MIT/LCS/TM-110

AN ANALYSIS OF PREEMPTIVE MULTIPROCESSOR JOB SCHEDULING

Jeffrey M. Jaffe

September 1978

An analysis of preemptive multiprocessor job scheduling

Jeffrey M. Jaffe, MIT *

Abstract

The preemptive scheduling of a partially ordered set of tasks is studied. A class of scheduling heuristics is introduced, and the performance of schedules in this class is analyzed with respect to the least finishing time optimality criterion. If there are m processors, then the finishing time of any schedule in the class is at most $\sqrt{m} + (1/2)$ times worse than optimal, independent of the speeds of the processors. Examples are given which indicate that there are schedules which may be as bad as $\sqrt{m-1}$ times worse than optimal even for machines with one fast processor.

Keywords. scheduling, maximal usage schedules, worst case performance bounds, preemption

1. Introduction

The problem of nonpreemptive job scheduling on a machine with m processors of different speeds was introduced by Liu and Liu [2,3]. They showed that any demand driven or list schedule has a finishing time that is at most $1+(b_1/b_m)-(b_1/(b_1+...+b_m))$ times worse than optimal where b_i is the speed of the ith fastest processor. In addition, examples were presented which showed that demand driven schedules could in fact perform as poorly as the

---

* This report was prepared with the support of a National Science Foundation graduate fellowship, and National Science Foundation grant no. MCS77-19754.

bound. This is a discouraging result since a large gap between the speeds of the fastest and slowest processors implies the relative ineffectiveness of demand scheduling, independent of the speeds of the other processors or number of processors.

While there have not been any heuristics studied that improve on [2,3] in the nonpreemptive case, Horvath, Lam, and Sethi [1] have studied the preemptive version of this problem. They define a "level algorithm" for the preemptive scheduling of tasks, an algorithm which is a generalization of that of [4,5]. It is shown in [1], that the level algorithm has worst case performance between $\sqrt{1.5m}$ and $\sqrt{m/8}$.

In this paper the gap of [1] is reduced, by obtaining an improved upper bound on the performance of the heuristic. This is accomplished using general methods that indicate that the $O(\sqrt{m})$ behavior of their algorithm results from the use of preemption - and not from the particular quality of the algorithm. Specifically, this paper analyzes a class of schedules that includes all "reasonable" schedules. The class is sufficiently general that any schedule may be easily transformed into a schedule in the class, where the new schedule has a finishing time at least as small as that of the original schedule. The main result is that any such "reasonable" schedule is at most $\sqrt{m} + (1/2)$ times worse than optimal.

Formal definitions of these concepts are provided in Section 2. The main result is proved in Section 3 and an example is presented in Section 4 that shows that the main result is almost best possible.

2. Definitions

A *task system* $(\mathcal{T},<,\mu)$ consists of:

(1) The set $\mathcal{T}=(T_1,\ldots,T_r)$; the elements $T_i\in\mathcal{T}$ are called *tasks.*

(2) A *partial ordering* $<$ on $\mathcal{T}$.

(3) A *time function* $\mu:\mathcal{T}\to\mathbb{R}$.

The set $\mathcal{T}$ represents the set of tasks or jobs that need to be executed. The partial ordering specifies which tasks must be executed before other tasks. The value $\mu(T)$ is the *time requirement* of the task T.

The execution of a task system takes place on a machine with a set of processors $\mathcal{P}=\{P_i:1\leq i\leq m\}$. The processor P_i has an associated speed b_i. For simplicity, assume $b_1\geq b_2\geq\ldots\geq b_m=1$. Let B_i denote $\Sigma_{j=1}^{i}b_j$, the *total processing power of the fastest i processors.*

The execution of a task system by processors of a machine is modelled by the notion of a schedule. In this paper schedules with preemptions are considered; that is the processing of a task may be temporarily suspended, and resumed at a later time (perhaps on a different processor). A *preemptive schedule* for $(\mathcal{T},<,\mu)$ is a total function S that maps each task $T\in\mathcal{T}$ to a finite set of interval, processor pairs. If

$S(T)=\{([i_1,j_1],Q_1),([i_2,j_2],Q_2),\ldots,([i_n,j_n],Q_n)\}$ then

(1) $i_p,j_p\in\mathbb{R}$ for $p=1,\ldots,n$.

(2) $i_p\leq j_p$ for $p=1,\ldots,n$ and $j_p\leq i_{p+1}$ for $p=1,\ldots,n-1$

(3) $Q_p\in\mathcal{P}$ for $p=1,\ldots,n$.

For $i_p\leq t\leq j_p$ T *is being executed on processor* Q_p at time t. The time i_1 is the *starting time* of T, and the time j_n is the *finishing time* of T.

A *valid* preemptive schedule for $(\mathcal{T},<,\mu)$ *on a set of processors* $\mathcal{P}=\{P_i:1\leq i\leq m\}$ is a preemptive schedule for $(\mathcal{T},<,\mu)$ with the properties:

(1) For all $t\in\mathbb{N}$, if two tasks are both being executed at time t, then they are being executed on different processors at time t.

(2) Whenever $T<U$, the starting time of U is not smaller than the finishing time of T.

(3) For $T\in\mathcal{T}$ (with $S(T)$ as above), $\mu(T)=((j_1-i_1)/r(Q_1))+\ldots+((j_n-i_n)/r(Q_n))$ where $r(Q_i)$ is the speed of Q_i (i.e. if $Q_i=P_j$ then $r(Q_i)=b_j$).

Condition one asserts that processor capabilities may not be exceeded. Condition two forces the obedience of precedence constraints. Condition three asserts that each task is processed exactly long enough to complete its time requirement.

The *finishing time* of a valid schedule is the maximum finishing time of the set of tasks. An *optimal* preemptive schedule is any valid preemptive schedule that minimizes the finishing time. For two valid preemptive schedules S and S', with finishing times w and w' the *performance ratio of S relative to* S' is w/w'.

(*Notation*: The total number of steps required by all the tasks of $\mathcal{T}$ is denoted by $\mu(\mathcal{T})$.)

In this paper the performance of the *maximal usage* heuristic is discussed. A *maximal usage preemptive schedule* is a valid preemptive schedule satisfying the following two requirements. The first requirement is that whenever i tasks are executable, then $\min(m,i)$ tasks are being executed. (A task is executable if all its predecessors have been finished, but the task itself has not been finished.) The second requirement is that whenever i processors are being used, it is always the fastest i processors that are in use. It is easy to see how to transform any schedule S into a maximal usage schedule that has a finishing time at least as small as that of S.

A *chain* C is a sequence of tasks $C=(U_1,\ldots,U_l)$ with $U_i\in\mathcal{T}$ such that for all j, $1\leq j<l$, $U_j<U_{j+1}$. C *starts* with task U_1. The *length* of C is equal to

$\Sigma_{j=1}^{l}\mu(U_j)$. The *height* of a task $T\in\mathcal{T}$ is the maximum over all chains starting with T of the length of the chain. The *height* of $(\mathcal{T},<,\mu)$ is the maximum over all tasks $T\in\mathcal{T}$ of the height of T.

While the notion of the height of a task is a static notion which is a property of $(\mathcal{T},<,\mu)$, we also associate a dynamic notion of the height of a task with any valid schedule for $(\mathcal{T},<,\mu)$. Specifically, let S be a valid schedule for $(\mathcal{T},<,\mu)$, and let t be less than the finishing time of S. Then the *height of the task T at the time t* is equal to the height of T in the unexecuted portion of the task system (that is, the maximum over all chains starting with T of the length of the chain, where the length of the chain considers only the unexecuted time requirements).

3. Performance of preemptive maximal usage schedules

When attempting to get a bound in terms of the number of processors in the machine, we must balance off two factors. There are times when all of the processors run at approximately the same speed, and it is then desirable to get a bound in terms of the fastest processor (which is not too much worse than the slower). On the other hand there are times when there is a large disparity between the speeds of the processors of the machine, and then it is desirable to get a bound in terms of some sort of average of the speeds of the processors. The first bound obtained is directed towards the latter goal, that of obtaining a bound which would still be a relatively good bound in the case that there is a wide disparity between the speeds of the different processors of the machine.

Lemma 1. Let $(\mathcal{T},<,\mu)$ be a task system. Let w be the finishing time of a maximal usage schedule S and let w_0 be the finishing time of an optimal schedule. Then $w/w_0 \leq (B_m/B_1)$.

Proof. Note first that $w_0 \geq \mu(\mathcal{T})/B_m$. This follows from the fact that the optimal schedule can do no better than executing B_m units of the time requirement of the task system in unit time. Also note that $w \leq \mu(\mathcal{T})/B_1$. This follows from the fact that the maximal usage heuristic forces the fastest processor to be in use at every moment before the finishing time. Thus the ratio between an arbitrary schedule and the optimal schedule is bounded by $(\mu(\mathcal{T})/B_1)/(\mu(\mathcal{T})/B_m) = B_m/B_1$. □

The second bound addresses the goal of being an effective bound in the situation when the speeds of the processors are about equal (in that case the bound of Lemma 1 reduces to about m).

Lemma 2. Let $(\mathcal{T},<,\mu)$ be a task system. Let w and w_0 as in Lemma 1. Then $w/w_0 \leq 1+((m-1)(B_1)/B_m)$.

Proof. As above, there is a lower bound of μ/B_m on w_0. Let h denote the height of $(\mathcal{T},<,\mu)$. Then $w_0 \geq h/B_1$ since none of the h units of time requirement of the longest chain may be executed concurrently. Thus the best that any schedule can do is to use the fastest processor at each unit of time.

To determine the value of w, let p_i denote the amount of time during which exactly i processors are being used in the schedule S $(i=1,\ldots,m)$. By definition $w=p_1+\ldots+p_m$. Due to the maximal usage discipline, p_iB_i units of the time requirement of the task system are executed during the p_i units of time that exactly i processors are used. Thus $p_1B_1+\ldots+p_mB_m=\mu(\mathcal{T})$. Solving for p_m in this equation and substituting for p_m in the equation $w=p_1+\ldots+p_m$ yields

$$w=(p_1+\ldots+p_{m-1})+((\mu(\mathcal{T})-(p_1B_1+\ldots+p_{m-1}B_{m-1}))/B_m).$$

Fix an interval of time during which only i processors are being used ($i<m$). Assume that there is a single task, T, that is at the greatest height among all non finished tasks during this time interval. Then T is being executed throughout this interval. This follows from the fact that at any point in time the unexecuted task at the greatest height is executable, and so the maximal usage discipline forces the execution of the task if less than m processors are being used. Using this fact, it is easy to conclude that when $i<m$ processors are being used, the greatest unexecuted height goes down at a rate of at least b_i units of height per unit time (since the greatest height tasks are being executed at a rate greater than or equal to b_i). Thus $b_1p_1+\ldots+b_{m-1}p_{m-1}\leq h$ since the total amount that the "greatest height" can go down during all of the times that fewer than m processors are used is at most h.

The second bound on the performance of arbitrary maximal usage schedules may now be derived from the ratio of:

$$\frac{w}{w_0} \leq \frac{(\mu/B_m)+p_1(1-B_1/B_m)+\ldots+p_{m-1}(1-B_{m-1}/B_m)}{\max(\mu/B_m,h/B_1)}$$

The first lower bound on the optimal schedule in the above ratio is used only as a comparison to the μ/B_m term in the numerator. After that comparison multiply numerator and denominator of the remaining fraction by $B_m B_1$ to obtain a bound on the performance ratio of:

$$\frac{w}{w_0} \leq 1 + \frac{B_1[(p_1)(B_m-B_1)+\ldots+(p_{m-1})(B_m-B_{m-1})]}{hB_m}$$

Now $B_m-B_i=b_{i+1}+\ldots+b_m\leq(m-i)b_i$ since $b_i\geq b_j$ for $j>i$. Thus

$$\frac{w}{w_0} \leq 1 + \frac{(B_1)[(p_1)(b_1)(m-1)+\ldots+(p_{m-1})(b_{m-1})(1)]}{hB_m}$$

Using $p_1b_1+\ldots+p_{m-1}b_{m-1}\leq h$ we conclude $w/w_0\leq 1+((m-1)B_1/B_m)$. □

Theorem. Let $(\mathcal{T},<,\mu)$ be a task system. Let w be the finishing time of a maximal usage schedule and let w_0 be the finishing time of an optimal schedule. Then $w/w_0\leq\sqrt{m} + (1/2)$.

Proof. By Lemmas 1 and 2 $w/w_0\leq B_m/B_1$ and $w/w_0\leq 1+((m-1)B_1/B_m)$. Let $r=B_m/B_1$. Then $w/w_0\leq r$ and $w/w_0\leq 1+(m-1)/r$. To maximize $\min(r,1+((m-1)/r))$, solve $r=1+((m-1)/r)$ and get $r=(1/2)+\sqrt{(1+4(m-1))}/2$. This value of r maximizes $\min(r,1+(m-1)/r)$ since r and $1+(m-1)/r$ are inversely related. Thus $w/w_0\leq(1/2)+\sqrt{(1+4(m-1))}/2\leq\sqrt{m} + (1/2)$. □

4. Achievability of the performance bound

Consider the situation where $b_1=\sqrt{m-1}$ and $b_i=1$ for $i>1$. Consider the task system of $2n$ tasks as diagrammed in Figure 1. A node represents a task and an arrow represents a precedence dependence. The time requirement of each of the n tasks in the long chain is $1/\sqrt{m-1}$. The time requirement of the other n tasks is 1. An optimal schedule proceeds as follows. P_1 executes every task in the long chain. Thus $m-1$ of the tasks in the long chain require time $((m-1)/\sqrt{m-1})/\sqrt{m-1}=1$. Meanwhile, $P_2,\ldots,P_m$ execute the tasks that are not in the long chain. Each of these processors requires unit time for one of the tasks. If $n=m-1$ then the long chain requires unit time as above, but P_m will not finish its task until two units of time have passed since its task is not executable until almost one unit of time elapses. For any value of n the finishing time is similarly bounded by $(n/(m-1))+1$.

A "bad" maximal usage schedule first tries to use P_1 on the "non-long chain" tasks, and P_2 on the chain tasks. After time $1/\sqrt{m-1}$, P_1 finishes the first non-chain element, and P_2 finishes the first chain element. Repeating this strategy for each pair of tasks requires time $1/\sqrt{m-1}$ for each pair. Thus the total time for the bad schedule is about $n/\sqrt{m-1}$ and the ratio between the finishing times of the "bad" schedule and the optimal schedule approaches $\sqrt{m-1}$ for large n. □

5. Conclusions

Now that the basis for coomparison of preemptive scheduling algorithms has been established it would be useful to find a polynomial time algorithm

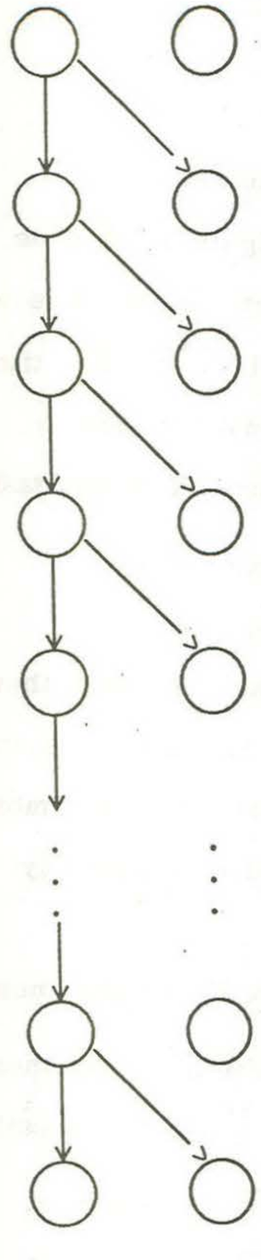

Figure 1.

whose performance is better than $O(\sqrt{m})$ times worse than optimal in the worst case. The level algorithm of [1] is not a candidate as there are known examples for which it performs $\sqrt{m/8}$ times worse than optimal. This problem seems quite difficult.

An easier problem might be to find an algorithm which is provably better than the worst schedules in this class (for example $\sqrt{m/2}$ times worse than optimal in the worst case). The level algorithm seems like a likely candidate for this problem. It can be shown (by combining our techniques with those of [1]) that the level algorithm is never worse than $(1/2)+\sqrt{(1+4(m-2))}/2$, but this is not a significant improvement over the performance of any maximal usage schedule.

References

1. Horvath, E. C., Lam, S., and Sethi, R. "A Level Algorithm for Preemptive Scheduling, *JACM 24*, 1, (Jan. 1977) pp 32-43.

2. Liu, J. W. S., and Liu, C. L. "Bounds on Scheduling Algorithms for Heterogeneous Computing Systems," TR No. UIUCDCS-R-74-632 Dept. of Comp. Sci., Univ. of Illinois, June 1974.

3. Liu, J. W. S., and Liu, C. L. "Bounds on Scheduling Algorithms for Heterogeneous Computing Systems," *IFIP74*, North Holland Pub. Co., pp349-353.

4. Muntz, R. R., and Coffman, E. G. Jr., "Optimal preemptive scheduling on two-processor systems, *IEEE Trans. Comptrs., C-18,* 11 (Nov. 1969) 1014-1020.

5. Muntz, R. R., and Coffman, E. G. Jr., "Preemptive scheduling of real time tasks on multiprocessor systems, *JACM 17*, 2 (April 1970) 324-338.