Vol.2010-AL-132 No.3
2010/11/19

資源使用量を削減するための耐故障スケジューリングアルゴリズムの提案

趙 来 平[†1] 櫻 井 幸 一[†2]

分散コンピューティングは大規模な科学計算を可能とする。様々なハードウェアは安価になったが、それでも資源使用量を減らすことは重要となる。たとえば、クラウドコンピューティングでは、省エネルギーや費用を節約するとが常に重要となる。しかしながら、これらを実現するスケジューリングアルゴリズムについてはそれほど多くは研究されていない。本稿では、ジョブの終了期限、信頼度と資源使用量を考えて、新しい耐障害のためのスケジューリングアルゴリズムを提案する。実験結果から、ジョブは必ず終了期限の前に終了することがわかった。また、我々のアルゴリズムは、FTSA と比べて、計算資源の 50 %と通信量の 70 %を節約することができた。

On the Design of Fault Tolerant Scheduling Algorithm to Reduce the Resource Usage

Laiping Zhao [†1] and Kouichi Sakurai[†2]

The distributed system made the large-scale scientific computing possible in a cost effective way. And the hardware resources in such systems are also getting much cheaper than years before. However, the problem of executing the job using minimum resources is still reasonable and important, especially for the cloud environment, who has to save energy and economic cost. Unfortunately, only a few existing scheduling algorithms has taken into account the resource usage issue. In this study, with considering the realistic network topology and communication model, we firstly propose the Deadline, Reliability, Resources-aware (DRR) scheduling algorithm.

The theory analysis fully demonstrate that, the output schedule of our algorithm can satisfy the user's requirement on reliability and deadline. Through the experiments, with setting the deadline less than the makespan of the MaxRe algorithm's output schedule, we find that our algorithm can complete the job under this deadline. Besides, our algorithm can save almost 50% computation resources and 70% communication resources than *FTSA(bl)* and *FTSA(tl+bl)* algorithms.

1. Introduction

1.1 Background

A heterogeneous system is composed of many different kinds of hardware and software working cooperatively to solve problems. With the new deployed machines and old, slow machines are replaced with new, fast ones continuously, the cloud environment is believed to become more heterogeneous in the future. And the efficient scheduling algorithm in heterogeneous system is critical in order to achieve both the users and systems' objectives. Especially, in order to provide high reliability, active replication scheme and backup/restart scheme[1)], which correspond to resource and time redundancy respectively, are already widely used in the literature.

Compared with the backup/restart scheme, the active replication scheme exploit several processors simultaneously to execution one task, which is much less sensitive to the deadline. Therefore, we can make use of the active replication scheme in certain job executions where the deadline is strictly demanded.

1.2 Motivation

However, the active replication scheme also brings another problem, i.e. the large resource usage, which has an adverse impact for the system performance. And the larger resource consumption usually comes with more energy consumption and higher economic cost.

The EPA(Environmental Protection Agency) report said, servers and data centers consumed 61 billion kwh (kilowatt hours) in 2006, which was 1.5 percent of total U.S. electricity consumption that year, amounting to $4.5 billion in electricity costs-

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

equivalent to 5.8 million average U.S. households. Therefore, it is necessary to do study on energy efficient computing, and reducing resources usage is one possible way.

In addition, Amazon, for example, claims that its S3 service stores three replicas of each file. That means, to store x gigabytes data, Amazon has to supply $3x$ gigabytes storage space located on three different drives, with x gigabytes corresponding to each drive. Assuming the economic cost of each drive is y, then 3 drives will cost $3y$, including extra $2y$ cost. It is believed that these extra cost will be passed on to the customers eventually. Besides in the storage service, the similar situation also happens on the computing service.

In this study, we address the resource minimization issue in scheduling algorithm. And similar with the existing works on scheduling[4),7)–9),11)], we specifically consider user's deadline and reliability constraint.

1.3 Previous works

Several algorithms have been proposed to cope with the deadline, reliability-biobjective scheduling problem. A. Dogan and F. Ozguner[7)] propose two biobjective scheduling algorithm: BDLS algorithm and the biobjective genetic algorithm. The BDLS schedules tasks using the weighted average order, which is computed from the time-related dynamic level and the reliability-related incremental cost. And the biobjective genetic algorithm aggregate the time and reliability into fitness function for task scheduling. J.J. Dongarra et al.[4)] design two algorithms that optimize both makespan and reliability. The first scheduling algorithm is targets to maximize the reliability subject to makespan inimization for independent unitary tasks. And the second one is based on the product failure rate unitary instruction execution time to trade off between reliability maximization and makespan minimization. Taking the makespan minimizing as the objective, Mourad Hakem and Frank Butelle[8)] propose the HAS algorithm, and taking the failure probability minimizing as the objective, they propose the RSA algorithm. At last, Based on the compromise function between failure probability and makespan, they propose the BSA algorithm. X. Qin and H. Jiang[9)] propose the DASAP and DALAP algorithms, which correspond to earliest finish time and latest finish time under the deadline constraint respectively. Then, DRCD algorithm is proposed to integrate both the reliability cost and makespan into scheduling. In summary, these biobjective scheduling algorithms take the processors' reliability analysis and execution time into scheduling. Compared with the active replication scheme, they do not offer greater reliability. In case one processor encounter a failure during his working, it may lead to the corruption of the whole job.

In the papers[1)] and[3)], FTSA and CAFT are proposed with directly applying the active replication scheme. The difference between them is that CAFT considers the more realistic bi-directional one-port model and communication overhead while FTSA not. To tolerant ε possible failures, they both schedule the task to $\varepsilon + 1$ processors. This wastes much resources in scheduling process, and causes more energy consumption and economic cost. L. Zhao et al.[10)] introduce the resource minimization problem for the first time. The proposed MaxRe scheduling algorithm selects some processors with maximum reliability for task execution. This reduces the resource cost to achieve a higher reliability. But the experiments show that, compared with FTSA algorithm, the MaxRe's output schedule does not performs well on makespan. It cannot be used when a deadline is specifically constrained. S. Swaminathan and G. Manimaran[11)] make use of the redundancy level for task scheduling, which is similar with dynamic number replicas in paper[10)]. However, there are no reliability and execution time analysis in this paper, and also the detail explanation on reward and penalty are not given.

In the papers[2),12),14)], the primary/backup scheme is used to improve the reliability of scheduling. Since only one failure is tolerated in these works, it is not reliable enough for restoring data.

1.4 Challenge issues

We study the problem of using minimum resources to achieve the reliability and deadline requirements. To satisfy the user's deadline requirement, the first challenge is how to decide the subdeadline for each task in the job based on the overall deadline. The second challenge is how to find the processors that can complete all tasks under the subdeadline constraint. Moreover, another challenge on user's reliability requirement is how to decide the number of replicas for each task. Obviously, the less replicas lead to less resources usage. Therefore, the last challenge is ensuring that the minimum resource usage should not violate the reliability requirement.

1.5 Our contribution with comparison to related works

In this study, we propose a reliability, deadline, resource usage-aware scheduling algorithm for the heterogeneous system. The contributions include:

- We consider the bi-directional one-port model and the three-tier network topology in our algorithms. This makes our algorithm more realistic than the FTSA and MaxRe algorithms.
- We compare the two priority methods on sorting the tasks of a job. And we found that the priority method with only *tl* (dynamic top level) value has a better performance on makespan than the one with both *tl+bl* (static bottom level) value. Thus we use the *tl* value for task sorting in our algorithm.
- We design the algorithm on computing each task's subdeadline based on the overall deadline. Compared with the four rules in deadline assignment method proposed by Yu Jia et al.[13], our method only states two rules. This reduces the complexity on deadline assignment.
- We propose the DRR scheduling algorithm. The analysis proves that the execution time and reliability achieved by DRR algorithm can guarantee the user's deadline and reliability requirements, respectively.
- The experiments results show that, when setting the deadline with less than the MaxRe schedule's makespan, DRR algorithm can find a schedule within the deadline constraint, but use 50% and 70% less computation and communication resources than *FTSA(tl)* and *FTSA(tl+bl)* algorithm, respectively.

2. System models

2.1 Processor model

In the system, denote by P the processors set, and $m = |P|$. Processors may have different kinds of hardware and software, and show various performances on capacity or other criterions. Processors are supposed to be fault-free while they are in the idle time. And each processor can execute 1 task at one time, another task can start to execute only after last task is completed or terminated. Crash failures are considered in this study, and the arrival of failures follows the Poisson distribution. The probability that k failures occur in a unit time t is represented as:

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

Where k is the number of occurrences of failures in unit time t, λ is the expected number of occurrences of failures in unit time t.

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. $n = |V|$ is the number tasks. $s = |E|$ 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*. Any task cannot start being executed before it received the output from its all its predecessors and the result of any task can be sent to its successor tasks after the task has been finished.

2.3 Network communication model

Different with the literature[10], this study discusses the communication link's reliability model, and takes the classic three-tier model as the network topology architecture. As mentioned by M. Al-Fares et al.[5], the network topology of a common data center consists of three tiers: *core tier*, *aggregation tier* and *edge tier* (Fig. 1). Switches at the edge tier (e.g. *H1, H2*) have some number of ports connected with processors (e.g. *E1,E2*), as well as some number of uplinks to the aggregation tier that aggregate and transfer packets between edge switches. Switches at the aggregation tier (e.g. A1) are connected with edge switches, while also have some uplinks to the core layer (e.g. C1, C2). The switches at higher level can be assumed with a higher capacity for the aggregate traffic.

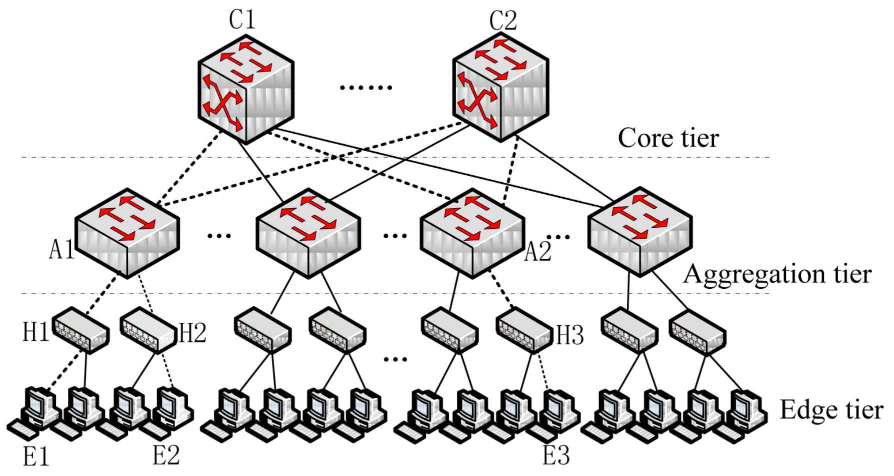


図 1 The three-tier network topology model

Without losing the central idea, we simply assume that the *oversubscription*[5] of the network is 1:1. It indicates that all hosts can communicate with arbitrary other hosts at the full bandwidth of their network interface. Therefore, the communication time between tasks only depends on the transmission load.

In order to make our algorithm more realistic, we also apply the bi-directional one-port model for the communication contention[1]:

- At a given time-step, any processor can send a message to, and receive a message from, at most one other processor. Network interfaces are assumed full-duplex in this bi-directional model.
- Communication and (independent) computation may fully overlap. As in the traditional model, a processor can execute at most one task at each time-step.
- Communications that involve disjoint pairs of sending/receiving processors can occur in parallel.

3. The scheduling objectives

Given the processor, job and communication link models, we seek a resource scheduling algorithm that can complete the job within the deadline and reliability requirements, but makes use of minimum resources. The scheduling objectives are discussed as follows.

3.1 Execution time

Execution time (also named as *makespan*) is directly related with the deadline constraint. Based on the one-port model, we compute the Earliest Start Time(EST) and the Latest Start Time(LST) for each task under the active replication scheme. Firstly, the arrive time $arrive(r,p)$, that the time of one parent task replica r's results arriving at processor p from its scheduled processor $p(r)$, is computed as the sum of communication cost $c_{p(r),p}$ and the maximum value between replica r's finish time (FT) and $line_{p(r),p}$'s ready time:

$$arrive(r,p) = \max\{FT(r), ready(line_{p(r),p})\} + c_{p(r),p}$$

If $p(r) = p$, we have $c_{p(r),p} = 0$. Then the earliest start time (EST) of the task x on processor p is computed as:

$$EST(x,p) = \max\left\{\max_{y\in parent(x)}\left\{\min_{r\in replica(y)}\{arrive(r,p)\}\right\}, ready(p)\right\} \quad (1)$$

The latest start time (LST) of a task x on processor p is computed as:

$$LST(x,p) = \max\left\{\max_{y\in parent(x)}\left\{\max_{r\in replica(y)}\{arrive(r,p)\}\right\}, ready(p)\right\} \quad (2)$$

3.2 Reliability

The reliability consists of communication reliability and processing reliability. Communication reliability is the probability that the message generated by the parent tasks can be successfully transferred to the processors where its child tasks are located. And processing reliability is the probability that the job is executed successfully on the processor.

The key point for computing the communication reliability is deciding the communication path. Considering the network topology shown in Fig. 1, there are 2 possible ways for transferring messages from parents tasks to children tasks.

3.2.1 E1-H1-A1-H2-E2 line

In this case, if the two communication related processors locate in the same domain (e.g. they are connected to the same hub, or the same aggregation tier switch), there is only one unique communication path between processors. And the communication reliability is computed from combined probability of all the resources on this path, such as, the communication reliability between $E1$ and $E2$ (Fig. 1) are computed as:

$$R_{comm}(E1, E2) = R_{E1-H1-A1} \cdot R_{A1-H2-E2} \tag{3}$$

Where $R_{E1-H1-A1} = R_{link_{E1,H1}} \cdot R_{H1} \cdot R_{link_{H1,A1}} \cdot R_{A1}$, and $R_{A1-H2-E2} = R_{link_{A1,H2}} \cdot R_{H2} \cdot R_{link_{H2,E2}}$.

Assume the failure occurring of all these communication resources follow a poison distribution: $R(k, \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$, then the probability of no failures occurring is $R = e^{-\lambda t}$.

3.2.2 E1-H1-A1-C1(C2)-A2-H3-E3 line

In this case, the two communication processors are located in different domains. If processor $E1$ sends a message to another processor $E3$, it has to pass through the core tier switches, and there are two available routing paths for transmission. The communication reliability is computed from the combined probability of all paths.

$$R_{comm}(E1, E3) = R_{E1-H1-A1} \cdot R_{A2-H3-E3} \cdot R_{C1\ or\ C2} \tag{4}$$

Where $R_{A2-H3-E3} = R_{A2} \cdot R_{link_{A2,H3}} \cdot R_{H3} \cdot R_{link_{H3,E3}}$, $R_{C1} = R_{link_{A1,C1}} \cdot R_{C1} \cdot R_{link_{C1,A2}}$, and $R_{C1\ or\ C2} = 1 - (1 - R_{C1})(1 - R_{C2})$.

3.3 Resource usage

The resource usage also consists of computation resource usage and the communication resource usage. The computation Resource Usage(RU) is computed based on all processors' execution time:

$$RU_{comp} = \sum_{x \in V} \sum_{p \in P} \{ET(x, p) \cdot \alpha_{xp}\} \tag{5}$$

Where $ET(x, p)$ indicates the execution time of task x on processor p, and α_{xp} is a binary choice, which equals to 1 if the task x is submitted on processor p, otherwise $\alpha_{xp} = 0$.

The communication resource usage is computed from all communication links and switches' communication time:

$$RU_{comm} = \sum_{e \in E} \sum_{l \in net} \{c(e, l) \cdot \alpha_{el}\} \tag{6}$$

Where $c(e, l)$ indicates the communication time of edge e on link (or switch) l, and $\alpha_{el} = 1$ if the edge e is communicated on network link (or switch) l, otherwise $\alpha_{el} = 0$.

4. The DRR scheduling algorithm

4.1 Deadline assignment

To support the deadline constraint, the *exit node* must be completed before deadline. Yu Jia et al.[13] gives one method on computing the subdeadline for each task, where the overall deadline is divided over task partitions in proportion to their minimum processing time. In practice, as long as the exit task can be finished before the deadline, the deadline requirement is satisfied. Therefore, in this study, different with their deadline assignment policies, we only concern about two points:

- The subdeadline can assure the overall deadline.
- The subdeadline for each task is not less than the finish time of each task.

Suppose that the overall deadline requirement is T, the scheduling algorithm should guarantee that the output schedule will complete the job under the deadline constraint. Alg. 1 gives the details of the deadline assignment algorithm. In line 1-3, we initialize all tasks' subdeadlines with T. Then learning from the broad first search (BFS) algorithm, we make use of the *queue* struct, and recursively compute the subdeadline set from each *exit node*(line 5-22). The big difference with BFS algorithm is the multiple *exit nodes* in our job graph. First, push the *exit node* into the *queue* (line 6). Then pop out the first task node z from the q. If task z is *exit node*, set its subdeadline with T (line 9-11). Then in line 12-20, we push z's parent y into q, and compute the y's subdeadline using $t = subdeadline[z] - \overline{c(e_{y,z}, l)} - \overline{ET\,(z,p)}$, where $\overline{c(e_{y,z}, l)}$ is the expected communication time from task y to task z, and $\overline{ET\,(z,p)}$ is the expected execution time of node z. At last, set the $subdeadline[y]$ with $min(t, subdeadline[y])$.

Algorithm 1: Subdeadline($\overline{ET\,(x,p)}, \overline{c(e,l)}$)(BFS-based deadline assignment algorithm)

```
Require:
    Deadline $T$, $G = (V, E)$, $\overline{ET\,(x,p)}$, $\overline{c(e,l)}$.
Ensure:
    subdeadline[i], where $\forall i \in V$.
 1: for $\forall i \in V$ do
 2:     $subdeadline[i] = T$;
 3: end for
 4: Queue q;
 5: for each exit node $x$ do
 6:     q.push(x);
 7:     while !q.empty() do
 8:         z = q.front(), q.pop();
 9:         if z is exit node then
10:             $subdeadline[z] = T$;
11:         end if
12:         if z is not entry node then
13:             for $\forall y \in parent(z)$ do
14:                 if $y \notin q$ then
15:                     q.push(y);
16:                 end if
17:                 $t = subdeadline[z] - \overline{c(e_{y,z}, l)}) - \overline{ET\,(z,p)}$;
18:                 $subdeadline[y] = min(t, subdealine[y])$;
19:             end for
20:         end if
21:     end while
22: end for
```

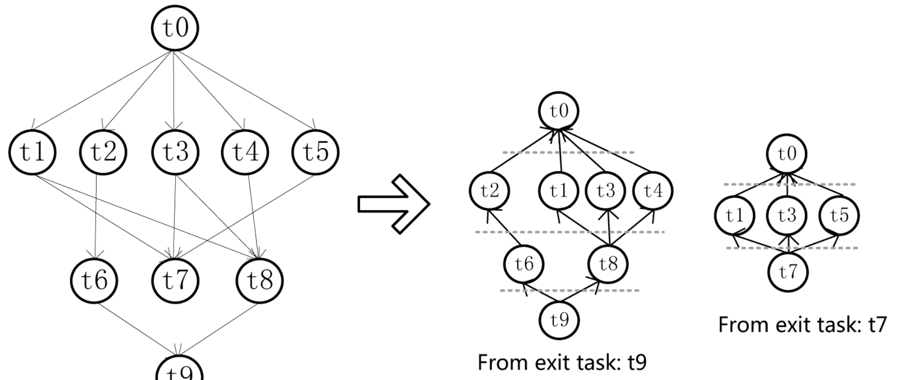


図 2 Case study: subdeadline assignment

Case study:

As shown in Fig.2, a sample workflow consists of 10 tasks, where t_0 is an *entry node*, and t_7, t_9 are *exit nodes*. According to the Alg. 1, the tasks' deadlines can be recursively computed from *exit task* t_7 and t_9. From node t_7, find his parent nodes are t_1, t_3, and t_5. Using the formula in line 17, we can get their subdeadlines. Then, the subdeadline of node t_0 can be gotten from t_1, t_3, and t_5, respectively. Choose the minimum one as his subdeadline. From *exit node* t_9, repeating this above procedure, compute the subdeadlines of t_9's parent nodes: t_6, t_8. From node t_6, we can get the subdeadline of t_2, and from t_8, we can get the subdeadline of t_1, t_3, and t_4. Based on the subdeadlines of t_2, t_1, t_3, and t_4, we will get the subdeadline of t_0. In this whole process, whenever need to set the subdeadline, we always choose the smallest value as his subdeadline.

4.2 Reliability assignment

Suppose the overall reliability requirement is R. Denote the task and its related previous communication link as a *group*, for example, in Fig. 2, $\{t_0\}$ is the t_0 group. $\{t_1,\ e_{t_0,t_1}\}$ is the t_1 group. And $\{t_7,\ e_{t_1,t_7},\ e_{t_3,t_7},\ e_{t_5,t_7}\}$ is the t_7 group. Generally speaking, *group* reliability is determined by the communication links' reliability and the computation reliability, except that the *entry node* group is only determined by computation reliability. Because for the exit tasks, we cannot arbitrary distinguish their importance without any preliminaries, and for the entry tasks and internal tasks, we believe that they are all equally important for their descendant tasks, the reliability for each *group* should not be less than: $\sqrt[n]{R}$.

4.3 Task priority

To order the execution of the tasks so that task precedence constraints are satisfied, the task priority is generally determined by the bottom level (*bl*, formula 7) and top level (*tl*, formula 8), which are also named as upward and downward ranking.

$$bl\,(x) = \overline{ET\,(x,p)} + \max_{y\in child(x)} (\overline{c_{x,y}} + bl\,(y)) \tag{7}$$

$$tl(x) = \max_{z\in parent(x)} (tl(z) + \overline{ET(z,p)} + \overline{c_{z,x}}) \tag{8}$$

Where $\overline{c_{x,y}}$ is the expected communication time between parent task x and child task y. $\overline{ET(x,p)}$ is the task x's expected execution time.

In summary, the HEFT[15], MaxRe[10] algorithms make use of *bl* to order tasks, while the FTSA[3], CAFT[1], CPOP[15] algorithms make use of $tl + bl$ value for task ordering. To tell the difference between the *bl*-based and the $(tl + bl)$-based task ordering, we study their effects on execution time through experiments. The detail experimental parameters are listed in section 5. Different with the original FTSA algorithm, we design a *bl*-revised version of FTSA algorithm, which makes use of *bl* value for task ordering. We compare the time performance between original FTSA(tl+bl) algorithm and the FTSA(bl) algorithm. As shown in Fig. 3, we find that, with the increasing number of tasks, the FTSA(bl)'s output schedule consumes less execution time than

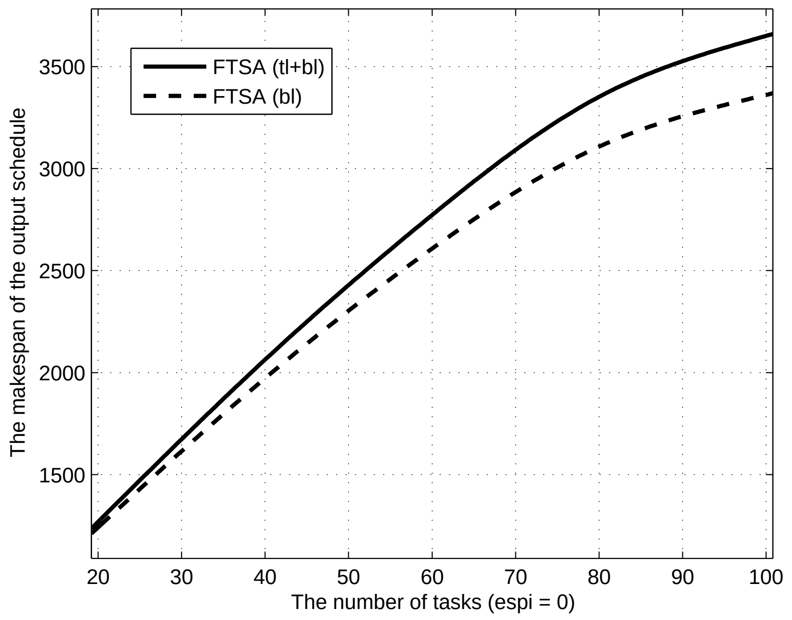


図 3 The comparison between FTSA(bl) and FTSA(tl+bl)

FTSA(tl+bl). Moreover, through multiple experiments (e.g. 100 times) with different parameters, the FTSA(bl) averagely performs better than FTSA(tl+bl). Therefore, we will apply the *bl* value into our algorithm's task priority computation.

4.4 DRR scheduling algorithm

Algorithm 2: The DRR scheduling algorithm.

```
Require:
    Deadline $T$, reliability $R$, $G = (V, E)$, P
Ensure:
    schedule
 1: for each node $x \in V$ do
 2:     for each $p_j \in P$ do
 3:         $ET(x, p_j) \leftarrow$ compute the execution time using $(x.load, p_j.speed)$;
 4:     end for
 5:     $\overline{ET(x,p)} \leftarrow ET(x, p_j), \forall p_j \in P$;
 6: end for//$Compute\ the\ expected\ communication\ time$;
 7: for each edge $e \in E$ do
 8:     $\overline{c(e,l)} \leftarrow$ compute the communication time using $(e.load, net.speed)$;
 9: end for
10: $Subdeadline(\overline{ET(x,p)}, \overline{c(e,l)})$; $(Assign\ the\ deadline\ to\ each\ task)$;
11: if $subdeadline(x) < \overline{ET(x,p)}, \forall x \in entry\ nodes$ then
12:     reject the job; return;
13: end if
14: $r \leftarrow root(R)$; $(Compute\ the\ reliability\ requirement\ for\ each\ task)$;
15: $TP \leftarrow rank(\overline{ET(x,p)}, \overline{c(e,l)})$; $(Compute\ the\ task\ priority)$;
16: $sort(V, TP)$; $(Sort\ all\ tasks\ according\ to\ the\ task\ priority\ values)$;
17: $\Theta = \emptyset$, $U = V$;
18: while $U \neq \emptyset$ do
19:     $x = head(U)$; $//Select\ the\ processors\ for\ current\ task\ x$;
20:     for each processor $p_i \in P$ do
21:         $CR(x, p_i) = CPR(x, p_i) \cdot CLR(x, p_i)$;
22:     end for
23:     $sort(P, CR(x, p_i))$;
24:     $fail \leftarrow 1.0$, $counter \leftarrow 0$, $j \leftarrow 0$;
25:     while $(1 - fail) < r$ && $counter < m$ do
26:         $EFT(x, p_j) = EST(x, p_j) + PT(x, p_j)$;
27:         if $EFT(x, p_j) < subdeadline[x]$ then
28:             $counter = counter + 1$;
29:             $fail = fail \times (1 - CR(x, p_j))$;
30:         end if
31:         $j{+}{+}$;
32:     end while
33:     $\varepsilon \leftarrow counter$, $counter \leftarrow 0$;
34:     if $\varepsilon = 0$ then
35:         reject the job; return;
36:     end if
37:     Schedule task $x$ to the corresponding $\varepsilon$ processors;
38:     Put $x$ into $\Theta$;
39:     $U \leftarrow U \backslash \{x\}$;
40: end while
```

In order to schedule task x to processors with satisfying the reliability requirement, we schedule the x *group* reliability based on Current processor reliability (CPR) and Current link reliability(CLR), which are computed from Current processor execution time(CPET) and Current communication time(CCT), respectively.

CPET is the running time that processor p has costed with the addition to task x's execution time:

$$CPET(x,p) = ET(x,p) + \sum_{y \in on(p)} ET(y,p)$$

Where $on(p)$ is the task set that scheduled on processor p.

$$CPR(x,p) = e^{-\lambda \cdot CPET} \tag{9}$$

For communication reliability, considering the network topology, the route path is determined by the location of both parent tasks and child tasks. Suppose l is one communication link (or switch) in the route path, the $CCT_l(x,p)$ indicates the running time that l has worked with addition to task x's communication time.

$$CCT_l(x,p) = c(e_{y,x}, l) + \sum_{e \in on(l)} c(e,l)$$

Where $on(l)$ indicates the edges set in the job graph that communicated on link(or switch) l.

$$CLR_l(x,p) = e^{-\lambda \cdot CCT}$$

Applying the $CLR_l(x,p)$ into formula 3 or 4, we can get the communication reliability $R_{comm}(r,p)$. And the current link reliability $CLR(x,p)$ for submitting task x to processor p is gotten by formula 10, which means the reliability value that for every x's parent task, at least one replica's output is successfully transmitted to processor p.

$$CLR(x,p) = \prod_{y \in parent(x)} (1 - \prod_{r \in replica(y)} (1 - R_{comm}(p(r),p))) \tag{10}$$

The details of DRR scheduling algorithm is shown in Alg. 2: In line 1-6, we compute the execution time of each task on each processor and each task's expected execution time($\overline{ET}(x,p)$). In line 7-9, we compute the expected communication time ($\overline{c(e,l)}$) of each edge in the task graph. Based on $\overline{ET}(x,p)$ and $\overline{c(e,l)}$, we can get all tasks' subdeadlines using Alg. 1 (line 10). If the *entry node*'s subdeadline is violated by its $\overline{ET}(x,p)$, then reject the job. In line 14, compute each task's subreliability requirement with R. In line 15-16, compute all tasks' priority using formula 7, and sort the tasks into order. From line 17 to 40, we will schedule the tasks to the processors. In line 19, select the head task from the sorted task list. In line 20-22, we compute the reliability achieved by simulating scheduling task x to every processor, and the processors are sorted into decreasing order according to the x *group* reliability value: $CR(x,p_i) = CPR(x,p_i) \cdot CLR(x,p_i)$. In line 24-32, we select the first ε processors, which can provide sufficient reliability for reliability requirement r (line 24, 29), and also can complete the task under the subdeadline constraint (line 27). The number of replicas for current task x is ε (line 33). If $\varepsilon = 0$, which means no processors can complete the current task within the constraints, reject the job. Or else schedule the task x to these ε processors, and repeat the whole process until all tasks are scheduled.

4.5 Analysis

Lemma 1 *In the case of $\exists x \in V, FT(x) > subdeadline(x)$, the whole job cannot be successfully completed within the deadline constraint T.*

proof: Suppose for the task x, we have $FT(x) > subdeadline(x)$. It is suffices to shown that if $x \in exit\ nodes$, then $subdeadline(x) = T$, and $FT(x) > T$, which means the job cannot be completed successfully.

If the $x \notin exit\ nodes$, according to the definition of EST (formula 1), we have $EST(z) \geq FT(x) + \overline{c(e_{x,z}, l)}$, $\forall z$, s.t. $z \in child(x)$. Applying it into $FT(z) = EST(z,p) + \overline{ET(z,p)}$, we have $FT(z) \geq FT(x) + \overline{c(e_{x,z}, l)} + \overline{ET(z,p)}$.

With $FT(x) > subdeadline(x)$ and $subdeadline(x) = \min_{z \in child(x)} \{subdeadline(z) - \overline{c(e_{x,z}, l)} - \overline{ET(z,p)}\}$, denoting z_{min} as the task that leads to the $subdeadline(x)$, we have

$$\begin{aligned} &FT(z_{min}) > subdeadline(x) + \overline{c(e_{x,z_{min}}, l)} + \overline{ET(z_{min}, p)} \\ &\Leftrightarrow FT(z_{min}) > subdeadline(z_{min}) \end{aligned} \tag{11}$$

Therefore, there exists at least one x's child task z_{min}, which cannot be completed before its $subdeadline(z_{min})$. By induction on the graph, at least one *exit* task will violate the deadline constraint T, therefore the lemma is established.

Lemma 2 *$\forall x \in entry\ nodes, \overline{ET(x,p)} < subdeadline(x)$ is just the necessary condition of job's successful completing within deadline T.*

proof: Lemma 1 directly tells that $\forall x \in entry\ nodes, \overline{ET(x,p)} < subdeadline(x)$ is

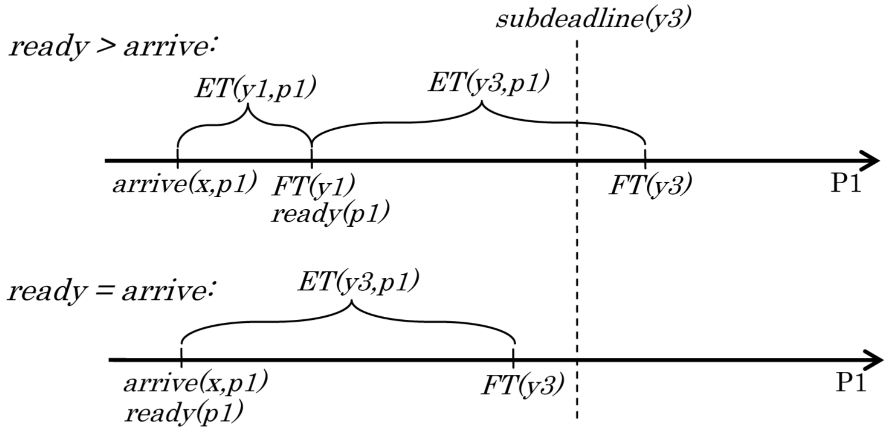


図 4 The $subdeadline(y_3)$ is violated for the reason of the later ready time

the necessary condition. The central idea on the not sufficient condition proof is that any task has to wait for executing until the processor is ready for it.

Assume for task x, there exists $y_1, y_2, y_3 \in child(x)$. And after task x is completed, assume there are only 2 available processors, i.e. p_1,p_2 on which task y_1 and y_2 are scheduled respectively. Whichever processor is scheduled for task y_3, it has to wait the finish of y_1 (if y_3 is scheduled on p_1). This leads to $ready(p_1) > arrive(x, p_1)$, then $EST(y_3) = ready(p_1)$. As shown in Fig. 4, if $subdeadline(y_3) - [arrive(x, p_1) + ET(y_3, p_1)] < ET(y_1, p_1)$, the $subdeadline(y_3)$ is violated. Similarly as the proof in theorem 1, at least one successor task will not be completed within the deadline. And the overall deadline is not guaranteed. Therefore, the lemma 2 holds. According to the Lemma 2, we add the feasibility judgement into Alg. 2 in line 30-31.

Theorem 1 *In the case of the job being accepted in Alg. 2, its output schedule can ensure the overall deadline T.*

proof: According to the Lemma 1 and Lemma 2, Alg. 2 rejects the job in the situation of $\exists x \in V, FT(x) > subdeadline(x)$ and situation of Fig. 4. Once the job is accepted and scheduled to some processors, the job will be executed as the schedule, which means the output schedule can ensure the overall deadline T.

Theorem 2 *Assume that for every task, the deserved number of replicas does not exceed n, and denote the reliability provided by Alg. 2 as $\Re$, then $\Re \geq R$.*

proof: From the description of Alg. 2, the subreliability requirement r follows that:

$$\begin{aligned} & r \leq 1 - \prod_{p_k \in sche(x)} (1 - CR(x, p_i)) \\ & \Leftrightarrow R = r^n \leq \prod_{x \in V} F(x) \end{aligned} \tag{12}$$

Where $F(x) = 1 - \prod_{p_i \in sche(x)} (1 - CR(x, p_i))$, and $sche(x)$ denotes the processors, communication links, and switches on which x *group* is scheduled. The $CR(x, p_i)$ implies the *group* probability that no failures occur on the x *group*, from the very beginning until the finish of task x on processor p_i. Thus $F(x)$ implies the probability that, for task x, at least one scheduled processor and its related communication path do not encounter failures from the very beginning to the x's finish on p_i.

For arbitrary two *groups* x and y corresponding to $F(x)$ and $F(y)$, respectively, assume the group y is scheduled later than the x. The relation between their success and the value $F(x) \cdot F(y)$ follows three situations:

(1) If $sche(x) \cap sche(x) = \emptyset$, the probability of both *groups*' success (Ψ) is equal to $F(x) \cdot F(y)$.

(2) If $sche(x) = sche(y)$, the probability of both *groups*' success would be equal to $F(y)$. Thus $F(y) > F(x) \cdot F(y)$.

(3) If $sche(x) \cap sche(y) \neq \emptyset$ and $sche(x) \neq sche(y)$. Since the double counting of the resources in $sche(x) \cap sche(y)$, their success probability is between $F(x) \cdot F(y)$ and $F(y)$. And with the higher repeatability of $sche(x)$ and $sche(y)$, their success probability is more closer to $F(y)$.

Generalized to the case of n tasks, $\prod_{x \in V} F(x) \leq \Re$. Thus we have $\Re \geq R$, with equality holding if and only if when all *groups* in set V are scheduled on total different processors, links and switches. Throughout the proof, we make use of *group* reliability instead of processor reliability, which is a big difference with the one in[10]. It is proved that the theorem is also established with *group* reliability.

Theorem 3 *The time complexity of Alg. 2 is* $O(MN\log M + N\log N)$.

proof: From line 2 to line 5, it takes $O(MN)$ to compute the execution time of each task on each processor. line 7-8 costs $O(s)$. In line 9, the *Subdeadline()* function costs $O(n+s)$. In line 13, it costs $O(n+s)$ for computing the task priority. Line 14 costs $O(N\log N)$ to sort the tasks into priority decreasing order. From line 22 to line 29, it costs $O(\varepsilon)$ to decide the number of replicas. And from line 15 to line 32, it takes $O(N(M + M\log M + \varepsilon))$ to schedule the task to processors.

Thus, the time complexity of DRR algorithm is $O(MN + n + s + N\log N + N(M + M\log M + \varepsilon))$, which is $O(MN\log M + N\log N)$.

5. Experiments

We study the performance of DRR algorithm by comparing the execution time and resource usage with 3 algorithms, namely FTSA(bl), FTSA(tl+bl) and MaxRe.

5.1 The simulation system

In our simulation, we use randomly generated application graphs to construct the DAGs, whose parameters are consistent with those used in literature[15].

- The number of tasks, $n = \{20, 40, 60, 80, 100\}$.
- Communication to computation ratio (CCR). $CCR = \{0.1, 0.5, 1.0, 5.0, 10.0\}$.
- Shape parameter of the graph, (α). $\alpha = \{0.5, 1\}$. The height of a DAG is randomly generated from a uniform distribution with a mean value equal to $\sqrt{n}/\alpha$, and the width for each level is randomly selected from a uniform distribution with mean value equal to $\alpha \times \sqrt{n}$.
- Out degree of a node. $out\_degree = \{1, 2, 3, 4, 5\}$.
- The average workload of the given graph, i.e., $\overline{w}$, is set randomly from the range $[1000, 2000]$, and the average workload of each task is set from a uniform distribution with the range $[0, 2 \times \overline{w}]$. Different with initializing computation cost of each task on each processor with random numbers in[15], the computation cost is determined by both task load and processing speed.
- The processor's speed is randomly selected from $[5, 20]$.
- The λ value of each processor, communication link, or switch is set with random values from the range $\{2, 4, 6, 8, 10\} \times 10^{-6}$.
- The network bandwidth is set with 1, and *oversubscription* is set with 1:1.

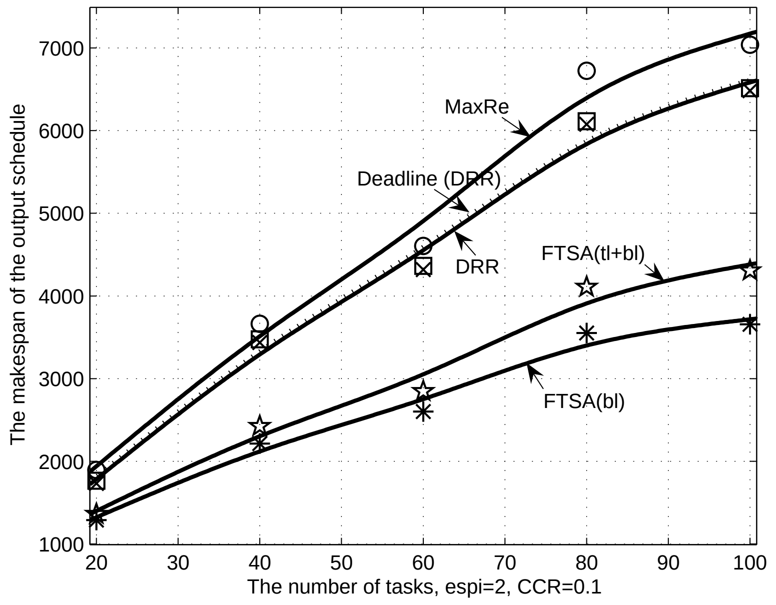
5.2 Execution time

FTSA (Fault tolerant scheduling algorithm) algorithm[3] is introduced as a fault tolerant extension of the classic HEFT algorithm[15]. In FTSA, at each step of the scheduling process, the free task x with the highest priority is simulated by mapping on all processors. The first $\varepsilon + 1$ (ε is the number of failures that FTSA can tolerate, i.e. *espi* in the figure) processors that allow the minimum finish time are scheduled.

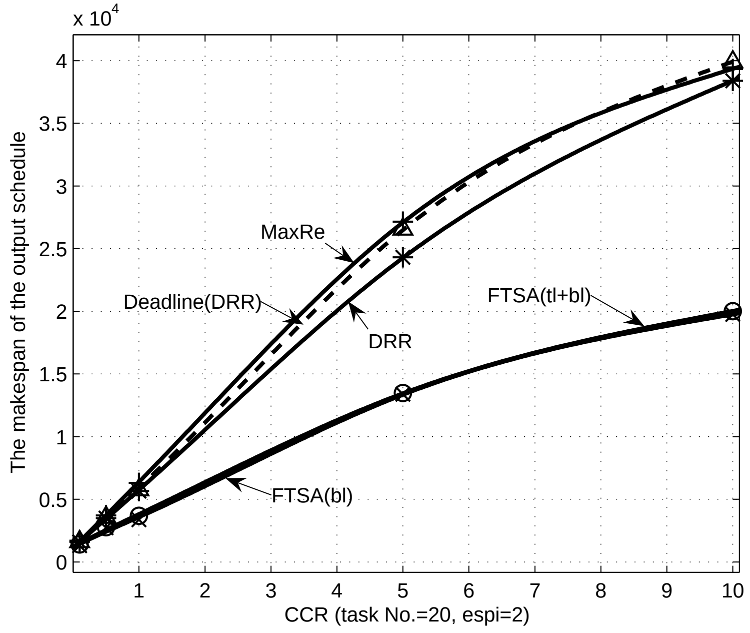
Fig. 5 shows the makespan of four algorithms' output schedule. All the experiment results are the average value from 100 times of executions. We evaluated the average makespan of FTSA(bl), FTSA(tl+bl), MaxRe and DRR algorithms, with $CCR = 0.1$ and $\varepsilon = 2$. Tolerating 2 failures by FTSA algorithms is transformed into the probability value of at most ε failures occur.

As shown in Fig. 5(a), the line of dashes denotes the deadline requirement of the DRR algorithm, and its value slightly shorter than MaxRe's makespan. It is shown that the MaxRe's schedule has the longest makespan. This is because the MaxRe algorithm is designed with one objective of minimizing the resource usage, and without taking into account the execution time. The makespan of DRR's schedule is slightly shorter than the deadline constraint. This proves that DRR algorithm can satisfy the deadline requirement whenever feasible. The FTSA(bl) and FTSA(tl+bl) algorithms shows the shortest makespan, which is because both two algorithms only take the execution time as the objective. And the makespan of FTSA(bl)'s schedule is shorter than FTSA(tl+bl). With the increasing number of tasks, the makespan is also continuously increasing.

In Fig. 5(b), set the task number with 20, and set the number of failures tolerated by FTSA with 2. With different CCR values, we observe the four algorithms'



(a)



(b)

図 5 The makespan comparision: (a)Different number of tasks; (b)Different CCR values;

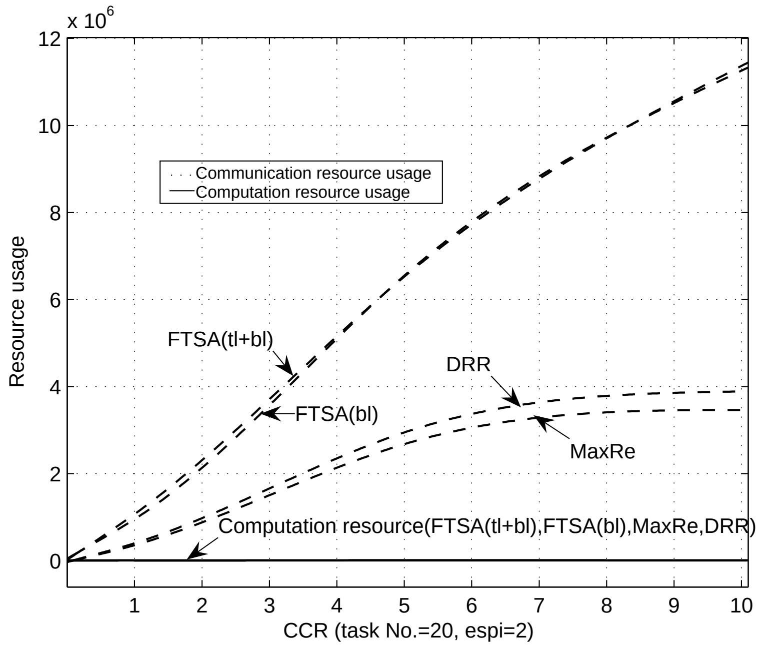


図 7 The resource usage with different CCR values

makespan. The MaxRe also shows the longest makespan in all algorithms. And the DRR algorithm can find the schedule with shorter makespan than deadline constraint. For FTSA(bl) and FTSA(tl+bl), the FTSA(bl) also has a shorter average makespan than FTSA(tl+bl), however, with the communication cost becoming gradually larger, the difference is not that big.

5.3 Resource usage

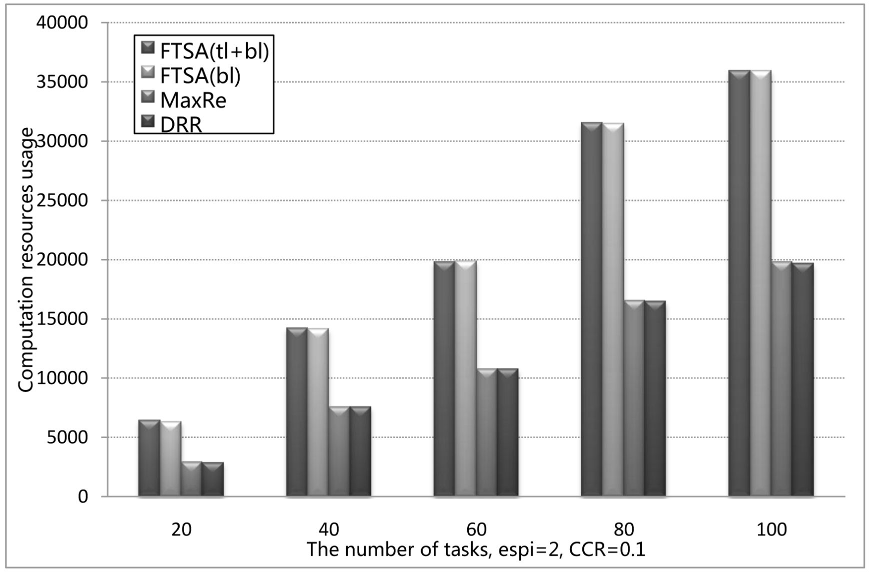
With $CCR = 0.1$ and $\varepsilon = 2$, we evaluate the resource usage of the four algorithms in the second experiment. As shown in Fig. 6, the computation resource usage (formula 5) and communication resource usage (formula 6) are shown respectively. The deadline constraint for DRR algorithm is the same with the first experiment. With the increasing number of tasks, both the computation resource usage and the communication resource usage are continuously increasing. For computation resources in Fig. 6(a), the FTSA(bl) and the FTSA(tl+bl) algorithms consume almost the same amount of resource, while the MaxRe and DRR algorithms use much less resource than FTSA. With different number of tasks, the DRR algorithm saves about 54.4%, 44.6%, 45.8%, 47.7%, and 45.4% computation resources, respectively. And compare the computation resource usage of MaxRe and DRR, for the deadline constraint reason, the DRR algorithm always select the processors within the subdeadline constraint, thus it consumes slightly less resource than MaxRe. For the communication resources in Fig. 6(b), with different number of tasks, the DRR algorithm saves about 73.2%, 66.1%, 62.7%, 62.1% and 54.8% communication resources, respectively. Compare the communication resources consumed by DRR and MaxRe, because the deadline constraint make the task not always scheduled on the processors with higher reliability but under the subdeadline constraint, the DRR use slightly more communication resource than MaxRe algorithm. And in summary, for the deadline constraint reason, the DRR uses more resources than the MaxRe algorithm.

In Fig. 7, the resource usage is evaluated based on different CCR values. Compared with the big amount on communication resources, the difference on computation resources is not so big, and does not clearly shown in Fig. 7. However, their actual difference is similar with Fig. 6(a). With the increasing of CCR value, it is obvious that the DRR and MaxRe algorithms can save more communication resources than FTSA algorithm. And for the deadline constraint reason, the DRR consumes more resources than MaxRe resources.

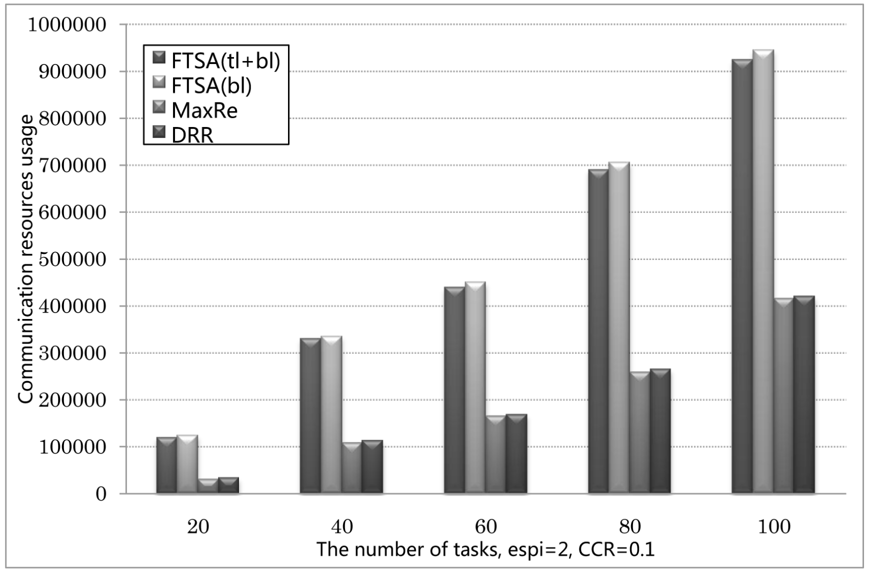
6. Conclusion

In this study, we firstly state the problem of resource minimizing scheduling with constraint on deadline and reliability. We consider the one-port model and three-tier network topology in the communication, which makes our solution more realistic than

Vol.2010-AL-132 No.3
2010/11/19



(a)



(b)

図 6 The resource usage: (a) describes the computation resource usage; (b) describes the communication resource usage;

previous works. We also study the performance of two different task priority methods, the experiment shows that *bottom level* based task priority has a better performance on makespan than *bottom level+top level* based one.

We design the DRR scheduling algorithm to solve the proposed problem. The analysis demonstrate that DRR can meet user's deadline and reliability constraint. And the experiment results show that the DRR algorithm can save almost 50% computation resources and 70% communication resources than *FTSA(tl)* and *FTSA(tl+bl)* algorithms.

Acknowledgment

The first and second authors of this research are supported by the governmental scholarship from China Scholarship Council.

参 考 文 献

1) A. Benoit, M. Hakem, Y. Robeert, Contention awareness and fault-tolerance scheduling for precedence constrained tasks in heterogeneous systems. Parallel Computing, vol. 35(2), pp.83-108, 2009.
2) 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.
3) A. Benoit, M. Hakem, Y. Robeert, Fault Tolerant Scheduling of Precedence Task Graphs on Heterogeneous Platforms. In Proceedings of the 10th Workshop on Advances in Parallel and Distributed Computational Models, APDCM08, Miami, USA, 2008.
4) 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.
5) M. Al-Fares, A. Loukissas, A. Vahdat, A Scalable, Commodity Data Center Network Architecture. In Proceedings of the ACM SIGCOMM conference on Data communication (SIGCOMM2008), pp. 63-74, Washington, 2008.
6) I. Foster, Yong Zhao, Ioan Raicu, et al., Cloud Computing and Grid Computing 360-Degree Compared. In 2008 Grid Computing Environments Workshop, pp. 1-10, 2008.
7) A. Dogan, F. Ozguner, Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. The Computer Journal, vol. 48(3)pp. 300-314, 2005.
8) M. Hakem, F. Butelle, Reliability and Scheduling on Systems Subject to Failures. International Conference on Parallel Processing, ICPP2007, Xi'an, 2007.
9) X. Qin, H. Jiang, A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. Journal of Parallel and Distributed Computing, vol. 65(8)pp. 885-900, 2005.
10) L. Zhao, Y. Ren, Y. Xiang et al. Fault tolerant scheduling with dynamic number of replicas in heterogeneous system. 12th IEEE International Conference on High Performance Computing and Communications, HPCC2010, Melbourne, 2010.
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(IPDPS2002). pp. 98-104, Florida, US, 2002.
12) G. Manimaran and C.S.R. Murthy,A Fault-Tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems and Its Analysis. IEEE Transactions on Parallel and Distributed Systems, vol. 9(11)pp. 1137-1152, 1998.
13) Y. Jia, R. Buyya, C.K. Tham, Cost-based scheduling of scientific workflow applications on utility grids. First International Conference on e-Science and Grid Computing. pp. 139-147, Melbourne, 2005.
14) 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.
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.