate to? In our work, the method as follow is proposed to solve the two problems aforementioned. First, $w_j\left(\sum_{k=1}^{\mu_j}\phi_j^k\right)(j=1,2,\cdots,N)$ are sorted in descending order and put them into a array Γ. Then, the processors are determined by Eq.(19).

$$n_p=\arg\min_{1\leq np\leq N}\left\{\sum_{j=1}^{np}\Gamma_j\geq\tau_0^\sigma\right\} \tag{19}$$

If $\sum_{j=1}^{n_p}\Gamma_j=\tau_0^\sigma$, we will assign the divisible task onto the time slot on processors before Γ_{n_p}. Otherwise, we will assign the divisible task onto the time slot on processors before Γ_{n_p-1}, and the reminder workload $(\tau_0^\sigma-\sum_{j=1}^{n_p-1}\Gamma_j)$ will allocate to processor Γ_{n_p-1}. In this case, the makespan of the hybrid tasks is T^{BoT}.

3.5.2. Large Divisible Task Scheduling

If the divisible task is regarded as a large task, it means that the divisible task τ_0 can not be completed in the time slot before T^{BoT}. In this case, we first allocate all the time slot before T^{BoT}. Then, the reminder workload $\tau^\sigma=\tau_0^\sigma-\sum_{j=1}^{N}\left(w_j\left(\sum_{k=1}^{\mu_j}\phi_j^k\right)\right)$ will allocate to the available time after T^{BoT} as shown in Fig.8(b).



(a) Allocation small divisible task to processors.

(b) Allocation large divisible task to processors.

Figure 8. Allocation Divisible task to processors.

Following previous work[35–39], an essential condition used in the related works in DLT(Divisible Load Theory, DLT) to derive optimal solution is as follows: for the sake of obtaining an optimal processing time, it is necessary and sufficient to demand that all the processors participating in the computation must finish their computing at the same time. So, we have $T_i=T_{i+1}(1\leq i\leq N-1)$. If $s_i(1\leq i\leq N)$ denotes the unavailable time between T^{BoT} and T_i of processor P_i, the Eq.(20) is obtained.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

In this case, the makespan of the tasks is T^{DT}, and we have $T^{DT} = T_1$. So the makespan of the hybrid tasks is calculated by Eq.(20).

$$T^{DT} = T_1 = T^{BoT} + s_1 + w_1\beta_1. \tag{20}$$

The makespan of hybrid tasks can be calculated easily through Eq.(20). s_i denotes the unavailable time between T^{BoT} and T^{DT} of processor P_i, and s_i is related to T^{DT}, so it is impossible to determine s_i when T^{DT} is not determined. For the sake of searching the T^{DT}, a binary search algorithm is proposed and its pseudocode is shown in algorithm 2.

Algorithm 2: T^{DT} and workload β_i are determined

Input: $w_i, [c_i^j d_i^j](i = 1, 2, \cdots, N; j = 1, 2, \cdots, m_i)$, reminder workload τ^σ of divisible task;
Output: Makespan T^{DT} and workload β_i assigned to processor P_i;

$min\_time = 0$, $max\_time = \left(\max_{1\le i\le N}\{w_i\}\right)\tau^\sigma$;
while $max\_time - min\_time > \xi$ **do**
  % ξ is a threshold of time;
  $mean\_time = (max\_time + min\_time)/2$;
  Calculate unavailable time $s_i(i = 1, 2, \cdots, N)$ between T^{BoT} and $T^{BoT} + mean\_time$ for every P_i;
  The fractions $\beta_i = (mean\_time - s_i)/w_i$ that can be allocated to P_i is calculated;
  if $\sum_{i=1}^{N}\beta_i < \tau^\sigma$ **then**
    $min\_time = mean\_time$;
  else
    $max\_time = mean\_time$;
  end
end
$T^{DT} = T^{BoT} + (max\_time + min\_time)/2$;
Calculate unavailable time $s_i(i = 1, 2, \cdots, N)$ between T^{BoT} and $T^{BoT} + mean\_time$ for every P_i;
The fractions $\beta_i = (mean\_time - s_i)/w_i$ that can be allocated to P_i is calculated;

4. GA for Hybrid Tasks Scheduling

Task scheduling is a NP hard problem in the well-known hardest combinatorial optimization problems. GA(generic algorithm, GA), which invented by Ehrlich and Raven[40], is employed to solve the task scheduling optimization model proposed in section 2.2. GA is a efficient technique for many realistic application problems such as Control and Decision, image processing, and machine learning, etc[41,42].

4.1. Encoding and Population Initialization

Generally speaking, a suitable encoding scheme, which encodes the solutions in problem domain to a chromosome, is much more significant. A better encoding scheme will make the search easier by limiting the search space and converge to the global optimal solution rapidly. Based on what characterizes this optimization model for hybrid tasks scheduling problem, the encoding scheme of integer is adopted for bag-of-tasks. A array $C^B_{2\times N_\tau} = (c^B_{ij})_{2\times N_\tau}$ is represented as a list of $2 \times N_\tau$ elements, called chromosome. For a specific task $\tau_j(1 \le j \le N_\tau)$, we have $c^B_{1j} = a$ and $c^B_{2j} = b$ if only and if the task τ_j allocate to the b^{th} available time segment of processor P_a. In addition, a real coding scheme is employed. A array $C^D_{N\times N_g} = (c^D_{ij})_{N\times N_g}$ is chromosome for divisible task, where $N_s = \max_{1\le i\le N}\{m_i\}$. In $C^D_{N\times N_g}$, we have $c^D_{ij} = \alpha_i^j(1 \le i \le N, 1 \le j \le m_i)$ and $c^D_{ij} = 0(1 \le i \le N, m_i \le j \le N_g)$. We can obtain the initial population Pop_BoT and Pop_DT of generic allocate according to the algorithm 3.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

Algorithm 3: Encoding and population initialization

Input: N, N_τ, population size Pop_{size};
Output: Initial population Pop_BoT and Pop_DT;
for $i = 1$ *to* Pop_{size} **do**
  Initial Pop_BoT**:**
  for $j = 1$ *to* N_τ **do**
    $flag\_allocated = 0$;
    while $flag\_allocated == 0$ **do**
      $p\Omega_j$ permutation of the elements in Ω_j;
      A random integer is generated in $[1, size(\Omega_j)]$, denoted as $k1$;
      $Pop\_BoT(1, j, i) = p\Omega_j(k1)$;
      $flag\_segment = 0$;
      while $flag\_segment == 0$ **do**
        Let $n_p = Pop\_BoT(1, j, i)$; pM is a permutation of elements $\{1, 2, \cdots, n_p\}$;
        A random integer is generated in $[1, m_{n_p}]$, denoted as $k2$;
        if $c_{n_p}^{k2} + \tau_j^\sigma w_{n_p} \leq d_{n_p}^{k2}$ **then**
          $flag\_segment = 1$;
          $Pop\_BoT(2, j, i) = pM(k2)$;
        end
      end
      if $flag\_segment == 1$ **then**
        $flag\_allocated = 1$;
        $c_{n_p}^{k2} = c_{n_p}^{k2} + \tau_j^\sigma w_{n_p}$;
      end
    end
  end
  Initial Pop_DT**:**
  $\left(c_{ij}^D\right)_{N \times N_g}$ is generated randomly, and all the elements range from 0 to 1;
  for $j = 1$ *to* N **do**
    for $k = 1$ *to* N_g **do**
      if $k < m_j$ **then**
        $$Pop\_DT(j, k, i) = c_{jk}^D \Big/ \left(\sum_{j=1}^{N} \sum_{k=1}^{m_j} c_{ij}^D \right);$$
      else
        $Pop\_DT(j, k, i) = 0$;
      end
    end
  end
end

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

4.2. Crossover Operator

For the sake of increasing diversity of the individuals in population, a crossover operator for divisible task(DT) coding presented in algorithm 4 respectively. To decrease the makespan of the tasks, the divisible load should be assigned to the former segment of the processors. So, two weight coefficients, which decrease gradually, are used in step 11 and 12.

Algorithm 4: Crossover operator for DT coding

Input: Individual C^1 in Pop_DT, probability p_n;
Output: Offsprings O^1 and O^2 for DT;
1 **if** $rand() < p_n$ **then**
2 | An individual C^2 is selected in the neighborhoods $neiber(C^1)$ of individual C^1;
3 **else**
4 | An individual C^2 is selected in the population except for the $neiber(C^1)$;
5 **end**
6 $O^1 = C^1, O^2 = C^2$;
7 **for** $i = 1$ *to* N **do**
8 | m_i numbers are generated randomly, and put them into W according to descend order.
9 | $W = W / \sum_{j=1}^{m_i} W(j)$;
10 | **for** $j = 1$ *to* m_i **do**
11 | | $O^1_{ij} = O^1_{ij} + W(j) \times |O^1_{ij} - O^2_{ij}|$;
12 | | $O^2_{ij} = O^2_{ij} + W(j) \times |O^1_{ij} - O^2_{ij}|$;
13 | **end**
14 **end**
15 $O^1 = O^1 \Big/ \left(\sum_{i=1}^{N} \sum_{j=1}^{m_i} O^1_{ij} \right), O^2 = O^2 \Big/ \left(\sum_{i=1}^{N} \sum_{j=1}^{m_i} O^2_{ij} \right)$;

4.3. Mutation Operator

Mutation, which can change some gene values in a parent individual to a new state, is a genetic operator. A better mutation operator can produce entirely novel offspring individuals and improve diversity of the population. With these new individuals, the genetic algorithm may obtain a better solution than previously one possible. Mutation is an essential operator in generic algorithm, and it can help to make the populations escape the local optimum. Suppose that the chromosome $C = (c_{ij})_{2\times N_\tau}$ and $C^D = (c^D_{ij})_{N\times N_g}$ are chosen to take part in mutation, and the offspring $C' = (c'_{ij})_{2\times N_\tau}$, $C^{D'} = (c^{D'}_{ij})_{N\times N_g}$ are obtained by the mutation as shown in algorithm 5 and algorithm 6 respectively. Similar to algorithm 4, weight coefficients are used too.

$$\{np_{best}, ns_{best}\} = \arg\min_{np,ns} \left\{ \max_{np\in\Omega_i, 1\le ns\le m_{np}} \{T_{np}\} \right\} \tag{21}$$

where np is the processor number of a tasks allocated to and ns is the available time segment on processor P_{np}. Eq.(21) is used to search a best processor $P_{np_{best}}$ and available time segment ns_{better} on processor $P_{np_{best}}$ for tasks τ_i to make the maximum processing finish time of the processors in Ω_i is minimum.

4.4. Local Search Operator

Local search is an important operator in generic algorithm, and it can help to jump out the local optima. In this paper, a local search operator, which can accelerate the convergence and enhance the searching ability of the proposed algorithm, is designed. If the local search operator applied to the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

Algorithm 5: Mutation operator for BoT coding

Input: Individual $C = (c_{ij})_{2\times N_\tau}$;
Output: Offspring $C' = (c'_{ij})_{2\times N_\tau}$;

$C' = C$;
Two random integer $i, j (i < j)$ are generated;
for $k = i$ *to* j **do**
  Let $n_p = i + j - k$;
  if $C(1, n_p) \in \Omega_i$ *and* $C(2, n_p) \leq m_{C(1,n_p)}$ **then**
    $C'_{1k} = C_{1n_p}$;
    $C'_{2k} = C_{2n_p}$;
  else
    Calculate np_{best} and ns_{best} by Eq.(21);
    $C'_{1k} = np_{best}$;
    $C'_{2k} = ns_{best}$;
  end
end

Algorithm 6: Mutation operator for DT coding

Input: Individual $C^D = (c^D_{ij})_{N\times N_g}$ and constant F;
Output: Offspring $C^{D'} = (c^{D'}_{ij})_{N\times N_g}$;

$C^{D'} = C^D$;
for $i = 1$ *to* N **do**
  m_i numbers are generated randomly, and put them into W according to descend order.
  $W = W \Big/ \sum_{j=1}^{m_i} W(j)$;
  for $j = 1$ *to* m_i **do**
    if $c^D_{ij} < 0.5$ **then**
      $\delta = (1 - c^D_{ij})^{1+F}$;
    else
      $\delta = (c^D_{ij})^{1+F}$;
    end
    $R = rand()$;
    if $R < 0.5$ **then**
      $\sigma = 1 - (2R + (1 - 2R)\delta)^{\frac{1}{1+F}}$;
    else
      $\sigma = 1 - (2(1 - R) + 2\delta(R - 0.5))^{\frac{1}{1+F}}$;
    end
    $c^{D'}_{ij} = c^D_{ij} + W(j)\sigma$;
  end
  $C^{D'} = C^{D'} \Big/ \left(\sum_{i=1}^{N} \sum_{j=1}^{m_i} c^{D'}_{ij} \right)$;
end

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

chromosome $C = (c_{ij})_{2\times N_\tau}$ and $C^D = (c^D_{ij})_{N\times N_g}$, the offspring $C' = (c'_{ij})_{2\times N_\tau}$ and $C^{D'} = (c^{D'}_{ij})_{N\times N_g}$ are obtained by local search operator as shown in algorithm 7.

$$ns_{better} = \arg\min_{ns}\left\{\max_{1\le ns\le m_{np}}\{T_{np}\}\right\} \tag{22}$$

where np is the processor number that the tasks is allocated to. Eq.(22) is used to search a better available time segment ns_{better} in processor P_{np}.

Algorithm 7: Local search operator

Input: $C = (c_{ij})_{2\times N_\tau}, C^D = (c^D_{ij})_{N\times N_g}$;
Output: $C' = (c'_{ij})_{2\times N_\tau}, C^{D'} = (c^{D'}_{ij})_{N\times N_g}$;

```
C' = C, C^{D'} = C^D;
for i = 1 to N_τ do
    Calculate ns_better by Eq.(22);
    if ns_better ≠ C(2, i) then
        C'_{2i} = ns_better;
    end
end
for i = 1 to N do
    Two integer j_1, j_2 and a random 0 < R < 1 are generated randomly, and 1 ≤ j_1 < j_2 ≤ m_i;
    c^{D'}_{ij_1} = c^D_{ij_2}(1 + R), c^{D'}_{ij_2} = c^D_{ij_2}(1 − R);
end
```

4.5. Modify Operator

For the sake of accelerating the convergence of generic algorithm and minimizing makespan of the tasks, a modify operator is designed. The pseudocode of the modify operator is shown in algorithm8. Chromosome $C = (c_{ij})_{2\times N_\tau}$ and $C^D = (c^D_{ij})_{N\times N_g}$ will modified as $C' = (c'_{ij})_{2\times N_\tau}$ and $C^{D'} = (c^{D'}_{ij})_{N\times N_g}$ according to the modify operator.

5. TSGA for Hybrid Tasks Scheduling

To solve the optimization model, a generic algorithm is designed in section 4. In GA, the allocation scheme of BoT and divisible task are both determined, a large search space exits. So, we propose a two step scheduling algorithm to minimizing makespan of the hybrid tasks. We first scheduling the bag-of-tasks using GA proposed in section 4, and then the divisible task is scheduled through the methods described in section 3.5.1 and section 3.5.2. The framework of the algorithm is shown in algorithm 9.

6. Experiments and Analysis

Several experiments are presented in this section to show the efficiency of the proposed algorithm. In section 6.1, some parameters used in the algorithms will be given. Experimental results are presented in section 6.2. Finally, the experimental results are analyzed in section 6.3.

6.1. Parameters Value

6.1.1. Tasks parameters

In this paper, we investigate the hybrid tasks scheduling problem in heterogeneous distributed system. In the experiments, the workload of the BoT ranges from 1000 to 7000. In simulation system, the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

Algorithm 8: Modify operator

Input: $C = (c_{ij})_{2\times N_\tau}, C^D = (c^D_{ij})_{N\times N_g}$;

Output: $C' = (c'_{ij})_{2\times N_\tau}, C^{D'} = (c^{D'}_{ij})_{N\times N_g}$;

$C' = C, C^{D'} = C^D$;
for $i = 1$ *to* N **do**
  for $j = m_i$ *to* 2 **do**
    Ψ_i^j is the tasks set allocated to j^{th} available time segment of processor P_i, and the tasks sorted descending order according to workload;
    for $k = size(\Psi_i)$ *to* 1 **do**
      if $\exists p < j$ *satisfy* $\tau^{\sigma}_{\Psi_i^j(k)} w_i < d^p - c^p$ **then**
        if $\Psi_i^j(k) \neq 0$ **then**
          $c'_{2\Psi_i^j(k)} = p$;
        else
          $c^{D'}_{ip} = c^{D'}_{ip} + c^{D'}_{ij}$;
          $c^{D'}_{ij} = 0$;
        end
      end
    end
  end
end

Algorithm 9: Framework of TSGA algorithm

Input: Tasks $\tau_i (i = 1, \cdots, N_\tau)$, speed index w_i and available time $[c_i^j, d_i^j] (j = 1, \cdots, m_i)$ of processor $P_i (i = 1, \cdots, N)$;

Output: a schedule scheme;

Scheduling BoT by GA;
if *divisible task workload is small* **then**
  Scheduling the divisible task according to the method proposed in section 3.5.1;
else
  Scheduling the divisible task according to algorithm 2;
end

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1

tasks number N_τ is varying between 50 and 500. In addition, the processors set $\Omega_i (1 \leq i \leq N_\tau)$, which can process task $\tau_i (1 \leq i \leq N_\tau)$, is generated as follow: n_r is a random number in $(0, 1]$, $n_{pro} = \lfloor n_r N \rfloor$. Then, n_{pro} processors are selected in P and put them in Ω_i. In our experiments, the workload of the divisible task is generated randomly in $\left[\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{m_i - 1} \left(w_i (d_i^j - c_i^j) \right), \sum_{i=1}^{N} \sum_{j=1}^{m_i - 1} \left(w_i (d_i^j - c_i^j) \right) \right]$.

6.1.2. System parameters

In this section, we show our simulation system parameters. The simulation system has 20 heterogeneous slave processors, and the time consuming for unit workload $w_i^s (1 \leq i \leq N^s)$ of $P_i (1 \leq i \leq N^s)$ in the heterogeneous distributed system is referred to Shang[43], where N^s denote the number of processors in simulation system. The available time segment m_i on processor P_i is generated randomly in [5, 12]. The length of $k^{th} (1 \leq k < m_i)$ on P_i is generated randomly in $\left[w_i \times \underset{1 \leq i \leq N_\tau}{\text{mean}} \{\tau_i^\sigma\}, 3w_i \times \underset{1 \leq i \leq N_\tau}{\text{mean}} \{\tau_i^\sigma\} \right]$. If $k = m_i$, d_i^k can equal to $+\infty$, that is to say, the processor P_i can processing tasks all the time in m_i^{th} available time segment.

6.1.3. Generic algorithm parameters

In this paper, GA denotes the algorithm of generic algorithm without local search operator, and generic algorithm with local search and modify operator denoted as GALM. Similarly, GAL and GAM are indicated as the generic algorithm with local search and modify operator proposed in section 4 respectively. TSGA denotes the algorithm proposed in section 5 without local search operator, TSGAL, TSGAM and TSGALM are denote the two step scheduling algorithm with local search or modify operator. In the algorithm of GA, GAL, GAM, GALM, TSGA, TSGAL, TSGAMA and TSGALM, the following parameters are chosen: population size $Pop_{size} = 100$, crossover probability $p_c = 0.8$, mutation probability $p_m = 0.05$, elitist number $E = 5$ and maximum iterations $G_{max} = N_\tau$. Since the concept of neighborhoods of a individual is employed, neighbor size $T = 10$ in our experiments.

6.2. Simulation Results

As there is no algorithm available in the literature for scheduling hybrid task in heterogeneous system with unavailable time segment constraints. To evaluate the effectiveness of the proposed scheduling algorithm, we first will present a performance evaluation study in simulation system. First, we will evaluate the nine algorithms (BoTAPDTA, GA, GAL, GAM, GALM, TSGA, TSGAL, TSGAM and TSGALM) on makespan for various tasks number (N_τ) and workload size, and the makespan obtained by the five algorithms are shown in Fig.9(a) to Fig.9(f).

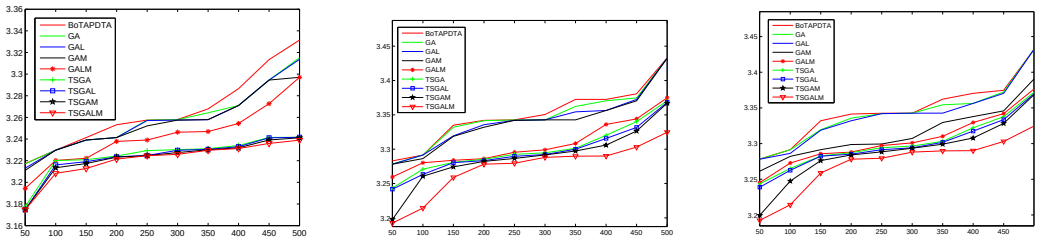
To evaluate the convenience of the eight algorithms(GA, GAL, GAM, GALM, TSGA, TSGAL, TSGAM, TSAGALM), convergence performance of the four algorithms are shown in Fig.10(a) to Fig.10(d). In this experiment, a specific number of task is selected, $N_\tau = 50$, and the maximum iterations $G_{max} = 1000$ in every group experiment.

What is more, we evaluate the robustness of the eight algorithm(GA, GAL, GAM, GALM, TSGA, TSGAL, TSGAM, TSAGALM). In the experiments, every algorithm is executed 30 times independently. Workload of the bag-tasks are uniform distribution in [1000 5000], and the number of task ranges from 50 to 500. The statistics results of the four algorithms are shown in Fig.11(a) to Fig.11(h) respectively using Box-whisker Plot.

6.3. Experimental Analysis

The makespans obtained by the four algorithms are shown in Fig.9. GA, GAL, GAM and GALM can obtain a better scheduling strategy according to all the information and the state of the processors. Since local search operator and modified operator are tailor-made, both of them are conducive to increasing the diversity of solutions and searching a local optimal solution in search space. So, GAL

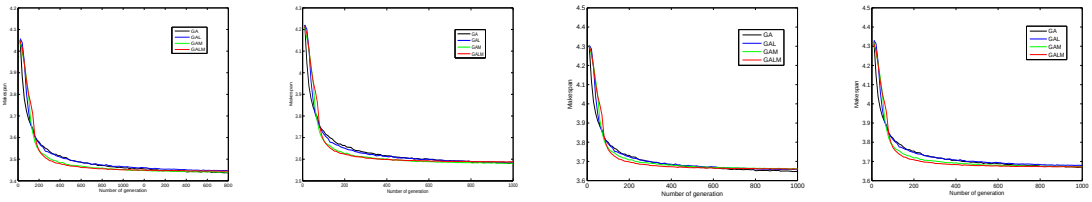
Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 September 2018 doi:10.20944/preprints201809.0331.v1



(a) $1000 \leq \tau_i^{\sigma} \leq 2000$ (b) $500 \leq \tau_i^{\sigma} \leq 3000$ (c) $1000 \leq \tau_i^{\sigma} \leq 4000$

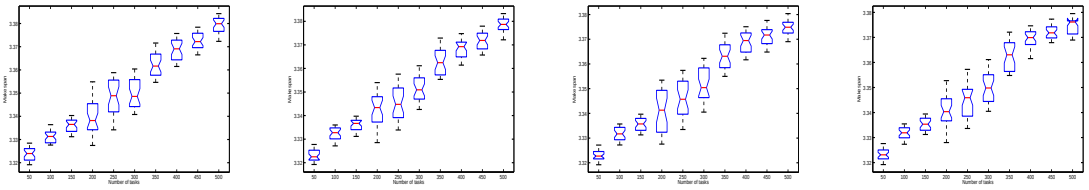
(d) $1000 \leq \tau_i^{\sigma} \leq 5000$ (e) $1000 \leq \tau_i^{\sigma} \leq 6000$ (f) $1000 \leq \tau_i^{\sigma} \leq 7000$

Figure 9. Makespan in small simulation system.

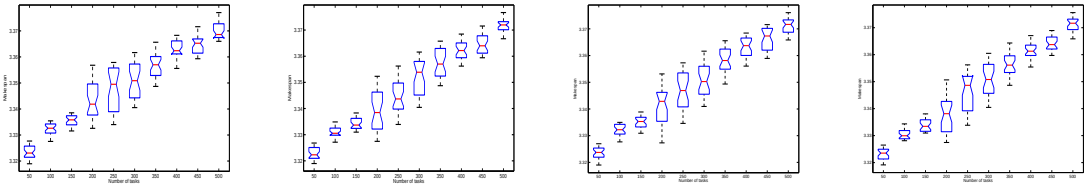


(a) $1000 \leq \tau_i^{\sigma} \leq 2000$ (b) $1000 \leq \tau_i^{\sigma} \leq 3000$ (c) $1000 \leq \tau_i^{\sigma} \leq 4000$ (d) $1000 \leq \tau_i^{\sigma} \leq 5000$

Figure 10. Convergence of the four algorithms in simulation system.



(a) The robustness of GA (b) The robustness of GAL (c) The robustness of GAM (d) The robustness of GALM



(e) The robustness of TSGA (f) The robustness of TSGAL (g) The robustness of TSGAM (h) The robustness of TSGALM

Figure 11. Robustness of the four algorithms in simulation system.

 doi:10.20944/preprints201809.0331.v1

and GAM can convergent to a better solution than GA. That is to say, makespan obtained by GAL and GAM both are smaller than that obtained by GA. GALM is a algorithm that local search and modified operator are added into GA. So, makespan obtained by GALM is smallest. However, GAL is hard to tell from GAM. Because local search operator and modified operator both are search a local optimal solution by changing scheduling scheme of a task. As shown in the Fig.9, we can see that makespan obtained by GALM is smallest, and makespan obtained by GA is largest among the four algorithms(GA, GAL, GAM and GALM). Makespan obtained by GAL is smaller than that obtained by GAM in some cases. But, the opposite results can be obtained in other cases.

In additional, convergence of proposed four algorithms including GA, GAL, GAM and GALM are investigated in a simulation system. In this paper, we design a local search operator and a modify operator, and the two optimization algorithms referred as GAL and GAM. Local search operator and modified operator are conducive to increasing the diversity of the solutions and searching a local optimal solution in search space. On the one hand, local search operator can generate a better offspring than its parent individual by changing the value of a gene. On the other hand, modified operator can also decrease the processing finish time of a processor as much as possible. For a specific generation, the offsprings obtained by local search operator and modified operator will have better fitness than their parents. So, GAL and GAM can convergent to global optimal solution quickly. That is to say, GAL and GAM have a higher convergent speed than GA. However, we can not tell good or bad for GAL and GAM. As we can see in the experimental results, GAL is better than GAM in some cases, and GAM is better than GAL in others case. Because local search operator and modified operator are both searching a local optimal solution by changing scheduling scheme of a task. GALM is a algorithm that comprise of GA, local search operator and modified operator. So, it can convergent to a global optimal solution as fast as possible. As shown in the Fig.9, GALM has a highest convergent speed among the four algorithms (GA, GAL, GAM and GALM), and the convergent speed of GA is lowest.

What is more, the robustness of the four algorithms are investigated in simulation system. Fig.11(a) to Fig.11(d) give the robustness of GA, GAL, GAM and GALM in simulation system for various task number. From the figures, we can see that the four algorithms have a high robustness for various task number. With increasing of the tasks and processors, much more local optimal solutions exit. So, the robustness of the four algorithms are much higher in a simulation system for the same task number.

7. Conclusion

In this paper, we investigate a hybrid tasks comprising both bag-of-tasks(BoT) and divisible tasks scheduling problem with unavailable time considered in heterogeneous distributed system. For the sake of minimizing the makespan of the tasks, a mathematical optimization model with the unavailable time constraint is established. To solve the optimization model effectively, a hybrid scheduling algorithms with local search or modified operator are designed. Makespan obtained by the four algorithms with various number of tasks and workload are evaluated. In addition, convergence and robustness of eight algorithm (GA, GAL, GAM and GALM) are evaluated in the simulation system. Experimental results show that the four algorithms proposed are efficiency. What is more, the four algorithms(GA, GAL, GAM and GALM) can converge to a better makespan and have a better robustness.

Author Contributions: Hejun Xuan (H.X.) contributed to the concept of the research, performed the data analysis and wrote the manuscript; Shiwei Wei (S.W) and Yanling Li (Y.L.) helped perform the data analysis with constructive discussions and helped performed the experiment. Ran Li (R.L.) and Wuning Tong (W.T.) revised the manuscript, and provided the necessary support for the research.

Acknowledgments: This work was supported in part by the National Natural Science Foundation of China, under Grants Nos. 61501393, 61601396 and 61572417, and in part by Nanhu Scholars Program for Young Scholars of Xingyang Normal University.

1. Foster, I.; Kesselman, C.; others. The Grid 2: Blueprint for a future computing infrastructure. *Waltham: Morgan Kaufmann Publishers* **2004**.
2. Wang, X.; Veeravalli, B. Performance Characterization on Handling Large-Scale Partitionable Workloads on Heterogeneous Networked Compute Platforms. *IEEE Transactions on Parallel & Distributed Systems* **2017**, *28*, 2925–2938.
3. Wang, X.; Wang, Y.; Cui, Y. A new multi-objective bi-level programming model for energy and locality aware multi-job scheduling in cloud computing. *Future Generation Computer Systems* **2014**, *36*, 91–101.
4. Sajid, M.; Raza, Z.; Shahid, M. Energy-efficient scheduling algorithms for batch-of-tasks (BoT) applications on heterogeneous computing systems. *Concurrency and Computation: Practice and Experience* **2015**.
5. Srinivasan, S.; Jha, N.K. Safety and reliability driven task allocation in distributed systems. *Parallel and Distributed Systems, IEEE Transactions on* **1999**, *10*, 238–251.
6. Lee, Y.C.; Zomaya, A.Y. Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. *Computers, IEEE Transactions on* **2007**, *56*, 815–825.
7. Anglano, C.; Canonico, M. Scheduling algorithms for multiple bag-of-task applications on desktop grids: A knowledge-free approach. Proceedings of Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on. IEEE, 2008, pp. 1–8.
8. Iosup, A.; Sonmez, O.; Anoep, S.; Epema, D. The performance of bags-of-tasks in large-scale distributed systems. Proceedings of the 17th international symposium on High performance distributed computing. ACM, 2008, pp. 97–108.
9. Beaumont, O.; Carter, L.; Ferrante, J.; Legrand, A.; Marchal, L.; Robert, Y. Centralized versus distributed schedulers for bag-of-tasks applications. *Parallel and Distributed Systems, IEEE Transactions on* **2008**, *19*, 698–709.
10. Benoit, A.; Marchal, L.; Pineau, J.F.; Robert, Y.; Vivien, F. Scheduling concurrent bag-of-tasks applications on heterogeneous platforms. *Computers, IEEE Transactions on* **2010**, *59*, 202–217.
11. Celaya, J.; Arronategui, U. Fair scheduling of bag-of-tasks applications on large-scale platforms. *Future Generation Computer Systems* **2015**, *49*, 28–44.
12. Yang, Y.; Peng, X.; Wan, X. Security-aware data replica selection strategy for Bag-of-Tasks application in cloud computing. *Journal of High Speed Networks* **2015**, *21*, 299–311.
13. Oxley, M.A.; Pasricha, S.; Maciejewski, A.A.; Siegel, H.J.; Apodaca, J.; Young, D.; Briceno, L.; Smith, J.; Bahirat, S.; Khemka, B.; others. Makespan and energy robust stochastic static resource allocation of a Bag-of-tasks to a heterogeneous computing system. *Parallel and Distributed Systems, IEEE Transactions on* **2015**, *26*, 2791–2805.
14. Zhang, F.; Cao, J.; Li, K.; Khan, S.U.; Hwang, K. Multi-objective scheduling of many tasks in cloud platforms. *Future Generation Computer Systems* **2014**, *37*, 309–320.
15. Schmidt, G. Scheduling with limited machine availability. *European Journal of Operational Research* **2000**, *121*, 1–15.
16. Hoong Chuin LAU, C.Z. Job Scheduling with Unfixed Availability Constraints. Proceedings of the 35th Meeting of the Decision Sciences Institute(DSI). ACM, 2004, pp. 4401–4406.
17. Qin, X.; Xie, T. An availability-aware task scheduling strategy for heterogeneous systems. *Computers, IEEE Transactions on* **2008**, *57*, 188–199.
18. Adiri, I.; Bruno, J.; Frostig, E.; Kan, A.R. Single machine flow-time scheduling with a single breakdown. *Acta Informatica* **1989**, *26*, 679–696.
19. Sheen, G.J.; Liao, L.W. Scheduling machine-dependent jobs to minimize lateness on machines with identical speed under availability constraints. *Computers & operations research* **2007**, *34*, 2266–2278.
20. Kacem, I.; Sadfi, C.; El-Kamel, A. Branch and bound and dynamic programming to minimize the total completion times on a single machine with availability constraints. Proceedings of Systems, Man and Cybernetics, 2005 IEEE International Conference on. IEEE, 2005, Vol. 2, pp. 1657–1662.
21. Vahedi-Nouri, B.; Fattahi, P.; Ramezanian, R. Minimizing total flow time for the non-permutation flow shop scheduling problem with learning effects and availability constraints. *Journal of Manufacturing Systems* **2013**, *32*, 167–173.

22. Wang, X.; Cheng, T.E. An approximation scheme for two-machine flowshop scheduling with setup times and an availability constraint. *Computers & operations research* **2007**, *34*, 2894–2901.
23. Tong, Z.; Li, K.; Xiao, Z.; Qin, X. H2ACO: An Optimization Approach to Scheduling Tasks with Availability Constraint in Heterogeneous Systems. *Journal of Internet Technology* **2014**, *15*, 115–124.
24. Tong, Z.; Li, K.; Xiao, Z.; Qin, X. A QoS Scheduling Scheme with Availability Constraint in Distributed Systems. Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2012 13th International Conference on. IEEE, 2012, pp. 481–486.
25. Yuan, H.; Wang, Y.; Chen, L. An availability-aware task scheduling for heterogeneous systems using quantum-behaved particle swarm optimization. In *Advances in Swarm Intelligence*; Springer, 2010; pp. 120–127.
26. Zhou, L.; Wang, H.; Lian, S.; Zhang, Y.; Vasilakos, A.; Jing, W. Availability-aware multimedia scheduling in heterogeneous wireless networks. *Vehicular Technology, IEEE Transactions on* **2011**, *60*, 1161–1170.
27. Robertazzi, T.G. Ten reasons to use divisible load theory. *Computer* **2003**, *36*, 63–68.
28. Chou, F.D. Particle swarm optimization with cocktail decoding method for hybrid flow shop scheduling problems with multiprocessor tasks. *International Journal of Production Economics* **2013**, *141*, 137–145.
29. Marichelvam, M.K.; Prabaharan, T.; Yang, X.S. A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. *Evolutionary Computation, IEEE Transactions on* **2014**, *18*, 301–305.
30. Hu, M.; Veeravalli, B. Dynamic scheduling of hybrid real-time tasks on clusters. *Computers, IEEE Transactions on* **2014**, *63*, 2988–2997.
31. Lee, W.Y.; Hong, S.J.; Kim, J. On-line scheduling of scalable real-time tasks on multiprocessor systems. *Journal of Parallel and Distributed Computing* **2003**, *63*, 1315–1324.
32. Doulamis, N.D.; Doulamis, A.D.; Varvarigos, E.A.; Varvarigou, T.A. Fair scheduling algorithms in grids. *Parallel and Distributed Systems, IEEE Transactions on* **2007**, *18*, 1630–1648.
33. Hu, M.; Veeravalli, B. Requirement-Aware Scheduling of Bag-of-Tasks Applications on Grids with Dynamic Resilience. *Computers, IEEE Transactions on* **2013**, *62*, 2108–2114.
34. Xiao, L.; Zhu, Y.; Ni, L.M.; Xu, Z. Incentive-based scheduling for market-like computational grids. *Parallel and Distributed Systems, IEEE Transactions on* **2008**, *19*, 903–913.
35. Veeravalli, B.; Li, X.; Ko, C.C. On the influence of start-up costs in scheduling divisible loads on bus networks. *Parallel and Distributed Systems, IEEE Transactions on* **2000**, *11*, 1288–1305.
36. Kim, H.; Mani, V. Divisible load scheduling in single-level tree networks: Optimal sequencing and arrangement in the nonblocking mode of communication. *Computers & mathematics with applications* **2003**, *46*, 1611–1623.
37. Charcranoon, S.; Robertazzi, T.G.; Luryi, S. Parallel processor configuration design with processing/transmission costs. *IEEE Transactions on Computers* **2000**, pp. 987–991.
38. Bharadwaj, V.; Xiaolin, L.; Ko, C.C. Design and analysis of load distribution strategies with start-up costs in scheduling divisible loads on distributed networks. *Mathematical and computer modelling* **2000**, *32*, 901–932.
39. Wang, M.; Wang, X.; Meng, K.; Wang, Y. New model and genetic algorithm for divisible load scheduling in heterogeneous distributed systems. *International Journal of Pattern Recognition and Artificial Intelligence* **2013**, *27*, 1359005.
40. Ehrlich, P.R.; Raven, P.H. Butterflies and plants: a study in coevolution. *Evolution* **1964**, pp. 586–608.
41. Sastry, K.; Goldberg, D.E.; Kendall, G. Genetic algorithms. In *Search methodologies*; Springer, 2014; pp. 93–117.
42. Ding, W.; Guan, Z.; Shi, Q.; Wang, J. A more efficient attribute self-adaptive co-evolutionary reduction algorithm by combining quantum elitist frogs and cloud model operators. *Information Sciences* **2015**, *293*, 214–234.
43. Mingsheng, S. Optimal algorithm for scheduling large divisible workload on heterogeneous system. *Applied mathematical modelling* **2008**, *32*, 1682–1695.