An Analysis of Preemptive Multiprocessor Job Scheduling
Author(s): Jeffrey M. Jaffe
Source: *Mathematics of Operations Research*, Vol. 5, No. 3 (Aug., 1980), pp. 415-421
Published by: INFORMS
Stable URL: http://www.jstor.org/stable/3689447
Accessed: 28-06-2017 00:57 UTC

MATHEMATICS OF OPERATIONS RESEARCH
Vol. 5, No. 3, August 1980
Printed in U.S.A.

AN ANALYSIS OF PREEMPTIVE MULTIPROCESSOR JOB SCHEDULING*†

JEFFREY M. JAFFE‡

Massachusetts Institute of Technology

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.

1. Introduction. Preemptive scheduling of a partially ordered set of tasks on m processors of different speeds was first studied by Horvath, Lam, and Sethi [2]. They define a "level algorithm" which is a generalization of the algorithm developed by Muntz and Coffman [7], [8]. The finishing time of any schedule produced by the level algorithm is at most $\sqrt{1.5m}$ times worse than optimal [2]. In addition, there are level schedules that are as bad as $\sqrt{m/8}$ times worse than optimal [2]. The level algorithm finds an optimal schedule in three cases: for two processors of different speeds even if the tasks are partially ordered [2], for m processors of different speeds if the tasks are independent [2], and for m identical processors if the partial order is a tree [8].

In this paper an improved upper bound on the performance of the level algorithm is obtained. This is accomplished by noticing that the $O(\sqrt{m})$ behavior of the level algorithm results from the use of preemption—and not from the particular way in which the level algorithm schedules tasks. Specifically, this paper analyzes a class of schedules, called the maximal usage schedules, that includes all "reasonable" preemptive schedules. Any preemptive schedule may be easily transformed into a maximal usage schedule with a finishing time at least as small as that of the original schedule [4]. The main result is that any maximal usage schedule is at most $\sqrt{m} + (1/2)$ times worse than optimal. Since the level algorithm is a special case of the maximal usage schedules, the level algorithm is at most $\sqrt{m} + (1/2)$ times worse than optimal.

It is interesting to contrast this situation with the problem of nonpreemptive job scheduling on a machine with m processors of different speeds. In that environment, there is no notion of a maximal usage schedule which encompasses all schedules. The closest analog is the notion of a list schedule. Both list schedules and maximal usage schedules work on the principle of local efficiency—never permitting a processor to be idle if there are unexecuted executable tasks to be handled. List scheduling was studied by Liu and Liu [5], [6]. They showed that any list schedule has a finishing time

* Received August 21, 1978.
AMS 1970 subject classification. Primary 90B35.
IAOR 1973 subject classification. Main: Scheduling.
OR/MS Index 1978 subject classification. Primary: 584 Production/scheduling, job shop, deterministic.
Key words. Scheduling, maximal usage schedules, worst case performance bounds, preemption.
† This report was prepared with the support of a National Science Foundation graduate fellowship and National Science Foundation Grant No. MCS77-19754.
‡ The author's current address is: IBM, T. J. Watson Research Center, Department of Computer Science, Yorktown Heights, New York 10598.

0364-765X/80/0503/0415$01.25

which is at most $1 + (b_1/b_m) - (b_1/(b_1 + \cdots + b_m))$ times worse than optimal where b_i is the speed of the ith fastest processor. In addition, examples were presented which showed that list schedules do in fact perform as poorly as the bound. This is a discouraging result since a large gap between the speeds of the fastest and slowest processors implies the ineffectiveness of list scheduling, independent of the speeds of the other processors or number of processors. While b_1/b_m may be smaller than $\sqrt{m}$, it turns out that if for a given set of processors b_1/b_m is small, the maximal usage schedules perform no worse than b_1/b_m.

Recent work by this author [3] presents a nonpreemptive algorithm whose performance is almost as good as the performance of maximal usage preemptive schedules. The basic idea is to devise a list schedule on a subset of the m processors. The worst case performance of this algorithm is $\sqrt{m} + O(m^{1/4})$ [3].

Formal definitions of preemptive schedules and maximal usage schedules are provided in §§2 and 3. The upper bound for maximal usage schedules is proved in §4 and an example of a maximal usage schedule whose performance is close to the bound is presented in §5.

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

(1) a set $\mathcal{T}$ of *n 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.

Associated with a task system is a set of *processors* $\mathcal{P} = \{P_i : 1 \leqslant i \leqslant m\}$. The processor P_i has a *speed* b_i. Assume $b_1 \geqslant b_2 \geqslant \cdots \geqslant b_m > 0$.

In this paper preemptive schedules 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, i.e., the first element of each pair is an interval and the second is a processor. If $S(T) = \{([i_1, j_1], Q_1), ([i_2, j_2], Q_2), \ldots, ([i_l, j_l], Q_l)\}$ then

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

(2) $i_p \leqslant j_p$ for $p = 1, \ldots, l$ and $j_p \leqslant i_{p+1}$ for $p = 1, \ldots, l-1$;

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

For $i_p \leqslant t \leqslant j_p$ task 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_l is the *finishing time* of T.

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

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

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

(c) For $T \in \mathcal{T}$ (with $S(T)$ as above),

$$\mu(T) = (j_1 - i_1)(\text{speed}(Q_1)) + \cdots + (j_l - i_l)(\text{speed}(Q_l))$$

where if $Q_i = P_j$ then $\text{speed}(Q_i) = b_j$.

Condition (a) asserts that processor capabilities may not be exceeded. Condition (b) forces the obedience of precedence constraints. Condition (c) 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 mini-

mizes the finishing time. For two valid preemptive schedules S and S', with finishing times w and w' the *performance ratio of S to S'* is w/w'.

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

A task T is *executable* at time t if all predecessors of T have been finished by time t, but T has not yet been completed.

A *chain* C is a sequence of tasks $C = (T_1, \ldots, T_l)$ with $T_i \in \mathfrak{T}$ such that for all j, $1 \leqslant j < l$, $T_j < T_{j+1}$. C *starts* with task T_1. The *length* of C is $\sum_{j=1}^{l} \mu(T_j)$. The *height* of a task $T \in \mathfrak{T}$ is the length of the longest chain starting at T. The *height* of $(\mathfrak{T}, <, \mu)$ is the length of the longest chain starting at any task $T \in \mathfrak{T}$.

While the notion of the height of a task is a static notion which is a property of $(\mathfrak{T}, <, \mu)$, we also associate a dynamic notion of the height of a task with any valid schedule for $(\mathfrak{T}, <, \mu)$. Specifically, let S be a valid schedule for $(\mathfrak{T}, <, \mu)$, and let t be less than the finishing time of S. Then the *height of the task T at time t* is equal to the height of T in the unexecuted portion of the task system. That is, the height equals the length of the longest chain starting at T, where the length of the chain considers only the unexecuted time requirements. Similarly: the *height of* $(\mathfrak{T}, <, \mu)$ *at time t* is the length of the longest chain starting at any task (not yet completed) $T \in \mathfrak{T}$. Note that if a portion of a task has been finished at time t, then it contributes to the height the proportion of the time requirement not yet completed.

3. Maximal usage schedules. In this paper the performance of the *maximal usage schedules* is discussed. A *maximal usage preemptive schedule* is a valid preemptive schedule which satisfies the following two additional requirements.

(1) Whenever i tasks are executable, then $\min(m, i)$ tasks are being executed.

(2) Whenever i processors are being used, the fastest i processors are in use.

Maximal usage schedules encompass all preemptive schedules in a very strong sense. Given any task system, at least one optimal preemptive schedule is a maximal usage schedule. Furthermore, given any preemptive schedule S (with finishing time w), one may easily transform S into a maximal usage schedule S' (with finishing time w') with the property $w' \leqslant w$. This transformation algorithm is based on very simple principles, as we proceed to sketch. The details may be found in [4].

Assume that S is given as a listing $U_1, \ldots, U_r$ where each listing element $U_k = ([start_k, end_k], T_k, Q_k)$. The interpretation of U_k is that task T_k is executed on Q_k between times $start_k$ and end_k.

The translation from an arbitrary schedule into a maximal usage schedule will proceed in stages. It starts with the schedule $S = S_0$, and produces schedules $S_1, \ldots, S_q$ where S_q is the resulting maximal usage schedule. The basic idea is that if S_i satisfies conditions (1) and (2) above before t, but does not at t, then S_i is modified at t to produce S_{i+1}. Specifically, let t' be the smallest time greater than t that equals $start_k$ for some k. Then, if S_i does not use all m processors at t and there are executable tasks that are not being executed at t, then an executable task is assigned to an idle processor for the interval $[t, t']$. Assignment of tasks to processors for this interval continues until either there are no more executable tasks or no more idle processors. Similarly, if the processors being used at t are not the fastest processors, then the tasks that are currently being executed on the slower processors are reassigned to the faster processors.

Next, S_i is further modified to insure that the total time requirement of each task T executed in S_{i+1} is $\mu(T)$. This is necessary to insure that S_{i+1} is in fact a valid schedule. For example, if a listing element $([t, t'], T, P)$ is added to S_{i+1} and the speed of P is b, then $(t' - t)b$ units of the time requirement of T are removed from later assignments. Similarly, if T is rescheduled onto a faster processor at t, then the added time requirement which is completed is removed from a later interval of T. If

executing T on P during $[t, t']$ causes the total time requirement of T that is assigned to processors before t' to exceed $\mu(T)$, then the interval $[t, t']$ is shortened somewhat.

After the reassignments are made, the translation algorithm determines the smallest time t'' greater than t and equal to end_k for some k. S_{i+1}, attains maximal usage before t'' and that portion of the schedule will not be changed again. Starting at t'', the procedure modifies S_{i+1} to produce S_{i+2} in the manner specified above.

The resulting finishing time of each task in S_q is at least as small as in S. From this it follows that the finishing time of S_q is at least as small as that of S.

Recall that a schedule S specifies a number of interval, processor pairs for each task. Let r be the total number of pairs in the original schedule S (summed over all tasks). The run-time of this algorithm is $O(n^2 + mr)$, where n is the number of tasks in $\mathfrak{T}$. Note that $n^2 + r + m$ is essentially the size of the input to the algorithm since the size of the partial order is $O(n^2)$. Obtaining the running time of $O(n^2 + mr)$ requires fairly detailed data structures [4] which are omitted here.

4. Performance of maximal usage preemptive schedules. In this section a bound on the performance of maximal usage schedules is obtained in terms of the number of processors. This is accomplished by finding two different bounds in terms of the speeds of the processors. The first bound is obtained by using the fact that in a maximal usage schedule the fastest processor is always in use. This bound essentially proves that maximal usage schedules are somewhat effective even if the slower processors are considerably slower than the fastest processor. The second bound uses techniques similar to those of [9], to get a bound which is effective if the processors are of approximately the same speed. It turns out that for any set of processor speeds, at least one of the two upper bounds is smaller than $\sqrt{m} + (1/2)$.

Let B_i denote $\sum_{j=1}^{i} b_j$, the *total processing power of the fastest i processors.*

LEMMA 1. *Let* $(\mathfrak{T}, <, \mu)$ *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* $w/w_0 \leqslant (B_m/B_1)$.

PROOF. Since the optimal schedule can complete at most B_m units of the time requirement of the task system during each unit of time, we get $w_0 \geqslant \mu(\mathfrak{T})/B_m$. Also $w \leqslant \mu(\mathfrak{T})/B_1$ because 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 maximal usage schedule and the optimal schedule is bounded by

$$(\mu(\mathfrak{T})/B_1)/(\mu(\mathfrak{T})/B_m) = B_m/B_1. \qquad \blacksquare$$

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

PROOF. As above, there is a lower bound of $\mu(\mathfrak{T})/B_m$ on w_0. Let h denote the height of $(\mathfrak{T}, <, \mu)$. Then, in addition, $w_0 \geqslant h/B_1$ since none of the h units of time requirement of the longest chain may be executed concurrently, and the best that the optimal schedule can do is to use the fastest processor on the longest chain from time 0 until time w_0.

To determine the value of w, let p_i, $i = 1, \ldots, m$, denote the amount of time during which exactly i processors are used in the schedule S. By definition $w = p_1 + \cdots + p_m$. Due to the maximal usage discipline, whenever i processors are in use B_i units of the time requirement are finished per unit time. Thus a total of $p_i B_i$ units of the time requirement of the task system are completed during the p_i units of time that exactly i processors are used. Thus $p_1 B_1 + \cdots + p_m B_m = \mu(\mathfrak{T})$. Solving for p_m in this equation

and substituting for p_m in the equation $w = p_1 + \cdots + p_m$ yields

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

Fix an interval of time during which exactly i processors are being used ($i < m$). Assume that during this interval, an unfinished task T is at the greatest height of the system (i.e., for every t in the interval, the height of T at time t equals the height of $(\mathfrak{T}, <, \mu)$ at t). Since T is executable during this interval, its execution is forced by the maximal usage discipline. Consequently, when $i < m$ processors are being used, the height of $(\mathfrak{T}, <, \mu)$ is reduced at a rate of at least b_i, the speed of the ith fastest processor. That is, a job from the longest remaining chain is always being executed if fewer than m processors are in use (where the length of each chain only considers the portion of $\mathfrak{T}$ not completed). Thus $b_1p_1 + \cdots + b_{m-1}p_{m-1} \leqslant h$ since the total amount that the "greatest height" can be reduced during all of the times that fewer than m processors are used is at most h.

Rewriting (1) and using the lower bounds on w_0 produces an upper bound on the performance of arbitrary maximal usage schedules of:

$$\frac{w}{w_0} \leqslant \frac{(\mu(\mathfrak{T})/B_m) + p_1(1 - (B_1/B_m)) + \cdots + p_{m-1}(1 - (B_{m-1}/B_m))}{\max(\mu(\mathfrak{T})/B_m, h/B_1)}. \quad (2)$$

The first lower bound on the optimal schedule in (2) is used only as a comparison to the $\mu(\mathfrak{T})/B_m$ term in the numerator. Thus:

$$\frac{w}{w_0} \leqslant 1 + \frac{(p_1(B_m - B_1)/B_m) + \cdots + (p_{m-1}(B_m - B_{m-1})/B_m)}{(h/B_1)}. \quad (3)$$

Now, multiply numerator and denominator of the fractional part of (3) by B_mB_1 to obtain:

$$\frac{w}{w_0} \leqslant 1 + \frac{B_1[(p_1)(B_m - B_1) + \cdots + (p_{m-1})(B_m - B_{m-1})]}{hB_m}. \quad (4)$$

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

$$\frac{w}{w_0} \leqslant 1 + \frac{(B_1)[(p_1)(b_1)(m - 1) + \cdots + (p_{m-1})(b_{m-1})(1)]}{hB_m}. \quad (5)$$

Since $p_1b_1 + \cdots + p_{m-1}b_{m-1} \leqslant h$ one may further increase the numerator of (5) and obtain $w/w_0 \leqslant 1 + ((m - 1)B_1/B_m)$. ∎

If all processor speeds are identical then $B_1/B_m = 1/m$. In that case, the bound of Lemma 2 is $1 + ((m - 1)/m) = 2 - (1/m)$. Thus it follows from Lemma 2 that when the processors are identical, maximal usage schedules are no worse than $2 - (1/m)$ times worse than optimal.

THEOREM. *Let $(\mathfrak{T}, <, \mu)$ 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 $w/w_0 \leqslant \sqrt{m} + (1/2)$.*

PROOF. By Lemmas 1 and 2 $w/w_0 \leqslant B_m/B_1$ and $w/w_0 \leqslant 1 + ((m - 1)B_1/B_m)$. Let $r = B_m/B_1$. Then $w/w_0 \leqslant r$ and $w/w_0 \leqslant 1 + (m - 1)/r$. To maximize $\min(r, 1 + ((m - 1)/r))$, solve $r = 1 + ((m - 1)/r)$ and obtain

$$r = (1/2) + \left(\sqrt{(1 + 4(m - 1))}\right)/2 < \sqrt{m} + (1/2). \quad ∎$$

5. Achievability of the performance bound. This section proves that the bound of $\sqrt{m} + (1/2)$ is almost achievable, even for machines with one fast processor and $m - 1$ identical slow processors. Let $b_1 = \sqrt{m-1}$ and $b_i = 1$ for $i > 1$. Consider the task system of $2n$ tasks as diagrammed in Figure 1, where a node represents a task and an arrow represents a precedence dependence. We shall assume time requirements of $1/\sqrt{m-1}$ for each of the n tasks in the long chain and 1 for each of the other n tasks. An asymptotically optimal schedule has P_1 executing every task in the long chain. Thus $m - 1$ of the tasks in the long chain require time

$$\left((m-1)/\sqrt{m-1}\right)/\sqrt{m-1} = 1.$$

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 $(n/(m-1)) + 1$.

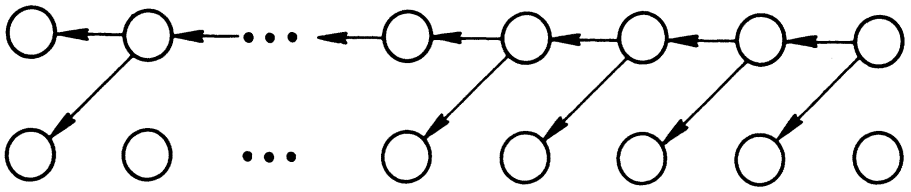

FIGURE 1

An ineffective maximal usage shcedule, S, first tries to use P_1 on the "nonlong chain" tasks, and P_2 on the chain tasks. After time $1/\sqrt{m-1}$, P_1 finishes the first nonchain 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 S is $n/\sqrt{m-1}$ and the ratio between the finishing times of S and the optimal schedule approaches $\sqrt{m-1}$ for large n. ∎

The fact that $m - 1$ of the m processors have the same speed in this example is important. In [1], it is shown that with independent tasks and almost identical processors (in the sense of $m - 1$ of them being identical) improved algorithms may be obtained for nonpreemptive schedules.

6. Conclusions. Now that the class of *all* preemptive scheduling algorithms has been analyzed, it would be useful to find a polynomial time algorithm whose performance is better than $O(\sqrt{m})$ times worse than optimal in the worst case. The level algorithm of [2] 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 whose worst cases are provably better than the worst maximal usage schedules (for example $\sqrt{m/2}$ times worse than optimal in the worst case). The level algorithm seems like a likely candidate for this problem. This algorithm, roughly speaking, executes the greatest height executable tasks on the fastest processors. It can be shown (by combining our techniques with those of [2]) 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] Gonzalez, T., Ibarra, O. H. and Sahni, S. (1977). Bounds for LPT Schedules on Uniform Processors. *SIAM J. Comput.* **6** 155–166.

[2] Horvath, E. C., Lam, S. and Sethi, R. (1977). A Level Algorithm for Preemptive Scheduling. *J. Assoc. Comput Mach.* **24** 32–43.

[3] Jaffe, J. M. (January 1979). Efficient Scheduling of Tasks without Full Use of Processor Resources. MIT, Laboratory for Computer Science, Technical Memo 122. To appear in *Theoretical Computer Science*.

[4] ———. (to appear 1979). Parallel Computation: Synchronization, Scheduling, and Schemes. Ph.D. thesis in preparation.

[5] Liu, J. W. S. and Liu, C. L. (June 1974). Bounds on Scheduling Algorithms for Heterogeneous Computing Systems. TR No. UIUCDCS-R-74-632 Dept. of Computer Science, University of Illinois.

[6] ——— and ———. Bounds on Scheduling Algorithms for Heterogeneous Computing Systems. *IFIP74*, North Holland Publishing Co. 349–353.

[7] Muntz, R. R. and Coffman, E. G., Jr. (1969). Optimal Preemptive Scheduling on Two-processor Systems. *IEEE Trans. Computers*. **11** 1014–1020.

[8] ——— and ———. Preemptive Scheduling of Real Time Tasks on Multiprocessor Systems. *J. Assoc. Comput. Mach.* **17** 324–338.

[9] Sethi, R. (1976). Algorithms for Minimal-Length Schedules. In *Computer and Job Shop Scheduling Theory*, E. G. Coffman, ed. J. Wiley and Sons, NY.

LABORATORY FOR COMPUTER SCIENCE, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 545 TECHNOLOGY SQUARE, ROOM 807, CAMBRIDGE, MA 02139