Fault-Tolerant Scheduling Algorithm with Re-allocation for Divisible Loads on Homogeneous Distributed System

Wuning Tong, Song Xiao, Hongbin Li

***Abstract*—High performance computing is facing a major challenge due to its increasing failure rate. Fault tolerance needs to be used to ensure the efficient progress and correct termination of its applications when failures occur. In this paper, the divisible load fault-tolerant scheduling problem on the homogeneous distributed system is addressed, where communication is in the non-blocking message receiving mode, and both processors and communication links have the same speed and start-up overhead. First, the workload for each processor has been derived with the fault checkout overhead and checkout time consumption taken into account. Second, a checkout strategy which works better for divisible loads has been employed to reduce time consumption and checkout overhead. Third, an efficient algorithm for fault workload re-distribution has been proposed. Finally, some simulation experiments have been conducted. The result shows that the proposed algorithm is effective. It can minimize the expected execution time and reduce the time consumed by fault-tolerance.**

***Index Terms*—divisible loads, distributed system, re-allocation, fault tolerance**

I. Introduction

In recent years, efforts have been made to handle large computation problems(e.g. big data) by using distributed computing systems[1], [2]. In this novel field, researchers have been working to find an optimized algorithm to end the load of massive processors to minimize the span[3]. Divisible loads are parallel tasks which can be partitioned into fractions discretionarily, all of which can be processed irrelevantly. The Divisible Loads Theory(DLT) has been investigated in the past decades[4].More literature has focused on the application of Divisible Loads Scheduling(DLS) with high-performance computing to heterogeneous and homogeneous distributed systems[5], [6], [7], [8], [9], [10].Varied scheduling algorithms for homogeneous or heterogeneous systems with different network topologies or conditions have been derived to minimize the makespan [11], [12]. Heterogeneous systems with different computation and communication speeds have been used to solve the realistic computation problems. For heterogeneous star/tree networks, some expressions for optimal processing time were obtained by a variety of methods. Meanwhile, some researchers analyzed the effect of load distribution sequences on the processing time of the load. The optimal distribution sequence of the task scheduling algorithm was obtained in some literature[13], [14]. It is shown that the distribution order depends not on the computation speed of the processors but on the communication speed between processors in the distributed system. The order of communication speed reduction is the optimal sequence of load distribution. Shang[12] took communication and computation start-up overheads into account and proposed a more common and practical model for heterogeneous distributed systems depending on the non-blocking mode of communications. Meanwhile, it is shown that the start-up overhead and workload distribution sequence have an effect on the processing time. For the purpose of fault-tolerance, we take checkout start-up overhead and checkout time consumption into account in this paper. In order to simplify the problem, a closed-form expression for optimal processing time is obtained on a homogeneous distributed system. High performance computing is facing a major challenge due to its increasing failure rate[15]. Fault tolerance needs to be used to ensure the efficient progress and correct termination of its application when failures occur. A large number of fault-tolerant techniques have been developed[16], [17], [18], [19]. Several techniques have been developed with varying levels of granularity. There are two major methods: (1)Primary backup (PB), and (2)Checkpoint. Fault-tolerant scheduling is designed on the heterogeneous system[18], [20]. Mohammad proposed a dynamic fault tolerant scheduling method[21], the loads classified into critical and noncritical ones based on load utilization and the time it takes the scheduler to allocate resources to the incoming load. Noncritical loads are scheduled on a single processor, and checkpoint with rollback is applied to them when a failure occurs. These methods are all designed for independent real-time tasks. In other words, they are not suitable for divisible loads. There is a large amount of literature focusing on the checkpoint strategies for divisible loads. The corresponding scheduling is to divide the load into several chunks and do checkpointing after each of the chunks. Daly[22] studied periodic checkpoint strategies(chunks of the same size) for exponentially distributed failures and improved his study on the effect of approximate optimal checkpoint periods. Robert investigated the complexity of scheduling computational loads for exponentially distributed failures in literature[23]. If a failure occurs, rollback and recovery are adopted for re-execution to be done from the last checkpoint. Guillaume in[24] aimed to minimize energy consumption when executing a divisible workload within a bound in the total execution time, and he took a checkpoint at the end of the execution of each chunk. A large amount of evidence shows

Manuscript received July 19, 2017; revised October 21, 2017. This work is supported by National Natural Science Foundation of China (No.81102610, 81503667 and 81473559)

Wuning Tong is with the Shaanxi University of Chinese Medicine, Xianyang, Shaanxi 712046, China, e-mail: successful77@163.com.

Song Xiao is with the Key Research Laboratory of Radar Armament and Utilization Engineering, Air Force Early Warning Academy, Wuhan 430019, China, e-mail: 15392961067@163.com.

Hongbin Li is with the Affiliated Hospital of Shaanxi University of Chinese Medicine, Xianyang, Shaanxi 712046, China

that divisible loads scheduling is an efficient approach to high performance computing in distributed parallel systems. A variety of scheduling algorithms have been proposed and an optimal scheduling algorithm has been determined for homogeneous and heterogeneous distributed systems for divisible loads applications in the past decades. However, no work has been done on the divisible loads scheduling algorithm when it is needed to take checkout start-up overhead and checkout time consumption into account. Re-execution on the same processor is a common strategy for the fault-tolerant scheduling algorithm for divisible loads. However, it is not an optimal technique for divisible loads' fault-tolerance, since it will take a long time to re-execute the fault tasks on the idle processor or on a processor whose finish time is early. In order to minimize time consumption, we can distribute parts of the tasks to another processor. The major contributions of this study are summarized as follows:

1) For the homogeneous system, we have derived a closed-form expression for optimal processing time when checkout start-up overhead and checkout time consumption are considered.
2) We employ a checkout strategy that works for divisible loads. The checkout is not performed until all the loads are executed.
3) In order to minimize time consumption, we propose an optimal algorithm with a fault load unit re-allocation strategy.

The rest of this paper is organized as follows: Section 2 presents the mathematical model, derives a closed-form expression for optimal processing time when checkout start-up overhead and checkout time consumption are considered. A checkout strategy applied to divisible loads is given in Section 3. Section 4 demonstrates the optimal algorithm for fault load unit re-allocation. In Section 5, several numeric experiments and discussions are provided to verify our result. Finally, conclusions are drawn in Section 6.

II. Scheduling Model

In this paper, the platform considered is a homogeneous distributed system. Processors are connected in a star/tree topology, where p_0 is the master processor, and $p = \{P_1, P_2, \cdots, P_m\}$ are worker/slave processors. The master processor divides the load W_{total} into $n(n \le m)$ load fractions, denoted as $\alpha_1, \alpha_2, \cdots, and\ \alpha_n$, and distributes them among all the n-$professors$. Therefore, we have

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

where W_{total} is the total workload size. The processors start computing their load fractions upon receiving them. The problem is how to determine the optimal sizes of these load fractions that are distributed to the slave processors to minimize the makespan. In Table 1, some notations to be used throughout the paper are introduced. What follows is an essential condition used in related works in the DLT to derive the optimal solution [14]: to obtain an optimal processing time, it is necessary and sufficient to require that all the processors participating in the computation finish their computing simultaneously. The time diagram of divisible loads scheduling on homogeneous distributed systems is shown in Fig.1. Because the finish times of all the slave processors are equal, Eq.(2) can be obtained using $T_i^0 = T_{i+1}^0 (i = 1, 2, \cdots, n-1)$.

TABLE I
Notations.

Notations	Description
α	Fraction of the load assigned to processor.
l_i	Processor P_0 connected with processor P_i by l_i.
g	Time taken to communicate with a unit load.
w	Time taken to process a unit workload.
s	A constant additive computation start-up overhead.
o	Communication start-up overhead.
c	Checkout start-up overhead.
βw	The time taken to process a unit load, $\beta < 1$.
T_i^0	The finish time of checkout on processor P_i.
T_i^c	The checkout time consumption on processor P_i.
T_i^1	The time from T_i^0 to the fault-tolerant finish time of P_i.

What follows is a primary principle used in earlier studies in DLT to derive an optimal solution [14]: in order to obtain an optimal processing time, it is necessary and sufficient to require that all the processors participating in the computation stop computing at the same time instantly. The time diagram of divisible loads scheduling on homogeneous distributed systems is shown in Fig.1.

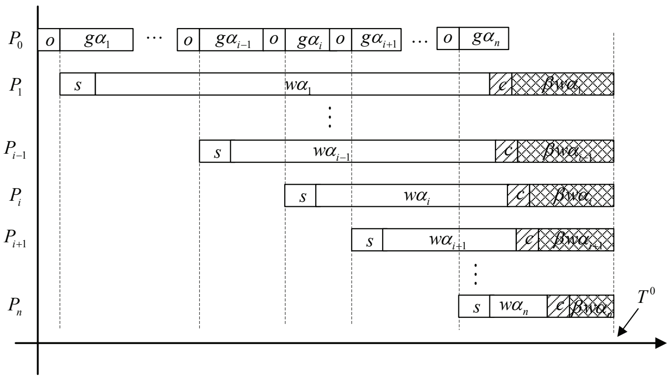


Fig. 1. Optimal timing diagram of DLS in a particular sequence.

Because the finish times (checkout finish time) of all processors are equal, Eq.(2) can be obtained using $T_i^0 = T_{i+1}^0$.

$$s + w\alpha_i + c + \beta w\alpha_i = g\alpha_i + s + w\alpha_{i+1} + c + \beta w\alpha_{i+1} \tag{2}$$

where $i = 1, 2, \cdots, n-1$. Using a similar approach in literature[12], [25], we obtain α_i as

$$\alpha_i = \mu_i \alpha_1 + \lambda_i \tag{3}$$

where

$$\alpha_1 = \frac{W_{total} - \sum_{k=2}^{n} \lambda_k}{1 + \sum_{k=2}^{n} \mu_k}, \tag{4}$$

$$\mu_i = \prod_{k=2}^{i} \left(\frac{-o}{(1+\beta)w} \right), \tag{5}$$

$$\lambda_i = \sum_{k=2}^{i} \left(\left(\frac{(1+\beta)w - g}{(1+\beta)w} \right) \prod_{j=k+1}^{i} \left(\frac{-o}{(1+\beta)w} \right) \right). \tag{6}$$

Thus, the closed-form expression of the checkout finish time T^0 is given by Eq.(7)

$$\begin{aligned} T^0 &= T_1^0 \\ &= o + s + c + (1+\beta)w\alpha_1 \\ &= o + s + c + (1+\beta)w\frac{W_{\text{total}} - \sum_{k=2}^{n}\lambda_k}{1 + \sum_{k=2}^{n}\mu_k} \end{aligned} \quad (7)$$

III. The checkout strategy for divisible loads

In order to detect the tasks carried out incorrectly, a strategy of checkpointing is adopted. As a technique for improving the reliability and availability of fault-tolerant computing systems, it has been an active area of research in the fault-tolerance aware task scheduling system. It is designed mostly for the real-time system[23], [26]. There are many related researches, but how to select the interval between the checkpointings remains unsolved. When a computing error occurs in some task interval it will be re-executed from the last checkpointing. If the interval between two checkpointings is bigger, the task re-execution time will be increased. On the other hand, if it is smaller, the checkout will start several times and the checkout overhead will be increased. Other researches on divisible loads focus mainly on the failure, which satisfies a certain distribution[23]. In this paper, a checkout strategy which works for divisible loads is employed and it does not depend on the distribution model of computing failures.

For divisible loads, tasks are independent of one another. We can check the task when the task execution is finished, as is shown in Fig.1. Therefore, the checkout time consumption is computed using Eq.(8).

$$T_i^c = c + \beta w\alpha_i. \quad (8)$$

where T_i^c is the checkout time consumption on Processor P_i. It should be noted that we do not consider the distribution model of computing failures since divisible loads do not depend on other tasks. To decrease the checkout time consumption, the checkout starts only once and the checkout is applied to the load unit. It does not divide the workload into several units. When a computing error occurs in some units, it will be marked as a fault load unit and re-executed or re-allocated when checkout finishes.

IV. Fault-tolerant task units re-allocation

A. The necessity for re-allocation

In previous work[23], fault-tolerant scheduling algorithms for divisible loads re-executed the tasks incorrectly on the original processor when some failures occurred. The fault-tolerant mechanism we have proposed can re-execute the units correctly. This indicates that the previous algorithms need to be further improved. Assume such a scenario: if a processor has several failure task units, it will take a long time to re-execute the failure load units. Meanwhile, some processors have few failure task units and their re-execution time is shorter. Thus, some processors are idle when the others are busy in re-executing the failure task units. In this case, the total time is bound to increase. The task units can be re-executed not only on the original processor but also on

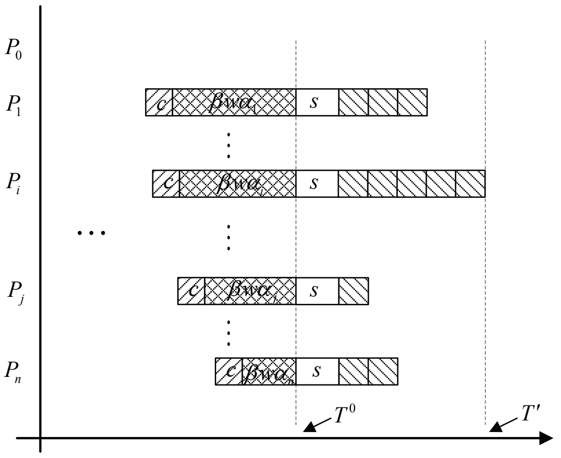


Fig. 2. The diagram of fault task units re-execution on original processor.

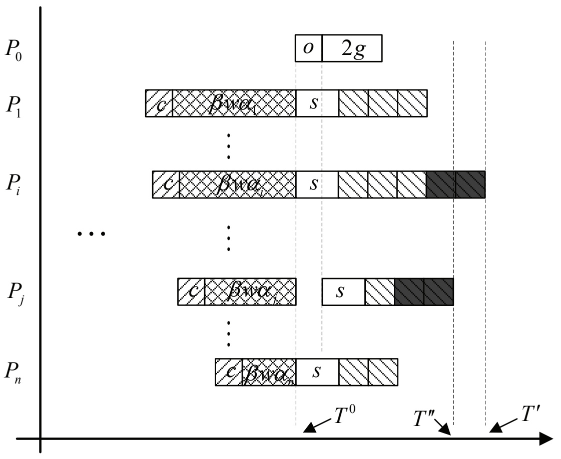


Fig. 3. The diagram of fault task units re-allocation.

other processors in order to minimize the total time. To do so, we selected two processors, one having failure task units and the maximum re-execution time, and another having the minimum sum of start-up overheads and computing speeds. Then, we re-allocated parts of the failure task units to the processor whose sum of start-up overheads and re-execution time was a minimum.

As is shown in Fig.2, Processor P_i has five fault task units and a small computing speed, so it has the longest finish re-execution time of all the processors. It will make execution of all the workloads continue for a long time and increase the total time. In this case, suppose the finish time of the system is $T^{'}$. Processor P_j has only one fault task unit and a much higher computing speed than Processor P_i. It has the shortest re-execution time of all the processors. It is reasonable to re-allocate parts of the fault task units of Processor P_i to Processor P_j. The master processor can transfer 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's finish time will be decreased. As shown in Fig.3, the finish time of the system is $T^{''}$. Obviously, $T^{''}$ is much smaller than $T^{'}$.

B. The principle of fault task units re-allocation

From the previous analysis, we know that task units re-allocation can decrease the system's execution time. However, there are three problems that need addressing. 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 should be re-allocated. The last is how many fault task units should be re-allocated to the target processor. In this section, these problems will be solved.

1) The source processor selected: The purpose of fault task units re-allocation is to minimize the time consumption of the system. The objective is to minimize the makespan, i.e., the processing time of the entire load. Let T denote the makespan, then

$$T = \max_{1\le i\le n}\left\{T^0 + s + w \times N_i^F\right\} \tag{9}$$

Eq.(9) shows that the makespan is the finish time of the processor which has the maximum processing time. Therefore, if we wish to minimize the makespan, we must minimize the maximum processing time of the processor. The makespan of the load can be reduced as long as the maximum processing time is reduced. According to this principle, the processor with the maximum processing time is selected as the source processor. Processor P_{sour} is selected as the source processor when sour satisfies Eq.(10) as follows.

$$sour = arg\max_{1\le i\le n}\left\{T^0 + s + w \times N_i^F\right\}, \tag{10}$$

where N_i^F is the number of fault task units on Processor $P_i(i = 1, 2, \cdots, n)$, n being the number of the processors required in computation.

2) The target processor selected and the number of task units re-allocated: The processor which has the maximum processing time among all the processors is selected as the source processor. Since our research is on the homogeneous distributed system, the processors have the same start-up overhead, communication, checkout and computing speed. Therefore, the selection of the target processor does not depend on the start-up overhead or the speed of the processors. The processor that has the minimum re-execution time should be selected as the target processor. We determine the target processor according to Eq.(11) as follows and P_{targ} is the target processor selected.

$$targ = arg\min_{1\le i\le m}\left\{j \times o + s + w \times N_i^F\right\} \tag{11}$$

where $j\times o$ is the sum of the start-up overheads of the j^{th} re-allocation, N_i^F is the number of fault task units on Processor P_i, and m is the number of processors in the distributed system.

How to determine the number of fault task units re-allocated to the target processor is another critical issue to be solved. From Eq.(9), we know that we must minimize the maximum time of the source processor and the finish time of the target processor after transferring the fault units from the source processor to the target processor. The number of fault task units that should be re-allocated to the target processor can be obtained by solving Eq.(12).

$$\begin{cases} \min\limits_{x} f(x) = \min\{\max\{s + w \times (N_{sour}^F - x), \\ \qquad\qquad j \times o + s + w \times (N_{targ}^F + x)\}\} \\ s.t. \\ \quad 1 \le x \le N_{sour}^F \\ \quad x \in Z \end{cases} \tag{12}$$

where $j \times o$ is the sum of the start-up overheads of the last but $(j-1)$ installment while re-allocation N_{sour}^F and N_{targ}^F are the number of fault task units on the source processor P_{sour} and target processor P_{targ}, respectively.

Theorem 4.1: In the homogeneous distributed system, to minimize the maximum processing time of the source and target processor, the number of fault task units x re-allocated from the source processor to the target processor must satisfy $x \in [\eta-1, \eta+1]$ and $x \in Z$, where $\eta = \frac{(N_{sour}^F - N_{targ}^F)}{2} - \frac{j\times o}{2w}$.

Proof: In order to minimize the makespan of the workload, we should minimize the maximum processing time of the source and target processor. Therefore, the source and target processor should terminate their re-execution simultaneously after a re-allocation, and the optimal finish time is $\left(w \times \left(N_{sour}^F + N_{targ}^F\right) + j \times o\right)/2 + s$. Since the start-up overhead exists, the finish time is not equal to but approximates the optimal finish time. What is more, assuming $s+w\times(N_{sour}^F - x) \ge j\times o + s + w\times(N_{targ}^F + x)$, Eq(13) is satisfied. That is to say, the finish time of the source processor is less than the sum of the optimal finish time and the time to process a task unit.

$$s + w \times (N_{sour}^F - x) \le \delta + s + w \tag{13}$$

$$j \times o + s + w \times (N_{targ}^F + x) \ge \delta + s - w \tag{14}$$

where $\delta = \frac{w\times(N_{sour}^F + N_{targ}^F) + j\times o}{2}$. The processing time of the target processor satisfies Eq.(14). In other words, the finish time of the target processor is greater than the difference between the optimal finish time and the time to process a task unit. According to Eq.(13) and Eq.(14), we have

$$\eta - 1 \le x \le \eta + 1 \tag{15}$$

From what has been discussed above, Theorem 4.1 is proved. ■

From Theorem 4.1, we can re-write Eq.(12) as Eq.(16). The optimal value of the number of fault task units re-allocated can be obtained quickly by solving the optimal model shown in Eq.(16).

$$\begin{cases} \min_x f(x) = \min\{\max\{s + w \times (N_{sour}^F - x), \\ \qquad\qquad j \times o + s + \omega \times (N_{targ}^F + x)\}\} \\ s.t. \\ \quad \eta - 1 \le x \le \eta + 1 \\ \quad x \in Z \end{cases} \tag{16}$$

Theorem 4.2: T^* is the makespan of fault-tolerant scheduling with the re-execution strategy for the former processor. T^{**} is the makespan of fault-tolerant scheduling with the re-allocation strategy. $T^{**} \le T^*$ is obtained.

Proof:

1) When the first re-allocation is executed and the number of fault load units re-allocated $x = 0$, that means $\forall x \neq$

0 cannot satisfy In-equation (17). Therefore, there is no need for re-allocation, and $T^{**} = T^{*}$.

$$\max\left\{T^1_{sour} - w \times x, T^1_{targ} + o + w \times x\right\} < T^1_{sour} \tag{17}$$

where T_i^1 is the time from T_i^0 to the fault-tolerant finish time of P_i and $i \in \{sour, targ\}$.

2) When the first re-allocation is executed and the number of fault load units re-allocated $x \neq 0$, that means $\exists x \neq 0$ satisfies In-equation (17). Therefore $T^{**} < T^{*}$.

3) From(2), we know that as long as one installment of re-allocations is implemented successfully, $T^{**} < T^{*}$ is satisfied. Therefore, if $j > 1$ (j being the number of successful re-allocations), In-equation (17) can be obtained.

$$T^{**} = T^{**}_j < T^{**}_{j-1} < \cdots < T^{**}_1 < T^{*} \tag{18}$$

where T^{**}_k is the makespan of the $k^{th}(1 \leq k \leq j)$ installment of re-allocations implemented successfully, and j is the number of successful re-allocations. From what has been discussed above, $T^{**} < T^{*}$ is proved.

■

C. Fault-tolerant scheduling algorithm

From the analysis above, if the fault task units are not re-allocated to other processors, the utilization of the processors will be decreased and the makespan of the system will be increased. Since the re-execution time of the source processor is much longer than that of the target processor, the method of fault task units re-allocation is designed to decrease the makespan and increase the utilization of the processors. When the fault task units are re-allocated to the target processor, they will be processed on the processor. The pseudocode of the fault-tolerant scheduling algorithm with task re-allocation for divisible loads scheduling (FTR_DLS) in heterogeneous computing systems is outlined in Algorithm 1.

Algorithm 1: Fault Tolerance with Task Re-allocation for DLS(FTR_DLS).

```
   Input: $o$, $s$, $g$, $w$, $c$, $\beta$
   Output: $Makespan$
1  Task scheduling according to the decrease of $g_i$ and computing $T^0$ using Eq.(7).
2  Initialization:$reallocated\_flag = 1$;$j = 1, K = \phi, sum\_overheads = 0$;
3  Computing $max\_Loc$,$min\_Loc$ and $num\_transfer$ using Eq.(10), Eq.(11) and Eq.(16)
4  while $reallocated\_flag == 1$ do
5      $current\_reallocated\_flag == 1$;
6      if $num\_transfer == 0$ then
7          $current\_reallocated\_flag = 0$;
8      else
9          update the $T^1_{max\_Loc}$ and $N^F_{min\_Loc}$
10         if $min\_Loc \in K$ then
11             $sum\_overheads1 = j \times o$; $flag = 0$;
               $sum\_overheads = m \times o$; %
               $m$ is the position in K.
12         else
13             $sum\_overheads = sum\_overheads + s$;
               $K(1, j) = min\_Loc$;
               $sum\_overheads1 = sum\_overheads$;
               $flag = 1, j = j + 1$;
14         end
15         updating $T^1_{min\_Loc}$ and computing $max\_Loc$ using Eq.(10);
16         Computing $min\_Loc$ and $num\_transfer$ by solving Eq.(11) and Eq.(16);
17         $current\_reallocated\_flag = 1$;
18     end
19     $reallocated\_flag = current\_reallocated\_flag$;
20 end
21 $Makespan = \max_i\{T^0 + T^1_i\}$;
```

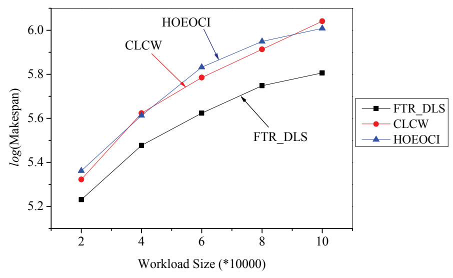
V. Experiments and Analysis

A. Experiments

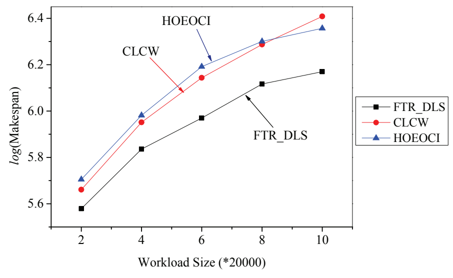
1) Comparison of Experiments: In this subsection, some experiments are compared. The experiments were carried out on the personal computer of HP with Intel(R) Core(TM) i7 CPU, 8G RAM and a 64-bit OS. The experimental parameters of the homogeneous distributed system in our simulation studies were generated randomly, for which please refer to literature[12]. Literature[12] does not provide the ratio of computing speed to checkout speed. Some data were generated randomly and added. Since the failure rate is an exponential distribution in literature[23], it is also an exponential distribution in the experiments compared.

Literature[24] proposes an algorithm (HOEOCI) for minimizing energy consumption when a divisible workload is executed within a bound in the total execution time. Failures may occur in the execution of a load. Re-execution of the fault load units is done only on the former processor, and not on other processors[24]. Therefore, the makespan is greater than that when the re-allocation strategy is employed. Literature[23] 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. The goal is to minimize the expected execution time(makespan). However, the fault loads will be re-executed on the former processor, and we can see that the makespan is greater than that by the re-allocation strategy employed by Theorem 1. For divisible loads, tasks are independent of one another. We check out the task when the task execution is finished, which also decreases the checkout time consumption. Therefore, the FTR_DLS algorithm has more advantages over the other algorithms in decreasing execution time. Figs. 4(a) to (f) show the makespan of several different workloads.

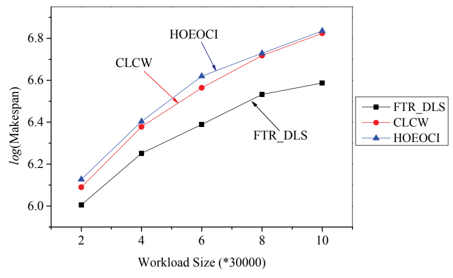
To evaluate the stability of the proposed algorithm and the compared algorithms, we give the statistical results (Mean and Variance) in the experiments with the different experimental scenes. Table II shows the mean and variance results of the makespan with different scenes. In the statistical results, 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



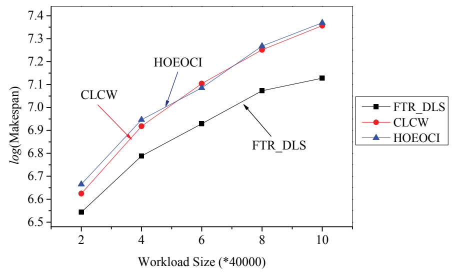
(a)



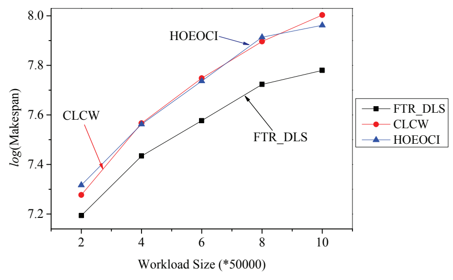
(b)



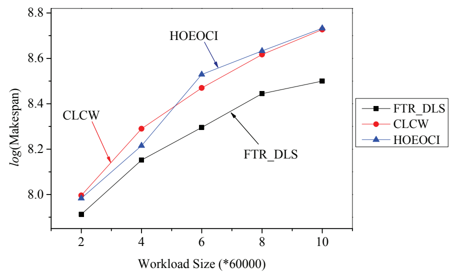
(c)



(d)



(e)



(f)

Fig. 4. The makespan of FTR_DLS, CSCW and HOEOCI.

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 II
STATISTICAL RESULTS (MEAN AND VARIANCE) OF THE MAKESPAN.

χ	ν	FTR_DLS	CLCW	HOEOCI
10000	2	5.2304 (2.58E-2)	5.3222 (3.13E-2)	5.3617 (3.24E-2)
	4	5.4771 (2.98E-2)	5.6232 (3.45E-2)	5.6128 (3.64E-2)
	6	5.6232 (3.37E-2)	5.7853 (3.95E-2)	5.8325 (4.02E-2)
	8	5.7482 (3.92E-2)	5.9138 (4.31E-2)	5.9494 (4.68E-2)
	10	5.8062 (4.36E-2)	6.0414 (4.81E-2)	6.0086 (4.79E-2)
20000	2	5.5788 (4.67E-2)	5.6610 (5.24E-2)	5.7051 (5.19E-2)
	4	5.8357 (4.68E-2)	5.9515 (5.34E-2)	5.9818 (5.42E-2)
	6	5.9697 (4.85E-2)	6.1438 (5.47E-2)	6.1919 (5.64E-2)
	8	6.1167 (4.99E-2)	6.2874 (5.85E-2)	6.3008 (5.79E-2)
	10	6.1702 (5.12E-2)	6.4086 (6.07E-2)	6.3572 (6.13E-2)
30000	2	6.0054 (5.58E-2)	6.0895 (6.48E-2)	6.1274 (6.34E-2)
	4	6.2512 (5.84E-2)	6.3779 (6.59E-2)	6.4031 (5.67E-2)
	6	6.3893 (5.93E-2)	6.5640 (6.79E-2)	6.6195 (6.81E-2)
	8	6.5325 (6.07E-2)	6.7179 (6.90E-2)	6.7289 (6.93E-2)
	10	6.5868 (6.37E-2)	6.8241 (7.09E-2)	6.8353 (7.15E-2)
40000	2	6.5430 (6.59E-2)	6.6244 (7.31E-2)	6.6651 (7.29E-2)
	4	6.7883 (6.71E-2)	6.9179 (7.54E-2)	6.9467 (7.66E-2)
	6	6.9289 (6.89E-2)	7.1037 (7.84E-2)	7.0853 (7.78E-2)
	8	7.0729 (7.06E-2)	7.2514 (8.14E-2)	7.2677 (8.07E-2)
	10	7.1277 (7.37E-2)	7.3571 (8.54E-2)	7.3696 (8.60E-2)
50000	2	7.1939 (7.60E-2)	7.2773 (8.91E-2)	7.3168 (8.86E-2)
	4	7.4343 (7.89E-2)	7.5668 (9.16E-2)	7.5627 (9.08E-2)
	6	7.5774 (8.14E-2)	7.7486 (9.42E-2)	7.7367 (9.39E-2)
	8	7.7232 (8.37E-2)	7.8964 (9.86E-2)	7.9136 (9.79E-2)
	10	7.7799 (8.96E-2)	8.0031 (1.09E-1)	7.9622 (1.08E-1)
60000	2	7.9128 (9.45E-2)	7.9962 (1.22E-1)	7.9835 (1.19E-1)
	4	8.1520 (9.86E-2)	8.2897 (1.38E-1)	8.2149 (1.33E-1)
	6	8.2955 (1.02E-2)	8.4694 (1.45E-2)	8.5290 (1.50E-2)
	8	8.4443 (1.09E-1)	8.6170 (1.59E-1)	8.6328 (1.62E-1)
	10	8.4999 (1.23E-1)	8.7267 (1.67E-1)	8.7329 (1.70E-1)

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 homogeneous distributed system given were generated randomly, for which please refer to literature[12]. To study the influence of the failure rate on Performance Improvement Ratio (PIR), several groups of experiments with different probabilities of failure were conducted. In this paper, the failure rate in every group of experiments was generated randomly among 0.5%-1%, 1%-2%, 2%-3%, 3%-4%, and 4%-5%, respectively.

To show the advantage of fault task units re-allocation, a definition of PIR is used. PIR is computed using Eq.(19) as follows

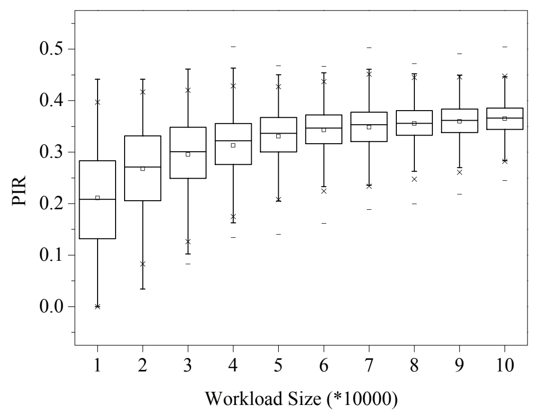
$$PIR = \frac{T_0^c - T_1^c}{T_0^c}. \tag{19}$$

where T_0^c is the fault-tolerant time from T^0 to $T^{'}$ in Fig.3 without the use of the re-allocation strategy, T_1^c is fault-tolerant time from T^0 to $T^{''}$ in Fig.3 with the re-allocation strategy employed.

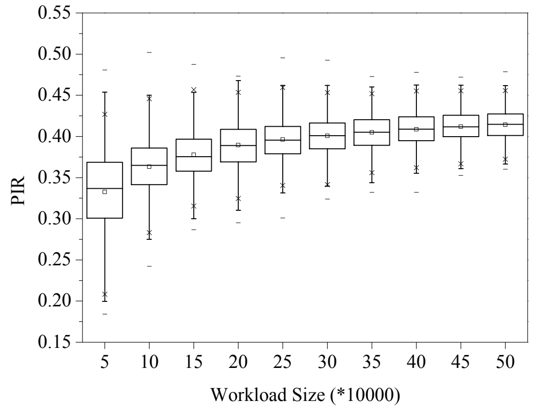
Figs.5(a), (b), (c) and (d) show the statistical results of PIR when the workload size ranges from 10^4 to 10^8, and the failure rate was generated randomly in 1%-2%. Some representative workload sizes were simulated and every workload size was re-executed 1000 times. Fig.6 shows the variation of the mean of PIR when the workload size ranges from 10^4 to 10^8 with different probabilities of failure.

B. *Experiment analysis*

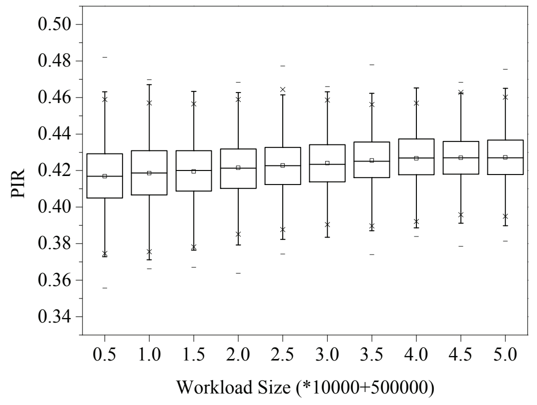
PIR increases monotonically because the number of fault load units is approximately equal to the expected number and the condition of re-allocation is easy to satisfy due to the increase of the workload size. Therefore, the variation of PIR is stable and increases slowly as shown in Figs.5(a), (b),



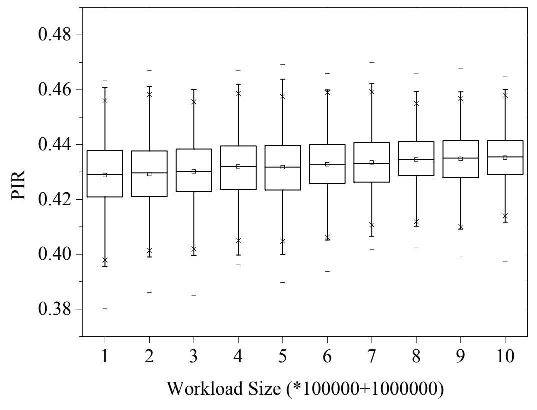
(a)



(b)



(c)



(d)

Fig. 5. The variation of PIR with the workload size.

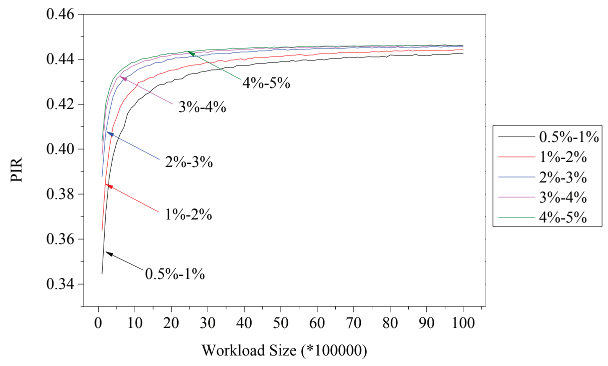


Fig. 6. The variation of PIR with the workload size ranges.

(c) and (d). Fig.6 shows the variation of the mean of PIR with the workload size and the failure rate. We can see that the mean of PIR reaches 44% when the load size is large enough. That is to say, fault-tolerant scheduling with the re-allocation strategy can save 44% of the time consumed by fault tolerance, compared with re-execution on the original processor without re-allocation. When the workload size is large enough, we can re-write Eq.(19) as Eq.(20).

$$PIR = \frac{\max\limits_{1\le i\le n}\{p\alpha_i\} - \left(p\overline{\alpha} + \sum\limits_{i=1}^{n}(c+s)\right)}{\max\limits_{1\le i\le n}\{p\alpha_i\}} \tag{20}$$

where $\overline{\alpha} = W_{total}/n$. When the workload size is large enough, $\theta \to 0$, where

$$\theta = \frac{\sum\limits_{i=1}^{n}(c+s)}{\max\limits_{1\le i\le n}\{p\alpha_i\}}, \tag{21}$$

then, Eq.(22) can be obtained.

$$\begin{aligned} PIR &\approx \frac{\max\limits_{1\le i\le n}\{p\alpha_i\} - p\overline{\alpha}}{\max\limits_{1\le i\le n}\{p\alpha_i\}} \\ &= \frac{p\alpha_{max} - p\overline{\alpha}}{p\alpha_{max}} \\ &= 1 - \frac{w_{total}}{n\alpha_{max}} = 1 - \frac{w_{total}}{n\alpha_1} \\ &= 1 - \frac{w_{total}\left(1+\sum_{k=2}^{n}\mu_k\right)}{n\left(w_{total}-\sum_{k=2}^{n}\lambda_k\right)} \end{aligned} \tag{22}$$

From Eq.(22), we know that the PIR does not depend on the failure rate. The PIR reaches the same value and is close to a constant as shown in Fig.(6) when the workload size is large enough.

VI. Conclusion

This paper aims to find an optimal fault-tolerant scheduling method for divisible loads in heterogeneous distributed systems. We have successfully achieved the aim by designing a scheduling algorithm with a fault task units re-allocation strategy. First, we derived a closed-form expression for the optimal processing time and optimal scheduling sequence. Second, we employed a checkout method which works for divisible loads. 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, we conducted a set of experiments. From the experimental results, we can see that fault-tolerant scheduling with the re-allocation strategy can save some time consumed by fault tolerance, as compared with re-execution on the original processor without re-allocation.

ACKNOWLEDGMENT

This work was supported by National Natural Science Foundation of China (No.81102610, 81503667 and 81473559).

REFERENCES

[1] G. Sanchez, E. Leal, and N. Leal, "A linear programming approach for 3d point cloud simplification," *IAENG International Journal of Computer Science*, vol. 44, no. 1, pp. 60–67, 2017.

[2] M. Kolar, M. Benes, D. Sevcovic, and J. Kratochvil, "Mathematical model and computational studies of discrete dislocation dynamics," *IAENG International Journal of Applied Mathematics*, vol. 45, no. 3, pp. 198–207, 2015.

[3] A. Shokripour, M. Othman, H. Ibrahim, and S. Subramaniam, "New method for scheduling heterogeneous multi-installment systems," *Future Generation Computer Systems*, vol. 28, no. 8, pp. 1205 – 1216, 2012.

[4] T. G. Robertazzi, "Ten reasons to use divisible load theory," *Computer*, vol. 36, no. 5, pp. 63–68, May 2003.

[5] Z. Zhang and T. G. Robertazzi, "Scheduling divisible loads in gaussian, mesh and torus network of processors," *IEEE Transactions on Computers*, vol. 64, no. 11, pp. 3249–3264, 2015.

[6] K. Wang and T. G. Robertazzi, "Scheduling divisible loads with nonlinear communication time," *IEEE Transactions on Aerospace & Electronic Systems*, vol. 51, no. 3, pp. 2479–2485, 2015.

[7] C. Y. Chen and C. P. Chu, "A novel computational model for non-linear divisible loads on a linear network," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 53–65, 2016.

[8] S. Vadde and S. Ganesan, "Effect of fault in single load distribution with fifo(first in, first out) back propagation of results," in *IEEE International Conference on Electro Information Technology*, 2016, pp. 0804–0810.

[9] L. Dai, Z. Shen, T. Chen, and Y. Chang, "Analysis and modeling of task scheduling in wireless sensor network based on divisible load theory," *International Journal of Communication Systems*, vol. 27, no. 5, pp. 721–731, 2014.

[10] O. Beaumont, L. Eyraud-Dubois, H. Rejeb, and C. Thraves, "Heterogeneous resource allocation under degree constraints," *IEEE Transactions on Parallel & Distributed Systems*, vol. 24, no. 5, pp. 926–937, 2013.

[11] H. J. Kim, "A novel optimal load distribution algorithm for divisible loads," *Cluster Computing*, vol. 6, no. 1, pp. 41–46, 2003.

[12] Mingsheng and Shang, "Optimal algorithm for scheduling large divisible workload on heterogeneous system," *Applied Mathematical Modelling*, vol. 32, no. 9, pp. 1682–1695, 2008.

[13] V. Bharadwaj, D. Ghose, and V. Mani, "Optimal sequencing and arrangement in distributed single-level tree networks with communication delays," *IEEE Transactions on Parallel & Distributed Systems*, vol. 5, no. 9, pp. 968–976, 1994.

[14] H. J. Kim and V. Mani, "Divisible load scheduling in single-level tree networks: Optimal sequencing and arrangement in the nonblocking mode of communication," *Computers & Mathematics with Applications*, vol. 46, no. 10, pp. 1611–1623, 2003.

[15] J. Dongarra, P. Beckman, P. Aerts, F. Cappello, T. Lippert, S. Matsuoka, P. Messina, T. Moore, R. Stevens, and A. Trefethen, "The international exascale software project: a call to cooperative action by the global high-performance community," *International Journal of High Performance Computing Applications*, vol. 23, no. 4, pp. 309–322, 2009.

[16] X. Zhu, X. Qin, and M. Qiu, "Qos-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters," *IEEE Transactions on Computers*, vol. 60, no. 6, pp. 800–812, 2011.

[17] B. Javadi, P. Thulasiraman, and R. Buyya, "Enhancing performance of failure-prone clusters by adaptive provisioning of cloud resources," *The Journal of Supercomputing*, vol. 63, no. 2, pp. 467–489, Feb 2013.

[18] J. Balasangameshwara and N. Raju, "Performance-driven load balancing with a primary-backup approach for computational grids with low communication cost and replication cost," *IEEE Transactions on Computers*, vol. 62, no. 5, pp. 990–1003, 2013.

[19] B. Nazir, K. Qureshi, and P. Manuel, "Replication based fault tolerant job scheduling strategy for economy driven grid," *Journal of Supercomputing*, vol. 62, no. 2, pp. 855–873, 2012.

[20] W. Sun, C. Yu, X. Dfago, and Y. Inoguchi, "Dynamic scheduling realtime task using primary-backup overloading strategy for multiprocessor systems," *Ieice Transactions on Information & Systems*, vol. 91-D, no. 3, pp. 796–806, 2008.

[21] M. H. Mottaghi and H. R. Zarandi, "Dfts: A dynamic fault-tolerant scheduling for real-time tasks in multicore processors," *Microprocessors & Microsystems*, vol. 38, no. 1, pp. 88–97, 2014.

[22] J. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 303 – 312, 2006.

[23] Y. Robert, F. Vivien, and D. Zaidouni, "On the complexity of scheduling checkpoints for computational workflows," in *Ieee/ifip International Conference on Dependable Systems and Networks Workshops*, 2012, pp. 1–6.

[24] G. Aupy, A. Benoit, R. Melhem, and P. Renaud-Goud, "Energy-aware checkpointing of divisible tasks with soft or hard deadlines," in *Green Computing Conference*, 2013, pp. 1–8.

[25] M. Wang, X. Wang, K. Meng, and Y. Wang, "New model and genetic algorithm for divisible load scheduling in heterogeneous distributed systems," *International Journal of Pattern Recognition & Artificial Intelligence*, vol. 27, no. 07, pp. 1 359 005–, 2013.

[26] J. M. Yang, "Probabilistic optimisation of checkpoint intervals for realtime multi-tasks," *International Journal of Systems Science*, vol. 44, no. 4, pp. 595–603, 2013.