1975

Some Insights Into Deterministic Scheduling Problems Involving Multiple Resources and Preemption

K. Omahen

Report Number:
75-155

Some Insights Into Deterministic Scheduling Problems Involving Multiple Resources & Preemption

K. Omahen
Computer Science Department
Purdue University
West Lafayette, IN 47907

CSD-TR 155
July 1975

Keywords: deterministic scheduling, preemptive-resume rules, multiple-resource scheduling, minimizing completion time

Abstract

This paper considers deterministic scheduling problems for systems having a number of different resource types and an arbitrary number of units of each resource. General results are obtained for preemptive-resume scheduling rules under the condition of zero preemption costs. Schedule completion time is the objective function considered, and the paper examines the relative effect of demand scheduling and precedence constraints on the minimum schedule completion time.

Introduction

With the exception of multiple-server systems, few scheduling problems arising from multiple-resource systems have been treated in the literature. This paper attempts to offer insights into one class of multiple-resource scheduling problems and to provide some general results as a by-product. A formal description of the problem to be considered is given below:

Problem.

We are given a system having the following resource characteristics:

I = No. of Types of Resources

R_i = Amount of Resource-Type-i in System, $i = 1,2,\ldots,I$.

A set of jobs as described below are to be scheduled so as to minimize schedule completion time (i.e., the time to finish all jobs):

N = No. of Jobs to be scheduled.

J_n = Symbol denoting the nth Job, $n = 1,2,\ldots,N$.

$V_n = [r_{n1}, r_{n2}, \ldots, r_{nI}]$ = Vector describing the resource requirements of J_n; i.e., r_{n1} units of Resource-Type-1, r_{n2} units of Resource-Type-2, etc.

P_n = Processing Time for J_n; this is the amount of time that the resources V_n are simultaneously required. If the resource requirement V_n is satisfied, job J_n progresses at unit rate.

The scheduling rule to be used is assumed to be of the preemptive-resume type with zero preemption costs, and the four cases to be considered are:

Case 1. No Precedence Constraints, Non-Demand Scheduling

Case 2. No Precedence Constraints, Demand Scheduling

Case 3. Precedence Constraints, Non-Demand Scheduling

Case 4. Precedence Constraints, Demand Scheduling

Problems examined in the past have assumed that the scheduling decision involves choosing the next job to be scheduled; such a viewpoint is a reasonable one to take when developing heuristic rules but ignores the fundamental nature of the problem. This author believes that the following alternative view is more appropriate;

Viewpoint. Scheduling rules for multiple-resource systems should, at a scheduling epoch, be concerned with the choosing of some combination of jobs whose characteristics allow for simultaneous processing.

In order to demonstrate the utility of this approach, additional notation must be introduced:

M = Number of distinct feasible job combinations

C_m = $N \times 1$ column vector denoting a feasible combination of jobs,

= a subset of the set of jobs $\{J_n | 1 \leq n \leq N\}$,

$$= \begin{bmatrix} X_{m1} \\ X_{m2} \\ \vdots \\ X_{mN} \end{bmatrix} \qquad \text{where } X_{mn} = \begin{cases} 1 \text{ if } J_n \text{ is included in } C_m, \\ 0 \text{ otherwise.} \end{cases}$$

For a job combination C_m to be "feasible," we require that

$$\sum_{j=1}^{N} X_{mj}{*}r_{ji} \leq R_i \ , \quad i = 1,2,\ldots,I.$$

That is, the total resources required of all jobs in the combination must not exceed the amount actually present in system.

If we pursue the viewpoint that the scheduling rule should be concerned with choosing combinations of jobs for processing, we may denote a schedule S as

$$S = [(i_1,t_1),(i_2,t_2),\ldots,(i_K,t_K)]$$

where each pair (i_k,t_k) is interpreted to be a <u>combination scheduling interval</u> in which feasible combination with index i_k is to be serviced for t_k units of time.

The above notation specifies the first through Kth combination scheduling intervals which constitutes the schedule.

Some Insights

Observe that it is immediately possible to make a statement concerning the minimum schedule completion time for the situation in which there are neither precedence constraints on the order in which jobs must be processed nor any requirement for demand processing.

<u>Lemma 1</u>. The minimum schedule completion time T_1 for Case I (Preemptive-Resume Scheduling, No Precedence Constraints, Non-Demand Scheduling) is the solution of the following Linear Programming (LP) problem:

$$T_1 = \mathrm{Min} \sum_{m=1}^{M} t_m$$

subject to the N equality constraints

$$\sum_{m=1}^{M} X_{mn} * t_m = P_n \quad ; \quad n = 1,2,\ldots,N$$

where t_m = (non-negative) amount of time that the mth combination, C_m, is to be serviced in total to obtain the minimum schedule completion time;

P_n = processing time for job J_n;

X_{mn} = element of column matrix C_m indicating whether job J_n is present in the combination.

Proof. The schedule completion time is the sum of the amount of time spent in processing each possible combination of jobs, and the constraints reflect the requirement that, for each job, the amount of time spent in combinations in which the job "progresses" must equal the processing time request of that job. □

Given a solution to the above LP problem, a schedule S may be easily formed; for example, any permutation of those combination scheduling intervals (m,t_m) for which t_m has a non-zero value would be acceptable. Well-known LP results also imply that no more than J of the combinations need be assigned non-zero values and that the set of J column vectors, A, will be linearly independent, where

$$A = \{C_m | t_m > 0\}.$$

It should be obvious that imposing demand scheduling and/or precedence constraints on the set of jobs can never result in a smaller schedule completion than would be obtained using Lemma 1. However, there might exist situations in which the minimum schedule completion time might be increased due to demand scheduling and/or precedence constraints, and therefore these factors will be next examined.

For multiple-server systems, demand scheduling has been taken to mean that no processor will remain idle if there is some job which might be serviced. For multiple-resource systems this definition is not adequate because ambiguities arise when there are many types of resources in system.

If we consider a scheduling epoch to be the instant at which a different combination of jobs goes into service, a more suitable definition of demand scheduling may be given in terms of a "maximal combination."

<u>Maximal Combination</u> (when no precedence constraints): For a given set of jobs in a multiple-resource system, each with resource-requirements and (remaining) processing time requests, a (feasible) combination C_i is said to be a maximal combination if there does not exist another (feasible) combination C_j such that C_i is properly contained in C_j (i.e., such that every job in C_i is also contained in C_j).

Having defined a maximal combination, demand scheduling may be better described as shown below.

<u>Demand Scheduling</u>: At each scheduling epoch, a maximal combination (for the remaining jobs in system) is chosen for processing.

The next result to be presented will apply to Case 2 (i.e., with demand scheduling imposed on Case 1); this result demonstrates that the minimum schedule completion time T_2 will always be the same as that possible for Case 1 where demand scheduling is not required.

<u>Lemma 2</u>. The minimum schedule completion time T_2 for Case 2 (Preemptive-Resume Rule, No Precedence Constraints, Demand Scheduling) will always be equal to the value T_1 of Lemma 1.

<u>Proof</u>. Given a schedule of minimum length T_1 given by Lemma 1, a schedule which meets the requirements of demand scheduling may always be constructed using the algorithm given in Appendix 1. $\square$

Demand scheduling has the effect of imposing constraints on the order in which combinations may be processed because, whenever a combination C_i is properly contained in some combination C_j, combination C_i may be chosen for servicing only if it is impossible to form combination C_j from the remaining uncompleted jobs in system.

Job precedence constraints specify a partial ordering on the set of jobs, where

$$J_i < J_n \qquad i \neq n$$

implies that the <u>i</u>th job must be processed to completion before the servicing of the <u>nth</u> job can commence. The effect of an arbitrary set of job precedence constraints is illustrated in the following lemma:

<u>Lemma 3</u>. The minimum schedule completion time T_3 for Case 3 (Preemptive-Resume Rule, Precedence Constraints, Non-Demand Scheduling) will always be greater than or equal to the value T_1 of Lemma 1.

<u>Proof</u>. Consider the effect of the job precedence constraints on the possible schedules that might be generated for Case 3.

(a) We must first eliminate from consideration any job combination C_m such that $J_i \in C_m$, $J_n \in C_m$, and $J_i < J_n$ for $i \neq n$.

(b) For the remaining job combinations, the job precedence relations have the following effect:

(i) A combination precedence relation between C_k and C_m will be denoted as $C_k < C_m$ if each job precedence relation between a job in C_k and one in C_m is of the form $J_i < J_n$, where $J_i \in C_k$ and $J_n \in C_m$.

(ii) If $J_i < J_n$ and $J_j < J_\ell$ for some $J_i, J_\ell \in C_k$ and $J_j, J_n \in C_m$, then combinations C_k and C_m may not be present in the same schedule. That is,

$$t_k > 0 \text{ implies } t_m = 0$$

and

$$t_m > 0 \text{ implies } t_k = 0$$

where t_k and t_m are combination processing times.

(iii) If there are no precedence relations between any pair of different jobs, one in combination C_i and the second in combination C_m, there is no combination precedence relation defined between C_i and C_m.

The meaning of a combination precedence relation, $C_i < C_m$, is that a schedule containing combination scheduling intervals for both C_i and C_m must be such that the interval for C_i precedes the one for C_m. The proof is completed as follows. A well-known LP result is that the solution to the minimization problem of Lemma 1 lies at an extreme point for which at most J combinations are assigned non-zero combination processing times. Situations (a) and (b - ii) described above cause one or more of the extreme points to be excluded due to the presence of the job precedence constraints, and it follows that $T_3 \geq T_1$. □

We complete the coverage of the four cases by considering the effect of having both precedence constraints and a requirement for demand processing (Case 4). The approach taken here is similar to that for Lemma 2 except that it will be shown that, given a solution to the Case 3 situation, one may always construct a demand schedule having the same completion time. We again interpret demand scheduling to mean that a maximal combination (for the remaining jobs in system) is chosen at each scheduling epoch. However, the presence of precedence constraints causes the definition of maximal combination to be modified:

<u>Maximal Combination</u> (when precedence constraints): Given a set A of jobs $\{J_n\}$ having non-zero (remaining) processing times, we consider only those combinations which can be formed from set B, a subset of A having the following properties:

$$B = \{J_n \mid J_n \in A \text{ and } \nexists\, J_m \ni (J_m < J_n \text{ and } J_m \in A)\}$$

That is (in the context of the jobs in A), we consider only those jobs for which either no precedence relations are defined or which no job precedes. A (feasible) combination C_i is said to be a maximal combination if there does not exist any other (feasible) combination C_j such that C_i is properly contained in C_j, where both C_i and C_j are formed from jobs in B only.

<u>Lemma 4</u>. The minimum schedule completion time T_4 for Case 4 (Pre-emptive-Resume Rule, Precedence Constraints, Demand Scheduling) will always be the same as the minimum schedule completion time T_3 for Case 3.

<u>Proof</u>. Given a minimum length schedule for Case 3, a schedule for Case 4 may always be constructed using the same algorithm given in Appendix 1. Further details of the algorithm as it applies to the case where both precedence constraints and demand scheduling are present is given in Appendix 2.

The preceding lemmas will allow the following summary to be given for the relative effects of job precedence constraints and demand scheduling on minimum schedule completion time for preemptive-resume rules for multi-resource systems.

<u>Theorem</u>. For a multiple-resource system with a set of N jobs $\{J_n | 1 \leq n \leq N\}$ to be scheduled, where each job J_n has an associated resource requirement V_n and processing time P_n, let T_i denote the minimum schedule completion time possible under each alternative-i listed below:

(1) Preemptive-Resume Rule, No Precedence Constraints, Non-Demand Scheduling

(2) Preemptive-Resume Rule, No Precedence Constraints, Demand Scheduling

(3) Preemptive-Resume Rule, Precedence Constraints, Non-Demand Scheduling

(4) Preemptive-Resume Rule, Precedence Constraints, Demand Scheduling

The relative values of T_i are given by the following relations:

$$T_1 = T_2; \qquad T_1 \leq T_3; \qquad T_2 \leq T_3;$$

$$T_1 \leq T_4; \qquad T_2 \leq T_4; \qquad T_3 = T_4.$$

<u>Proof</u>. Immediately follows from Lemmas 1-4. $\square$

No mention has been made concerning the effect of non-preemptive scheduling; this has been avoided because non-preemptive disciplines considerably complicate matters. Non-preemptive scheduling requires that one become aware of the allowable transitions between job combinations, and the description of such transitions is somewhat unwieldy for the general case. It is interesting to note that no anomalies have been observed in the above theorem; this suggests that anomalies such as those observed for multiple-server systems are due to the interaction between the constraints imposed by nonpreemptive scheduling and those imposed by precedence constraints and demand scheduling.

Summary

This paper considered the deterministic scheduling problem for a multiple-resource system when preemptive-resume rules may be employed. The suggestion was made that the proper viewpoint to be taken for such a system is that the scheduling rule should be concerned with choosing the next combination of jobs to be serviced rather than with the selection of the next job for processing. Under the condition that schedule completion time is taken to be the objective function to be minimized, the following statements may be made when preemptive-resume rules are employed:

--- Job precedence constraints may cause an increase in the minimum schedule completion time over that possible if the constraints were not present.

--- Demand scheduling never causes the minimum schedule completion time to be greater than that possible if demand scheduling were not required.

--- The problem of minimizing schedule completion time when there are neither job precedence relations nor the requirement for demand processing is a conventional Linear Programming problem in which one determines the amount of time that each job combination should be serviced.

Bibliography

1. Omahen, K. Capacity bounds for multi-resource queues. Tech. Rpt. CSD-TR 137, Computer Science Department, Purdue University, March 1975.
2. Kafura, D. G. Analysis of Scheduling Algorithms for a Model of a Multiprocessor Computer System. Ph.D. Dissertation, Computer Science Department, Purdue University, August 1974.

Appendix 1

Suppose that we are given a schedule S_1 for Case 1 having minimum completion time and that we wish to construct a schedule for Case 2 by means of an algorithm which, at the kth step, insures that the combination chosen for the kth epoch meets the requirements of demand scheduling. Define:

$P_n(k)$ = remaining processing time for job J_n after the completion of the kth combination scheduling interval in schedule $S_2(k)$.

$S_2(k) = [(i_1,t_1),(i_2,t_2),\ldots,(i_K,t_K)]$

= schedule which results after k steps of the algorithm given below, where $S_2(0) = S_1$. Note that K need not have the same value for all $S_2(k)$.

The algorithm consists of repeating the following step for k = 1,2,... until a demand schedule is obtained.

Step - k: Assume that the first k-1 combination scheduling intervals each satisfy the requirements of demand scheduling. Examine the kth combination in schedule $S_2(k-1)$, i_k:

(a) If i_k is a maximal combination for the set of jobs $\{J_n | P_n(k-1) > 0, 1 \le n \le N\}$, $S_2(k)$ is taken to be $S_2(k-1)$, and we proceed to (c). Otherwise, go to (b).

(b) Given C_{I_k} is properly contained in some maximal combination $C_{I'_k}$, we may redistribute the remaining processing times of those jobs included in combination i'_k but not in i_k. Let B denote a set of job indices defined as follows:

$$B = \{n | J_n \in C_{I'_k} , J_n \notin C_{I_k}\}$$

We replace the kth combination scheduling interval in $S_2(k-1)$, (i_k,t_k), with the interval (i'_k,t'_k), where

$$t'_k = \min[t_k, \min_{n\in B} [P_n(k-1)]]) .$$

For each $n \in B$, we systematically modify the remainder of the schedule $S_2(k-1)$ so as to subtract the portion of the processing

(b) Continued.

times which are now included in the $\underline{k}$th interval. The quantity t_k' represents the amount of processing time for job J_n which must be subtracted off from the remaining combination scheduling intervals. These remaining intervals (i_{k+1},t_{k+1}), etc. must be sequentially examined and modified until this amount of processing time t_k' for job J_n has been properly subtracted. Suppose that we are examining (i_j,t_j) for $j > k$ and that w_n represents the processing time for job J_n which has yet to be subtracted ($w_n = t_k'$ initially). If $J_n \notin C_{i_j}$, we proceed to the next combination interval. Otherwise, one of the following actions takes place:

(i) If $t_j \leq w_n$, we replace (i_j,t_j) in the schedule with interval (i_j',t_j), where $C_{i_j'} = C_{i_j} - J_n$. Also, we decrement quantity w_n by amount t_j.

(ii) If $t_j > w_n$, replace (i_j,t_j) in the schedule with two combination scheduling intervals $(i_j',w_n),(i_j,t_j-w_n)$ where $C_{i_j'} = C_{i_j} - J_n$. Also, quantity w_n is set to zero.

For a given value of n, additional combination scheduling intervals are examined & modified until the amount of processing time t_k' for job J_n has been subtracted entirely. This process is then repeated until all values of n included in the set of indices B have been treated. The resulting schedule is then defined as $S_2(k)$.

(c) Update the remaining processing times; i.e.,

$$P_n(k) = P_n(k-1) - X_{mn}*t_k \text{ for } n = 1,2,\ldots,N$$

where (i_k,t_k) is the $\underline{k}$th combination scheduling interval in $S_2(k)$ and $m = i_k$.

This process is guaranteed to terminate in a finite number of steps, and the final value of $S_2(k)$ is a demand schedule of the same length as S_1 because the algorithm never causes the completion time to increase and a decrease in completion time would contradict the assumption that S_1 is of minimum length.

Appendix 2

The algorithm described in Appendix 1 works equally well when we are given a schedule S_3 for Case 3 having minimum completion time possible and wish to construct the corresponding demand schedule S_4 for Case 4. The key to believing that the algorithm still works properly is to understand the implications of the definition of demand scheduling when precedence constraints are present. As mentioned in the main body of the paper, the jobs included in a maximal combination are not preceded by any of the remaining jobs in system. It is this characteristic that prevents any conflicts in precedence constraints from arising when the algorithm systematically modifies the schedule.