

SCHEDULING ALGORITHMS FOR MASTER-SLAVE SYSTEMS*

Joseph Y-T. Leung and Hairong Zhao

*Department of Computer Science
New Jersey Institute of Technology
Newark, NJ 07102, USA*

Abstract We consider scheduling problems in the master-slave model. In this model each job has to be processed sequentially in three stages. In the first stage, a preprocessing task runs on a master machine; in the second stage, a slave task runs on a dedicated slave machine; in the last stage, a postprocessing task again runs on a master machine, possibly different from the master machine in the first stage. It has been shown that the problem of minimizing makespan or total completion time is NP-hard in the strong sense even if preemption is allowed. In this paper we design efficient approximation algorithms to minimize the total completion time in various settings. These are the first general results for total completion time problem in the master-slave model. We also show that these algorithms generate schedules with small makespan as well.

Keywords: Master-slave, scheduling, NP-hard, approximation algorithms, makespan, total completion time

1. Introduction

We consider scheduling problems in the master-slave model which was recently introduced by Sahni [13]. In this model, each job has to be processed sequentially in three stages. In the first stage, the preprocessing task runs on a master machine; in the second stage, the slave task runs on a dedicated slave machine; and in the last stage, the postprocessing task again runs on a master machine, possibly different from the master machine in the first stage. We use a_i , b_i and c_i to denote the preprocessing, slave and postprocessing tasks and task times of job i , respectively. We assume that $a_i > 0$, $b_i > 0$ and $c_i > 0$.

A job may have a release time $r_i \geq 0$, i.e., a_i cannot start until r_i . In this paper, we assume every job has release time 0, unless stated otherwise. There are two cases when arbitrary release time is present. The first case deals with offline problems, i.e., the release times and processing times of all jobs are known in advance. The second case deals with online problems, i.e., we have no knowledge of a job i until it arrives at r_i , and when it arrives, we know everything about job i . We use the quadruple (r_i, a_i, b_i, c_i) to denote job i . For simplicity, if $r_i = 0$, we use the triplet (a_i, b_i, c_i) to represent job i .

Each machine is either a master machine or a slave machine. The master machines are used to run preprocessing and/or postprocessing tasks, and the slave machines are used to run slave tasks, one slave machine for each slave task. In a single-master system, there is a single master to execute all preprocessing tasks (a tasks) and postprocessing tasks (c tasks). In a multi-master system, there are more than one master, each of which is capable of processing both a tasks and c tasks. Finally, in some systems, there are distinct preprocessing masters (preprocessors) and postprocessing masters (postprocessors), which are used to process a tasks and c tasks, respectively.

*This work was partially supported by NSF Grant DMI-0300156.

The master-slave model is closely related to the flow shop model. The system which has a single preprocessing master and a single postprocessing master can be seen as a two-machine flow shop with transfer lags ([17]). When there are more than one preprocessing and postprocessing masters, the model is a two-stage hybrid flow shop with transfer lags. In this sense, we can regard the previous case as a three-stage hybrid flow shop where the first and the last stage has a single machine and the second stage has n machines. In recent years, hybrid flow shop has received significant attention due to its applications, see [1], [9], [10] and [16].

The master-slave model finds many applications in parallel computer scheduling and industrial settings such as semiconductor testing, machine scheduling, transportation maintenance, etc. For more applications, see [13], [14], [15] and [18].

1.1 Preliminaries

Given a set of jobs in the master-slave system and a schedule S of the jobs, the completion (or finish) time of job i in S is the time when the postprocessing task c_i finishes. We denote the completion time of i in S by $C_i(S)$. If S is clear from the context, we use C_i instead of $C_i(S)$. The *makespan* of S is the earliest time when all the tasks have been completed. We denote the makespan of S by $C_{\max}(S)$, or C_{\max} if S is clear from the context. The *total completion time* of S , denoted by $C(S)$, is the sum of the completion times of all n jobs, i.e., $C(S) = \sum_{j=1}^n C_j$. Makespan and total completion time are two common objectives to minimize. Throughout this paper we use C_{\max}^* and C^* to denote the minimum makespan and the minimum total completion time, respectively. Without loss of generality, we always assume that the b tasks are scheduled immediately as soon as the corresponding a tasks complete.

An α -*approximation* algorithm for makespan (or total completion time) is an algorithm that for any set of jobs generates a schedule S whose makespan (or total completion time) is at most α times the optimal makespan (or total completion time). It is an (α, β) -*approximation* algorithm if it is an α -approximation algorithm for makespan and at the same time a β -approximation algorithm for total completion time. For a schedule S , if $C_{\max}(S) \leq \alpha C_{\max}^*$ and $C(S) \leq \beta C^*$, then we say that S is an (α, β) -schedule.

The Shortest-Processing-Time (SPT) rule, and the Shortest-Remaining-Processing-Time (SRPT) rule are two well-known algorithms for minimizing total completion time. Suppose each job consists of a single task. If all jobs are available at time 0, then the SPT rule is optimal for total completion time in the single-machine or multi-machine environment. If the release times are arbitrary, then the SRPT rule is optimal for a single machine and it is a 2-approximation (see [12]) in the multi-machine environment.

In this paper, we *adapt* the above two rules for our setting. We apply both rules to generate *preemptive* schedules. A scheduling decision is made when a task completes so that a master machine becomes free, or when a new a task or c task becomes available. At any such time instant, the SPT rule schedules, from the set of available tasks (including those that have been preempted but have not yet completed), the one with the smallest processing time. Depending on how we choose from the set of available jobs, we have the SPT_a rule and the SPT_c rule. Specifically, in the SPT_a rule, preemption occurs only among the a tasks. Similarly we have SPT_c rule. On the other hand, the SRPT rule schedules, from the set of available tasks, the one with the smallest remaining processing time. We define the $SRPT_a$ rule and the $SRPT_c$ rule similarly as above. Both the SPT rule and the SRPT rule may generate schedules with *migration* when there are multiple machines, i.e., after interrupted on one machine, a task is resumed on a different machine.

In this paper, we frequently sort the jobs in certain nondecreasing order of some parameters x_i of job i , e.g. $x_i = c_i$. For convenience, we say that $x_i < x_j$ as long as x_i comes before x_j in the sorted order, even though it may be the case that $x_i = x_j$.

Given a set of jobs in the master-slave system, we may have some constraints or requirements about how these jobs should be scheduled. In the following, we discuss these constraints in detail.

Canonical schedule versus non-canonical schedule. A *canonical* schedule on the single master system is one such that all the preprocessing tasks have to finish before any postprocessing tasks can start (Note that our definition of canonical schedule is slightly different from the one given in [13]). In the multi-master system, a canonical schedule is one that is canonical on each master.

Both canonical and non-canonical schedules have some applications. For both makespan and total completion time problems, it has been shown that the problems are NP-hard for both canonical and non-canonical schedules. It is easy to see that for a single master, one can always arrange a schedule to be canonical without increasing the makespan. Thus, for the problem of minimizing the makespan, we only need to focus on canonical schedules. However, for the problem of minimizing total completion time, the ratio of the total completion time of the best canonical schedule versus that of the best non-canonical schedule can be arbitrarily large.

Preemptive schedule versus non-preemptive schedule. For both makespan and total completion time problems, it has been shown that the problems are NP-hard even if preemption is allowed. For some instances, preemption does not help to reduce the makespan or total completion time. However, in most cases, preemption can reduce the total completion time and makespan.

1.2 Related works

So far the main research efforts to the master-slave model are for minimizing makespan and for special cases of total completion time. The general makespan problem has been shown to be NP-hard by Kern and Nawijn [8]. Leung and Zhao [11] strengthened the result and showed that the problem remains NP-hard in the strong sense, even if all the preprocessing and postprocessing tasks have unit time. In other words, the problem remains NP-hard in the strong sense even if preemption is allowed.

Sahni and Vairaktarakis [14], [18] proposed several heuristics for the makespan problem in the single-master, multi-master and distinct preprocessor and postprocessor systems. The total completion time minimization problem is considered only in [11] by Leung and Zhao. They showed that the problem of finding the optimal canonical or non-canonical schedule is NP-hard even if all preprocessing and postprocessing tasks have unit time. They designed efficient approximation algorithm to minimize the total completion time of canonical schedules on a single-master system when $a_i = a$ and $c_i = c$ for all jobs i .

For the two-stage flow shop with transfer lags model, some research has been done, most of which is about makespan minimization. Dell'Amico [4] proved that the makespan problem is NP-hard, even if preemption is allowed and each stage has only one machine. Later, Yu, Hoogeveen and Lenstra [19] showed that the problem is NP-hard even if all tasks have unit length. This is in contrast to the fact that the problem is solvable in polynomial time when there is no transfer lags. As we mentioned before, this model is the same as the master-slave model when the preprocessing and postprocessing masters are distinct. Thus, the heuristics given in [18] for the master-slave model also work here. Little is known about the total completion time minimization problem.

1.3 New results

In this paper, our main objective is to minimize total completion time under various scenarios. This includes single-master systems, multi-master systems, and distinct preprocessing and postprocessing master systems. Since the problem is strongly NP-hard, we can only hope to have approximation algorithms. For each type of system, we try to approximate the best preemptive

Table 1. New results for single-master system.

	preemptive	non-preemptive
$r_i = 0$	$(\frac{3}{2}, 2)$, canonical	$(\frac{5}{2}, \frac{2e}{e-1})$
$r_i \geq 0$	$(2, 2)$, online and non-canonical	$(3, 4)$

Table 2. New results for multi-master system, all schedules are non-canonical.

	preemptive	non-preemptive
$r_i = 0$	$(3, 2)$, no migration	$(5, 4)$
$r_i \geq 0$	$(3, 2)$, offline with no migration	$(5, 4)$
$r_i \geq 0$	$(3, 2)$, online with migration	-

Table 3. New results for distinct preprocessor and postprocessor, $m_1 = m_2 = 1$.

	preemptive	non-preemptive
$r_i = 0$	$(3, 2)$	$(4, \frac{2e}{e-1})$
$r_i = 0$	$(3, 2)$, offline	$(4, 4 + \frac{2e}{e-1})$
$r_i \geq 0$	$(3, 2)$, online	$(4, 4 + \frac{2e}{e-1})$

and non-preemptive schedules. We consider both the case when all jobs have the same release time and the case when the jobs have different release times.

Preemptive relaxation is an important technique for getting constant-factor approximations for total completion time of non-preemptive schedules; see [12], [2], and [6]. The advantage of preemptive relaxation is that usually there are very efficient algorithms to generate optimal or near optimal schedules. In most cases, these algorithms (both preemptive and non-preemptive) can be implemented to run in an online fashion, see [12].

In this paper, we use preemptive relaxation. We first develop efficient algorithms to generate preemptive offline or online schedules with good approximation. We show that these schedules have small makespan as well. Then, by applying the ideas in [12] and [3] to our models, we convert these preemptive schedules into non-preemptive schedules with certain degradation in the quality of approximation. Our results are summarized in Tables 1, 2, 3 and 4. In these tables, e denotes the base of natural logarithm.

The organization of this paper is as follows. In Sections 2, 3 and 4, we consider preemptive schedules. Section 2 is devoted to the single-master case, Section 3 is devoted to the multi-master case, and Section 4 is devoted to the case of distinct preprocessor and postprocessor. In Section 5, we consider the conversions from preemptive schedules into non-preemptive schedules. Finally, we draw some conclusions in the last section.

Table 4. New results for distinct preprocessor and postprocessor, $m_1 \geq 1$ and $m_2 \geq 1$.

	preemptive	non-preemptive
$r_i = 0$	$(4, 2)$, with migration	$(4, \frac{14}{3})$
$r_i \geq 0$	$(4, 3)$, offline with migration	$(5, 13)$
$r_i \geq 0$	$(4, 3)$, online with migration	$(5, 13)$

2. Single-master systems

In this section we assume that there is a single master. We first study canonical schedules, and then non-canonical schedules. Finally, we study the case when jobs have different release times.

2.1 Canonical preemptive schedules

We begin with two lower bounds of canonical schedules. Suppose we are given n jobs $1, 2, \dots, n$. Throughout this paper, we let $A = \sum_{j=1}^n a_j$, $B = \sum_{j=1}^n b_j$ and $C = \sum_{j=1}^n c_j$. For canonical schedules, preemption allowed or not, we have the following lower bound

$$C^* \geq nA + \sum_{j=1}^n \sum_{c_i \leq c_j} c_i, \quad (1)$$

which assumes that there is no idle time in the schedule and that the c tasks are scheduled in ascending order of their lengths. Another lower bound is

$$C^* \geq \sum_{j=1}^n \sum_{a_i \leq a_j} a_i + B + C, \quad (2)$$

which assumes that jobs are scheduled in increasing order of the a_i , and that the b tasks and the c tasks are scheduled immediately after they are available. Finally, we have the following trivial lower bound for any schedules which is simply the summation of all the processing times.

$$C^* \geq A + B + C. \quad (3)$$

In canonical schedules, all the a tasks are scheduled first. Since all the a tasks are available at time 0 and the c tasks cannot start until all the a tasks finish, there is no need to preempt the a tasks. Hence, only a c task can be preempted by another c task in this case.

Heuristic 1: Schedule the a tasks in an arbitrary order. After all the a tasks finish, schedule the available c tasks by the SPT_c rule.

THEOREM 1 *Heuristic 1 generates a (2, 2) canonical preemptive schedule in $O(n \log n)$ time.*

Proof : Let C_{a_j} denote the time a_j completes. Then at time $t_j = \max(A, (C_{a_j} + b_j))$, all the a tasks finish and the c_j is available to be scheduled. Since $C_{a_j} \leq A$, we have $t_j \leq A + b_j$. According to the heuristic, if there is another available task c_i that hasn't finished at time t_j and $c_i < c_j$, then c_j has to wait until c_i finishes. Also, during the execution of c_j , if there is another task $c_i < c_j$ that becomes available, then c_i preempts c_j . In both cases, we say that c_j is *delayed* by c_i . Let C_j be the completion time of c_j in the schedule generated by Heuristic 1. Then, we have

$$C_j = t_j + c_j + \sum_{c_i \text{ delays } c_j} c_i \leq (A + b_j + c_j) + \sum_{c_i < c_j} c_i.$$

The total completion time is

$$\sum_{j=1}^n C_j \leq \sum_{j=1}^n \left(A + b_j + c_j + \sum_{c_i < c_j} c_i \right) = \left(nA + \sum_{j=1}^n \sum_{c_i < c_j} c_i \right) + (B + C) < 2C^*,$$

where the last inequality comes from the lower bounds (1) and (3).

It has been shown in [14] that any canonical schedule without preemption is a 2-approximation for makespan. Since preemption among the c tasks can not increase the makespan, the schedule generated by Heuristic 1 has makespan at most two times the optimal. Together with the above bound of total completion time, this concludes the theorem. \square

In [14], an algorithm is developed which generates schedule with makespan at most $3/2$ times the optimal schedule. Thus in Heuristic 1, if we schedule the a tasks in the same order as the algorithm in [14], we will get a $3/2$ -approximation for makespan, since preemption on the available c tasks will not increase the makespan.

COROLLARY 2 *There is an $O(n \log n)$ time algorithm that generates a $(3/2, 2)$ canonical preemptive schedule.*

Note that when preemption occurs we use the SPT rule instead of the SRPT rule. This is for the purpose of analysis only. In practice, one can use the SRPT rule to get a better approximation for total completion time.

2.2 Non-canonical preemptive schedules

In this model, the a tasks and the c tasks can be scheduled alternatively. A lower bound on the C^* can be obtained by assuming all the b tasks have length 0:

$$C^* \geq \sum_{j=1}^n \sum_{a_i + c_i \leq a_j + c_j} (a_i + c_i) \quad (4)$$

Heuristic 2: For any two jobs, if $(a_j + c_j) < (a_i + c_i)$, then we say that both a_j and c_j have higher priority than a_i and c_i . At any time, if the master processor is free for assignment, assign the available task with the highest priority. If a new task becomes available and has higher priority than the currently running task, the new task preempts the currently running task.

THEOREM 3 *Heuristic 2 generates a $(2, 2)$ preemptive schedule for a single-master system when all jobs have the same release times.*

Proof : Due to space limit, we sketch the proof. Note that an a task or c task can be delayed only by those tasks with higher priority. Using the lower bounds (4) and (3), one can show that the total completion time is at most two times the optimal. For the makespan, pick the last job j such that c_j is scheduled immediately when it is ready. Such a job always exists since the job i with the highest priority must be such a job. Then, the interval $I_1 = [0, C_{a_j})$ and the interval after c_j starts till the end, i.e., $I_2 = [(C_{a_j} + b_j), C_{\max})$, contain no idle time. Denote their lengths by $|I_1|$ and $|I_2|$, respectively. Then $|I_1| + |I_2| \leq C_{\max}^*$. Thus, the makespan is

$$C_{\max} = |I_1| + b_j + |I_2| \leq C_{\max}^* + b_j < 2 C_{\max}^*$$

\square

2.3 Arbitrary release times

When arbitrary release times are present, it is not meaningful to have canonical schedules any more. Thus, we consider only non-canonical schedules. Apparently, the lower bound (4) still holds for this case. Let $R = \sum_{i=1}^n r_i$. Then, a trivial lower bound for the minimum total completion time of any schedule is

$$C^* \geq \sum_{i=1}^n (r_i + a_i + b_i + c_i) = R + A + B + C \quad (5)$$

Observe that Heuristic 2 makes no assumptions about the release time of a job. So, we can still apply Heuristic 2 when jobs have arbitrary release times. Unlike the case when all jobs have the same release times, in this case, a higher priority a task may also delay a lower priority a task or c task. Using similar argument, one can prove the following theorem.

THEOREM 4 *Heuristic 2 generates a (2, 2) preemptive schedule online for a single-master system even when the jobs have arbitrary release times.*

3. Multi-master systems

In this section, we assume that there are $m \geq 2$ masters, each of which is capable of processing both the a tasks and the c tasks.

3.1 Non-canonical preemptive schedules

Let us assume that the jobs are indexed in *nondecreasing* order of $a_j + c_j$. That is, $a_j + c_j \leq a_{j+1} + c_{j+1}$ for $1 \leq j \leq n - 1$.

Heuristic 3: (1) Without loss of generality, we may assume that n is a multiple of m . Otherwise, we add dummy jobs with $a_i = b_i = c_i = 0$. (2) We assign the jobs to the machines such that jobs $1, 2, \dots, m$ go to machines $1, 2, \dots, m$, respectively; jobs $m + 1, m + 2, \dots, 2m$ go to machines $m, m - 1, \dots, 1$, respectively; jobs $2m + 1, 2m + 2, \dots, 3m$ go to machines $1, 2, \dots, m$, respectively; and so on until all jobs are assigned. (3) Apply Heuristic 2 to each master machine to schedule the jobs assigned to it.

THEOREM 5 *Heuristic 3 generates a (3, 2) preemptive schedule for multi-master systems without migration when all jobs have release time 0.*

Proof : Without loss of generality, we may assume that $n = mk$ for some integer k . For convenience, we reindex the jobs assigned to each machine p in the form of (p, q) such that $a_{(p,q)} + c_{(p,q)} \leq a_{(p,q+1)} + c_{(p,q+1)}$. A lower bound of the total completion time comes from the fact that Heuristic 3 is optimal if $b_{(p,q)} = 0$ for every job (p, q) .

$$C^* \geq \sum_{p=1}^m \sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \quad (6)$$

Fix a machine p . Let $C_{(p,q)}$ denote the completion time of job (p, q) and $B_p = \sum_{q=1}^k b_{(p,q)}$. Following the analysis of Heuristic 2 in Section 1.2.2, we have

$$\sum_{q=1}^k C_{(p,q)} \leq \left(\sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \right) + B_p .$$

Thus, the total completion time is

$$\begin{aligned} \sum_{p=1}^m \sum_{q=1}^k C_{(p,q)} &\leq \sum_{p=1}^m \left(\sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) + B_p \right) \\ &\leq \sum_{p=1}^m \left(\sum_{q=1}^k (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \right) + B \quad \text{by (6) and (3)} \\ &< 2C^* . \end{aligned}$$

Now we analyze the makespan of the schedule. Let $\Delta = A + C$. Then,

$$C_{\max}^* \geq \frac{\sum_{j=1}^n (a_j + c_j)}{m} = \frac{\Delta}{m} .$$

Let $\Delta_p = \sum_{j \text{ scheduled on } p} (a_j + c_j)$. By the pigeon hole principle, $\min_{1 \leq p \leq m} \Delta_p \leq \Delta/m < C_{\max}^*$. For any two machines p and q , by the way we assign jobs to the machines

$$\Delta_p - \Delta_q \leq \max_{1 \leq k \leq n} (a_k + c_k) - \min_{1 \leq k \leq n} (a_k + c_k) \leq \max_{1 \leq k \leq n} (a_k + c_k) < C_{\max}^* .$$

This means that for any machine p , $\Delta_p \leq \min_{1 \leq q \leq m} \Delta_q + C_{\max}^* \leq 2C_{\max}^*$. Suppose the job with the maximum completion time among all jobs is assigned to machine p . Let l be the last job on machine p so that c_l is scheduled immediately after it is ready. Define C_{a_l} as before. Then machine p is busy during the intervals $I_1 = [0, C_{a_l}]$ and $I_2 = [(C_{a_l} + b_l), C_{\max}]$. The total length of the two intervals is $|I_1| + |I_2| \leq \Delta_p < 2C_{\max}^*$. Therefore, the makespan is

$$C_{\max} = |I_1| + b_l + |I_2| < 3C_{\max}^* .$$

□

3.2 Arbitrary release times

Offline schedule without migration. In this case, we can still apply Heuristic 3. However, note that to assign jobs to machines, Heuristic 3 requires that we have full knowledge of all the jobs at the beginning. Thus, Heuristic 3 is an offline algorithm. Combining the arguments in Sections 2.3 and 3.1, we have the following theorem.

THEOREM 6 *Heuristic 3 generates a (3, 2) offline preemptive schedule without migration for multi-master systems when jobs have arbitrary release times.*

Online schedule with migration. In this case, we apply Heuristic 2 to multi-master systems. Note that if a new task becomes available and one or more currently running tasks have lower priority than the new task, then the task with the lowest priority will be preempted.

THEOREM 7 *Heuristic 2 generates a (3, 2) online preemptive schedule with migration on multi-master systems when jobs have arbitrary release times.*

Proof : (sketch). First one can easily show the following bound:

$$C^* \geq \sum_{j=1}^n \sum_{a_i + c_i \leq a_j + c_j} (a_i + c_i)/m . \quad (7)$$

Let C_{a_j} denote the completion time of a_j in the schedule generated by Heuristic 2. Let $I_1 = [r_j, (C_{a_j} - a_j))$ and $I_2 = [C_{a_j} + b_j, (C_j - c_j))$. Then one can show there is no idle time during these two intervals, Furthermore, only tasks with higher priority than j can be scheduled during this intervals. Let $|I_1|$ and $|I_2|$ denote the lengths of I_1 and I_2 , respectively. Then, we must have $|I_1| + |I_2| \leq \sum_{a_i + c_i < a_j + c_j} (a_i + c_i)/m$. Thus, we can bound the completion time C_j as follows:

$$\begin{aligned} C_j &\leq r_j + |I_1| + a_j + b_j + |I_2| + c_j \\ &\leq \left(\sum_{a_i + c_i < a_j + c_j} (a_i + c_i)/m \right) + (r_j + a_j + b_j + c_j) . \end{aligned}$$

Summing up all C_j together and using the bounds (7) and (5), we can show that $\sum_{j=1}^n C_j < 2C^*$. For the makespan of the schedule, let k be the job with the maximum completion time among all jobs. Let l be the last job among all jobs so that c_l is scheduled immediately after it is ready. Define C_{a_l} as before. Then, all machines must be busy during the intervals $I_1 = [r_l, C_{a_l} - a_l]$ and $I_2 = [(C_{a_l} + b_l), C_{\max} - c_k]$. The total length of the two intervals is $|I_1| + |I_2| \leq C_{\max}^*$. Therefore, the makespan is

$$C_{\max} = r_l + |I_1| + a_l + b_l + |I_2| + c_k = (r_l + a_l + b_l) + (|I_1| + |I_2|) + c_k < 3C_{\max}^* .$$

□

4. Distinct preprocessing and postprocessing masters

In this section, we study the model with distinct preprocessing masters and postprocessing masters. Different from the previous cases, here an a task can only be preempted by another a task and a c task can only be preempted by another c task. In all cases, we apply the following heuristic to obtain a preemptive schedule.

Heuristic 4. Schedule the available a tasks using the $SRPT_a$ rule on the preprocessors; Schedule the available c tasks using the SPT_c rule on the postprocessors.

Let m_1 and m_2 denote the numbers of preprocessing masters and postprocessing masters, respectively. We first study the simple case $m_1 = m_2 = 1$.

THEOREM 8 *Heuristic 4 generates a (3, 2) online preemptive schedule when $m_1 = m_2 = 1$ and jobs have the same or arbitrary release times.*

Proof : We first consider the total completion time. Let C_{a_j} be the time a_j finishes in the schedule obtained by Heuristic 4. The key observation is that our schedule has the minimum $\sum_{j=1}^n C_{a_j}$ among all possible schedules. Let $C_{a_j}^*$ be the time a_j finishes in an optimal schedule. Thus, we have

$$C^* \geq \sum_{j=1}^n (C_{a_j}^* + b_j + c_j) = \left(\sum_{j=1}^n C_{a_j}^* \right) + B + C > \left(\sum_{j=1}^n C_{a_j} \right) + B + C = \sum_{j=1}^n (C_{a_j} + b_j + c_j) .$$

Notice c_j can only be delayed by smaller c tasks. Thus, the total completion time of the schedule obtained by Heuristic 4 is at most

$$\sum_{j=1}^n \left(C_{a_j} + b_j + c_j + \sum_{c_i \text{ delays } c_j} c_i \right) \leq \sum_{j=1}^n (C_{a_j} + b_j + c_j) + \sum_{j=1}^n \sum_{c_i < c_j} c_i \leq 2C^* .$$

For the makespan, consider the last job l such that c_l runs immediately when it is available at time $C_{a_l} + b_l$. There is no idle time in the interval $I_1 = [r_l, C_{a_l}]$ and the interval $I_2 = [(C_{a_l} + b_l), C_{\max}]$. The length of each interval is at most C_{\max}^* . Therefore, the makespan is

$$C_{\max} = r_l + |I_1| + b_l + |I_2| \leq 3C_{\max}^* .$$

□

THEOREM 9 *When $m_1 > 1$ and $m_2 > 1$, Heuristic 4 generates a (4, 2) preemptive schedule with migration if the jobs have the same release times and a (4, 3) preemptive schedule with migration if the jobs have arbitrary release times.*

Proof : Using similar argument as in the previous theorem, and the fact that $SRPT_a$ rule minimize the total completion time of C_{a_j} when all jobs have the same release time and that it is a 2-approximation when the jobs have arbitrary release times in the multi-machine environment. □

5. Converting preemptive schedules into non-preemptive schedules

As we mentioned before, we obtain non-preemptive schedules by converting from preemptive schedules. Our approach is based on the technique that was introduced by Phillips et. al in [12], and improved by Chekuri et. al in [3]. The model studied in [12] consists of one or more identical machines and n simple jobs. Let S be a preemptive schedule. To obtain a non-preemptive schedule S' , they form a list of jobs in increasing order of their completion times in S and then list schedule the jobs in this list one by one, respecting their release times. They showed that if S is a β -approximation for total completion time, then S' is a 2β -approximation for total completion time in the single-machine environment, and a 3β -approximation for total completion time in the multi-machine environment. In addition, this conversion also yields an online non-preemptive algorithm if the preemptive schedule can be generated online.

Later, Chekuri et. al [3] improved the above results in the single machine case. The difference between the two approaches in [3] and [12] is how to obtain the list of jobs. Chekuri et. al designed a deterministic $O(n^2)$ time *offline* algorithm such that the schedule obtained has total completion time at most $\frac{e}{e-1}$ times that of the preemptive schedule, where e is the base of natural log. As well, they gave a randomized *online* algorithm with *expected* performance $\frac{e}{e-1}$. In the multi-machine case, Chakrabarti et al. [2] showed that the convert procedure given in [12] has a bound of $7/3$, instead of 3 times that of S . In both cases one can show that the makespan of the S' is not much larger than that of S .

In the following sections, we will describe how to convert preemptive schedules generated by Heuristics 1-4 to non-preemptive schedules in the master-slave model. The difficulty of our conversion is that we need to respect not only the release time of a_i , $1 \leq i \leq n$, but also respect the constraint that the interval between the finish time of a_i and the start time of c_i has length at least b_i . By our convention, we use S to denote a (α, β) preemptive schedule, C_{a_j} to denote the completion time of a_j in S , C_j to denote the completion time of j in S , and C_{\max} to be the makespan of S . Our goal is to convert S into a non-preemptive schedule S' . We define C'_{a_j} , C'_j , C'_{\max} similarly for the non-preemptive schedule S' .

5.1 Single master and multi-master systems

For a canonical schedule S on a single machine, all jobs have the same release time and preemption occurs only among the c tasks. We can think of the c tasks as having release time $\max(A, (C_{a_j} + b_j))$ with processing time c_j . Thus, we can do the conversion for the c tasks using the approach of [3], respecting the release time $\max(A, (C_{a_j} + b_j))$. The obtained schedule is a $(1 + \alpha, \frac{e}{e-1}\beta)$ -schedule.

If we take S to be the $(\frac{3}{2}, 2)$ canonical schedule in Corollary 2 in Theorem 3 and apply the above conversions, we have the following theorem.

THEOREM 10 *In $O(n^2)$ time, one can obtain a $(\frac{5}{2}, \frac{2e}{e-1})$ non-preemptive canonical schedule when there is a single master and all jobs have the same release time.*

To get a non-canonical schedule without preemption, we let S to be the preemptive $(2, 2)$ non-canonical schedule in Theorem 4. Then we obtain a λ -schedule S' , $\lambda = 1$. Using similar argument as [12], one can show that S' is a $(3, 4)$ -schedule. Furthermore, we can generate S' in an online fashion. Thus we have the following theorem.

THEOREM 11 *In $O(n \log n)$ time, one can obtain a $(3, 4)$ online non-preemptive schedule when there is a single master and $r_j \geq 0$ for all job j .*

The following theorem shows how to get offline non-preemptive schedules for multi-master systems.

THEOREM 12 *When there are multi-masters, one can obtain a $(5, 4)$ non-preemptive offline schedule.*

Proof : Let S be the (3,2)-schedule generated by Heuristic 3. Then S has no migration. We can obtain the non-preemptive schedule S' by converting the schedule on each machine separately in the same way as described in the proof of Theorem 11. Thus, the total completion time of S' is at most two times that of S . Since S is a 2-approximation for total completion time, S' is a 4-approximation for total completion time. For the makespan, $C'_j \leq C_j + \max_{i=1}^m \Delta_i$ where Δ_i is the total length of the a tasks and the c tasks assigned on machine i . We have shown in Section 1.3 that $\Delta_i \leq 2C_{\max}^*$. Thus, $C'_j \leq C_j + \max_{i=1}^m \Delta_i \leq C_j + 2C_{\max}^*$. Since S is 3-approximation for makespan, S' is a 5-approximation for the makespan. This concludes the proof. \square

5.2 Distinct preprocessors and postprocessors

THEOREM 13 *In $O(n^2)$ time, one can obtain a $(4, \frac{2e}{e-1})$ non-preemptive schedule when $r_j = 0$ for all j , and $m_1 = m_2 = 1$.*

Proof : Since all jobs have the same release time, we can assume there is no preemption on the single preprocessor. So we simply do conversion on the single postprocessor using the approach given in [3]. Let S be the (3, 2)-schedule generated by Heuristic 4. Then S' is a $(4, \frac{2e}{e-1})$ non-preemptive schedule. \square

When the release times are arbitrary, we need to do the conversion carefully so as to make sure that the difference between the finish time of a_j and the start time of c_j is at least b_j .

THEOREM 14 *When $m_1 = m_2 = 1$, and $r_j \geq 0$, one can obtain a $(4, 4 + \frac{2e}{e-1})$ non-preemptive offline schedule or an online non-preemptive schedule with expected performance $(4, 4 + \frac{2e}{e-1})$.*

Proof : We first remove the preemptions among the a tasks to get a λ -schedule of a tasks, respecting the release times of the a tasks. Let C'_{a_j} be the new completion time of a_j . Then, we must have $C'_{a_j} \leq \frac{e}{e-1} C_{a_j}$, by [3]. Next, we remove the preemptions among the c tasks to get a λ -schedule of the c tasks, where $\lambda = 1$, and make sure the interval between C'_{a_j} and the start time of c_j is at least b_j for each job j .

Let C_j be the completion time in the preemptive schedule S . Suppose the jobs are indexed such that $C_i < C_{i+1}$. Then, $C_j \geq \max_{i=1}^j r_i$, $C_j \geq \max_{i=1}^j b_i$, $C_j > \sum_{i=1}^j a_i$ and $C_j > \sum_{i=1}^j c_i$.

Let C'_j be the new completion time of job j in S' . Then we have

$$C'_j \leq C'_{a_j} + \max_{1 \leq i \leq j} b_i + \sum_{i=1}^j c_i \leq (2 + \frac{e}{e-1}) C_j .$$

For the makespan, we have

$$C'_j \leq \max_{1 \leq i \leq j} r_i + \sum_{C_{a_i} \leq C_{a_j}} a_i + \max_{1 \leq i \leq j} b_i + \sum_{i=1}^j c_i \leq 4 C_{\max}^* .$$

Let S be the (3,2)-schedule generated by Heuristic 4. Then the theorem follows. \square

THEOREM 15 *In $O(n \log n)$ time, one can obtain a $(4, 14/3)$ non-preemptive schedule when $r_j = 0$ for all j , $m_1 \geq 1$ and $m_2 \geq 1$.*

Proof : Since all jobs have the same release time, we can assume there is no preemption on the preprocessors. So we do conversion on the postprocessors using the approach given in [2]. Let S

be the (4, 2)-schedule generated by Heuristic 4. Then the total completion time of S' is at most $\frac{7}{3}$ times that of S , i.e., S' is a 14/3-approximation for total completion time.

We now consider the makespan. Suppose the jobs are indexed such that $C_i \leq C_{i+1}$ in S . By the heuristic, for any $1 \leq i \leq n$, $C_{a_i} + b_i \leq \sum_{k \neq i} a_k/m_1 + (a_i + b_i) \leq 2C_{max}^*$ and $\sum_{i=1}^j c_i/m_2 \leq C_{max}^*$. In S' , the latest time the postprocessors become busy is $\max_{i \leq j} (C_{a_i} + b_i)$. Therefore, we have $C'_j \leq \max_{i \leq j} (C_{a_i} + b_i) + \sum_{i=1}^{j-1} c_i/m_2 + c_j \leq 4C_{max}^*$ \square

THEOREM 16 *In $O(n \log n)$ time, one can obtain a (5, 13) non-preemptive online schedule when $m_1 \geq 1$ and $m_2 \geq 1$.*

Proof : We first remove preemptions among the a tasks to get a λ -schedule of the a tasks, respecting the release times of the a tasks. Let C'_{a_j} be the new completion time of a_j . Then, we must have $C'_{a_j} \leq (\frac{7}{3})C_{a_j}$, by the result of [2]. Next, we remove preemptions among the c tasks to get a λ -schedule of the c tasks, where $\lambda = 1$, and make sure that the interval between C'_{a_j} and the start time of c_j is at least b_j for each job j .

Let C_j be the completion time in the preemptive schedule. Suppose the jobs are indexed such that $C_i < C_{i+1}$ in S . Then, $C_j \geq \max_{1 \leq i \leq j} (C_{a_i} + b_i + c_i)$ and $C_j > \sum_{i=1}^j c_i/m_2$. Let C'_j be the new completion time of job j . Then we have

$$C'_j \leq \max_{1 \leq i \leq j} (C'_{a_i} + b_i) + \sum_{1 \leq i < j} c_i/m_2 + c_j \leq \max_{1 \leq i \leq j} (\frac{7}{3} C_{a_i} + b_i) + \sum_{1 \leq i < j} c_i/m_2 + c_j \leq \frac{13}{3} C_j .$$

For the makespan, we have $C_{max}^* \geq A/m_1$, $C_{max}^* \geq C/m_2$ and $C_{max}^* \geq a_j + b_j + c_j$. Therefore, we have

$$\begin{aligned} C'_j &\leq \max_{1 \leq k \leq n} r_k + \left(\sum_{C_{a_k} < C_{a_j}} a_k/m_1 \right) + a_j + \max_{C_i < C_j} b_j + \left(\sum_{1 \leq i < j} c_i/m_2 \right) + c_j \\ &\leq 5 C_{max}^* \end{aligned}$$

Thus, if S is a (4, 3)-schedule generated by Heuristic 4, then S' is a (5, 13)-schedule. \square

6. Conclusion

In this paper we have considered the problem of minimizing total completion time in the master-slave model in various settings. We designed efficient algorithms to generate preemptive schedules with good performance ratio. In most cases, these schedules have small makespan as well. Then we convert the preemptive schedules into non-preemptive schedules using the techniques developed in [12], [2] and [3].

Several problems remain open. We are not able to find good canonical schedules in multi-master systems. The difficulty is in obtaining a good lower bound of the optimal schedule in this case. In some cases, the non-preemptive schedules obtained by the conversion procedure give us a rather large performance ratio. It is worthwhile to consider using different techniques such as linear programming relaxation. Most of the performance bounds are quite loose. For future research, it is worthwhile to tighten the performance bounds.

References

- [1] R.E. Buten and V.Y. Shen. A scheduling model for computer systems with two classes of processors. Sagamore Computer Conference on Parallel Processing, pp. 130-138, 1973.

- [2] S. Chakrabarti, C. Phillips, A. Schulz, D.B. Shmoys, C. Stein and J. Wein. Improved scheduling algorithms for minsum criteria. In *Proc. 23rd ICALP*, pp. 646-657, 1996
- [3] C. Chekuri, R. Motwani, B. Natarajan and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31(1), pp. 146-166, 2001.
- [4] M. Dell'Amico. Shop problems with two machine and time lags. *Operations Research*, 44(5), pp. 777-787, 1996.
- [5] J. Du and J.Y-T. Leung. Minimizing mean flow time in two-machine open shops and flow shops. *Journal of Algorithms*, 14:24-44, 1993.
- [6] M.X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proc. 8th SODA*, pp. 591-598, 1997.
- [7] J.N.D. Gupta. Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, 38, pp. 359-364, 1988.
- [8] W. Kern and W. Nawijn. Scheduling multi-operation jobs with time lags on a single machine. University of Twente, 1993.
- [9] M.A. Langston. Interstage transportation planning in the deterministic flow-shop environment. *Operations Research*, 35(4), pp. 556-564, 1987.
- [10] C.-Y. Lee and G.L. Vairaktarakis. Minimizing makespan in hybrid flowshops. *Operations Research Letters*, 16, pp. 149-158, 1994.
- [11] J.Y-T. Leung and H. Zhao. Minimizing mean flowtime on master-slave machines. In *Proc. 2004 PDPTA*, Vol. II, pp. 939-945, 2004.
- [12] C. Phillips, C. Stein and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199-223, 1998.
- [13] S. Sahni. Scheduling master-slave multiprocessor systems. *IEEE Trans. on Computers*, 45(10), pp. 1195-1199, 1996.
- [14] S. Sahni and G. Vairaktarakis. The master-slave paradigm in parallel computer and industrial settings. *Journal of Global Optimization*, 9, pp. 357-377, 1996.
- [15] S. Sahni and G. Vairaktarakis. The master-slave scheduling model. In J. Y-T. Leung (Ed): *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, Boca Raton, FL, 2004.
- [16] C. Sriskandarajah and S.P. Sethi. Scheduling algorithms for flexible flowshops: worst and average case performance. *European Journal of Operational Research*, 43, pp. 143-160, 1989.
- [17] W. Szwarc. Flow-shop problems with time lags. *Management Science*, 29, pp. 447-491, 1983.
- [18] G. Vairaktarakis. Analysis of algorithms for master-slave system. *IIE Transactions*, 29(11), pp. 939-949, 1997.
- [19] W. Yu, H. Hoogeveen and J.K. Lenstra. Minimizing makespan in a two-machine flowshop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5), 333 - 348, 2004.