IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 23, 2018, accepted November 8, 2018, date of publication November 14, 2018,
date of current version December 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2881268

Fault-Tolerant Scheduling Algorithm With Re-Allocation for Divisible Task

HEJUN XUAN[1], SHIWEI WEI[2], WUNING TONG[3], DAOHUA LIU[1], AND CHUANDA QI[1]

[1]School of Computer and Information Technology, Xinyang Normal University, Xinyang 464000, China
[2]School of Computer and Technology, Guilin University of Aerospace Technology, Guilin 541000, China
[3]School of Science, Shaanxi University of Chinese Medicine, Xinyang 712000, China

Corresponding author: Hejun Xuan (xuanhejun0896@126.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61602422 and Grant 61572417, in part by the Science and Technology Department of Henan Province under Grant 182102210537, in part by the Innovation Team Support Plan of the University Science and Technology of Henan Province under Grant 19IRTSTHN014, in part by the Guangxi Natural Science Foundation of China under Grant 2016GXNSFAA380226, in part by the Guangxi Young and Middle-aged Teachers' Basic Ability Improvement Foundation of China under Grant 2017KY0866, in part by the Internet of Things and Big Data Application Research Foundation of the Guilin University of Aerospace Technology under Grant KJPT201809, and in part by the Nanhu Scholars Program for Young Scholars of XYNU.

ABSTRACT Divisible task fault-tolerant scheduling problems for a heterogeneous system on a general and realistic platform are addressed in this paper, where the communication is in non-blocking message receiving mode, and the processors and communication links may have different speeds and startup overheads. For this kind of problems, the optimal sequence and the fraction of task for each processor are derived first when the fault checkout overhead and checkout time consumption are considered. Then, to decrease the time consumption and checkout overheads, a checkout strategy, which is more suitable for divisible task, is employed. Moreover, an efficient algorithm with the fault fraction units re-allocated is proposed. Finally, the experiments on some simulation examples are conducted and the experimental results indicate that the proposed algorithm is effective, can minimize the expected execution time, and can save the time on fault-tolerant consumption.

INDEX TERMS Divisible task, optimal sequence, task re-allocated, fault-tolerant.

I. INTRODUCTION

Divisible task is the parallel task which can be divided into any number of fractions, and can be processed independently on the processors in parallel since there are no precedence relationships among these fractions [1]. A divisible task is a task that can be arbitrarily split linearly among any number of processors. The applications of the divisible task model occur in many scientific and engineering applications [2], [3], [7], including signal processing, image processing, video and multimedia broad casting, linear algebra computation, and the processing of large distributed files, etc. Divisible task theory has been intensively studied in the past decades. There are many papers focusing on studying high-performance divisible task scheduling on heterogeneous distributed system [4]–[9].

Most systems or platforms used in the real world are heterogeneous systems with different computation or communication speeds. For heterogeneous star/tree networks, a closed form expression for optimal processing time was derived, meanwhile, the effect of task distribution sequences on the processing time was analyzed and an algorithm was developed to find the optimal distribution sequence [11]. It was proved that the distribution order depends only on the communication speeds between nodes but not on the computation speed of each node [10]. The sequence of task distribution should follow the order in which the communication speeds decrease. Based on the non-blocking mode of communications, Shang proposed a more general and realistic model for heterogeneous systems with both communication and computation start-up overheads [12], [13]. The influence of start-up overheads and task distribution sequence on processing time was demonstrated. For the purpose of considering fault-tolerance, we will take checkout start-up overhead and the checkout time consumption into account in this paper, and a closed form expression for optimal processing time will be obtained and also the optimal scheduling sequence will be analyzed.

Nowadays, high performance computing is facing a major challenge with the increasing frequency of failures [14], [15]. There is a need to use fault tolerance or resilience mechanisms

to ensure the efficient progress and correct termination of the applications in the presence of failures. A large number of fault-tolerant techniques have been developed [16]–[21]. Several techniques have been developed with different levels of granularity, and the two representative methods are: (1) Primary backup (PB), and (2) Checkpoint. Fault-tolerant scheduling on heterogeneous system is designed in papers [19], [21]–[23], [26]. In [24], Mohammad proposed a dynamic fault tolerant scheduling. Each task is categorized into critical or noncritical ones based on the task utilization and the time at which scheduler is used to allocate resources to the task. Noncritical tasks are scheduled on a single core, and checkpoint with rollback recovery will be applied to them. These methods are all designed for the independent real-time tasks, and has a very high performance to schedule the real-time task. However, it has a lower performance to schedule the non-real-time task and divisible task. There is a large body of literatures on checkpoint strategies for divisible task. The corresponding scheduling problem is to partition the task into several chunks and to checkpoint after each of them. Daly studies periodic checkpoint policies (same-size chunks) for exponentially distributed failures in [25]. In [27], the authors develop an 'optimal' checkpoint policy, based on the popular assumption that optimal checkpoint must be periodic. Robert in [28] deals with the complexity of scheduling computational work flows in the presence of exponentially distributed failures. When such a failure occurs, rollback and recovery are used so that the execution can resume from the last checkpoint state.

A. MOTIVATION

Growing evidence shows that scheduling is an efficient approach to achieving high performance of applications in parallel systems. A wide variety of scheduling algorithms have been developed to provide optimal scheduling for heterogeneous system supporting divisible tasks applications in the past decade. Unfortunately, to the best of our knowledge, no work has been done on designing scheduling algorithm which takes checkout start-up overhead and the checkout time consumption into account. Re-execution on the same processor is a commonly strategy to fault-tolerant scheduling algorithms for divisible task. But it is not an optimal technique for fault-tolerance. In order to minimizing the time consuming, we can re-distribute those parts of the tasks on the processor with a long re-execution time of the fault tasks to the idle processor or a processor whose finishing time is early.

B. CONTRIBUTIONS

The major contributions of this study are summarized as follows:

(1) For heterogeneous system, we derive a closed form expression for optimal processing time and analyze the optimal distributed sequence when checkout start-up time and checkout time consumption are considered.

TABLE 1. Notations.

Notations	Description
α_i	Fraction of the task assigned to processor P_i.
l_i	link between P_0 and P_i.
g_i	communication time of unit task from P_0 to P_i.
w_i	processing time of P_i to process a unit task.
s_i	start-up overhead before computation P_i.
o_i	start-up overhead during communication to P_i.
c_i	start-up overhead before checkout of P_i.
$\beta_i w_i$	checkout time of P_i to check a unit task, where $\beta_i < 1$
T_i^0	finish time of checkout on P_i.
T_i^c	checkout time consumption on P_i.
T_i^1	time from T_i^0 to fault-tolerant finish time of P_i.

(2) To save checkout time, a checkout strategy that is fit for the divisible task is employed. The checkout is not performed until all the tasks are executed over.

(3) In order to minimize the time consuming, an efficient algorithm with a fault task units re-allocated strategy is proposed.

II. SCHEDULING MODEL AND OPTIMAL SEQUENCE

A. MATHEMATICAL MODEL

The platform considered in this paper is heterogeneous distributed systems. Processors are connected in a star topology, where the center is the master processor P_0, while $P = \{P_1, P_2, \cdots, P_m\}$ are slave processors. The master processor divides the task into $n(n \leq m)$ task fractions, denoted as $\alpha_1, \alpha_2, \cdots, \alpha_n$, and distributes them among all the *n-processors* in a particular sequence. Therefore, we have

$$\sum_{i=i}^{n} \alpha_i = W_{total}, \tag{1}$$

where W_{total} is the entire worktload. Upon receiving their respective task fractions, the slave processors start computing their respective task fractions. The problem is then to determine the optimal sizes of these task fractions that are assigned to the processors and the particular sequence of the task fractions assigned to each slave processor such that the total processing time is minimized. We now introduce some notations that will be used throughout the paper in Table 1.

A primary principle used in the earlier studies in DLT to derive optimal solution is as follows [29]–[33]: in order to obtain an optimal processing time, it is necessary and sufficient to require that all the processors participating in the computation must stop computing at the same time instant. The time diagram of divisible task scheduling in heterogeneous distributed systems is shown in Fig.1.

Because finish times for all processors are equal, the following equation can be obtained intuitively using $T_i^0 = T_{i+1}^0$.

$$s_{i-1} + w_i\alpha_{i-1} + c_{i-1} + \beta_{i-1}w_{i-1}\alpha_{i-1} = g_{i-1}\alpha_{i-1} + o_i + s_i + w_i\alpha_i + c_i + \beta_i w_i\alpha_i \tag{2}$$

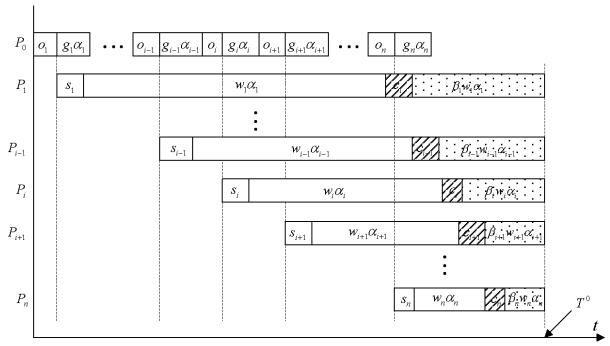


FIGURE 1. Optimal timing diagram of DLS in a particular sequence.

where $i = 2, 3, \cdots, n$. We can rewrite this equation as

$$\alpha_i = \frac{(1+\beta_{i-1})w_{i-1} - g_{i-1}}{(1+\beta_i)w_i}\alpha_{i-1} - \frac{s_{i-1} + c_{i-1} - s_i - c_i - o_i}{(1+\beta_i)w_i} \tag{3}$$

Let

$$\gamma_i = \frac{(1+\beta_{i-1})w_{i-1} - g_{i-1}}{(1+\beta_i)w_i} \tag{4}$$

$$\eta_i = \frac{s_{i-1} + c_{i-1} - s_i - c_i - o_i}{(1+\beta_i)w_i} \tag{5}$$

Thus, Eq.(3) can re-written as

$$\alpha_i = \gamma_i\alpha_i + \eta_i \tag{6}$$

So,

$$\begin{aligned}\alpha_i &= \gamma_i\alpha_i + \eta_i \\ &= \gamma_i\gamma_{i-1}\alpha_{i-2} + \gamma_i\eta_{i-1} + \eta_i \\ &= \gamma_i\gamma_{i-1}\gamma_{i-2}\alpha_{i-3} + \gamma_i\gamma_{i-1}\eta_{i-2} + \gamma_i\eta_{i-1} + \eta_i \\ &\vdots\end{aligned} \tag{7}$$

We can obtain

$$\alpha_i = \mu_i\alpha_1 + \lambda_i, \tag{8}$$

where

$$\mu_i = \prod_{k=2}^{i}\gamma_k, \tag{9}$$

$$\lambda_i = \sum_{k=2}^{i}\left(\rho_k \prod_{j=k+1}^{i}\gamma_j\right). \tag{10}$$

Combining Eq.(1) and Eq.(8), we can obtain α_1 as follows

$$\alpha_1 = \frac{W_{total} - \sum_{k=2}^{n}\lambda_k}{1 + \sum_{k=2}^{n}\mu_k}, \tag{11}$$

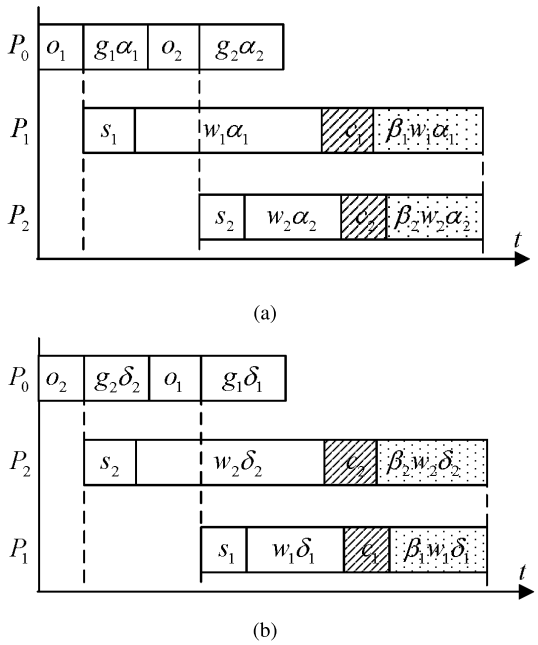


FIGURE 2. Comparison of the two possible sequences.

Thus, the closed-form expression of the checkout finish time T_i^0 is given by Eq(12)

$$\begin{aligned}T^0 &= T_1^0 \\ &= o_1 + s_1 + c_1 + (1+\beta_1)w_1\alpha_1 \\ &= o_1 + s_1 + c_1 + (1+\beta_1)w_1\frac{W_{total} - \sum_{k=2}^{n}\lambda_k}{1 + \sum_{k=2}^{n}\mu_k}\end{aligned} \tag{12}$$

B. OPTIMAL DISTRIBUTED SEQUENCE

Task scheduling problems are among the well-known hardest combinatorial optimization problems in heterogeneous system. An optimal distributed sequence of divisible task is discussed. Using the similar analysis to that in [12], optimal distributed sequence of divisible task is analyzed as follows.

Firstly, we consider the simplest case of scheduling divisible workload on heterogeneous system with only two processors P_1 and P_2. As shown in Fig.2, there are two possible sequences for scheduling.

If T' donate the finish time of the checkout, from Fig.2(a), we can obtain Eq.(13) and Eq.(14) as follows:

$$T' = o_1 + s_1 + c_1 + (1+\beta_1)w_1\alpha_1, \tag{13}$$

$$T' = o_1 + g_1\alpha_1 + o_2 + s_2 + c_2 + (1+\beta_2)w_1\alpha_2. \tag{14}$$

Then

$$\alpha_1 = \frac{T' - (o_1 + s_1 + c_1)}{(1+\beta_1)w_1}, \tag{15}$$

$$\alpha_2 = \frac{T' - (o_1 + o_2 + s_2 + c_1)}{(1+\beta_2)w_2} - \frac{T' - (o_1 + s_1 + c_1)}{(1+\beta_1)(1+\beta_2)w_1w_2}g_1. \tag{16}$$

Let $s'_i = s_i + c_i$, $w'_i = (1+\beta_i)w_i$, $i = 1, 2$, Eq.(15) and Eq.(16) can be rewritten as Eq.(17) and Eq.(18).

$$\alpha_1 = \frac{T' - (o_1 + s'_1)}{w'_1}, \tag{17}$$

$$\alpha_2 = \frac{T' - (o_1 + o_2 + s'_2)}{w'_2} - \frac{T' - (o_1 + s'_1)}{w'_1 w'_2} g_1. \tag{18}$$

Hence the entire workload W^0_{total} to be processed in T' is

$$\begin{aligned} W^0_{total} &= \alpha_1 + \alpha_2 \\ &= \frac{T' - (o_1 + s'_1)}{w'_1} + \frac{T' - (o_1 + o_2 + s'_2)}{w'_2} \\ &\quad - \frac{T' - (o_1 + s'_1)}{w'_1 w'_2} g_1. \end{aligned} \tag{19}$$

As shown in Fig.2(b), using the same way above, we can obtain δ_1 and δ_2. Thus, the entire workload W^1_{total} to be processed in the same time period T' in this alternate sequence is

$$\begin{aligned} W^1_{total} &= \delta_1 + \delta_2 \\ &= \frac{T' - (o_2 + s'_2)}{w'_2} + \frac{T' - (o_1 + o_2 + s'_1)}{w'_1} \\ &\quad - \frac{T' - (o_2 + s'_2)}{w'_1 w'_2} g_2. \end{aligned} \tag{20}$$

The difference between W^0_{total} and W^1_{total} is

$$\begin{aligned} W^0_{total} - W^1_{total} &= \frac{T'(g_2 - g_1) + o_2(w'_2 - w'_1)}{w'_1 w'_2} \\ &\quad + \frac{o_1(g_1 - w'_1) + s'_1 g_1 - s'_2 g_2}{w'_1 w'_2} \end{aligned} \tag{21}$$

Even when the checkout start-up overhead and checkout time consuming are considered, we have also derived the similar computing formula of $W^0_{total} - W^1_{total}$ as that in literature [12]. We can see from this formula that if the workload is large, then, the finishing time T' is long enough, which makes item $T'(g_2 - g_1)$ the most significant and the other items may be neglected. The following conclusion can be obtained if T' is large enough.

$$W^0_{total} > W^1_{total} \Leftrightarrow g_1 < g_2$$

From the conclusion above, we know that for the $2 - processors$, the optimal sequence is $g_1 < g_2$ when the workload is large enough. For the heterogeneous system which has $n - processors$ and each processor has arbitrary checkout, computation and communication start-up overheads, as well as computation and communication speeds, if the workload is large enough, the sequence of workload distribution according to the decreasing order of the link speeds in the non-blocking mode of communication is optimal too. The conclusion can be proven using the similar method in literature [12].

III. A PROPER CHECKOUT STRATEGY FOR DIVISIBLE TASK

A strategy of checkpointing is adopted in order to detect whether the tasks are carried out incorrectly or not. Checkpointing, as a technique for improving the reliability and availability of fault-tolerant computing systems, has been an active area of research in fault-tolerant aware task scheduling system. Most of the works on checkpointing are designed for the real-time system [26], [27], [29], [34]–[37]. Although there have been many researches on this area, how to select the interval of the checkpointing is still a critical issue. Firstly, when a computing error is occurred in some task interval it needs to make the re-execution from the last checkpointing. If the interval of the checkpointing is larger, it will result in the increase of the task re-execution time. On the other hand, if the interval of the checkpointing is smaller, the checkout will start several times and this will increase the checkout overheads. Although there are some other researches on studying the divisible task, they focus only on the probability of failure satisfying a certain distribution [26], [28]. In this paper, a checkout strategy, which is suitable for divisible task, is employed and it has nothing to do with the distribution model of computing failure.

For the divisible task, task fractions are independent and do not depend on the other fractions. We can checkout the task when the tasks execution is finished, as is shown in Fig.1. So the checkout time consumed can be computed by using Eq.(22)

$$T^c_i = c_i + \beta_i w_i N^F_i. \tag{22}$$

where N^F_i is the number of fault task units on processor P_i.

To decrease the checkout time consumed, the checkout only starts one time and is applied to the task unit. The workload is not divided into several chunks. When a computing error occurs in some units, these units will be marked as fault tasks units and re-execution or re-allocation starts when checkout finishes.

IV. FAULT-TOLERANT TASK UNITS RE-ALLOCATION

A. THE NECESSITY OF RE-ALLOCATION

In the work [28], fault-tolerant scheduling algorithms for divisible workload re-execute the tasks incorrectly on the original processor when some failures occur. It is necessary to improve these methods further. Let us imagine a scenario, if a processor has several failure task units and its speed of computing is slow, it will take a long time to re-execute the failure task units. Meanwhile, some high speed processors may have no failure task units. Thus, these processors will be idle when the others are busy in re-executing the failure task units. In this case, it is bound to increase the total time. Thus, the task units can be re-executed not only on the original processor, but also on other processors in order to minimize the total time. To do so, we select two processors, ideally, one has the failure task units and the maximum re-execution time, and another has the minimum sum of the start-up overheads and the highest computing speed. Then, the failure task units

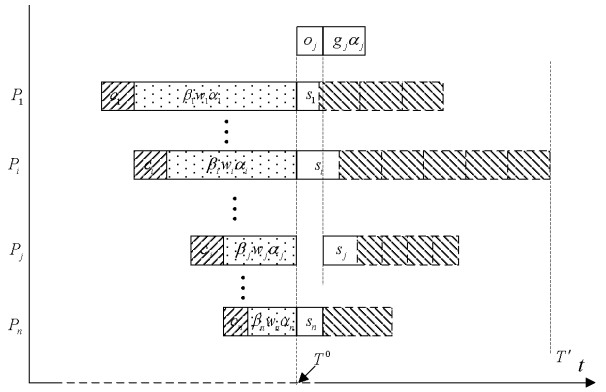


FIGURE 3. The diagram of fault task units re-execution on original processor.

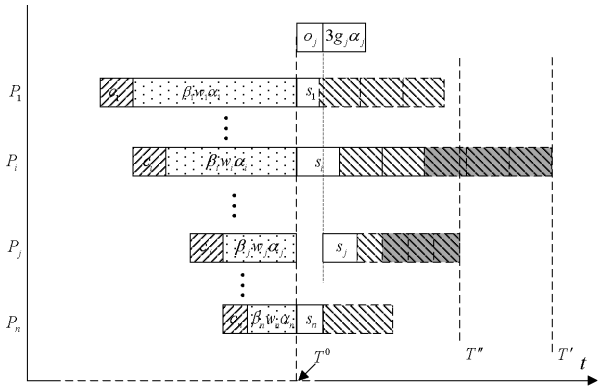


FIGURE 4. The diagram of fault task units re-allocation.

are re-allocated on the processor which has the minimum sum of the start-up overheads and the minimum re-execution time.

As is shown in Fig.3. The processor P_i has five fault task units and a very slow computing speed. So it has the biggest finish re-execution time among all the processors. This will make execution of all the workload continue a long time and increase the complete time. In this case, suppose the finish time of the system is T'. The processor P_j has only one fault task unit and a much higher computing speed than processor P_i. It has a shortest re-execution time of all the processors. It is reasonable to re-allocate three fault task units of processor P_i to processor P_j. As a result, the finish time of processor P_i will be shortened and the system finish time will be decreased. As shown in Fig.4, the finish time of the system is T''. Obviously, T' is much larger than T''.

B. THE PRINCIPLE OF THE FAULT TASK UNITS RE-ALLOCATION

From previous analysis, we know that the reasonable task units re-allocation can decrease the system execution time. However, there are three problems need to be addressed. First, which processor should be selected as the source processor. Second, which processor should be selected as the target processor to which the fault task units will be re-allocated. The last is how many fault task units should be re-allocated from the source processor to the target processor. In this section, we will deal with these problems.

1) THE SOURCE PROCESSOR TO BE SELECTED

The purpose of fault task units re-allocation is to minimize the time consumption of the system. Thus, the objective is to minimize the makespan, i.e., the processing time of the entire task. Let T donates the optimal makespan, we have

$$T = \min\left(\max_{1\le i\le n}\left\{T_i^0 + T_i^1\right\}\right) \tag{23}$$

where the definitions of T_i^0 and T_i^1 are given in Table 1. Eq.(23) shows that the makespan is the maximum processing time of all the processors. So, if we want to minimize the makespan, we must minimize the maximum processing time among all the processors. So it is reasonable to select the processor with maximum processing time as the source processor. That is, the processor P_s is selected as the source processor when s satisfy the Eq.(24)

$$s = arg\max_i\left\{T_i^0 + s_i + w_i N_i^F\right\}, \tag{24}$$

where N_i^F is the number of fault task units on processor P_i. Since $T_i^0 = T_j^0 (i \neq j \ and \ i, j = 1, 2, \cdots, n)$, so the source processor P_s can be selected according to Eq.(25)

$$s = arg\max_i\left\{s_i + w_i N_i^F\right\}. \tag{25}$$

2) TARGET PROCESSOR AND NUMBER OF RE-ALLOCATED TASK UNITS DETERMINED

If a processor has the minimum processing time, it may have a lower computing speed. Similarly, if a processor has a high speed, it may have a longer processing time. Thus, how to select the target processor is not an easy task. Also, considering the processing time and the computing speed separately is unreasonable. Moreover, start-up overheads cannot be neglected and they should be taken into account. So, we must take the processing time, computing speed and all start-up overheads into account together while we select the target processor.

Another critical issue is how to determine the number of fault task units, which will be re-allocated to the target processor. In this paper, an optimization model is set up to determine the target processor and the number of fault task units to be re-allocated. As shown in Eq.(26), the purpose of the model is to minimize the maximum processing time of the source processor and target processor. The fault task units will be re-allocated to the target processor immediately once the target processor and the number of fault task units

are determined.

$$\begin{cases} \min\limits_{l,x} f(l,x) = \min\left(\max\left\{T_s^c - w_s x, \Delta\right\}\right) \\ s.t. \\ \quad 1 \le x \le N_s^F \\ \quad 1 \le l \le n \\ \quad x \in Z, \quad l \in Z \end{cases} \tag{26}$$

where x is the number of fault task units to be re-allocated to the target processor, P_l is the target processor, $\sum_{k=1}^{j-1} o_{s_k}$ is the sum of start-up overheads of the last $(j-1)$ installment re-allocation, P_s is the source processor and N_s^F is the number of fault task units on the processor P_s, $\Delta = \sum\limits_{k=1}^{j-1} o_{s_k} + o_l + T_l^1 + w_l x$

If m and N_s^F are larger, the model (26) has a lager search space and it is hard to solve. So, a simpler and equivalent model is given.In order to minimize the maximum processing time of the source and target processor, the processing time of the source and target processors should be equal. Since the start-up overheads exist, the finish time is hard to be equal, but it is better to re-allocate fault task units such that the finish time is approximately equal. If let T^m denote the mean of the processing time of source and target processors, so the processing time of source and target processors should approach to T^m as much as possible after re-allocation. In order to minimize the finish time of source and target processors, the the finish time must be greater than $T^m - \max\{w_s, w_t\}$ and less than $T^m + \max\{w_s, w_t\}$ as shown Fig.(5). Thus, from Fig.(5) (a) and (b), Eq.(27) and Eq.(28) can be obtained.

$$\begin{cases} \sum\limits_{k=1}^{j-1} o_k + o_t + T_t^c + w_t x > T^m - \max\{w_s, w_t\} \\ T_s^c - w_s x < T^m + \max\{w_s, w_t\} \\ T_s^c - w_s x < T^m + \sum\limits_{k=1}^{j-1} o_k + o_t + T_t^c + w_t x \end{cases} \tag{27}$$

$$\begin{cases} \sum\limits_{k=1}^{j-1} o_k + o_t + T_t^c + w_t x < T^m + \max\{w_s, w_t\} \\ T_s^c - w_s x > T^m - \max\{w_s, w_t\} \\ T_s^c - w_s x < T^m - \sum\limits_{k=1}^{j-1} o_k + o_t + T_t^c + w_t x \end{cases} \tag{28}$$

From Eq.(27) and Eq.(28), we can obtain Eq. (29) as follow.

$$C_c - C_w \le x \le C_c + C_w \tag{29}$$

where $C_c = \frac{(T_s^c - T_t^c) - \sum_{k=1}^{j-1} o_k - o_t}{w_s + w_t}$ and $C_w = \frac{\max\{w_s, w_t\}}{w_s + w_t}$. From analysis above, we can re-write Eq.(26) as Eq.(30). The larger search space of the can be reduced to $\lceil 2C_w \rceil \times n$. Since $C_w < 1$, the search space of Eq.(30) is $2n$. In this paper, the method that is proposed in literature [38] is used to solve this "Min-Max" model.

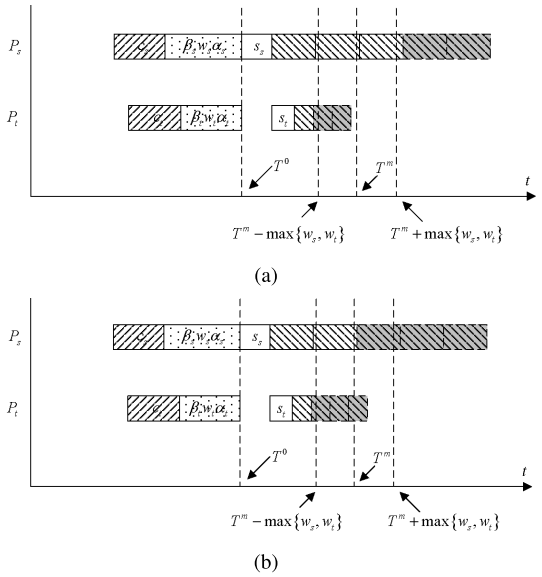


FIGURE 5. Comparison of the two possible sequences.

$$\begin{cases} \min\limits_{l,x} f(l,x) = \min\left(\max\left\{T_s^1 - w_s x, \Delta\right\}\right) \\ s.t. \\ \quad C_c - C_w \le x \le C_c + C_w \\ \quad 1 \le l \le n \\ \quad x \in Z, \quad l \in Z \end{cases} \tag{30}$$

The solution of the optimization model above is to determine the target processor and the number of fault task units to be re-allocated. If the optimization model has the solution of $\{sour, 0\}$, it means that there is no need to do the re-allocation. If the optimization model has more than one solution, then, a further judgement is needed. Assuming that $\{sour_i, x_i\}$ is the i^{th} solution of the optimization model, we will select a solution which has the minimum of $\{x_i\}$ as the optimal solution in order to save energy consumption when the task units are re-allocated.

Theorem 1: T^* is the makespan of fault-tolerant scheduling of the original strategy. T^{**} is the makespan of fault-tolerant scheduling with the above re-allocation strategy. Then we have $T^{**} \le T^*$.

Proof: (1) when the first re-allocation is executed and the number of fault task units re-allocated is $x = 0$, that is, $\forall x \ne 0, l \ne s$ (P_s is the source processor) and l cannot satisfy the following Eq.(31)

$$\max\left\{T_s^1 - w_s x, T_l^1 + o_l + w_l x\right\} < T_s^1 \tag{31}$$

It means that there is no need to do the re-allocation. So $T^{**} = T^*$.

(2) When the first re-allocation is executed and the number of fault task units re-allocated is $x \ne 0$, that is, $\exists l \ne s$ and l satisfies the Eq.(31). So $T^{**} < T^*$.

TABLE 2. Parameters of the heterogeneous distributed system.

P	o	s	g	w	β	c
P_1	30.19	1.40	0.77	7.60	0.10	30.02
P_2	81.44	4.53	0.70	6.14	0.09	20.69
P_3	52.48	5.35	0.76	5.92	0.08	32.58
P_4	46.87	62.26	0.29	6.47	0.11	28.36
P_5	26.37	82.98	0.27	8.24	0.08	25.23
P_6	58.91	91.09	0.98	9.26	0.12	48.36
P_7	69.51	24.39	0.98	5.33	0.07	25.84
P_8	10.63	67.61	0.99	5.57	0.09	19.99
P_9	57.51	10.30	0.10	7.98	0.08	37.23
P_{10}	28.44	29.57	0.04	7.82	0.09	28.68
P_{11}	30.09	97.98	0.94	7.01	0.10	31.23
P_{12}	27.82	16.28	0.16	6.46	0.09	64.23
P_{13}	70.55	57.95	0.53	10.89	0.08	26.38
P_{14}	86.26	37.35	0.79	9.61	0.10	39.21
P_{15}	87.14	94.95	0.05	9.64	0.09	41.23

(3) From(2), we know that as long as one installment re-allocation is implemented successfully, $T^{**} < T^*$ is satisfied. So if $j > 1$ (j is the number of the successful re-allocations), Eq. (32) can be obtained.

$$T^{**} = T_j^{**} < T_{j-1}^{**} < \cdots < T_1^{**} < T^* \tag{32}$$

where T_k^{**} is the makespan after k installments re-allocations are implemented successfully. Thus, $T^{**} \leq T^*$. □

C. FAULT-TOLERANT SCHEDULING ALGORITHM

To obtain the optimal makespan of the task, frictions of the task should be distributed in the decreasing order of the link speeds. Then, checkout the tasks among all the processors. If the fault task units are not re-allocated to other processors, it will decrease the utilization of the processors and increase the makespan of the task. So, fault-tolerant with the fault task units re-allocated strategy should be implemented reasonably. When the fault task units are re-allocated to the target processor, they will be processed on the target processor. The pseudocode of fault-tolerant scheduling algorithm with task re-allocation for divisible task scheduling(FTR_DLS) in heterogeneous computing systems is outlined in Algorithm1.

Algorithm 1 Fault-Tolerant With Task Re-Allocation for DLS(FTR_DLS)

Input: o, s, g, w, c, β
Output: *Makespan*

Task scheduling according to decreasing order of g_i.
Initialization:$reallocate\_flag = 1; j = 1, K = \phi, sum\_overheads = 0$;
$max\_Loc = arg\ \min_i\{T^0 + s_i + w_i \times N_i^F\}$;
min_Loc and *num_transfer* are obtained by solving Eq.(30);
while $reallocate\_flag == 1$ **do**
  $current\_reallocate\_flag == 1$;
  if $num\_transfer == 0$ **then**
    $current\_reallocate\_flag = 0$;
  else
    update the $T^1_{max\_Loc}$ and $N^F_{min\_Loc}$
    if $min\_Loc \in K$ **then**
      $sum\_overheads1 = \sum_{k=1}^{j-1} o_k; flag = 0;$
      $sum\_overheads = \sum_{k=1}^{m-1} o_k$; %
      m is the position in K.
    else
      $sum\_overheads = sum\_overheads + o_{min\_Loc} + s_{min\_Loc}$;
      $K(1, j) = min\_Loc$;
      $sum\_overheads1 = sum\_overheads$;
      $flag = 1, j = j + 1$;
    end
    $T^1_{min\_Loc} = T^1_{min\_Loc} + flag \times sum\_overheads + w_i \times num\_transfer$;
    $max\_Loc = arg\ \min_i\{T^0 + s_i + w_i \times N_i^F\}$;
    min_Loc and *num_transfer* are obtained by solving Eq.(30);
    $current\_reallocated\_flag = 1$;
  end
  $reallocate\_flag = current\_reallocate\_flag$;
end
$Makespan = \max_i\{T^0 + T_i^1\}$;

V. EXPERIMENTS AND ANALYSIS

A. EXPERIMENTS

1) COMPARED EXPERIMENT

In this subsection, all experiments are carried out on a personal computer of HP with Intel(R) Core(TM) i7 CPU, 8G RAM and a 64-bit OS. The parameters in Shang [12] are adopted in the heterogeneous distributed system in our simulation studies and shown in Table 2. Since Shang [12] does not provide the parameter of checkout speed, some data is generated randomly and added in Table 2. Also, since the probability of failure often obeys Exponential distributions in literature [28], we also use this assumption in the experiments.

Literature [25] proposes an algorithm aiming at optimizing checkpoint interval (HOEOCI) for restart dumps. Failures may occur during the execution of a task. Re-execution of the fualt task units only on the former processor is employed, and not on other processors [25]. So the makespan is bigger than that of the re-allocated strategy. Literature [28] proposes a method (CSCW) to deal with the complexity of scheduling computational workflows in the presence of Exponentially distributed failures. When such a failure occurs, rollback and recovery are used so that the execution can resume from the last checkpointing state. The goal is to minimize the expected execution time (makespan). However, the fault tasks will be re-executed on the former processor, and we can know from theorem 1 that the makespan is larger than that of the re-allocated strategy. For the divisible task, tasks do not depend on the other tasks. Making check on the tasks will also decrease the checkout time consumption when the tasks execution is finished. So the FTR_DLS algorithm has more advantages compared with the other algorithms in decreasing execution time. Fig.6 shows the makespan of the workload ranges from 10^4 to 10^6.

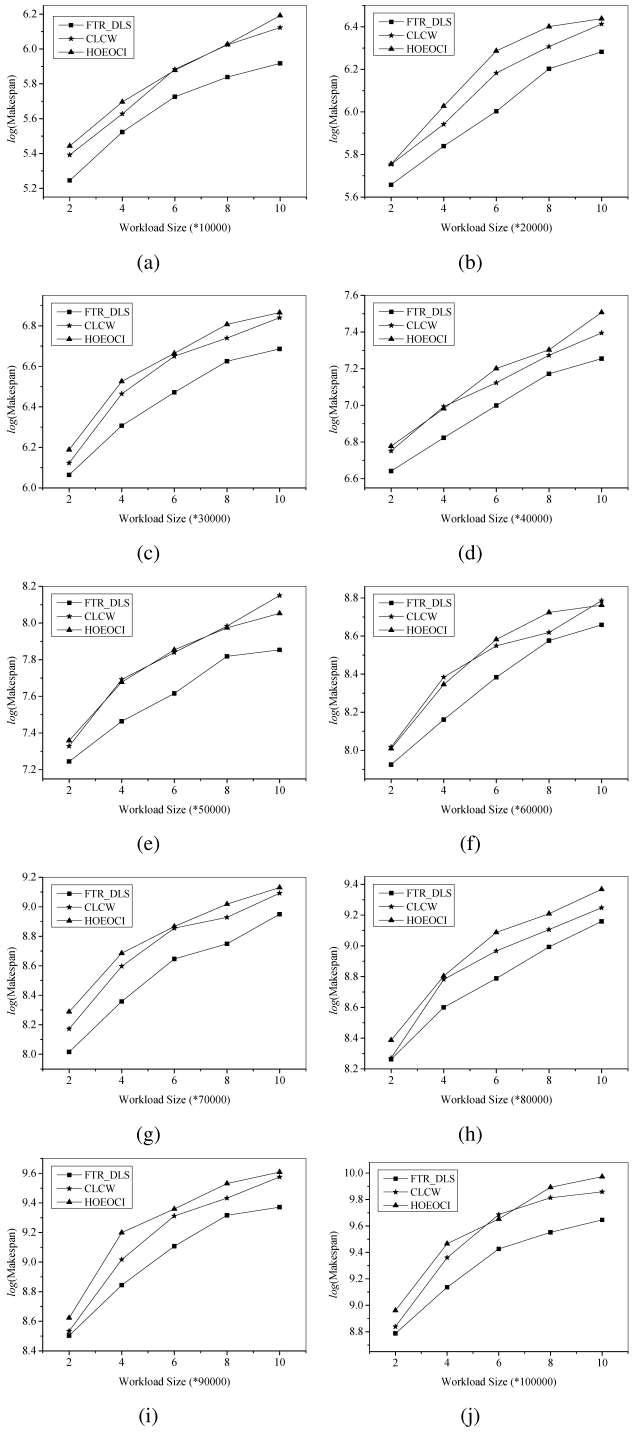


FIGURE 6. The makespan of FTR_DLS, HOEOCI and CSCW.

To evaluate the stability of the proposed algorithm and the compared algorithms, we give the statistical results (Mean and Variance) in the experiments with the different experimental scenes. Table 3 shows the mean and variance results of the makespan with different scenes. In the statistical results, the workloads are set as $W_{total} = \chi \times \nu$, and $\chi = 10000, 20000, \cdots, 60000$; $\nu = 2, 4, \cdots, 10$. From the statistical results, we can see that proposed algorithm (BiHMA) is better than the compared algorithms. Not only the statistical results of mean but also the statistical results of variance are smaller than the compared algorithms. In addition, the variance are increased with the workload increased.

TABLE 3. Statistical results (Mean and Variance) of the makespan.

χ	ν	FTR_DLS	CLCW	HOEOCI
10000	2	5.2452 (1.23E-2)	5.3920 (1.56E-2)	5.4430 (1.54E-2)
	4	5.5233 (1.86E-2)	5.6272 (2.12E-2)	5.6962 (2.30E-2)
	6	5.7262 (2.07E-2)	5.8835 (2.35E-2)	5.8783 (2.52E-2)
	8	5.8393 (2.45E-2)	6.0243 (2.76E-2)	6.0274 (2.81E-2)
	10	5.9176 (2.86E-2)	6.1234 (3.24E-2)	6.1917 (3.35E-2)
20000	2	5.6576 (2.34E-2)	5.7542 (2.79E-2)	5.7552 (2.82E-2)
	4	5.8394 (2.86E-2)	5.9421 (3.23E-2)	6.0275 (3.40E-2)
	6	6.0028 (3.05E-2)	6.1828 (3.67E-2)	6.2867 (3.71E-2)
	8	6.2024 (3.44E-2)	6.3069 (3.96E-2)	6.4010 (3.88E-2)
	10	6.2822 (3.89E-2)	6.4130 (4.37E-2)	6.4381 (4.43E-2)
30000	2	6.0642 (2.46E-2)	6.1231 (3.09E-2)	6.1885 (3.12E-2)
	4	6.3069 (2.84E-2)	6.4646 (3.56E-2)	6.5260 (3.47E-2)
	6	6.4719 (3.49E-2)	6.6500 (3.94E-2)	6.6646 (3.90E-2)
	8	6.6252 (3.97E-2)	6.7397 (4.32E-2)	6.8077 (4.29E-2)
	10	6.6862 (4.16E-2)	6.8403 (4.69E-2)	6.8659 (4.58E-2)
40000	2	6.6414 (2.78E-2)	6.7516 (3.35E-2)	6.7772 (3.49E-2)
	4	6.8230 (3.22E-2)	6.9937 (3.89E-2)	6.9820 (3.96E-2)
	6	6.9991 (3.81E-2)	7.1234 (4.48E-2)	7.2007 (4.56E-2)
	8	7.1718 (4.16E-2)	7.2731 (4.96E-2)	7.3032 (4.89E-2)
	10	7.2548 (4.47E-2)	7.3950 (5.25E-2)	7.5066 (5.34E-2)
50000	2	7.2442 (2.92E-2)	7.3285 (3.64E-2)	7.3587 (3.70E-2)
	4	7.4636 (3.39E-2)	7.6925 (4.05E-2)	7.6772 (4.11E-2)
	6	7.6154 (3.94E-2)	7.8393 (4.84E-2)	7.8534 (4.79E-2)
	8	7.8184 (4.25E-2)	7.9832 (5.26E-2)	7.9738 (5.32E-2)
	10	7.8536 (4.66E-2)	8.1499 (5.67E-1)	8.0526 (5.59E-1)
60000	2	7.9248 (3.25E-2)	8.0170 (3.98E-1)	8.0094 (3.91E-1)
	4	8.1608 (3.67E-2)	8.3840 (4.43E-1)	8.3454 (4.37E-1)
	6	8.3835 (4.12E-2)	8.5489 (5.15E-2)	8.5821 (5.23E-2)
	8	8.5759 (4.59E-2)	8.6190 (5.58E-2)	8.7240 (5.62E-2)
	10	8.6586 (4.83E-2)	8.7856 (6.07E-2)	8.7618 (6.12E-2)
70000	2	8.0151 (3.85E-2)	8.1719 (4.42E-2)	8.2880 (4.49E-2)
	4	8.3582 (4.26E-2)	8.5965 (4.98E-2)	8.6857 (5.03E-2)
	6	8.6466 (4.62E-2)	8.8549 (5.45E-2)	8.8667 (5.50E-2)
	8	8.7491 (4.92E-2)	8.9284 (5.98E-2)	9.0177 (6.02E-2)
	10	8.9500 (5.23E-2)	9.0920 (6.37E-2)	9.1301 (6.30E-2)
80000	2	8.2624 (4.12E-2)	8.2708 (4.98E-2)	8.3869 (5.09E-2)
	4	8.6000 (4.66E-2)	8.7817 (5.58E-2)	8.8039 (5.49E-2)
	6	8.7883 (4.89E-2)	8.9672 (5.97E-2)	9.0878 (6.14E-2)
	8	8.9927 (5.32E-2)	9.1059 (6.36E-2)	9.2090 (6.33E-2)
	10	9.1585 (5.64E-2)	9.2479 (6.71E-2)	9.3680 (6.64E-2)
90000	2	8.5032 (4.44E-2)	8.5335 (5.23E-2)	8.6228 (5.32E-2)
	4	8.8445 (4.89E-2)	9.0168 (5.87E-2)	9.1989 (5.93E-2)
	6	9.1072 (5.16E-2)	9.3114 (6.55E-2)	9.3588 (6.48E-2)
	8	9.3158 (5.47E-2)	9.4321 (7.09E-2)	9.5308 (6.98E-2)
	10	9.3709 (5.93E-2)	9.5750 (7.42E-2)	9.6083 (7.39E-2)
100000	2	8.7875 (4.83E-2)	8.8385 (5.62E-2)	8.9609 (5.59E-2)
	4	9.1365 (5.14E-2)	9.3596 (6.28E-2)	9.4662 (6.23E-2)
	6	9.4264 (5.43E-2)	9.6866 (6.79E-2)	9.6526 (6.84E-2)
	8	9.5514 (5.99E-2)	9.8135 (7.33E-2)	9.8925 (7.42E-2)
	10	9.6459 (6.25E-2)	9.8583 (7.97E-2)	9.9731 (7.90E-2)

2) PERFORMANCE EVALUATION

In this subsection, we present several groups of experimental results obtained from extensive simulations to evaluate the performance of FTR_DLS. The parameters of the heterogeneous distributed system are shown in Table 2. To study the influence of the probability of failure to *PIR*, several groups of the experiments with different probabilities of failure are conducted. In this paper, the probability of failure in every group of the experiments is generated randomly among 0.5%-1%, 1%-2%, 2%-3%, 3%-4%, 4%-5% respectively.

To show the advantage of the proposed scheduling of the fault task units re-allocation, a definition of the performance improvement ratio (*PIR*) is used. *PIR* is defined as follows.

$$PIR = \frac{T_0^c - T_1^c}{T_0^c} \tag{33}$$

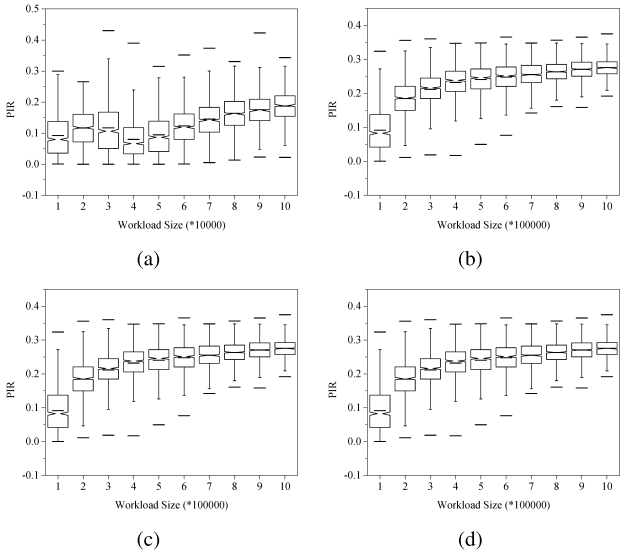


FIGURE 7. The variation of *PIR* with the task size ranges when the probability of failure is in1%-2%.

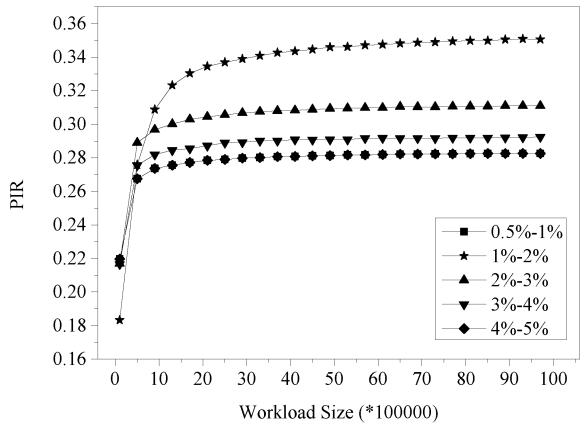


FIGURE 8. The mean *PIR* variation with the workload size and the probability of failure.

where T_0^c is fault-tolerant time (from T^0 to T' as shown in Fig.4) without the re-allocation strategy, T^1 fault-tolerant time (from T^0 to T'' as shown in Fig.4) with the re-allocation strategy employed.

Figs.7(a), (b), (c) and (d) show the statistical results of *PIR* when the workload size ranges from 10^4 to 10^7, and the probability of failure is generated randomly among 1%-2%. Some representative workload sizes are simulated and every workload size is executed 1000 times.

Fig.8 shows the variation tendency of the mean of *PIR* when the workload size ranges from 10^4 to 10^8 with the different probabilities of failures.

B. EXPERIMENT ANALYSIS

As shown in Fig.7(a), *PIR* does not monotonically increase because the failure is random and the condition of re-allocation is difficult to satisfy when the workload size is smaller. The number of fault task units is approximately equal to the ideal one and the condition of re-allocation is easy to satisfy with the increase of workload size, so the variation of *PIR* is stable and increases slowly as shown in Fig.7(b), (c) and (d), respectively. Fig.8 shows the mean *PIR* varies with the workload size and the probability of failure, we can observe that when the task size is considerably large, the mean of *PIR* reaches 35% when the probability of failure is in 1%-2%. That is to say, fault-tolerant scheduling with re-allocation strategy can save 35% time consuming than without re-allocation. Moreover, for the fixed workload size, the *PIR* decreases with the increase of $\overline{p}/p_{max}$, When the workload size is large enough, we can re-write Eq.(33) as

$$PIR = \frac{\max\limits_{1\le i\le n}\{p_i\alpha_i\} - \left(\overline{p\alpha} + \sum\limits_{i=1}^{n}(o_i + s_i)\right)}{\max\limits_{1\le i\le n}\{p_i\alpha_i\}} \tag{34}$$

where $\overline{p} = \sum_{i=1}^{n} p_i/n$ and $\overline{\alpha} = W_{total}/n$. When the workload size is large enough, we have

$$\frac{\sum\limits_{i=1}^{n}(o_i + s_i)}{\max\limits_{1\le i\le n}\{p_i\alpha_i\}} \to 0, \tag{35}$$

then

$$\begin{aligned} PIR &\approx \frac{\max\limits_{1\le i\le n}\{p_i\alpha_i\} - \overline{p\alpha}}{\max\limits_{1\le i\le n}\{p_i\alpha_i\}} \\ &\ge 1 - \frac{\overline{p\alpha}}{p_{max}\alpha_{max}} \\ &= 1 - \frac{\overline{p}}{p_{max}}\delta, \end{aligned} \tag{36}$$

where $\delta = W_{total}/n\alpha_{max}$ and it has nothing to do with the the probability of failure. Since $\delta = W_{total}/n\alpha_{max}$ is close to a constant gradually when the workload size gradually becomes large enough, thus, the *PIR* will be close to a constant. From Eq.(26), we know that the *PIR* decreases with the increases of $\overline{p}/p_{max}$, In our experiments $\overline{p}/p_{max}$ increases when the probability of failures is in 0.5%-1%, 1%-2%, 2%-3%, 3%-4%, 4%-5%, respectively. The *PIR* decreases for the same workload as shown in Fig.8.

VI. CONCLUSION

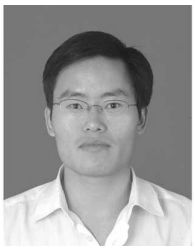
The goal of this paper is to find an optimal fault-tolerant scheduling for divisible task in heterogeneous distributed systems. By setting up an optimization model for fault task units re-allocation, we propose an effective scheduling algorithm. First, a closed form expression for optimal processing time and optimal scheduling sequence are derived. Then, a check-out method which is suitable for divisible task is employed. Finally, we proposed a novel fault-tolerant scheduling algorithm with a fault task units re-allocation strategy. In order to examine the performance of the proposed algorithm, a set of experiments are carried out. From the experimental results,

we can see that the proposed fault-tolerant scheduling algorithm with re-allocation strategy can save time compared with re-execution on original processor without re-allocation.

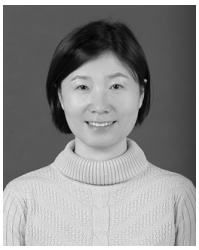
REFERENCES

[1] T. G. Robertazzi, "Ten reasons to use divisible load theory," *Computer*, vol. 36, no. 5, pp. 63–68, May 2003.
[2] C.-Y. Chen and C.-P. Chu, "Divisible nonlinear load distribution on heterogeneous single-level trees," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 54, no. 4, pp. 1664–1678, Aug. 2018.
[3] J. Li, W. Yao, Y. Zhang, H. Qian, and J. Han, "Flexible and fine-grained attribute-based data storage in cloud computing," *IEEE Trans. Services Comput.*, vol. 10, no. 5, pp. 785–796, Sep./Oct. 2017.
[4] X. Kong, C. Lin, Y. Jiang, W. Yan, and X. Chu, "Efficient dynamic task scheduling in virtualized data centers with fuzzy prediction," *J. Netw. Comput. Appl.*, vol. 34, no. 4, pp. 1068–1077, 2011.
[5] G. Lizheng, S. Zhao, S. Shen, and C. Jiang, "Task scheduling optimization in cloud computing based on heuristic algorithm," *J. Netw.*, vol. 7, no. 3, pp. 547–553, 2012.
[6] X. Wang and B. Veeravalli, "Performance characterization on handling large-scale partitionable workloads on heterogeneous networked compute platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2925–2938, Oct. 2017.
[7] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, pp. 931–945, Jun. 2011.
[8] D. Liang, S. Zhong, C. Ting, and Y. Chang "Analysis and modeling of task scheduling in wireless sensor network based on divisible load theory," *Int. J. Commun. Syst.*, vol. 27, no. 5, pp. 721–731, 2014.
[9] O. Beaumont, L. Eyraud-Dubois, C. T. Caro, and H. Rejeb, "Heterogeneous resource allocation under degree constraints," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 5, pp. 926–937, May 2013.
[10] H. J. Kim and V. Mani, "Divisible load scheduling in single-level tree networks: Optimal sequencing and arrangement in the non-blocking mode of communication," *Comput. Math. Appl.*, vol. 46, nos. 10–11, pp. 1611–1623, 2003.
[11] K. Sung-Soo, B. Ji-Hwan, Y. Hong, and L. Hongbo, "Biogeography-based optimization for optimal job scheduling in cloud computing," *Appl. Math. Comput.*, vol. 247, pp. 266–280, Nov. 2014.
[12] S. Mingsheng, "Optimal algorithm for scheduling large divisible workload on heterogeneous system," *Appl. Math. Model.*, vol. 32, no. 9, pp. 1682–1695, 2008.
[13] H. Han, W. Bao, X. Zhu, X. Feng, and W. Zhou, "Fault-tolerant scheduling for hybrid real-time tasks based on CPB model in cloud," *IEEE Access*, vol. 6, pp. 18616–18629, 2018.
[14] J. Dongarra *et al.*, "The international exascale software project: A call to cooperative action by the global high-performance community," *Int. J. High-Perform. Comput. Appl.*, vol. 23, no. 4, pp. 309–322, 2009.
[15] G. Yao, Y. Ding, and K. Hao, "Using imbalance characteristic for fault-tolerant workflow scheduling in Cloud Systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3671–3683, Dec. 2017.
[16] H. Xiaomin, Q. Xiao, and Q. Meikang, "QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters," *IEEE Trans. Comput.*, vol. 60, no. 6, pp. 800–812, Jun. 2011.
[17] S. Haider and B. Nazir, "Dynamic and adaptive fault tolerant scheduling with QoS consideration in computational grid," *IEEE Access*, vol. 5, pp. 7853–7873, 2017.
[18] J. Bahman, T. Parimala, and B. Rajkumar, "Enhancing performance of failure-prone clusters by adaptive provisioning of cloud resources," *J. Supercomput.*, vol. 63, no. 2, pp. 467–489, 2013.
[19] B. Jasma and R. Nedunchezhian, "Performance-driven load balancing with a primary-backup approach for computational grids with low communication cost and replication cost," *IEEE Trans. Comput.*, vol. 62, no. 5, pp. 990–1003, May 2013.
[20] N. Babar, Q. Kalim, and M. Paul, "Replication based fault tolerant job scheduling strategy for economy driven grid," *J. Supercomput.*, vol. 62, no. 2, pp. 855–873, 2012.
[21] Q. Zheng, B. Veeravalli, and C. K. Tham, "On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs," *IEEE Trans. Comput.*, vol. 58, no. 3, pp. 380–393, Mar. 2009.
[22] Z. Xiaomin, H. Chuan, G. Rong, and L. Peizhong, "Boosting adaptivity of fault-tolerant scheduling for real-time tasks with service requirements on clusters," *J. Syst. Softw.*, vol. 84, no. 10, pp. 1708–1716, 2011.
[23] S. Wei, Y. Chen, D. Xavier, and I. Yasushi, "Dynamic scheduling real-time task using primary-backup overloading strategy for multiprocessor systems," *IEICE Trans. Inf. Syst.*, vol. E91.D, no. 3, pp. 796–806, 2008.
[24] M. H. Mottaghi and H. R. Zarandi, "DFTS: A dynamic fault-tolerant scheduling for real-time tasks in multicore processors," *Microprocessors Microsyst.*, vol. 38, no. 1, pp. 88–97, 2014.
[25] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Gener. Comput. Syst.*, vol. 22, no. 3, pp. 303–312, 2006.
[26] W. Tongquan, M. Piyush, W. Kaijie, and Z. Junlong, "Quasi-static fault-tolerant scheduling schemes for energy-efficient hard real-time systems," *J. Syst. Softw.*, vol. 85, no. 6, pp. 1386–1399, 2012.
[27] P. M. M. Shastry and K. Venkatesh, "Analysis of dependencies of checkpoint cost and checkpoint interval of fault tolerant MPI applications," *Analysis*, vol. 2, no. 8, pp. 2690–2697, 2010.
[28] Y. Robert, F. Vivien, and D. Zaidouni, "On the complexity of scheduling checkpoints for computational workflows," in *Proc. IEEE/IFIP 42nd Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2012, pp. 1–6.
[29] V. Bharadwaj, D. Ghose, and T. G. Robertazzi, "Divisible load theory: A new paradigm for load scheduling in distributed systems," *Cluster Comput.*, vol. 6, no. 1, pp. 7–17, 2003.
[30] B. Veeravalli, X. Li, and C. C. Ko, "On the influence of start-up costs in scheduling divisible loads on bus networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 12, pp. 1288–1305, Dec. 2000.
[31] S. Charcranoon, T. G. Robertazzi, and S. Luryi, "Parallel processor configuration design with processing/transmission costs," *IEEE Trans. Comput.*, vol. 49, no. 9, pp. 987–991, Sep. 2000.
[32] C. A. Moritz and M. I. Frank, "LoGPG: Modeling network contention in message-passing programs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 4, pp. 404–415, Apr. 2001.
[33] V. Bharadwaj, L. Xiaolin, and C. C. Ko, "Design and analysis of load distribution strategies with start-up costs in scheduling divisible loads on distributed networks," *Math. Comput. Model.*, vol. 32, nos. 7–8, pp. 901–932, 2000.
[34] S. W. Kwak and J.-M. Yang, "Probabilistic optimisation of checkpoint intervals for real-time multi-tasks," *Int. J. Syst. Sci.*, vol. 44, no. 4, pp. 595–603, 2013.
[35] N. Babar, Q. Kalim, and M. Paul, "Adaptive checkpointing strategy to tolerate faults in economy based grid," *J. Supercomput.*, vol. 50, no. 1, pp. 1–18, 2009.
[36] P. Paul, I. Viacheslav, E. Petru, and P. Zebo, "Design optimization of time- and cost-constrained fault-tolerant embedded systems with checkpointing and replication," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 3, pp. 389–402, Mar. 2009.
[37] M. Chtepen, F. H. A. Claeys, B. Dhoedt, F. D. Turck, P. Demeester, and P. A. Vanrolleghem, "Adaptive task checkpointing and replication: Toward efficient fault-tolerant grids," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 2, pp. 180–190, Feb. 2009.
[38] X. Wang, E. Aboutanios, and M. G. Amin, "Reduced-rank STAP for slow-moving target detection by antenna-pulse selection," *IEEE Signal Process. Lett.*, vol. 22, no. 8, pp. 1156–1160, Aug. 2015.

HEJUN XUAN received the B.Sc. degree in computer science and technology from Xinyang Normal University, China, in 2012, and the Ph.D. degree in computer software and theory from Xidian University, China, in 2018. He is currently with the School of Computer and Information Technology, Xinyang Normal University. His research interests include cloud/grid/cluster computing and scheduling in parallel and distributed systems.



SHIWEI WEI received the B.Sc. and M.Sc. degrees in computer science and technology from the Guilin University of Electronic Technology, China, in 2004 and 2007, respectively. He is currently an Associate Professor in computer and technology with the Guilin University of Aerospace Technology. His research interests include cloud computing and machine learning.



WUNING TONG received the B.Sc. and M.Sc. degrees in computer science and technology from the Shaanxi University of Science and Technology, China, in 2004 and 2007, respectively. She is currently an Associate Professor in science with the Shaanxi University of Chinese Medicine. Her research interests include cloud computing and machine learning.

DAOHUA LIU received the Ph.D. degree in computer science and technology from the Xi'an University of Architecture and Technology, China. He is currently a Professor with the School of Computer and Information Technology, Xinyang Normal University. His research interests include machine learning and image processing.

CHUANDA QI received the B.Sc. degree in mathematics and applied mathematics from Xinyang Normal University, China, and the Ph.D. degree in computer science and technology from PLA Information Engineering University, China. He is currently a Professor with the School of Computer and Information Technology, Xinyang Normal University. His research interests include machine learning and image processing.