Vol.2010-AL-129 No.9
2010/3/5

A reliability analysis based scheduling algorithm in heterogeneous system

Laiping Zhao †1 and Kouichi Sakurai†2

In heterogeneous systems, if under active replication schema, tolerating t failures often needs t+1 processors. However, using so many replicas for one task is wasting resources, especially when the reliability of processors is high. In this paper, we propose a new scheduling algorithm based on the reliability analysis of each processor. For each task in a workflow, a dynamic number of replicas will be scheduled simultaneously. The experiments show that our scheduling algorithm uses much less resources than original active replication schema.

1. Introduction

1.1 Background

Generally, faults can be categorized into four types: crash faults, omission faults, timing faults and byzantine faults. For the crash faults, the component either completely stops operating or never returns to a valid state. for omission faults, the component fails to perform its service; for timing faults, the component does not complete its service on time; and byzantine faults are faults of an arbitrary nature.

To tolerant the failures caused by the crash faults, active replication and backup/restart[1] are the most widely used schemas. The active replication schema exploits resource redundancy to tolerant failures: several processors are scheduled simultaneously, and the task will succeed at least one processor doesn't encounter a failure. The backup/restart schema exploits time redundancy to tolerant failures, the task is scheduled on one processor each time. If a failure occurs on the processor, the scheduling algorithm will schedule another new processor to replace the failing processor[2].

We consider and improve the active replication schema in this research, and incorporate it into the scheduling algorithm. The objective is to design a fault tolerant scheduling algorithm to satisfy the reliability requirement.

1.2 Motivation

Active replication schema is exploited in the algorithms discussed in[3)5)6)16)] and[17]. In order to tolerant ε failures, these algorithms have to schedule $\varepsilon + 1$ replicas for each task of the workflow. As we have discussed in the last part, this causes a large resource redundancy, which is quite a problem for the system especially when the resources are limited. How to achieve a higher reliability with minimum resources is quite a challenge for the scheduling algorithm.

1.3 Previous work

Reliability analysis based scheduling algorithms are addressed by many works. J.J. Dongarra et al.[8] designs algorithms that optimize both makespan and reliability. The first scheduling algorithm targets to maximize the reliability subject to makespan minimization. And the second one is based on the product *failure rate* × *unitary instruction execution time* to trade off between reliability maximization and makespan minimization. A. Dogan et al.[9] also develops matching and scheduling algorithms which account for both the execution time and the failure probability. A genetic algorithm based scheduling algorithm is developed to trade off the execution time and the reliability. In[10], MCMS and PRMS are developed to achieve the maximum system reliability while satisfying a given time constraint. S. Swaminathan et al.[11] proposes a reliability-aware value-based dynamic scheduling algorithm, which aims to maximize the overall PI of the system. M. Hakem et al.[14] presents the BSA scheduling algorithm, which takes into account not only the time makespan but also the failure probability of the application.

Primary and backup scheduling algorithm can tolerate one failure in the sys-

†1 PH.D. candidate, Department of Informatics, Kyushu University
zlp@itslab.csce.kyushu-u.ac.jp.
†2 Professor, Department of Informatics, Kyushu University
sakurai@csce.kyushu-u.ac.jp.

tems. Q. Zheng et al.[3)4)] consider the response time and replication cost in the scheduling process. In[12)], although the dynamic number of replicas are scheduled for each task, only one failure can be tolerated by the scheduling result. In order to reduce the schedule length, X. Qin et al.[13)] puts the emphasis on the conditions for the backup copies to safely overlap with each other, and proposes the eFRD scheduling algorithm. However obviously only tolerating one failure is far from enough for resource scheduling problem.

Considering crash failures, the active replication schema is combined with the scheduling algorithm in[5)] and[6)], and FTSA and CAFT scheduling algorithm are proposed respectively, where FTSA is a extended version of the classic HEFT algorithm[15)]. And CAFT put emphasis on the one-port communication model. In[16)], active replication and standby parallel replication strategies are exploited in managing the redundancy existing in each task. And based on the analysis on price, reliability and response time, algorithms are developed to meet user-specified QoS requirements. In[17)], Alain Girault et al. proposes the FTBAR scheduling algorithm, which produces automatically a static distributed fault-tolerant schedule of a given algorithm on distributed architecture. However, the high demand for resources is not considered by these algorithms.

1.4 Challenging issues

In a workflow based scheduling, if using the minimum resources to achieve the user required reliability, the number of replicas for each task should be as few as possible. How to decide the number of replicas for each task is one challenge. And even using minimum resources, the scheduling algorithm should also guarantee that user's required reliability is satisfied. This is another challenge. Moreover, while meeting the user's reliability requirement, the scheduling algorithm should not be much terrible on execution time performance, which is the third challenge.

1.5 Our contribution with comparing to related works

In our research, we analyze the reliability of processors, and schedule the dynamic number of replicas for each task. We design the MaxRe scheduling algorithm, which targets to maximize the reliability of scheduling results. To the best of our knowledge, the MaxRe algorithm is the first scheduling algorithm that combines active replication and reliability analysis together. The analysis of MaxRe scheduling algorithm shows that reliability achieved by MaxRe algorithm must not less than the user's required reliability.

In our simulation, we use four typical workflow examples to verify the algorithm. Experiments results show that:

(1) The reliability achieved by MaxRe algorithm certainly can satisfy user's requirement, which has already been proved in the analysis.

(2) Resource usage
While exhibiting the same reliability degree, it is obvious that the MaxRe scheduling algorithm use much less resources than FTSA algorithm. From 30 to almost 70 percentage resources are saved by the MaxRe algorithm.

(3) Performance on time
The experiments also observe the time performance of MaxRe algorithm's scheduling result. The latest finish time and earliest finish time are observed respectively. With compared with FTSA algorithm, the earliest finish time of MaxRe is only worse than FTSA algorithm for less than 10-20 percent. For the latest finish time, The MaxRe even exceeds the FTSA algorithm.

2. System model and problem statement

The processor model, job model and the system model is given below.

2.1 Processor model

Suppose the heterogeneous system consists of m processors:

$P = \{p_0, p_1, p_2, ..., p_{m-1}\}$

And for each processor $p_i (1 \leq i \leq m-1)$, the arrival of failures follows a Poisson distribution with λ_i, which is a positive real number, and equal to the expected number of occurrences of failures in unit time t. So the failure distribution in unit time t can be represented as:

$$f(k, \lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \quad (1)$$

where k is the number of occurrences of failures in unit time t.

In a heterogeneous system, processors may have different λ values: $\Lambda = \{\lambda_0, \lambda_1, \lambda_2, ..., \lambda_{m-1}\}$. Therefore, for the processor p_i, we get the failure distribution is: $F(k, \lambda_i) = \frac{\lambda_i^k e^{-\lambda_i}}{k!}$.

A processor may encounter a failure more likely when it is executing tasks. We assume that the processor is reliable while it is idle.

2.2 Job model

A job is represented as a weighted directed acyclic graph (DAG): $G = (V, E)$, where V is the set of nodes corresponding to the tasks, and E is the set of edges corresponding to the precedence relations between the tasks. Suppose: $V = \{\tau_0, \tau_1, \tau_2, ..., \tau_{n-1}\}$, $n = |V|$ is the number tasks. $e = |E|$ is the number of edges. The node without any predecessor is called an entry node, and the node without any successor node called an exit node. Task τ_i cannot start being executed before it received the output from its all its predecessors, and the result of task τ_i can be sent to its successor tasks after the task has been finished.

2.3 The system model and its some properties

In the system, suppose all processors are fully connected, each processor can communicate with every other processors. And the communication device is reliable, we do not consider the failures on communication device.

Proposition 1 *When submitting a DAG-based workflow to a heterogenous system, if each task in the workflow can be replicated and scheduled on multiple processors, the total number of scheduling methods is* $(2^m - 1)^n$.

proof: For each task in the workflow, for example: τ_i, it can be either scheduled on processor p_j or not. For all m processors, there are 2^m possibilities. Excluding the one possibility that the task τ_i is not scheduled on any processors, we have $2^m - 1$ possibilities to schedule this task. Therefore, for all the n tasks in the workflow, the total number of scheduling methods is $(2^m - 1)^n$. *((end proof))*

Definition 1 (Full-schedule) *Every task in the workflow has m replicas, which means every task is replicated and scheduled on all the m processors.*

The probability of no failures occur in time period T is: $f(k = 0, \lambda_i T) = \frac{\lambda_i^k e^{-\lambda_i T}}{k!} = e^{-\lambda_i T}$, So the reliability of the full-schedule is: $1 - \prod_{i=0}^{m-1} (1 - f_i(k = 0, \lambda_i T_i))$. To tolerant more failures, traditional research must schedule more replicas for every task. However, more replicas may not always provide higher reliability. In the extreme, a full schedule may not more reliable than a non-full schedule.

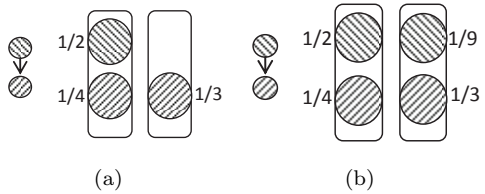


Fig. 1 More replicas do not always lead to a higher reliability: (a) reliability of less replicas is 0.25; (b) reliability of more replicas is 0.1574.

Proposition 2 *when submitting a DAG-based workflow to a heterogenous system, the more replicas for a task may not lead to a higher reliability.*

proof: This conclusion can be proved by an example. As shown in Fig.1, the workflow consists of two tasks: τ_0, τ_1, and the system consists of two processors: p_0, p_1. The workflow is scheduled to the system in two ways: Fig. 1(a) and Fig. 1(b), and the corresponding probability of no failures occur during each execution is as shown in the figure.

In the first situation (Fig. 1(a)), τ_0 is only scheduled on processor p_0, while τ_1 has two replicas, and is scheduled on both processors. The achieved reliability of this situation is: $Reliability(a) = \frac{1}{2} \times \left(1 - \left(1 - \frac{1}{4}\right) \times \left(1 - \frac{1}{3}\right)\right) = 0.25$.

In the second situation (Fig. 1(b)), both task τ_0 and τ_1 are replicated and scheduled on the two processors (full-schedule). The reliability of this situation is: $Reliability(b) = 1 - \left(1 - \frac{1}{2} \times \frac{1}{4}\right) \times \left(1 - \frac{1}{9} \times \frac{1}{3}\right) \approx 0.1574$.

From $0.1574 < 0.25$, we get that the more replicas for a task do not always lead to a higher reliability. *(end proof)*

Actually, Proposition 1 can be interpreted as: the more work that one proces-

sor executes, the greater risk is faced by the processor. Therefore, a full-schedule may not give a higher reliability.

2.4 Problem statement

Based on the job model, processor model and the heterogeneous system model, we analyze the fault distribution of each processor in the heterogeneous system, and seek a fault-tolerant scheduling algorithm, which employs dynamic number of replicas and targets to satisfy user's reliability requirement with the minimum processors.

3. The MaxRe scheduling algorithm

In this section, firstly, we analyze the task priority of all tasks in a workflow, and propose the MaxRe scheduling algorithm, which can satisfy user's required reliability. Secondly, we analyze the reliability of the MaxRe algorithm's result, and give a method to compute the exact reliability value of the scheduling result at last.

3.1 Task priority

The scheduling order of each task is identified by its priority values, which is called *upward rank*. The *upward rank* value of task τ_i is computed by Formula 2[15].

$$rank(\tau_i) = \overline{w_i} + \max_{\tau_j \in succ(\tau_i)} (\overline{c_{i,j}} + rank(\tau_j)) \tag{2}$$

Where $rank(\tau_i)$ is the priority value of task τ_i, $\overline{w_i}$ is the average computation cost of task τ_i, $succ(\tau_i)$ is the successor tasks set of task τ_i, and $\overline{c_{i,j}}$ is the average communication cost of from task τ_i to task τ_j. The $rank(\tau_i)$ can be gotten by traversing the task graph upward recursively.

3.2 The MaxRe scheduling algorithm

In many cases, a required reliability is specified by the user when submitting a job. For example, the required reliability of China ChangZheng II F rocket is 0.97. Suppose that the user required reliability is: $\Re$. Then for each task in the workflow, the required reliability is the n-geometric mean of $\Re$ (Formula 3):

$$r = \sqrt[n]{\Re} \tag{3}$$

Our objective is to design a scheduling algorithm which satisfies the $\Re$ requirement with minimum resource usage. To decide the processors that the task τ_i will be scheduled on, first we introduce the conception *Total time (TT)* and *Current reliability (CR)*:

Definition 2 (TT) *Total time:*

$$TT(p_j) = ET(\tau_i, p_j) + \sum_{\tau_k \in on(p_j)} ET(\tau_k, p_j) \tag{4}$$

where $ET(\tau_i, p_j)$ is the execution time when scheduling task τ_i to processor p_j. $on(p_j)$ is the previous tasks that have already been scheduled on processor p_j.

Definition 3 (CR) *$CR[p_j]$ is the probability that no failure occur on processor p_j during $TT[p_j]$ period. Because of the memoryless property of Poisson distribution, we also can get the current reliability value as belows:*

$$\begin{aligned} & CR(p_j) \\ &= e^{-\lambda_j TT[p_j]} \\ &= e^{-\lambda_j ET(\tau_i,p_j)} \times e^{-\lambda_j \sum_{\tau_k \in on(p_j)} ET(\tau_k,p_j)} \\ &= R(\tau_i, p_j) \times \prod_{\tau_k \in on(p_j)} R(\tau_k, p_j) \end{aligned} \tag{5}$$

Algorithm 1: The MaxRe Scheduling Algorithm

Require:

$G = (V, E)$, $\Re$, and $\Lambda = \{\lambda_1, \lambda_2, \lambda_3, ..., \lambda_m\}$.

Ensure:

To what processors the tasks will be scheduled.

1: **for** each task $\tau_i \in V$ **do**
2: **for** each processor $p_j \in P$ **do**
3: $ET(\tau_i, p_j) \leftarrow$ compute the execution time using $(\tau_i.load, p_j.speed)$;
4: $R(\tau_i, p_j) \leftarrow$ compute the reliability using $(ET(\tau_i, p_j), \lambda_j)$
5: **end for**

6: **end for**
7: $C \leftarrow$ compute the average communication time using (G, P);
8: $TP \leftarrow$ compute the priority value for all tasks using Formula 2;
9: $sort\,(V, TP)$; (Sort all tasks according to the task priority value)
10: $r \leftarrow root\,(\Re)$; (Compute the geometric mean of $\Re$ using Formula 3)
11: $\Theta = \emptyset$, $U = V$;
//Start scheduling
12: **while** $U \neq \emptyset$ **do**
13: $\tau_i = head(U)$;
14: **for** each processor $p_j \in P$ **do**
15: $TT\,(\tau_i, p_j) \leftarrow$ compute the total execution time using Formula 4;
16: $CR\,(\tau_i, p_j) \leftarrow (TT\,(\tau_i, p_j)\,, \lambda_j)$; (Compute the current reliability for each processor using Formula 5)
17: **end for**
18: $sort\,(P, CR)$; (Sort all processors according to $CR\,(\tau_i, p_j)$)
19: $\xi \leftarrow replica\_num\,(r, \tau_i, CR)$; (Compute the number of replicas)
20: $S \leftarrow$ select the first ξ maximum reliability processors from sorted P;
21: Schedule task τ_i on processors in S;
22: Put τ_i into Θ;
23: $U \leftarrow U \backslash \{\tau_i\}$;
24: **end while**

Algorithm 2: Decide the number of replicas for task τ_i: $\xi \leftarrow replica\_num\,(r, \tau_i, CR)$

Require:
r, τ_i, CR.

Ensure:
The number of replicas for task τ_i.
//Variable *counter* stores the number of replicas
1: $counter = 0$;
//Variable *fail* represents the probability of all scheduled processors failing
2: $fail = 1 - CR\,(\tau_i, p_0)$;
3: **while** $(1 - fail) < r \&\& counter < m$ **do**
4: $counter = counter + 1$;
5: $fail = fail \times (1 - CR\,(\tau_i, p_{counter}))$;
6: **end while**
7: return counter;

The MaxRe scheduling algorithm is given in Alg. 1. In lines 1-6, we computed the task τ_i's execution time $ET\,(\tau_i, p_j)$ when it is scheduled on processor p_j, and based on the execution time, we compute its corresponding probability $R\,(\tau_i, p_j)$ which represents the probability of no failures occur during the execution time period. In line 7, compute the communication time between two consecutive tasks. Based on the previous computed execution time set and the communication time, we get the task priority set TP in line 8 using Formula 2, and all the tasks will be sorted according to the task priority value in line 9. In line 10, get user's required reliability value $\Re$, and calculate the n-geometric mean of the reliability value. We use the n-geometric mean as the required reliability value for each task of the workflow.

In lines 11-24, we select a task from sorted set U, and schedule it onto some processors. First select the head task τ_i from U in line 13, then use Formula 4 and Formula 5 to compute the *total time* $TT\,(\tau_i, p_j)$ and the *current reliability* $(CR\,(\tau_i, p_j))$ for each processor. The $CR\,(\tau_i, p_j)$ will be used in algorithm $replica\_num\,(r, \tau_i, CR)$ (Alg. 2) to decide the number of replicas for task τ_i. In line 20-21, select the first ξ maximum $CR\,(\tau_i, p_j)$ value's processors, then schedule task τ_i to the ξ processors. Delete the task τ_i from U, and repeat this whole process until all tasks are scheduled.

The $replica\_num\,(r, \tau_i, CR)$ algorithm is given in Alg. 2. The processors with maximum reliability value will be selected to execute user's task (Line 2). And this process is repeated until the achieved reliability is not less than the n-geometric mean of $\Re$ (Lines 3-6). The number of replicas for task τ_i is ξ.

3.3 Analysis of the MaxRe

Proposition 3 *When the required number of replicas for each task does not exceed the total number of processors, suppose the reliability value provided by the MaxRe algorithm is Ψ, then we have $\Re \leq \Psi$.*

proof: From the description of the MaxRe algorithm, we have that the reliability for the task τ_i follows:

$$r \leq 1 - \prod_{k=0}^{k<\xi} (1 - CR(\tau_i, p_k)) \tag{6}$$

Let $F(\tau_i) = 1 - \prod_{k=0}^{k<\xi} (1 - CR(\tau_i, p_k))$, we have:

$$\Re = r^n \leq \prod_{i=0}^{n-1} F(\tau_i) \tag{7}$$

The next, we only need to prove that: $\prod_{i=0}^{n-1} F(\tau_i) \leq \Psi$.

In Formula 5, $CR(\tau_i, p_j)$ implies the probability that no failures occur on processor p_j from the beginning time of the first task that is scheduled on processor p_j, until the finish time of the task τ_i (if task τ_i is scheduled on processor p_j). So $F(\tau_i)$ is the probability that at least one scheduled processor does not encounter failures in the same period.

If randomly select two tasks (τ_i and τ_j) from the set V, we can get $F(\tau_i)$ and $F(\tau_j)$ respectively. And $F(\tau_i) \times F(\tau_j)$ implies the probability that both task τ_i and τ_j succeed in the same period. There are three cases to compute the probability:

(1) If both task τ_i and τ_j are scheduled on the totally different processors, the probability of both tasks' success is equal to $F(\tau_i)F(\tau_j)$.

(2) If both tasks are scheduled on the exactly the same processors and the task τ_j is later than the τ_i, the success probability would be equal to $F(\tau_j)$, following $F(\tau_j) > F(\tau_i)F(\tau_j)$.

(3) Without loss of generality assume that task τ_i is scheduled on processor p_0 and p_1, and task τ_j is scheduled on processor p_1 and p_2, then the probability of both success meet the following conditions: $1 - \prod_{k=0}^{k\leq 2} (1 - CR(\tau_i, p_k)) > (1 - \prod_{k=0}^{k\leq 1} (1 - CR(\tau_i, p_k))) \times (1 - \prod_{k=1}^{k\leq 2} (1 - CR(\tau_i, p_k)))$.

Therefore, we have:

$$\prod_{i=0}^{n-1} F(\tau_i) \leq \Psi \tag{8}$$

with equality holding if and only if when all tasks in set V are scheduled on total different processors. Above all, $\Re \leq \Upsilon$. *(end proof)*

Theorem 1 *The time complexity of the MaxRe scheduling algorithm is $O(MN \log M + N \log N + eN)$.*

proof: First, the time complexity of Alg. 2 is: $O(\xi)$, where ξ is the number of replicas, and $\xi \leq M$.

Second, in the MaxRe algorithm, from line 1 to line 6, the time complexity is $O(MN)$. And from line 7 to line 8, it is $O(e + eN)$. If we use the quick sort algorithm, the time complexity of line 9 is $O(N \log N)$. And from line 12 to line 23, the time complexity is: $O(N(M + M \log M + \xi))$.

Above all, the time complexity of the MaxRe scheduling algorithm is $O(MN + e + eN + N \log N + MN + MN \log M + \xi N)$. That is $O(MN \log M + N \log N + eN)$, where M is the number of processors, and N is the number of tasks in the workflow. *(end proof)*

3.4 The reliability of the MaxRe's result

Proposition 4 *The reliability Ψ of the MaxRe scheduling algorithm meet the following conditions:*

$$\Re \leq \Psi \leq \prod_{i=1}^{i<n} (1 - \prod_{p_j \in sche(\tau_i)} (1 - R(\tau_i, p_j))) \tag{9}$$

where $sche(\tau_i)$ is the processors set on which the task τ_i's replicas are scheduled.

proof: The left part of the formula has been given in Proposition 2. Here we

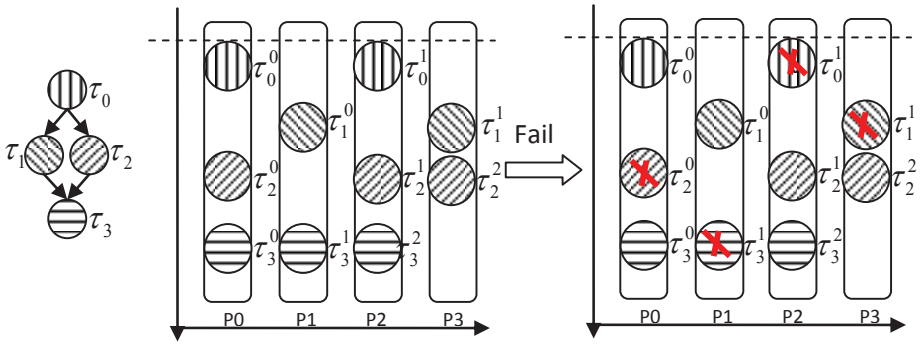


Fig. 2 An example of MaxRe scheduling result

only need to prove the right part: $\Psi \leq \prod_{i=1}^{i<n} (1 - \prod_{p_j \in sche(\tau_i)} (1 - R(\tau_i, p_j)))$.

Let $F'(\tau_i) = 1 - \prod_{p_j \in sche(\tau_i)} (1 - R(\tau_i, p_j))$, $F'(\tau_i)$ implies the probability that at least one replica of the task τ_i succeeds if not considering the relationship with the other tasks. Therefore, $\prod_{i=1}^{i<n} F'(\tau_i)$ implies the probability that at least one replica succeed for all tasks.

(1) If all the replicas of all tasks are scheduled on totally different processors, equation is satisfied: $\Psi = \prod_{i=1}^{i<n} F'(\tau_i)$.

(2) If at least two replicas are scheduled on the same processor. For example, as shown in Fig. 2, suppose a workflow is comprised of 4 tasks: $\tau_0, \tau_1, \tau_2, \tau_3$. And there are 4 processors in the heterogeneous system. After MaxRe scheduling, τ_0 is arranged 2 replicas and is scheduled on processor p_1 and p_3 respectively; τ_0 is also arranged 2 replicas and is scheduled on processor p_2 and p_4 respectively; τ_2 need 3 replicas and is scheduled on processor p_1, p_3, and p_4 respectively; τ_3 need 3 replicas and is scheduled on processor p_1, p_2 , and p_3 respectively.

Table 1 The parameters for the task and processor

Task		Processor			
Load	Com_load	No.	Speed	$\lambda \times 10^3$	Com_speed
$100 \sim 500$	$9 \sim 29$	10/20	$5 \sim 19$	$2 \sim 8$	$0.8 \sim 1.2$

In this case, if τ_0^1, τ_1^1, τ_2^0 and τ_3^1 encounter failures, the job will fail. This is because the processors follow the fail-stop property, failure of τ_0^1 leads that both replica τ_2^1 and τ_3^2 fail too. And the failure of τ_0^1, τ_1^1, τ_2^0 and τ_3^1 will lead to all processors' failing. So there exists $\Psi \leq \prod_{i=1}^{i<n} (1 - \prod_{p_j \in sche(\tau_i)} (1 - R(\tau_i, p_j)))$.

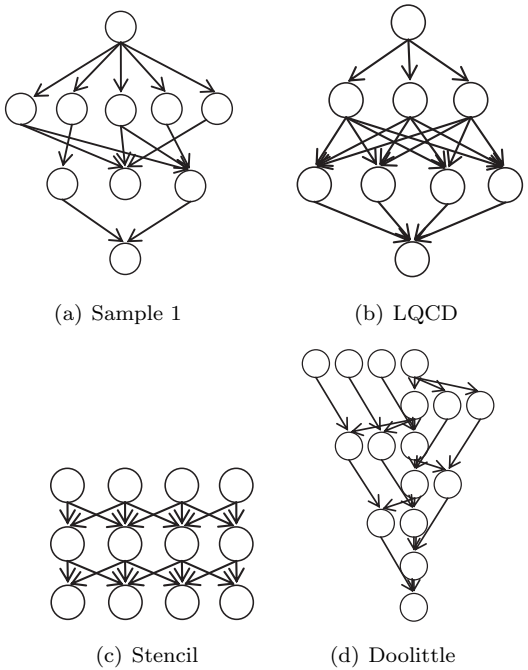
Above all, $\Re \leq \Psi \leq \prod_{i=1}^{i<n} (1 - \prod_{p_j \in sche(\tau_i)} (1 - R(\tau_i, p_j)))$ is established, with equality holding if and only if all replicas of all tasks are scheduled on totally different processors. *(end proof)*

4. Experiments

We select 4 typical workflow examples with considerable complexity to evaluate the MaxRe scheduling algorithm. As shown in Fig. 3(a): the first workflow is an typical workflow example from[15], the second is from LQCD application (Fig. 3(b)), which is simplified prototype of LQCD workflow[18]. The third workflow *stencil* is from the stencil algorithm[5], and the fourth workflow *doolittle* is from doolittle reduction[5].

The parameters for the tasks and processors are shown in Tab. 1. *load* is the computation load for a task, *Com_load* is the transmission load from a parent task to one of his child task. *No.* is the number of processors in the system. *Speed* represents the computation speed of one processor. λ represents the parameter of the Possion distribution. *Com_speed* is the communication speed between two processors. All values of these parameters are initialized with random numbers in the corresponding range.

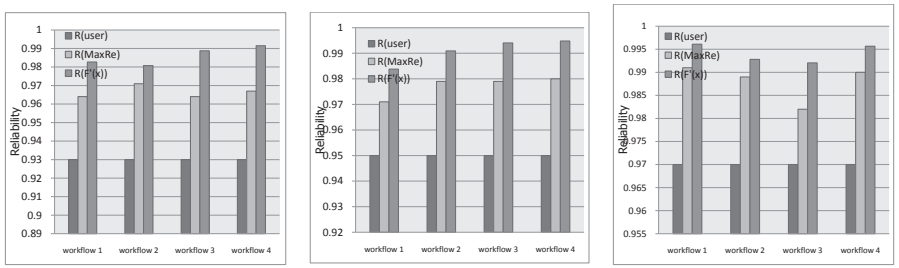
The MaxRe scheduling algorithm is evaluated from three aspects: the verification to Proposition 4, the resource usage with compared to FTSA algorithm,



(a) Sample 1 (b) LQCD

(c) Stencil (d) Doolittle

Fig. 3 The workflows used in the experiments



(a) (b) (c)

Fig. 4 The verification to Proposition 4: (a) describes the situation when user required reliability is 0.93; (b) describes the situation when user required reliability is 0.95; (c) describes the situation when user required reliability is 0.97;

and the execution time with compared to FTSA algorithm.

4.1 The verification to Proposition 4

We evaluate the correctness of Proposition 4 through three experiments, in which we set the user required reliability are 0.93, 0.95 and 0.97 respectively. In each experiment, the workflow is executed for 1000 times, and the exact reliability of the MaxRe's scheduling result is equal to the success rate, which follows $successrates = success/1000$. From Fig. 4, we can see that the experiments results completely meet the Proposition 4: $\Re \leq \Psi \leq \prod_{i=1}^{i<n} F'(\tau_i)$. When the user required reliability is 0.93, the reliability by the MaxRe algorithm could be more than 0.96, and $\prod_{i=1}^{i<n} F'(\tau_i)$ could be more than 0.98. When the user required reliability is 0.95, we have $0.97 < R(MaxRe) < 0.98$ and $R(\prod_{i=1}^{i<n} F'(\tau_i)) > 0.98$. When the user required reliability is set with 0.97, we have $0.98 < R(MaxRe) < 0.99$ and $R(\prod_{i=1}^{i<n} F'(\tau_i)) > 0.99$. Therefore, the Proposition 4 is completely verified by this experiments.

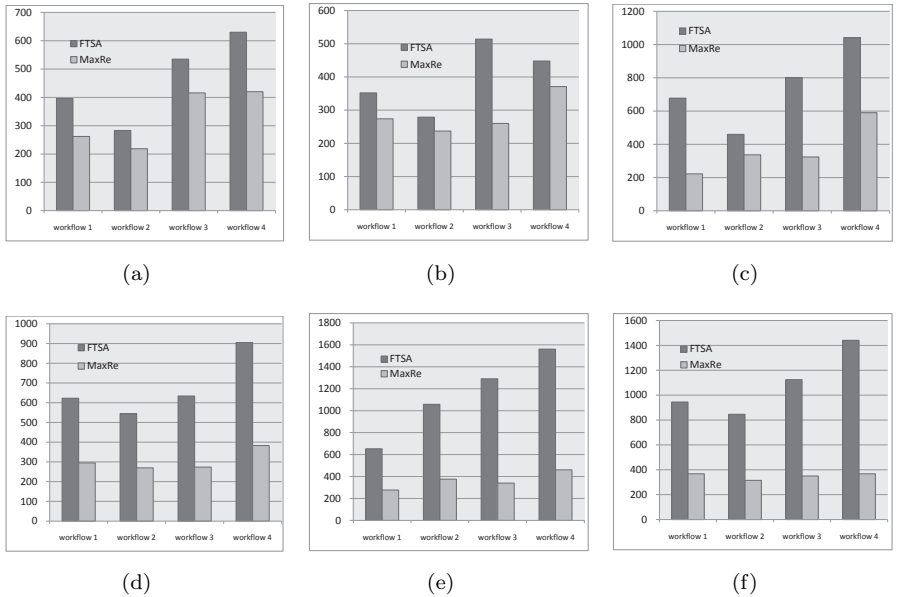


Fig. 5 The resource usage: (a) describes the resource usage when $\varepsilon = 1, m = 10$; (b) describes the resource usage when $\varepsilon = 1, m = 20$; (c) describes the resource usage when $\varepsilon = 2, m = 10$; (d) describes the resource usage when $\varepsilon = 2, m = 20$; (e) describes the resource usage when $\varepsilon = 3, m = 10$; (f) describes the resource usage when $\varepsilon = 3, m = 20$;

4.2 The resource usage with compared to the FTSA algorithm

FTSA (Fault tolerant scheduling algorithm) algorithm is introduced in[6] as a fault tolerant extension of the classic HEFT algorithm[15]. In FTSA, at each step of the mapping process, the free task τ_i with the highest priority is simulated by mapping on all processors. The first $\varepsilon + 1$ (ε is the number of failures that FTSA can tolerant) processors that allow the minimum finish time are kept.

The resource usage is measured by the metric of CUA (CPU usage amount), which is defined in Formula 10:

$$CUA = \sum_{i=0}^{i<n} (ET(\tau_i, p_j) \times x_{ij}) \tag{10}$$

where $x_{ij} = 1$ if the task τ_i is submitted on processor p_j, otherwise $x_{ij} = 0$.

The experiments is organized into 6 groups. We set the ε with value 1, 2, and 3 respectively, and the number of processors in the system with $m = 10$ and $m = 20$. The ε failures are translated into reliability value, which is also the reliability requirement for the MaxRe algorithm. From the Fig. 5, we can see that, to achieve the same reliability, the resource usage of the MaxRe algorithm is less than the FTSA algorithm in all experiments. Specifically, we have the following conclusions:

(1) To tolerant more failures, the increased resource usage by FTSA algorithm is much more dramatically than by the MaxRe algorithm. As shown in Fig. 5(a), Fig. 5(c), and Fig. 5(e), the average ratio of $CUA(MaxRe)/CUA(FTSA)$ is 71.3%, 49.4% and 31.9% respectively. And the average ratio of Fig. 5(b), Fig. 5(d) and Fig. 5(f) is 71.7%, 45% and 32.2%.

(2) Obviously, the workflow that consists of more tasks usually requires more CUA.

(3) Generally, the workflow consumes less CUA if more processors exist in the system. This is because a larger system always have more faster processors.

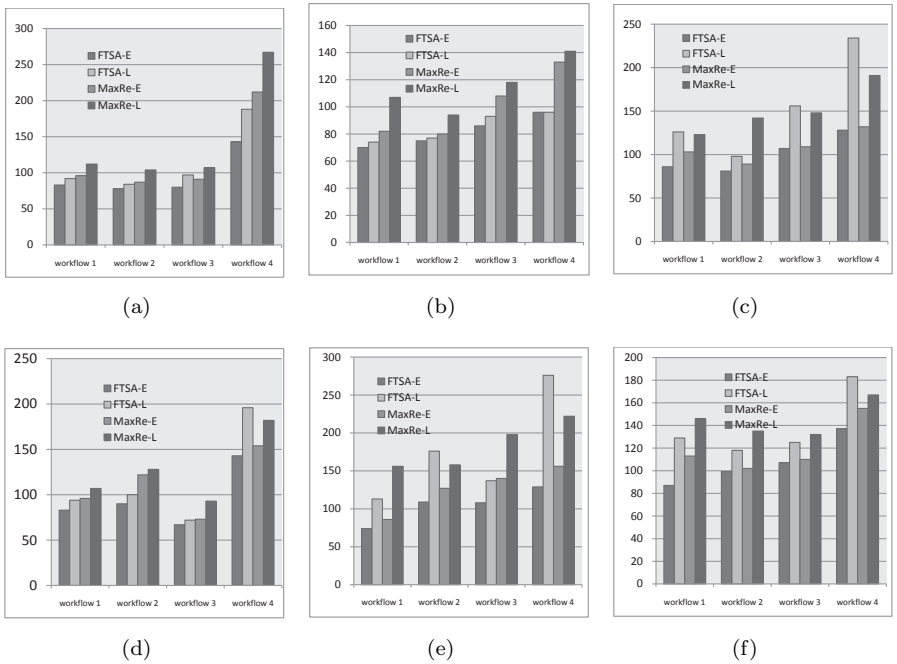


Fig. 6 The execution time: (a) describes the execution time when $\varepsilon = 1, m = 10$; (b) describes the execution time when $\varepsilon = 1, m = 20$; (c) describes the execution time when $\varepsilon = 2, m = 10$; (d) describes the execution time when $\varepsilon = 2, m = 20$; (e) describes the execution time when $\varepsilon = 3, m = 10$; (f) describes the execution time when $\varepsilon = 3, m = 20$;

4.3 The execution time with compared to the FTSA algorithm

The original FTSA algorithm is a earliest finish time version (FTSA-E), which means once one replica of the τ_i finishes successfully, its results will be sent to its successors. We also design a latest finish time version of FTSA algorithm (FTSA-L), in which only after all replicas of task τ_i finish, the result will be sent to its successors. In the same way, we extend the MaxRe algorithm with the MaxRe-E version and the MaxRe-L version.

Table 2 The comparision on time between the FTSA and the MaxRe

Experiment	a	b	c	d	e	f
ar_e	0.79	0.81	0.93	0.86	0.825	0.896
ar_f	0.82	0.74	1.02	0.906	0.96	0.96

As shown in Fig. 6, In all experiments, the execution time of FTSA-E is always the shortest. However, the difference between FTSA-E and the MaxRe-E is not that large. We compute the average ratio on earliest finish time $ar\_e = time(FTSA - E)/time(MaxRe - E)$ and on the latest finish time $ar\_f = time(FTSA - L)/time(MaxRe - L)$, which is shown in Tab. 2. we can see that the average ratio on execution time between the FTSA algorithm and the MaxRe algorithm is mostly higher than 0.8. For the latest finish time, The MaxRe scheduling algorithm even perform better than the FTSA algorithm in Fig.6(c), Fig. 6(d), Fig. 6(e), and Fig. 6(f). This is because the FTSA algorithm has to schedule more processors, and will spend more time on waiting for the finish of the last replica.

Therefore, even the MaxRe algorithm performs worse than the FTSA algorithm on execution time, but compared with the resource usage, it is also considerable.

5. Concluding and future works

To tolerant ε failures, existing research always need to schedule $\varepsilon + 1$ replicas for each task. However, based on the reliability analysis of each processor's failure distribution, we can achieve the corresponding reliability with much less resources. In this paper, we propose a reliability based scheduling algorithm: MaxRe, which employs dynamic number of replicas and targets to satisfy user's reliability requirement with the minimum processors. Both the theorem analysis and experiments show that the scheduling result of the MaxRe algorithm can satisfy user's reliability requirement. And to achieve the same reliability, the resource usage by the MaxRe algorithm is at least 30 percent less than by the FTSA algorithm. If the system is required to tolerant 3 failures, the resource usage by the MaxRe algorithm is almost 70 percent less than by the FTSA

algorithm. The experiments show that the earliest finish time of the MaxRe algorithm's result is just about 30 percent longer than FTSA, while the latest finish time of the MaxRe's result is even shorter than the FTSA's result. Therefore, the MaxRe algorithm is much more suitable in such situation, where the user has special requirements on higher reliability, but the processors is limited.

Our future work will concentrate on the scheduling algorithms that combine reliability and time together. We also should do more work on the system performance analysis.

Acknowledgments The first author of this research is supported by the governmental scholarship from China Scholarship Council.

References

1) C. Dabrowski, Reliability in grid computing systems. Concurrency and Computation: Practice and Experience, vol. 21(8), pp.927-959, 2009.
2) I. Koren, C. Mani Krishna, Fault-Tolerant Systems. Organ Kaufmann, San Francisco, 2006.
3) Q. Zheng, B. Veeravalli, On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs. IEEE TRANSACTIONS ON COMPUTERS, vol. 58(3), pp.380-393, 2009.
4) Q. Zheng, B. Veeravalli, Chen-Khong Tham. On the design of communication-aware fault-tolerant scheduling algorithms for precedence constrained tasks in grid computing systems with dedicated communication devices. Journal of Parallel and Distributed Computing, vol. 69(3), pp.282-294, 2009.
5) A. Benoit, Contention awareness and fault-tolerance scheduling for precedence constrained tasks in heterogeneous systems. Parallel Computing, vol. 35(2), pp.83-108, 2009.
6) A. Benoit, M. Hakem, Y. Robeert, Fault Tolerant Scheduling of Precedence Task Graphs on Heterogeneous Platforms. INRIA Research Report No. 2008-03, 2007.
7) H. Jin, X.H. Sun, Z. Zheng et al., Performance under Failures of DAG-based Parallel Computing. Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, Paris, pp. 236-243, 2009.
8) J.J. Dongarra, E. Jeannot, E. Saule, et al., Bi-objective Scheduling Algorithms for Optimizing Makespan and Reliability on Heterogeneous Systems. In proceedings of the 19th annual ACM symposium on Parallel algorithms and architectures, ACM Press, San Diego, pp. 280-288, 2007.
9) A. Dogan, F. Ozguner, Biobjective scheduling algorithms for execution time-reliabilty trade-off in heterogeneous computing systems. The Computer Journal, vol. 48(3), pp. 300-314, 2005.
10) Y. He, Z. Shao, B. Xiao, et al., Reliability Driven Task Scheduling for Tightly Coupled Heterogeneous Systems. In Proc. IASTED International Conference on Parallel and Distributed Computing and Systems (IASTED PDCS), pp. 465-470, Marina Del Ray, CA, Nov. 2003.
11) S. Swaminathan, G. Manimaran, A Reliability-aware Value-based Scheduler for Dynamic Multiprocessor Real-time Systems. In Proceedings International Parallel and Distributed Processing Symposium(IPDPS). pp. 98-104, Florida, US, 2002.
12) K. Hashimito, T. Tsuchiya, T. Kikuno, Effective scheduling of duplicated tasks for fault-tolerance in multiprocessor systems, IEICE Transactions on Information and Systems E85-D (3), pp. 525-534, 2002.
13) X. Qin, H. Jiang, A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems, Parallel Computing, vol.32, pp. 331-356, 2006.
14) M. Hakem, F. Butelle, Reliability and Scheduling on Systems Subject to Failures, In proceedings of the 2007 International Conference on Parallel Processing, pp. 38-47, 2007.
15) H. Topcuoglu, S. Hariri, M.Y. Wu. Performance effective and low complexity task scheduling for heterogeneous computing. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, vol. 13(3), pp.260-274, 2002.
16) C.M. Wang, S.T. Wang, H.M. Chen, et al., A Reliability-Aware Approach for Web Services Execution Planning. 2007 IEEE Congress on Services, pp.278-283, Salt Lake City, USA, 2007.
17) A. Girault, H. Kalla, M. Sighireanu et al., An Algorithm for Automatically Obtaining Distributed and Fault-Tolerant Static Schedules. The 2003 International Conference on Dependable Systems and Networks (DSN'03), pp.1-10, San-Francisco, USA, 2007.
18) L. Piccoli, X.-H. Sun, J. N. Simone, D. J. Holmgren, et al., The LQCD Workflow Experience: What Have We Learned, Posters of ACM/IEEE SuperComputing Conference(SC07), Nov. 2007.