A Polynomially Solvable Case of Scheduling Multiprocessor Tasks in a Multi-Machine Environment

Xiao Xin[a, *], Min Mou[b] and Guohua Mu[c]

College of Foreign Studies, Shandong Institute of Business and Technology, Yantai, 264005, China

[a]xinxiaoyt@hotmail.com, [b]mouminyt@hotmail.com, [c]muguohuayt@hotmail.com, [*]Corresponding author

Keywords: Parallel Processing, Scheduling, Multiprocessor tasks, Makespan, Polynomially solvable case.

Abstract. The problem of scheduling multiprocessor tasks in a multi-machine environment is considered. Each machine contains a number of identical processors. Each task requires a number of processors on a single machine for its processing. The objective is to minimize the overall task completion time, i.e. the makespan. The general problem has been known to be strongly NP-hard. A linear time optimal algorithm is presented for a special case of the problem where all the tasks have unit processing times and each task requires one or *k* (*k* is part of the input) processors. The small computational effort of the algorithm is valuable in some practical applications.

Introduction

Multiprocessor task scheduling problem is an important issue in a distributed computing environment [1]. The problem can be formally defined as follows. There are n independent multiprocessor tasks $T_1, T_2, \ldots, T_n$ which are to be processed on m machines $M_1, M_2, \ldots, M_m$. Machine M_i ($i = 1, 2, \ldots, m$) consists of ρ_i identical processors. All tasks are available at time zero. Task T_j ($j = 1, 2, \ldots, n$) requires δ_j processors on a single machine for its processing and its processing time is equal to t_j. The objective is to find a feasible schedule so as to minimize the overall task completion time, i.e. the *makespan*.

Multiprocessor task scheduling is an extension of scheduling tasks on uniprocessor machines. In the latter problem, each task T_j must be processed on a single uniprocessor machine without preemption. It is NP-hard even if there are only two machines [2], and strongly NP-hard for an arbitrary number of machines [3]. The former problem is therefore strongly NP-hard, too.

Blazewicz et al. [4] studied the problem of scheduling multiprocessor tasks in a single machine environment for minimizing makespan. They presented a linear time algorithm for a special case where all the tasks have unit processing times and each task requires one or k (k is part of the input) processors. They also presented a linear time algorithm for scheduling tasks with unit processing times requiring an arbitrary number of processors between 1 and k at a time where k is a fixed integer. Drozdowski [5] gave a detailed review on the problems of scheduling multiprocessor tasks in a single machine environment for optimizing various objective functions. Pascual et al. [6] and Rzadca [7] studied the problem of scheduling multiprocessor tasks in a multi-machine environment for minimizing makespan where all the machines have the same number of processors. Pascual et al. [6] proposed a 4-approximation algorithm and Rzadca [7] proposed a 3-approximation algorithm. Later, the problem of scheduling multiprocessor tasks in a multi-machine environment for minimizing makespan was studied in [8-10], and three 3-approximation algorithms were obtained. Lin and Liaw [11] proposed a 2.5-approximation algorithm for the problem of scheduling multiprocessor tasks in a multi-machine environment for minimizing makespan, under the assumption that $\delta \le \left\lceil \frac{\rho}{2} \right\rceil$, where $\delta = \max_j \delta_j$ and $\rho = \min_i \rho_i$.

In this paper, we study the problem of scheduling multiprocessor tasks in a multi-machine environment for minimizing makespan, allowing the machines have unequal number of processors. We present a linear time optimal algorithm for a special case of the problem where all the tasks have unit processing times and each task requires one or k (k is part of the input) processors on a single machine. The result generalizes the first algorithm presented in [4] for the single machine case.

The remainder of this paper is organized as follows. In Section 2, we present the algorithm. In Section 3, we prove the correctness of the algorithm. We conclude this paper in Section 4.

The Algorithm

For the purpose of easier descriptions, we assume a machine indexing for machines $M_1, M_2, \dots, M_m$ such that $\rho_{i-1} \le \rho_i$ holds and introduce $\rho_0 = 0$, $i = 1, 2, \dots, m$. Let n_1 denote the number of the tasks requiring one processor at a time. Similarly, let n_k denote the number of the tasks requiring k (k is not fixed) processors on a single machine at a time. Let OPT denote the makespan of an optimal schedule.

Let $LB_1 = \left\lceil n_k / \sum_{i=1}^{m} \lfloor \rho_i / k \rfloor \right\rceil$, and $LB_2 = \left\lceil (n_1 + kn_k) / \sum_{i=1}^{m} \rho_i \right\rceil$. Further, let $SOL = \max\{LB_1, LB_2\}$. In the next section, we will show that LB_1 and LB_2 are lower bounds on OPT, and $SOL = OPT$ indeed.

Algorithm $A1$:

Step 1. Calculate SOL as described above.

Step 2. Schedule the tasks requiring k processors on the machines which have at least k processors in time interval $[0, LB_1]$.

Step 3. Assign the tasks requiring one processor to the remaining free processors on the machines in time interval $[0, SOL]$.

We use the following two instances to give an example.

In the first instance, there are 3 machines M_1, M_2, M_3 which have 2, 4, 8 processors, respectively. There are 7 tasks requiring 3 processors, and 5 tasks requiring 1 processor. Hence, we have: $LB_1 = \lceil 7/(\lfloor 4/3 \rfloor + (\lfloor 8/3 \rfloor) \rceil = 3$, $LB_2 = \lceil (5 + 3 \times 7)/(2+4+8) \rceil = 2$, $SOL = 3$. Algorithm *A1* first schedules the tasks requiring 3 processors in time interval $[0,3]$: It assigns 6 tasks requiring 3 processors on machine M_3. It also assigns 1 task requiring 3 processors on machine M_2. Algorithm *A1* then schedules the tasks requiring 1 processor in time interval $[0,3]$: It assigns 5 tasks requiring 1 processor on machine M_3. All the tasks complete no later than time 3.

In the second instance, there are 3 machines M_1, M_2, M_3 which have 2, 4, 8 processors, respectively. There are 5 tasks requiring 3 processors, and 15 tasks requiring 1 processor. Hence, we have: $LB_1 = \lceil 5/(\lfloor 4/3 \rfloor + (\lfloor 8/3 \rfloor) \rceil = 2$, $LB_2 = \lceil (15 + 3 \times 5)/(2+4+8) \rceil = 3$, $SOL = 3$. Algorithm *A1* first schedules the tasks requiring 3 processors in time interval $[0,2]$: It assigns 4 tasks requiring 3 processors on machine M_3. It also assigns 1 task requiring 3 processors on machine M_2. Algorithm *A1* then schedules the tasks requiring 1 processor in time interval $[0,3]$: First in time interval $[0,2]$, it assigns 4 tasks requiring 1 processor on machine M_3, 5 tasks requiring 1 processor on machine M_2, 4 tasks requiring 1 processor on machine M_1. Then it has to assign 2 tasks requiring 1 processor on machine M_3 in time interval $[2,3]$. All the tasks complete no later than time 3.

The Analysis

We now prove the correctness of the algorithm.

Theorem 1. *Algorithm A1 is a linear time optimal algorithm for a special case of the problem of scheduling multiprocessor tasks in a multi-machine environment where all the tasks have unit processing times and each task requires one or k processors.*

Proof. First, we show that LB_1 and LB_2 are lower bounds on OPT , and $SOL \le OPT$. Consider an optimal schedule Σ^* for the problem. By a simple task interchange argument, Σ^* can be easily transformed into another feasible schedule (without increasing makepsan) in which the tasks requiring k processors are scheduled just as Step 2 of Algorithm *A1* does. Thus we get: $LB_1 \le OPT$. To prove $LB_2 \le OPT$, we cut each task requiring k processors into k *unitized tasks* each of which requires one processor. Obviously, the original tasks requiring one processor together with the unitized tasks cannot be completed before time LB_2 in any feasible schedule. Therefore we get: $LB_2 \le OPT$. It follows that $SOL \le OPT$.

Next, we show that $SOL = OPT$ indeed and thus Algorithm *A1* generates an optimal schedule. Since $SOL \le OPT$, we only need to prove that Algorithm *A1* generates a feasible schedule.

Since there is an optimal schedule in which the tasks requiring k processors are scheduled just as Step 2 of Algorithm *A1* does, all the tasks requiring k processors can be scheduled by Step 2 of Algorithm *A1*. Consider Step 3 of Algorithm *A1*. Suppose that there is a task requiring one processor, say task j , which cannot be assigned in time interval $[0, SOL]$. Then, all the processors on machines $M_1, M_2, \ldots, M_m$ must be busy during the interval $[0, SOL]$. It follows that $n_1 + kn_k \ge \sum_{i=1}^{m} \rho_i \cdot SOL + 1 \ge \sum_{i=1}^{m} \rho_i \cdot LB_2 + 1$, a contradiction.

Conclusion

In this paper, we studied the problem of scheduling multiprocessor tasks in a multi-machine environment. The objective is to minimize makespan. Since the general problem is strongly NP-hard, we presented a linear time optimal algorithm which solves a special case of the problem where all the tasks have unit processing times and each task requires one or *k* (*k* is part of the input) processors. It would be interesting to search for other polynomially solvable cases of the problem. Another interesting direction is to design fast algorithms with approximation ratios better than 3 for the general problem.

References

[1] J. Blazewicz, K. H. Ecker, G. Schmidt and J. Weglarz, Scheduling in computer and manufacturing systems, Springer Science & Business Media, 2012.

[2] M. R. Garey, D. S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, Freeman, New York, 1979.

[3] E. L. Lawler, J. K. Lenstra, A. H. R. Kan and D. B. Shmoys, Sequencing and scheduling: Algorithms and complexity, Handbooks in operations research and management science. 4 (1993) 445-522.

[4] J. Blazewicz, M. Drabowski and J. Weglarz, Scheduling multiprocessor tasks to minimize schedule length, IEEE Transactions on Computers. C-35 (1986) 389-393.

[5] M. Drozdowski, Scheduling multiprocessor tasks—An overview, European Journal of Operational Research. 94 (1996) 215-230.

[6] F. Pascual, K. Rzadca and D. Trystram, Cooperation in multi-organization scheduling, Euro-Par 2007 Parallel Processing. (2007) 224-233.

[7] K. Rzadca, Scheduling in multi-organization grids: measuring the inefficiency of decentralization, in International Conference on Parallel Processing and Applied Mathematics. (2007) 1048-1058.

[8] U. Schwiegelshohn, A. Tchernykh and R. Yahyapour, Online scheduling in grids, in IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, Miami, Florida Usa, April, 2008, pp. 1-10.

[9] J. F. Lin, List scheduling multiprocessor tasks in grid computing environments, ICIC Express Letters. 4 (2010) 245-248.

[10] J. F. Lin, Performance analysis and discussion on a heuristic approach for scheduling multiprocessor tasks in a grid computing environment, International Journal of Innovative Computing Information & Control. 6 (2010) 5451-5462.

[11] J. F. Lin and H. Liaw, Worst performance analysis of scheduling multiprocessor tasks in a multi-machine environment using LPT policy, in ICIM 2012 International Conference on Information Management, Kaohsiung, Taiwan, 2012.