

STOCHASTIC DISPATCHING OF MULTI-PRIORITY JOBS TO HETEROGENEOUS PROCESSORS

SUSAN H. XU,* *Pennsylvania State University*
PITU B. MIRCHANDANI,** *The University of Arizona*
SRIKANTA P. R. KUMAR,*** *Northwestern University*
RICHARD R. WEBER,**** *University of Cambridge*

Abstract

A number of multi-priority jobs are to be processed on two heterogeneous processors. Of the jobs waiting in the buffer, jobs with the highest priority have the first option of being dispatched for processing when a processor becomes available. On each processor, the processing times of the jobs within each priority class are stochastic, but have known distributions with decreasing mean residual (remaining) processing times. Processors are heterogeneous in the sense that, for each priority class, one has a lesser average processing time than the other. It is shown that the non-preemptive scheduling strategy for each priority class to minimize its expected flowtime is of threshold type. For each class, the threshold values, which specify when the slower processor is utilized, may be readily computed. It is also shown that the social and the individual optimality coincide.

STOCHASTIC SCHEDULING; MULTI-SERVER SYSTEMS

1. Introduction

Consider a system in which two heterogeneous processors are available for processing a number of jobs. The jobs have stochastic processing requirements, but fall into n classes, C_1, \dots, C_n , such that jobs of the same class have processing requirements whose distributions are identical. Jobs are dispatched to the processors in a non-preemptive manner and jobs of lower index classes have priority: a class C_j job may be dispatched to an idle processor only if it has been rejected by all jobs from classes C_1, \dots, C_{j-1} .

Received 15 March 1988; revision received 2 November 1989.

* Postal address: Department of Management Science, Pennsylvania State University, University Park, PA 16802, USA.

** Postal address: Systems and Industrial Engineering, The University of Arizona, Tucson, AZ 85721, USA.

*** Postal address: Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA.

**** Postal address: Department of Engineering, University of Cambridge, Mill Lane, Cambridge CB2 1RX, UK.

This work has been partially supported by the FMS Program in the Center for Manufacturing Productivity and Technology Transfer, Rensselaer Polytechnic Institute, Troy, NY 12180–3590, USA.

The flowtime of the jobs in a class is the sum of their completion times. The paper addresses the following problem: what non-preemptive dispatching rule for each class minimizes the expected flowtime for the class? So far as class C_l jobs are concerned, the presence of jobs in classes C_{l+1}, \dots, C_n , does not affect their expected flowtime. Class C_l jobs are to be dispatched in a manner that minimizes the expected flowtime of that class, subject to the priority use of the processors by jobs of classes C_1, \dots, C_{l-1} .

This study was motivated by the following scenario in production scheduling with regard to expediting high-priority jobs. When a set of jobs falls behind schedule for whatever reason or when a set of 'hot' jobs is released to the shop-floor for manufacturing, special effort must be taken for these jobs to make up the lost time or to meet the required schedule as close as possible, without causing major disruption to 'routine' jobs waiting to be processed. In manufacturing scheduling this special function is usually referred to as 'expediting'. Although expediting is a common practice in many control systems, the problem of expediting high-priority jobs has not been sufficiently studied, nor are the consequences of expediting these jobs well understood. Consequently, more attention to studying job expedition may be useful, not only for designing better scheduling/dispatching algorithms, but also for more realistic modeling and understanding of the manufacturing and the production scheduling environment.

There has been some interest on the optimal dispatching and related problems for systems with heterogeneous processors. Weiss and Pinedo (1980) and Weber (1981), (1982) focused on preemptive schedules where the processing of a job may be preempted to begin the processing of another job. Some research has also been reported with regard to non-preemptive schedules for heterogeneous processors. Agrawala et al. (1984) considered the problem of executing a number of identical jobs with exponential processing times on a set of parallel processors which differ only in speed. They showed that the optimal non-preemptive policy to minimize expected flowtime is of threshold type (the threshold values can be explicitly computed as a function of the processing rates). This (threshold) socially optimal policy is also individually optimal for each job, and this fact was shown by Kumar and Walrand (1985). Coffman et al. (1987) have considered the minimization of the expected makespan criterion, and have showed that for a system with two or three processors with dissimilar speeds the optimal policy is of threshold type. Among other studies in non-preemptive dispatching, Weber et al. (1986) consider the scheduling of tasks with stochastically ordered processing times, but the processors are taken to be identical there.

Nevertheless, most of the work done on systems with heterogeneous processors has focused either on dispatching one class of identical jobs non-preemptively or dispatching multi-class jobs preemptively. Very little attention has been paid to the non-preemptive dispatching of multi-class jobs. Mirchandani and Xu (1987) studied the problem of non-preemptive dispatching of jobs of n priority classes to two heterogeneous processors when the processing times of class C_j jobs on processor P_i are assumed to be i.i.d. exponential random variables with rate parameter μ_{ij} , where $\mu_{1j} \geq \mu_{2j}$, $i = 1, 2$ and $j = 1, 2, \dots, n$. They show that if an optimal threshold policy is applied to every higher priority class C_j job, $j = 1, 2, \dots, l-1$, then the expected flowtime of class C_l jobs is also minimized by a threshold dispatching policy. That is, at

each decision epoch, available processors are accepted or rejected according to some readily computable threshold functions.

In this paper we consider the non-preemptive dispatching of multi-priority jobs with more general processing time distributions. Specifically, the processing times of jobs within each priority class are assumed to be independent and identically distributed random variables with the property that the expected remaining processing time is non-increasing in age (or elapsed processing time). A sufficient condition for this property is that the processing time distribution has increasing hazard rate. The processors are heterogeneous in the sense that they differ in the expected processing time for each job in each priority class. It is shown that the optimal non-preemptive dispatching policy is of threshold type for each class, and these thresholds can be readily computed. This socially optimal policy for each priority class is also individually optimal for each job in that class; and this result in part proves the conjecture in Kumar and Walrand (1985), p. 95 regarding socially and individually optimal policies.

2. Definitions and notations

Throughout this paper we shall let X_{ij} represent the processing time of a class C_j job on processor P_i , $i = 1, 2$, and $j = 1, 2, \dots, n$, where X_{ij} , $0 < X_{ij} \leq M_{ij}$ is an integer-valued random variable with hazard rate $h_{ij}(m) = P(X_{ij} = m + 1 | X_{ij} > m)$. Define $\chi_{ij} \equiv E(X_{ij})$ and $\chi_{ij}(m) \equiv E(X_{ij} - m | X_{ij} > m)$, for $m \geq 0$. The processing times are assumed to satisfy the following conditions:

$$(A.1) \quad \chi_{ij}(m) \text{ is non-increasing in } m, \text{ for } i = 1, 2 \text{ and } j = 1, 2, \dots, n.$$

$$(A.2) \quad \chi_{1j} \leq \chi_{2j}, \quad \text{for } 1 \leq j \leq n.$$

The first condition implies that the expected remaining processing time of each job is non-increasing in its ‘age’ (the amount of processing it has already received). This condition is satisfied if the processing times have increasing hazard rate, i.e., $h_{ij}(m)$ is increasing in m , for all i and j (see Ross (1983)). With (A.2), we are assuming that P_1 is faster, on the average, than P_2 for all job classes.

The dispatching decisions are made at time epochs $t = 0, 1, \dots$. We identify the state of the system at time $t = 0, 1, \dots$ by $x = (\alpha m; k)$, where $\alpha m = (\alpha_1 m_1, \alpha_2 m_2)$ is a two-dimensional vector representing the processors’ states

$$\alpha_i m_i = \begin{cases} 00 & \text{if } P_i \text{ is idle} \\ jl & \text{if } P_i \text{ is processing a class } C_j \text{ job of age } l; 1 \leq j \leq n, 0 \leq l \leq M_{ij} \end{cases}$$

and $k = (k_1, k_2, \dots, k_n)$ is called the buffer state, where k_j denotes the number of C_j jobs waiting in buffer j . Note that α_i indicates the type of job on P_i and m_i its age. Finally, let S denote the state space which is the set of all possible states.

At each decision time, class C_j jobs receive dispatching preference over C_{j+1} jobs. We define for class C_j jobs a non-preemptive dispatching function $f_j: S \rightarrow S$. In other words, if $f_j(x) = y$ then the state of a processor is the same in both x and y , if the processor is

busy in x . At a decision epoch, f_j is applied after f_1, f_2, \dots, f_{j-1} have been applied sequentially to their corresponding job classes.

We shall say that a state x is stable under f_j if $f_j(x) = x$, and unstable otherwise. At any decision epoch, the dispatching functions are sequentially applied to all classes until the state becomes stable under the dispatching functions. Then the processors operate on the jobs assigned to them for one unit of time, at which instant the state evolves and may be unstable.

The expected flowtime of class C_j jobs under $f \equiv (f_1, f_2, \dots, f_n)$ with the initial state x will be denoted by $c_j(x; f)$. It is convenient to define the following operators E_1 and E_2 . Let

$$(1) \quad E_1 c_j(\alpha_1 m_1, \alpha_2 m_2; k; f) = h_{1\alpha_1}(m_1) c_j(00, \alpha_2 m_2; k; f) + \bar{h}_{1\alpha_1}(m_1) c_j(\alpha_1(m_1 + 1), \alpha_2 m_2; k; f)$$

and

$$(2) \quad E_2 c_j(\alpha_1 m_1, \alpha_2 m_2; k; f) = h_{2\alpha_2}(m_2) c_j(\alpha_1 m_1, 00; k; f) + \bar{h}_{2\alpha_2}(m_2) c_j(\alpha_1 m_1, \alpha_2(m_2 + 1); k; f).$$

In the above equations, $\bar{h}_{ij}(m) \triangleq 1 - h_{ij}(m)$. In addition, h_{i0} is taken as unity for the case when the processor is idle. The expected flowtimes for any given dispatching function can now be written in the following compact recursive form, for a stable state. For $1 \leq j \leq n$,

$$(3) \quad c_j(\alpha_1 m_1, \alpha_2 m_2; k; f) = k_j + I_j(\alpha_1) + I_j(\alpha_2) + E_2 E_1 c_j(\alpha_1 m_1, \alpha_2 m_2; k; f)$$

where $I_j(\alpha_i)$ is an indicator function which equals 1 if $\alpha_i = j$, and 0 otherwise. (Note that $E_1 E_2 \equiv E_2 E_1$.)

3. Optimal dispatching of class C_l jobs, $1 \leq l \leq n$

In this section, we analyze the dispatching strategy which minimizes the expected flowtime for class C_l jobs. The analysis here is also applicable to the case when there is only one class of jobs. We first introduce a threshold-type dispatching function for each job class, and then establish its optimality.

Definition 1. Let $x = (\alpha m; k)$ be any state of the system. Let the dispatching function π_1 assign class C_1 jobs to available processors in the following manner:

- (i) If $\alpha_1 = 0$ and $k_1 > 0$, assign a class C_1 job to processor P_1 ;
- (ii) If $\alpha_1 > 0, \alpha_2 = 0$ and $k_1 > 0$, assign a class C_1 job to P_2 if and only if $\chi_{1\alpha_1}(m_1) + k_1 \chi_{11} > \chi_{21}$.

Based on the dispatching function π_1 , we recursively define the following dispatching function $\pi_l, 1 < l \leq n$, for the class C_l jobs.

Definition 2. Let $x = (\alpha m; k)$ be any state of the system. Let the dispatching function π_j be used for class C_j jobs, $1 \leq j \leq l - 1$. The policy π_l which assigns class C_l jobs is defined as follows:

- (i) if $\alpha_1 = 0, \sum_{j=1}^{l-1} k_j = 0$ and $k_l > 0$, assign a class C_l job to P_1 ;
- (ii) if $\alpha_1 > 0, \alpha_2 = 0, k_l > 0$, and P_2 is rejected by all higher priority jobs, dispatch a C_l job to P_2 if and only if

$$(4) \quad \chi_{1\alpha_1}(m_1) + \sum_{j=1}^l k_j \chi_{1j} > \chi_{2l}.$$

From Definition 1, it is evident that if $\chi_{1\alpha_1}(m_1) + k_1 \chi_{11} \leq \chi_{21}$, then P_2 is not assigned to class C_1 jobs, and in addition, it is rejected at all future times by class C_1 jobs. This is due to assumption (A.1) and the fact that the number of C_1 jobs that remain to be processed does not increase over time. Likewise, if P_2 is rejected by all higher priority jobs, and if $\chi_{1\alpha_1}(m_1) + \sum_{j=1}^l k_j \chi_{1j} \leq \chi_{2l}$, then P_2 will be declined by the class C_l jobs, in x and all subsequent states of x . We refer to this property as the ‘monotonicity’ of $\pi_l, 1 \leq l \leq n$. This property implies that the set

$$(5) \quad B_l \triangleq \left\{ x = (\alpha m; k) \in S \mid \chi_{1\alpha_1}(m_1) + \sum_{i=1}^j k_i \chi_{1i} \leq \chi_{2j}, 1 \leq j \leq l \right\}$$

is closed with respect to the evolution of the state. Due to this property, the flowtime of class C_l jobs under π_l does not depend on the dispatching function and the buffer states of the lower priority jobs.

In state x , let $c_l(x)$ denote the expected flowtime of C_l jobs when policies π_1, \dots, π_{l-1} , and π_l are used. Then, for $x = (\alpha m; k) \in B_l$, the flowtime of C_l jobs can be explicitly written as:

$$(6) \quad \begin{aligned} c_l(\alpha m; k) &= I_l(\alpha_1) \chi_{1\alpha_1}(m_1) + I_l(\alpha_2) \chi_{2\alpha_2}(m_2) \\ &+ \sum_{j=1}^l \sum_{i=1}^{k_j} \left(\chi_{1\alpha_1}(m_1) + \sum_{p=1}^{j-1} k_p \chi_{1p} + i \chi_{1j} \right) \end{aligned}$$

where $\sum_{p=1}^{j-1} k_p \chi_{1p} = 0$, for $j = 1$. Before we establish the optimality of π_l , the following observation is of interest. For class C_l jobs, consider the class F_l of threshold policies with the property: (i) P_1 is used whenever it is offered to the C_l jobs, and (ii) once P_2 is rejected by the C_l jobs, it is never used by the class again. For every job class, if the attention is restricted to the threshold dispatching strategies, the scheduling problem for each job class may be viewed as an optimal stopping problem, i.e., when to stop using P_2 . Clearly, π_l belongs to F_l . Its optimality among policies in F_l can be shown via the one-step look-ahead rule for stopping problems (Ross (1983)).

Lemma 1. For $1 \leq l \leq n$, the policy π_l minimizes the expected flowtime of class C_l jobs among dispatching strategies in F_l .

Proof. Consider the state of the form $x = (\alpha_1 m_1, 00; k)$, and let P_2 be rejected by every higher priority class C_j under the policy $\pi_j, j = 1, \dots, l-1$ (such that $\chi_{1\alpha_1}(m_1) + \sum_{i=1}^{j-1} k_i \chi_{1i} \leq \chi_{2j}, 1 \leq j \leq l-1$). Consider the set, \tilde{B}_l , of states for which declining P_2 is at least as good as using P_2 exactly once. Using Equation (6), the expected flowtime of the class C_l from declining P_2 is

$$I_l(\alpha_l)\chi_{l\alpha_l}(m_l) + \sum_{j=1}^l \sum_{i=1}^{k_j} \left(\chi_{l\alpha_l}(m_l) + \sum_{p=1}^{j-1} k_p \chi_{lp} + i\chi_{lj} \right).$$

On the other hand, if P_2 is used exactly once, the expected flowtime is

$$I_l(\alpha_l)\chi_{l\alpha_l}(m_l) + \chi_{2l} + \sum_{j=1}^{l-1} \sum_{i=1}^{k_j} \left(\chi_{l\alpha_l}(m_l) + \sum_{p=1}^{j-1} k_p \chi_{lp} + i\chi_{lj} \right) + \sum_{i=1}^{k_l-1} \left(\chi_{l\alpha_l}(m_l) + \sum_{p=1}^{l-1} k_p \chi_{lp} + i\chi_{ll} \right).$$

The difference of the above expressions is $\chi_{l\alpha_l}(m_l) + \sum_{i=1}^{l-1} k_i \chi_{li} - \chi_{2l}$. From (5) it is clear that $\tilde{B}_l = B_l$. In this Markov optimal stopping problem, the set \tilde{B}_l is closed under the evolution of the state, and hence Theorem 2.2 in Ross ((1983), p. 54, Chapter 3) is applicable. This yields the optimality of the following one-step look-ahead policy: stop using P_2 when the state enters the set \tilde{B}_l for the first time. This policy is identical to π_l , hence the lemma is established.

Note that the scheduling problem at hand is not a pure stopping problem, because P_2 can be rejected at a certain time and used later. Thus, we need to establish the optimality of π_l amongst all dispatching functions.

To facilitate proofs by induction we consider any one ordering of the state space S , namely $x_0 = (0; 0), x_1, x_2 \dots$, such that if x_i is reachable under some sample path from x_j then $x_i < x_j$. It is clear that such an ordering of S is possible.

The following lemma is helpful in establishing the optimality of π_l . Intuitively, Lemma 2 states that if P_i is operated for one unit of time costlessly (instantaneously) keeping the other components of the state x fixed, the resulting flowtime of C_i jobs (which equals $E_i c_i(x)$, see (1) and (2)) is reduced, since the delay or completion time of the job in P_i , and also of other C_i jobs subsequently processed on P_i , is decreased by one unit. Define $\Delta_i c_i(x) = c_i(x) - E_i c_i(x)$, for $i = 1, 2$.

Lemma 2. If dispatching strategies π_1, \dots, π_{l-1} , and π_l are followed, then for any stable state $x = (\alpha m; k)$,

$$(7) \quad I_l(\alpha_l) \leq \Delta_l c_l(x) \leq k_l + I_l(\alpha_l), \quad i = 1, 2, \quad l = 1, \dots, n.$$

Proof. Consider the ordering of S defined earlier. For each given l , the proof is by induction on states. The lemma trivially holds for x_0 and also for all states with $\sum_{i=1}^l k_i = 0$. This is because, for such states, $\Delta_l c_l(x) = I_l(\alpha_l)[\chi_{l\alpha_l}(m_l) - \tilde{h}_{l\alpha_l}(m_l)\chi_{l\alpha_l}(m_l + 1)] = I_l(\alpha_l)$, for $i = 1, 2$. Now, suppose that the lemma holds for all states x_t , with the index $t \leq I - 1$, where I is an integer. We need to show that it holds for $x_I = x = (\alpha m; k)$, with $\sum_{i=1}^l k_i > 0$. Let $C_j, 1 \leq j \leq l$, be the highest indexed job class which has (non-zero) waiting jobs in its buffer (i.e., $\sum_{i=1}^{j-1} k_i = 0$ and $k_j > 0$). Let ϵ_j denote a vector with a 1 in component j and 0 in all other components. From (1) and (3), one obtains

$$\Delta_l c_l(x) = k_l + I_l(\alpha_l) + I_l(\alpha_2) - h_{l\alpha_l}(m_l)\Delta_2 c_l(j0, \alpha_2 m_2; k - \epsilon_j) - \tilde{h}_{l\alpha_l}(m_l)\Delta_2 c_l(\alpha_l(m_l + 1), \alpha_2 m_2; k).$$

Let y denote either of the two states which appears as an argument to c_l above. Note that y is stable and is such that $y < x_l = x$. Hence the induction hypothesis is applicable. Now, the lower bound (upper bound) for $\Delta_l c_l(x)$ in the lemma is trivially verified by using the upper bound (lower bound) for $\Delta_l c_l(y)$ from (7). This establishes the lemma for $i = 1$. The proof for the case $i = 2$ is similar.

We are now ready to establish the main result.

Theorem 1. For $1 \leq l \leq n$, the dispatching function π_l minimizes the expected flowtime of class C_l jobs among all non-preemptive dispatching functions, i.e.,

$$c_l(x) \leq c_l(x; f_n \cdots f_l \pi_{l-1} \cdots \pi_1).$$

Remark. Suppose we relax our definition of admissible policies by allowing class C_l jobs to preempt jobs of lower priority classes (except the jobs assigned before time 0). This relaxation means that when determining the optimal policy for class C_l jobs we can pretend that no jobs of lower priority classes are present. Since it will be shown that the optimal policy is π_l and this policy never assigns a class C_l job to a processor which has been previously declined for class C_l jobs, it is clear that the option to preempt is never exercised. Thus π_l is also optimal in the restricted class of policies which do not allow preemption. We write $c_l(x; f_n \cdots f_l \pi_{l-1} \cdots \pi_1) = c_l(x; f_l)$, when no lower priority jobs are present.

Proof of Theorem 1. The proof by induction on l is employed. The base of the induction is the trivial case: no jobs need to be assigned. Assume strategy π_j minimizes the expected flowtime of class C_j jobs, among all non-preemptive dispatching functions, for $j < l$. By the remark above, we may suppose that no jobs of classes $j > l$ are present. Using Lemma 1 we see that π_l minimizes the expected flowtime of class C_l jobs within the class of threshold policies for assignments of class C_l jobs. Now consider the class of all policies for C_l jobs given that policy π_j is used to assign C_j jobs, $j < l$. Let f_l be an optimal policy and let x be a *least state* for which π_l does not equal f_l . By this we mean that π_l is not optimal in state x but that it is optimal for all states y that are reachable from x . If π_l is not optimal in the class of all non-preemptive policies then there is at least one such x . If processor P_1 is idle and offered to C_l jobs, then x must be of the form $x = (00, \alpha_2 m_2, k)$, with $\sum_{i=1}^{l-1} k_i = 0$ (since the higher priority jobs, if there are any left, never decline P_1). It is clear that the optimal policy for C_l jobs should never decline the faster available processor P_1 . This statement can be justified formally as follows. Suppose that f_l prescribes not using P_1 in state x , so that $f_l(x) = (00, \beta_2 n_2; k^*)$, where k^* equals k or $k - \epsilon_l$ depending on whether or not a C_l job is assigned to P_2 . Consider a policy $\tilde{\pi}_l$ which is the same as f_l except that it prescribes the assignment of a C_l job to P_1 in state x . Then,

$$\begin{aligned} c_l(x; f_l) - c_l(x; \tilde{\pi}_l) &= c_l(00, \beta_2 n_2; k^*; f_l) - c_l(10, \beta_2 n_2; k^* - \epsilon_l; \tilde{\pi}_l) \\ &= E_2[c_l(10, \beta_2 n_2; k^* - \epsilon_l) - E_1 c_l(10, \beta_2 n_2; k^* - \epsilon_l)] \geq 0, \end{aligned}$$

where the inequality to zero follows from Lemma 2. This implies that using P_1 whenever it is available is optimal.

Now let x be of the form $x = (\alpha_1 m_1, 00; k)$ and let x be stable under π_j (i.e., $x \in B_j$), for $j < l$. There are two cases to consider, depending on whether or not $(\alpha_1(m_1 + 1), 00; k) = y$ belongs to B_l .

(i) Suppose $y \in B_l$. Due to monotonicity property, all states that result after one unit of processing of state x must be in B_l , regardless of the action taken in state x . Thus $f_l = \pi_l$ makes no further assignments of C_l jobs to P_2 from the next decision epoch. This implies that f_l must be of a threshold type. Consequently, the optimality of π_l follows from Lemma 1.

(ii) Suppose $y \notin B_l$. In this case we have $y < x \notin B_l$. Let P_2 be rejected by f_l in state x . Since $y \notin B_l$, with some positive probability $f_l = \pi_l$ will assign a C_l job to P_2 at the next decision epoch. Let C_j be the highest indexed class with (non-zero) waiting jobs. We compare the expected flowtime of class C_l jobs under the two policies:

$$(8) \quad \begin{aligned} & c_l(\alpha_1 m_1, 00; k; f_l) - c_l(\alpha_1 m_1, l0; k - \varepsilon_l) \\ & = h_{l\alpha_1}(m_1)[c_l(j0, 00; k - \varepsilon_j) - E_2 c_l(j0, l0; k - \varepsilon_j - \varepsilon_l)] \\ & \quad + \bar{h}_{l\alpha_1}(m_1)[c_l(\alpha_1(m_1 + 1), l0; k - \varepsilon_l) - E_2 c_l(\alpha_1(m_1 + 1), l0; k - \varepsilon_l)]. \end{aligned}$$

Now if $(j0, 00; k - \varepsilon_j) \notin B_l$ then with probability 1 a C_l job will be assigned to P_2 at the next step. In this circumstance it is better to assign a C_l job to P_2 initially. This can be formally verified by using Lemma 2 in Equation (8). On the other hand, if $(j0, 00; k - \varepsilon_j) \in B_l$, then using Lemma 2 in Equation (8) again we see that the difference of the flowtimes under the two policies is no lesser than

$$h_{l\alpha_1}(m_1) \left[\sum_{i=1}^l k_i \chi_{li} - \chi_{2l} + 1 \right] + \bar{h}_{l\alpha_1}(m_1).$$

The above quantity is negative only if

$$\sum_{i=1}^l k_i \chi_{li} < \chi_{2l} - \frac{1}{h_{l\alpha_1}(m_1)}.$$

But using (A.1), $\chi_{l\alpha_1}(m_1) = 1 + \bar{h}_{l\alpha_1}(m_1)\chi_{l\alpha_1}(m_1 + 1) \leq 1 + \bar{h}_{l\alpha_1}(m_1)\chi_{l\alpha_1}(m_l)$, so $1/h_{l\alpha_1}(m_1) \geq \chi_{l\alpha_1}(m_1)$. Therefore the above expression requires $\sum_{i=1}^l k_i \chi_{li} < \chi_{2l} - \chi_{l\alpha_1}(m_1)$, which contradicts $x \notin B_l$. This completes the proof of Theorem 1.

4. Individual optimality

The socially optimal policy π_l , $1 \leq l \leq n$, is also individually optimal for each job in class C_l in the following sense. Suppose jobs in priority class C_l occupy buffer positions $1, 2, \dots, k_l$ (with position 1 being the head of the queue). A policy for a job which specifies when an offered processor should be used or declined, is called an individual policy. An individually optimal policy is one which for each job minimizes its own expected completion time (delay), with processor preference given to other jobs ahead of it in the buffer (of the same priority class). It may be noted that the policy π_l can be implemented as an individual policy γ_l as follows: for a class C_l job in any position k_l ,

(i) accept P_1 whenever offered, and (ii) accept P_2 when offered if and only if (4) is satisfied.

It is not true in general that individually and socially optimal policies coincide. However, Kumar and Walrand (1985) have shown that this is the case when there are no arrivals, and the socially optimal policy π can be implemented as an individual policy γ , having the property that once a job rejects an offered processor it never wishes to use it thereafter. In this case, γ is an individually optimal policy.

In the priority class system analyzed in the previous sections, the policy π_l , $1 \leq l \leq n$ is individually implementable. Furthermore, due to the monotonicity property, a job will never accept a processor which was declined previously. Thus, from Theorem 1 in Kumar and Walrand (1985), we can conclude as follows.

Theorem 2. For $1 \leq l \leq n$, the policy γ_l , which is the individual implementation of π_l , is individually optimal for each job in class C_l .

If we extend the model to include job arrivals to each class, with FCFS priority within each class, then in general π_l will not be socially optimal and γ_l will not be individually optimal. (Although, by Theorem 2 in Kumar and Walrand (1985), γ_1 will be individually optimal for the highest priority class.) However, if jobs of all classes are held in just one queue and free processors are offered to jobs in order of their position in the queue, without regard to class, then the individually optimal policy for each job is simply to accept P_2 if its expected processing time on P_2 is less than the expected total delay which will be incurred by waiting for itself and all jobs ahead of it to be processed on P_1 . However, in general, this policy is not socially optimal.

5. Concluding remarks

In this paper we have studied the non-preemptive scheduling of jobs of multi-priority classes, having non-increasing mean residual processing time distributions, on two heterogeneous processors. The optimal strategy for each priority class was shown to be of a threshold type. These thresholds can be derived off-line and be implemented as a table of dispatching rules. In the model considered, social and individual optimality coincide. The conjecture of Kumar and Walrand ((1985), p. 995) is thus proved for the two-processor case under conditions weaker than they had suggested. Standard limiting arguments can be used to show that the results derived here for discrete processing time distributions also hold for continuous processing time distributions.

The problem of optimally dispatching priority jobs to multiple heterogeneous processors ($m > 2$) poses an interesting research topic. In such circumstance to assure the optimal dispatching strategy for each job class to be of a threshold type, we believe that we need to strengthen (A.1) to

(A.1') $h_{ij}(t)$ is non-decreasing in t for $i = 1, \dots, m, j = 1, \dots, n$.

The proof of the optimality of the threshold policy may be carried out as follows. Define $c_{ij}(x)$ as the minimal expected flowtime of C_j jobs given that all further C_j job assignments are made only to processors P_1, \dots, P_i . Define

$$B_{ij} = \{x = (\alpha m, k) \in S \mid c_{ij}(\alpha m, k) - c_{ij}(\alpha m, k - \varepsilon_j) \leq \chi_{(i+1)j}\}.$$

That is, B_{ij} is the set of states for which the confining of all future C_j job assignments to P_1, \dots, P_i is preferable to assigning one more C_j job to P_{i+1} . If it can be shown that B_{ij} is a closed set, it would follow that π_j (with stopping set B_{ij} for processor P_{i+1} , $i = 1, \dots, m - 1$) is optimal in the class of threshold policies. Similar arguments to those in this paper would show that π_j is optimal in the class of all policies.

There are several other interesting topics for future research. In this paper the class priorities were taken as given. The problem of assigning priorities to each class (given that each class minimizes its flowtime) to minimize the total expected flowtime of the entire system is a problem of interest. If priority constraints are removed, the problem is one of scheduling heterogeneous jobs on heterogeneous processors. This problem, which has received very little attention, does not appear to have a solution with a simple characterization. Optimal scheduling to minimize expected makespan in this context is also a topic for further research.

References

- AGRAWALA, A. K., COFFMAN, E. G. JR., GAREY, M. R. AND TRIPATHI, S. K. (1984) A stochastic optimization algorithm minimizing expected flowtime on uniform processors. *IEEE Trans. Computers* **33**, 351–357.
- COFFMAN, E. G., FLATTO, L., GAREY, M. R. AND WEBER R.R. (1987) Minimizing expected makespan on uniform processors systems. *Adv. Appl. Prob.* **19**, 177–201.
- KUMAR, P. R. AND WALRAND, J. (1985) Individually optimal routing in parallel systems. *J. Appl. Prob.* **22**, 989–995.
- MIRCHANDANI, P. B. AND XU, S. H. (1989) Optimal dispatching of multi-priority jobs to two heterogeneous workstations. *IIFMS* **2**, 25–42.
- ROSS, S. M. (1983) *Introduction to Stochastic Dynamic Programming*. Wiley, New York.
- WEBER, R. R. (1981) Scheduling jobs on parallel machines to minimize makespan or flowtime. In *Proc. Conf. on Applied Probability, Computer Science: The Interface, Boca Raton*.
- WEBER, R. R. (1982) Scheduling jobs with stochastic process requirements on parallel machines to minimize makespan or flowtime. *J. Appl. Prob.* **19**, 167–182.
- WEBER, R. R., VARAIYA, P. AND WALRAND, J. (1986) Scheduling jobs with stochastically ordered processing times on parallel machines to minimize expected flowtime. *J. Appl. Prob.* **23**, 841–847.
- WEISS, G. AND PINEDO, M. (1980) Scheduling tasks with exponential service times on non-identical processors to minimize various cost functions. *J. Appl. Prob.* **17**, 187–202.