Far East Journal of Applied Mathematics
Volume 33, Number 3, December 2008, Pages 395-409
Published online: November 19, 2008
This paper is available online at http://www.pphmj.com

PARALLEL TASKS SCHEDULING IN GRID ENVIRONMENTS

JIANN-FU LIN

Department of Management Information System
Takming University of Science and Technology
No. 56, Sec. 1, HuanShan Rd., NeiHu
Taipei, 11451, Taiwan
e-mail: alfu@takming.edu.tw

Abstract

With new technological advances in parallel processing and distributed systems, more and more problems in Grid environments are being discussed and are gradually setting a new trend of scientific applications. In this paper, we will investigate the problem of non-preemptively scheduling independent parallel tasks in a Grid environment, in which the Grid environment consists of several machines with each machine containing a number of identical processors and each parallel task could only be processed in a single machine with its required number of processors simultaneously. Message communicating among processors is unavoidable whenever a task is processed in parallel. Hence, communication overhead among processors is taken into consideration in this problem. Such a problem of finding an optimal schedule is NP-hard. Hence, we propose a heuristic scheduling algorithm to this problem and analyze its performance bound.

1. Introduction

The problems of scheduling tasks in a single machine with multiple processors have been extensively studied for decades. In the conventional

2000 Mathematics Subject Classification: 68Wxx, 41-XX.

Keywords and phrases: Grid environments, parallel task scheduling, communication overhead, performance bound.

Received August 7, 2008

scheduling approach, each task is assumed that it can be processed on only one processor at a time. However, with new technological advances in parallel processing, the conventional task scheduling problem has evolved to become parallel task scheduling problem [3, 9, 10, 16, 17]. In the parallel tasks scheduling problem, each task can be assigned on more than one processor at a time. This problem assumes that there are n parallel tasks, $T_1, T_2, ..., T_n$, to be scheduled in a single machine with multiple identical processors, and each task T_i has its maximum degree of parallelism Δ_i and computation requirement t_i. The maximum degree of parallelism Δ_i means that a task T_i may be scheduled to process on up to Δ_i processors and this degree of parallelism, once decided for T_i, will not be altered during its processing. The computation requirement t_i denotes that the processing time of the task T_i if it is processed on only one processor, that is, the processing time required by a task is equal to the computation requirement if it is processed on only one processor. Thus, under linear speedup assumption, the processing time required by T_i will be (t_i/δ_i) if the task T_i is scheduled to be processed on δ_i processors, where δ_i is called the *scheduled parallelism* of T_i and $1 \le \delta_i \le \Delta_i$. For this problem type, a schedule is feasible if the scheduled parallelism of each task is not greater than its maximum degree of parallelisms. A feasible schedule is called an *optimal schedule* if it has the earliest finish time. Besides, if tasks are scheduled from time 0, the finish time of a schedule is also the length of that schedule. Thus, an optimal schedule also means that it has the shortest schedule length. Finding an optimal schedule for this problem type in a single machine with p identical processors is NP-hard [3], where $p \ge 2$. Hence, polynomial time heuristic scheduling algorithms are usually used to get approximate solutions. The performance evaluation of a heuristic algorithm is the ratio of a heuristic schedule length to the optimal schedule length. A heuristic scheduling algorithm H is said to have a performance bound of γ if $(S_H/S_{OPT}) \le \gamma$ for all problem instances, where S_H and S_{OPT} denote the schedule lengths of heuristic algorithm H and an optimal schedule, respectively.

If message communicating among processors is negligible, any parallel task can be assigned to the requisite number of processors simultaneously so that the total processing time is reduced. This situation may occur in shared memory systems [4] and this is the basis of many heuristic algorithms that do not consider communication overhead in scheduling parallel tasks. Under the above assumption, Wang and Cheng [16] showed that the performance bound of scheduling parallel tasks in a single machine with p identical processors by applying the concept of Graham's List Scheduling (LS) algorithm [7] is $(\Delta + (p - \Delta)/p)$, where $\Delta = \max\{\Delta_i \mid i = 1, 2, ..., n\}$. Later, Wang and Cheng [17] proposed the Earliest Completion Time (ECT) algorithm for the same problem and derived the performance bound as $(3 - 2/p)$. On the other hand, if message communicating among processors cannot be neglected [4] such as message passing systems, scheduling algorithms need to consider the communication overhead among processors. The problem of scheduling parallel tasks with the consideration of communication overhead in a single machine with p identical processors is also an NP-hard problem. Lin et al. [10] gave a communication overhead assumption and proposed the Largest Scheduled Parallelism First (LSPF) algorithm for the problem of scheduling independent parallel tasks with communication overhead. They showed that the performance bound of the LSPF is $(4 + 2/k - 2/p)$, where p is the number of processors in the machine and k is a given positive constant.

The above studies only discussed the problems of scheduling parallel tasks on a single machine with multiple identical processors. With the great improvements in the performance of wide area network and the technologies of computers, the Grid environment has emerged as a promising computing platform that can support the execution of next generation scientific applications and will open up avenues in many research fields [6, 15]. Grid environment is a large virtual organization that integrates a large amount of distributed resources and high performance computing capabilities into a super service, which can provide huge computing services, storage capability and so on. Grid environment allows the use of geographically distributed computing

systems belonging to multiple organizations as a single system. Thus, for simplicity, a Grid environment can be seen as a multi-machine environment in which each machine contains multiple processors. In such a multi-machine environment, users submit their tasks from any one of machine and the scheduler checks whether the tasks can be processed on the available resources and meet their requirements before really assigning them. Thus, instead of processing locally, the scheduler dispatches tasks to the remote machines. To achieve the potentials of a multi-machine environment, an effective and efficient scheduling framework within a multi-machine environment is fundamentally important.

With the importance of scheduling tasks in a Grid environment, several studies have discussed the problem of scheduling multiprocessor tasks [1] in such an environment. In 2001, Braun et al. [2] made a performance comparison among eleven heuristic algorithms for scheduling a set of independent tasks onto heterogeneous distributed computing systems by simulation. In 2004, Martino and Mililotti [11] developed a simulation Grid environment to study the usefulness of genetic algorithms for scheduling tasks in a distributed group of parallel machines. They found that the genetic algorithm for scheduling 32 tasks does not converge to the optimal schedule within a limited number of trial performed, only a sub-optimal schedule could be got. In 2005, Weng and Lu [18] proposed a heuristic to schedule independent tasks in the Grid environment. According to the experimental results, they showed that their heuristic algorithm could obtain a better performance compared to the other four existing heuristic algorithms. In 2007, Pascual et al. [12] discussed the problem of scheduling multiprocessor tasks (rigid parallel tasks) in a Grid environment, in which each machine with the same number of identical processors. They proposed the Multi-Organization Load Balancing Algorithm (MOLBA) and derived the performance bound as 3 if the last completed task requires at most half of the available processors and 4 in the general case. Recently, Lin [8] discussed a similar problem and proposed the More Processor Required first (MPR) scheduling algorithm. He showed that the performance of the MPR is

bounded by $\left(3-1/\sum_{j=1}^{M} p_j\right)$, where p_j is the number of processors in machine m_j and M is the number of machines in the Grid environment.

The above studies discussed the problems of scheduling multiprocessor tasks in a Grid environment. In this paper, we consider n independent parallel tasks to be non-preemptively scheduled with the consideration of communication overhead in a Grid environment with multiple machines, in which each machine contains a number of processors and all processors in every machine are identical. The problem of scheduling parallel tasks is quite similar to the problem of scheduling multiprocessor tasks. The difference is that the parallelism of the multiprocessor task is rigid but the parallelism of the parallel task is malleable. The problem of scheduling independent parallel tasks with communication overhead in a Grid environment is also NP-hard because scheduling independent non-preemptive parallel tasks without considering communication overhead in a single machine with multiple identical processors, which is a special case of this problem, has been known as an *NP-hard problem* [3]. Therefore, we are interested in developing a polynomial time heuristic algorithm and in deriving its performance bound. The rest of this paper is organized as follows. Under the consideration of communication overhead, a policy is employed in calculating the maximum degree of advantageous parallelism of each parallel task in each machine in Section 2. In Section 3, we propose the Just Fit scheduling algorithm and discuss its performance bound. Finally, some concluding remarks are given in Section 4.

2. The Maximum Degree of Advantageous Parallelism

In this paper, we assume that a set of n independent non-preemptable parallel tasks $\boldsymbol{T} = \{T_1, T_2, ..., T_n\}$ are to be scheduled in an M-machine environment, where each machine m_j, $j = 1, 2, ..., M$, consists of p_j identical processors and each parallel task T_i can only be processed in a single machine with its requisite number of processors

simultaneously. Each parallel task is assumed processable in any machine with its maximum degree of parallelism, namely, $\max\{\Delta_i \mid i = 1, 2, ..., n\} \leq \min\{p_j \mid j = 1, 2, ..., M\}$. For simplicity, the term "parallel task" is sometimes denoted by "task". Whenever a task is processed in parallel in a machine, message communicating among processors is unavoidable. Generally, communication overhead depends on the characteristics of a machine and a task, and the degree of parallelism adopted by a task [4, 5, 13, 14]. As a consequence, an assumption of the average communication overhead among processors of a task in machine m_j is given as $\text{Comm}(x, j) = c_j x^{k_j}$ [10], where x is the degree of parallelism adopted by a task, and c_j and k_j are two given positive constants which depend on the characteristics of a machine. Thus, the total time required for processing the task T_i with a degree of parallelism x in machine m_j is defined as below:

$$f(x, i, j) = \begin{cases} t_i, & \text{if } x = 1, \\ \dfrac{t_i}{x} + c_j x^{k_j}, & \text{if } x > 1, \end{cases}$$

where t_i is the computation requirement of task T_i.

It may be quite intuitive that it is more advantageous to execute a task with more processors. However, the more processors involved for processing a task, the more message communicating also incurred. Hence, the more processors involved for processing a task is not necessarily shortening the total processing time. In fact, the total processing time of a task for different numbers of processors is typically decreasing if it does not exceed a certain smallest number of processors [5]. Therefore, we are going to find out the number of processors for which the total processing time of a task is the smallest in a particular machine. By simple calculus, we can get that

$$\frac{\partial f(x, i, j)}{\partial x} = \begin{cases} 0, & \text{if } x = 1 \\ \dfrac{-t_i}{x^2} + c_j k_j x^{k_j - 1}, & \text{if } x > 1 \end{cases}$$

and

$$\frac{\partial^2 f(x,\, i,\, j)}{\partial x^2} = \begin{cases} 0, & \text{if } x = 1, \\ \dfrac{2t_i}{x^3} + c_j k_j (k_j - 1) x^{k_j - 2}, & \text{if } x > 1. \end{cases}$$

Since $\dfrac{\partial f(x,\, i,\, j)}{\partial x} = 0$ and $\dfrac{\partial^2 f(x,\, i,\, j)}{\partial x^2} > 0$ at $x = \left(\dfrac{t_i}{c_j k_j}\right)^{1/(k_j+1)}$, $f(x,\, i,\, j)$ reaches its minimum value at $x = \left(\dfrac{t_i}{c_j k_j}\right)^{1/(k_j+1)}$. For the reason that the degree of parallelism adapted by the task T_i must be an integer, either $\left\lfloor \left(\dfrac{t_i}{c_j k_j}\right)^{1/(k_j+1)} \right\rfloor$ or $\left\lceil \left(\dfrac{t_i}{c_j k_j}\right)^{1/(k_j+1)} \right\rceil$ will make T_i with the smallest total processing time if it is processed in machine m_j. Here, we will always choose $\left\lfloor \left(\dfrac{t_i}{c_j k_j}\right)^{1/(k_j+1)} \right\rfloor$ as the maximum degree of advantageous parallelism of the task T_i even though the degree of parallelism $\left\lceil \left(\dfrac{t_i}{c_j k_j}\right)^{1/(k_j+1)} \right\rceil$ will really lead to the smallest total processing time. The reasons are: we try to remain more processors in a machine for the allocation of the other unassigned tasks and the less processors used, the less communication overhead required. In addition, the degree of parallelism adapted by the task T_i must obey the restriction that it cannot be greater than its maximum degree of parallelism Δ_i. Thus, $\phi_{i|j} = \min\left(\left\lfloor \left(\dfrac{t_i}{c_j k_j}\right)^{1/(k_j+1)} \right\rfloor, \Delta_i\right)$ is chosen as the maximum degree of advantageous parallelism of the task T_i for processing in the machine m_j. The procedure of calculating maximum degree of advantageous parallelism of task T_i for processing in the machine m_j is described below.

Procedure maximum-advantageous-parallelism

{ Let $\phi_{i|j} = \min\left(\left\lfloor\left(\frac{t_i}{c_j k_j}\right)^{1/(k_j+1)}\right\rfloor, \Delta_i\right)$ be the maximum degree of advantageous parallelism of task T_i for processing in the machine m_j, where $1 \le i \le n$ and $1 \le j \le M$.

}

3. The Just Fit Scheduling Algorithm

In order to get a higher utilization of processors in each machine in the Grid environment, an intuitive way is trying to let processors busy all the time, that is, do not let any processor idle until tasks are all assigned. Hence, we propose the Just Fit (JF) scheduling algorithm for the problem of non-preemptively scheduling independent parallel tasks with communication overhead in a Grid environment with M-machine, which let processors in each machine busy all the time until tasks are all assigned. The major policy of the JF scheduling algorithm is that it always chooses the first task from the tasks list and assigns the chosen task to the machine which owns the maximum number of free processors at this moment. However, the number of free processors in the selected machine is not always greater than or equal to the degree of maximum advantageous parallelism of the chosen task. In order to get a higher utilization of processors in each machine, processors allocation process needs to be flexible, that is, the number of processors really allocates to a task is adjusted depending on the state of that machine. Hence, the processors allocation process of the JF scheduling algorithm is that if the number of free processors f_j in the machine m_j is not less than the maximum degree of advantageous parallelism $\phi_{i|j}$ of the task T_i, the machine m_j allocates $\phi_{i|j}$ processors to task T_i; otherwise, the machine m_j only allocates f_j processors to task T_i. After processor allocation process, the task T_i starts to be executed and then to be removed from the tasks list. This process continues until no more tasks unassigned in

the tasks list. The JF scheduling algorithm is described as follows:

Algorithm JF

{

Input the computation requirements t_i and maximum degree of parallelisms Δ_i of parallel tasks T_i, where $i = 1, 2, ..., n$;

While (task list is not empty) **do**

{

Choose the first task T_i from the task list;

Wait until there exists at least one machine with free processors;

Choose the machine m_i, which has the maximum number of free processors f_j at this moment;

Call the *maximum-advantageous-parallelism* procedure to calculate the maximum degree of advantageous parallelism $\phi_{i|j}$ of the parallel task T_i for processing in the machine m_j;

If $(f_j \geq \phi_{i|j})$ **then** the machine m_j allocates $\phi_{i|j}$ processors to T_i;

else the machine m_j allocates f_j processors to T_i;

Execute T_i, and remove T_i from the task list;

}

}

Before showing the performance bound of the JF scheduling algorithm, an assumption and symbol definitions are given as follows: Tasks are assumed to be assigned from time 0, and S_{JF} and S_{OPT} denote the finish times of the JF schedule and an optimal schedule of the task set $\boldsymbol{T}$, respectively. Since tasks are scheduled from time 0, S_{JF} and S_{OPT} can also be seen as the schedule lengths of the JF schedule and an optimal schedule of the task set $\boldsymbol{T}$, respectively. Let $\delta_{i|j}$ be the number of processors that the machine m_j really allocates to the task T_i, that is, $\delta_{i|j}$ is the scheduled parallelism of the task T_i in the machine m_j. In addition, from the processor allocation process of the JF scheduling algorithm, it is obvious that $\delta_{i|j} = \min\{f_j, \phi_{i|j}\}$.

Lemma 1. $\phi_{i|j} \times f(\phi_{i|j}, i, j) \le \left(1 + \frac{1}{k_j}\right) t_i$, *where* $1 \le i \le n$ *and* $1 \le j \le M$.

Proof. Since

$\phi_{i|j} = \left\lfloor \left(\frac{t_i}{c_j k_j}\right)^{1/(k_j+1)} \right\rfloor$, $\phi_{i|j} \le \left(\frac{t_i}{c_j k_j}\right)^{1/(k_j+1)}$. That is, $c_j \phi_{i|j}^{k_j+1} \le \frac{t_i}{k_j}$.

Thus,

$$\phi_{i|j} \times f(\phi_{i|j}, i, j) = \phi_{i|j} \times \left(\frac{t_i}{\phi_{i|j}} + c_j \phi_{i|j}^{k_j}\right) \le \left(1 + \frac{1}{k_j}\right) t_i.$$

Lemma 2. $\delta_{i|j} \times f(\delta_{i|j}, i, j) \le \left(1 + \frac{1}{k_j}\right) t_i$.

Proof. Since $\delta_{i|j} = \min\{f_j, \phi_{i|j}\}$, we know that $1 \le \delta_{i|j} \le \phi_{i|j}$,

$$\delta_{i|j}^{k_j+1} \le \phi_{i|j}^{k_j+1}$$

$$\Rightarrow (t_i + c_j \delta_{i|j}^{k_j+1}) \le (t_i + c_j \phi_{i|j}^{k_j+1})$$

$$\Rightarrow \delta_{i|j}\left(\frac{t_i}{\delta_{i|j}} + c_j \delta_{i|j}^{k_j}\right) \le \phi_{i|j}\left(\frac{t_i}{\phi_{i|j}} + c_j \phi_{i|j}^{k_j}\right)$$

$$\Rightarrow \delta_{i|j} \times f(\delta_{i|j}, i, j) \le \phi_{i|j} \times f(\phi_{i|j}, i, j).$$

By Lemma 1, $\delta_{i|j} \times f(\delta_{i|j}, i, j) \le \left(1 + \frac{1}{k_j}\right) t_i$.

Lemma 3. *If the task* T_i *starts to be processed at time* τ_i *in the machine* m_b, *then none of the processors are free in any machine* m_j *before the time* τ_i, *where* $1 \le i \le n, 1 \le b, j \le M$.

Proof. According to the processors allocation process of the JF scheduling algorithm, none of the processors are free in any machine before the time τ_i; otherwise; the task T_i should have been started earlier than the time τ_i.

Lemma 4. *If the task* T_z *is finished at time* S_{JF} *in the machine* m_b, *then none of the processors are free before the time* $(S_{JF} - f(\delta_{z|b}, z, b))$ *in any machine* m_j, *where* $1 \le z \le n$ *and* $1 \le b, j \le M$.

Proof. $(S_{JF} - f(\delta_{z|b}, z, b))$ is the starting time of T_z in the machine m_b. By Lemma 3, none of the processors are free in any machine m_j before the time $(S_{JF} - f(\delta_{z|b}, z, b))$.

Lemma 5. $f(\delta_{i|b}, i, b) \le \Delta_i \cdot S_{OPT}$, *where* S_{OPT} *is the length of an optimal schedule,* $\Delta = \max\{\Delta_a \mid a = 1, 2, ..., n\}$, $1 \le i \le n$ *and* $1 \le b \le M$.

Proof. Since the scheduled parallelism of task T_i in the machine m_b is $\delta_{i|b}$, $1 \le \delta_{i|b} \le \phi_{i|b}$, we have $f(\phi_{i|b}, i, b) \le f(\delta_{i|b}, i, b) \le f(1, i, b) \le t_i \le \Delta_i(t_i/\Delta_i)$. For that $(t_i/\Delta_i) \le S_{OPT}$, we can get that $\Delta_i(t_i/\Delta_i) \le \Delta_i \cdot S_{OPT}$.

Lemma 6. *The performance bound of JF scheduling algorithm on the problem of non-preemptively scheduling independent parallel tasks with communication overhead is* $(1 + 1/\kappa) + \left(1 - 1/\sum_{j=1}^{M} p_j\right)\Delta_z$, *where* $1 \le z \le n$.

Proof. Assume that the task T_z finishes at time S_{JF} in the machine m_b. $S_{JF} \le \left[\sum_{i=1}^{n} \delta_{i|j} \times f(\delta_{i|j}, i, j) + \left(\sum_{j=1}^{M} p_j - \delta_{z|b}\right) \times f(\delta_{z|b}, z, b)\right] / \sum_{j=1}^{M} p_j$, where $1 \le z \le n$ and $1 \le b, j \le M$.

$$S_{JF} \le \left[\sum_{i=1}^{n} \delta_{i|j} \times f(\delta_{i|j}, i, j) + \left(\sum_{j=1}^{M} p_j - 1\right) \times f(\delta_{z|b}, z, b)\right] / \sum_{j=1}^{M} p_j.$$

Since $\sum_{i=1}^{n} t_i / \sum_{j=1}^{M} p_j \le S_{OPT}$, we have $\sum_{i=1}^{n} \delta_{i|j} \times f(\delta_{i|j}, i, j) / \sum_{j=1}^{M} p_j \le (1 + 1/\kappa) S_{OPT}$.

By Lemma 5, $S_{JF} \le (1 + 1/\kappa) S_{OPT} + \left(1 - 1/\sum_{j=1}^{M} p_j\right) \times \Delta_z \cdot S_{OPT}$.

Hence, $\frac{S_{JF}}{S_{OPT}} \le (1+1/\kappa) + \left(1 - 1/\sum_{j=1}^{M} p_j\right)\Delta_z$.

Theorem 1. *The derived performance bound of JF scheduling algorithm on the problem of non-preemptively scheduling independent parallel tasks with communication overhead falls between* $\left(2+1/\kappa - 1/\sum_{j=1}^{M} p_j\right)$ *and* $(1+1/\kappa) + \left(1 - 1/\sum_{j=1}^{M} p_j\right)\Delta$, *where* $\Delta = \max\{\Delta_i \mid i = 1, 2, \ldots, n\}$.

Proof. Since $1 \le \Delta_i \le \Delta$, the results are obvious.

Example 1. Consider a simple case that there are 3 independent parallel tasks scheduling in an environment with 2 machines m_1 and m_2. Let $p_1 = p_2 = p$; $k_1 = k_2 = \kappa$; $c_1 = c_2 = c = t/p^{\kappa+3}$; $t_1 = ((p-1)/p)t - c(p-1)^{\kappa+1}$ and $\Delta_1 = p - 1$; $t_2 = t/p$ and $\Delta_2 = p$; $t_3 = t - cp^{\kappa+1}$ and $\Delta_3 = p$. A possible JF schedule and an optimal schedule are illustrated in Figure 1 (a) and (b), respectively.

$$\text{Thus, } \frac{S_{JF}}{S_{OPT}} = \frac{t - cp^{\kappa+1}}{t/p} = \frac{(pt - cp^{\kappa+2})}{t} = \frac{(t - cp^{\kappa+2}) + (p-1)t}{t}$$

$$= \left(1 - \frac{cp^{\kappa+2}}{t}\right) + (p-1).$$

Since $p = \Delta = \max\{\Delta_1, \Delta_2\}$,

$$\left(1 - \frac{cp^{\kappa+2}}{t}\right) + (p-1)$$

$$= \left(1 - \frac{cp^{\kappa+2}}{t}\right) + \left(1 - \frac{1}{p}\right)p = \left(1 - \frac{1}{p}\right) + \left(1 - \frac{1}{p}\right)\Delta, \text{ because } c = \frac{t}{p^{\kappa+3}}$$

$$= \left(1 - \frac{1}{p}\right) + \left(1 - 2/\sum_{j=1}^{2} p_j\right)\Delta \le \left(1 - \frac{1}{p}\right) + \left(1 - 1/\sum_{j=1}^{2} p_j\right)\Delta$$

$$\le \left(1 + \frac{1}{\kappa}\right) + \left(1 - 1/\sum_{j=1}^{2} p_j\right)\Delta.$$

This example shows that the performance bound derived in Theorem 1 is not tight.

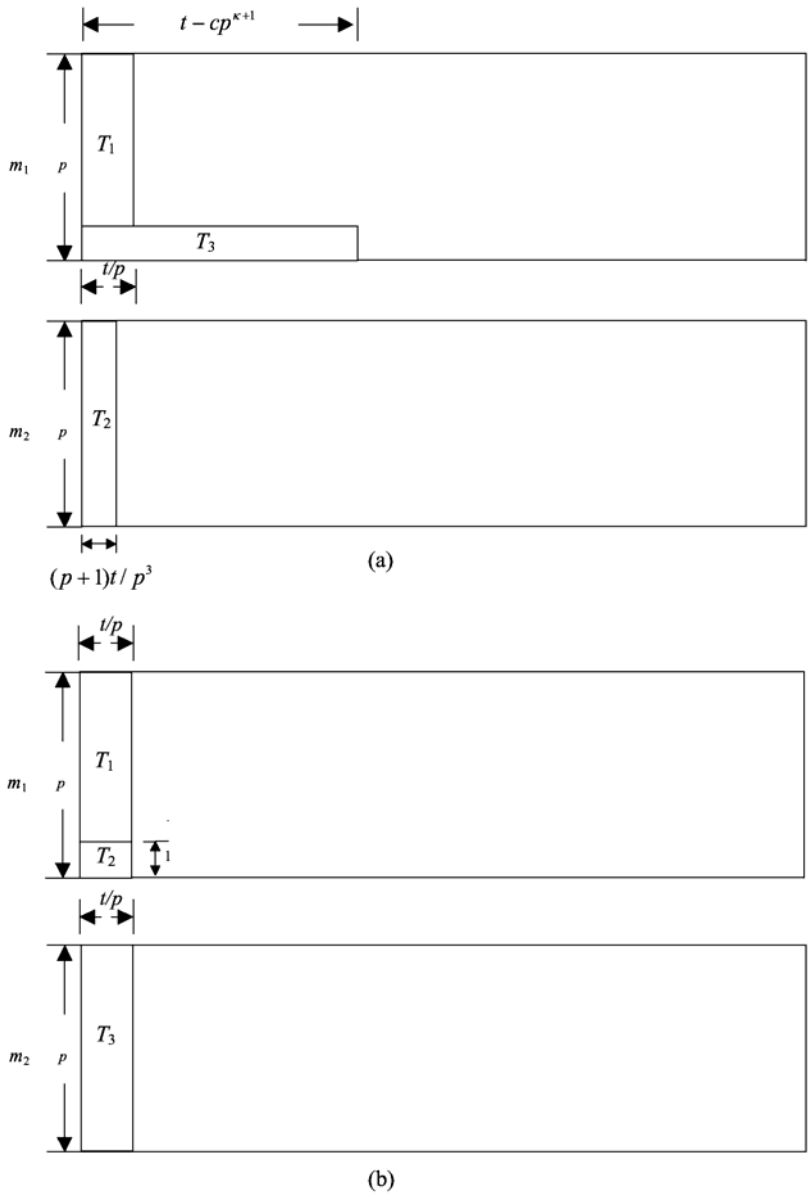

Figure 1. (a) A possible JF schedule of Example 1 and (b) an optimal schedule of Example 1.

4. Conclusion

In this paper, the problem of scheduling independent parallel tasks with the consideration of communication overhead in a Grid environment with multiple machines is discussed. Considering the utilization of processors in each machine, we developed the JF scheduling algorithm for such a problem and showed that the performance bound is $\left(1+\frac{1}{\kappa}\right)+\left(1-1/\sum_{j=1}^{M} p_j\right)\Delta$, where $\kappa = \min\{k_j \mid j = 1, 2, ..., M\}$ and $\Delta = \max\{\Delta_i \mid i = 1, 2, ..., n\}$. Although the JF scheduling algorithm could get a high utilization on processors in each machine, the performance of the JF scheduling algorithm completely depends on the last finished task. In addition, an example illustrated that the derived performance bound is not tight.

References

[1] J. Blazewicz, M. Drabowski and J. Weglarz, Scheduling multiprocessor tasks to minimize schedule length, IEEE Trans. Comput. 35(5) (1986), 389-393.

[2] T. D. Braun et al., A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distributed Computing 61 (2001), 810-837.

[3] J. Du and J. Y. Leung, Complexity of scheduling parallel task system, SIAM J. Discrete Math. 2 (1989), 473-487.

[4] H. El-Rewini and M. Abd-El-Barr, Scheduling and Task Allocation, Advanced Computer Architecture and Parallel Processing, John Wiley and Sons, Inc., (2005) pp. 235-265.

[5] S. M. Figueira, Optimal partitioning of nodes to space-sharing parallel tasks, Parallel Computing 32 (2006), 313-324.

[6] I. Foster and C. Kesselman, The Grid 2: Blueprint for a New Computing Infrastructure, Elsevier, Amsterdam, 2004.

[7] R. L. Graham, Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math. 17(2) (1969), 416-429.

[8] J. F. Lin, Performance bounds of scheduling multiprocessor tasks in a multiple machines environment, submitted to Information Science (under review after revising).

[9] J. F. Lin and S. J. Chen, Performance bounds on scheduling parallel tasks with setup time on hypercube systems, Informatica 19 (1995), 313-318.

[10] J. F. Lin, W. B. See and S. J. Chen, Performance bounds on scheduling parallel tasks with communication cost, IEICE Trans. Information Systems E78-D(3) (1995), 263-268.

[11] V. Di Martino and M. Mililotti, Sub optimal scheduling in a grid using genetic algorithms, Parallel Computing 30 (2004), 553-565.

[12] F. Pascual, K. Rzadca and D. Trystram, Cooperation in Multi-organization Scheduling, Euro-Par 2007, Rennes, France, August (2007), pp. 224-233.

[13] G. S. Sajjan, Array Processors, Advanced Computer Architectures, Taylor and Franics Group, 2006, pp. 167-220.

[14] J. P. Singh, J. L. Hennessy and A. Gupta, Scaling parallel programs for multiprocessors: methodology and examples, computer, IEEE Computer (1993), 42-50.

[15] B. Tierney, W. Johnston, J. Lee and M. Thompson, A data intensive distributed computing architecture for 'Grid' applications, Future Generation Computer Systems 16 (2000), 473-481.

[16] Q. Wang and K. H. Cheng, List scheduling of parallel tasks, Information Processing Letters 37(5) (1991), 291-297.

[17] Q. Wang and K. H. Cheng, A heuristic of scheduling parallel tasks and its analysis, SIAM J. Comput. 21(2) (1992), 281-294.

[18] C. Weng and X. Lu, Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid, Future Generation Computer Systems 21 (2005), 271-280.

■