The Master-Slave Paradigm in Parallel Computer and Industrial Settings*

Sartaj Sahni† George Vairaktarakis‡

Abstract

The master-slave paradigm finds important applications in parallel computer scheduling, semiconductor testing, machine scheduling, transportation, maintenance management and other industrial settings. In the master-slave model considered in this paper a set of jobs is to be processed by a system of processors. Each job consists of a preprocessing task, a slave task and a postprocessing task that must be executed in this order. The pre- and post-processing tasks are to be processed by a master processor while the slave task is processed by a slave processor. In this paper, we motivate the master-slave model and develop bounded performance approximation algorithms for the unconstrained makespan minimization problem as well as for multiple master systems.

*This work was supported in part by the National Science Foundation under grant MIP-9103379 and the Army Research Office under grant DAA H04-95-1-0111.

†Department of Computer and Information Sciences, University of Florida, Gainesville, FL 32611.

‡College of Business Administration, Management Department, Marquette University, Milwaukee, WI 53233.

1 Introduction

The master-slave paradigm involves two sets of processors. The master processors that are responsible for pre- and post-processing of work orders, and the slave processors that are responsible for the actual execution of the orders. The number of slave processors is no less than the number of work orders. Applications of this paradigm include parallel computing, semiconductor testing and problems in transportation as will be described shortly.

First we give a brief description of the model under consideration. A set of jobs is to be processed by a system of master and slave processors. Each job has three tasks associated with it. The first is a preprocessing task, the second is a slave task, and the third a postprocessing task. The tasks of each job are to be performed in the order: preprocessing, slave, postprocessing. Let a_i, b_i, and c_i, respectively, denote the preprocessing, slave, and postprocessing tasks (and task times) of job i. All task times are assumed to be greater than zero (i.e., $a_i > 0$, $b_i > 0$, and $c_i > 0$, for all i). The available processors are divided into two categories: master and slave. If n denotes the number of jobs, then no schedule can use more than n slaves. Hence we may assume that there are exactly n slaves. The *makespan* or *finish time* of a schedule is the earliest time at which all tasks have been completed. The case where there is a single master processor has been considered in [19]. In this paper we consider the problem of minimizing makespan in a system that consists of several master processors; we shall refer to this generalization as *multiple master systems*.

Several applications of the master-slave model are found in parallel computer scheduling. A common parallel programming paradigm involves the use of a single main computational thread that employs the fork and join operations to spawn parallel tasks/threads and then to synchronize following the completion of these tasks. The fork operation involves the passing of varying amounts of data to remote processors that will execute the spawned threads (we assume that each spawned thread will be executed on a different processor). These processors will, in turn, return the results to the main thread. So, associated which each of the spawned threads, we have three amounts of work:

1. Preprocessing by main thread. This is the work needed to initiate the thread. It includes the effort expended in collecting the data needed by the remote processor (in case of a distributed memory environment); overheads involved in transmitting this data to the remote processor, etc.
2. Work done in the thread. This includes the computational activity assigned to the remote processor, the work this processor must do to receive the data and send back the results, and the transmission times in receiving and sending.
3. Post-processing by the main thread. This represents the effort expended in receiving the answers and performing any postprocessing on them.

Since the different threads may execute very different pieces of code, the relative values of the amounts of work involved in preprocessing, in thread execution, and in postprocessing can vary widely from thread to thread.

The fork-join paradigm can be used to model, for example, one of the modes of operation of the nCube hypercube computer. In this, the main program thread runs on the host computer which serves as the master processor. This program initiates parallel tasks on the hypercube processors when the host computation reaches a point where parallelism can be exploited. For each parallel task, the host needs to gather the data needed by the task and also consume the results from the tasks when they are complete. These correspond to pre- and post-processing activities. The tasks themselves run on the hypercube processors and correspond to slave activities. The number of parallel tasks created is generally equal to the number of available hypercube processors.

For example, if the master processor reaches a point in its computation when two matrices A and B are to be multiplied, then it would partition the matrix multiplication problem into p (p is the number of slave processors) multiplication problems each involving a submatrix of A and B. These submatrix pairs together with the multilpication code would be transmitted to the p slaves (one pair per slave); the slaves would execute the code once they have received the data and code; the slaves would transmit the product submatrix back to the master; and finally the master would store the received submatrix of C into the proper locations in C. Since matrix multiplication is a highly structured problem, it is possible to partition the matrices so that the amount of preprocessing work for each slave task is

the same, the amount of postprocessing work is the same for each task, the amount of work done by each slave is the same (this assumes uniform data transmission times between the master and slaves). When the submatrices are square, the task preprocessing time is roughly twice the postprocessing time.

As another parallel computing example, suppose we are working with a computer vision or VLSI CAD problem that involves objects in a two-dimensional region. To process these objects, the region may be divided into p parts; each part is sent to a slave processor; the results are returned to the master. Because of the nonuniform distribution of objects and an often imposed requirement that the region be partitioned using regular geometries (for example, we may require a rectangular region be partitioned using either vertical or horizontal cut lines so that the pre- and post-processing tasks are simplified), the number of objects in each partition may vary widely. As a result, the amount of preprocessing work varies widely from task to task, and so also does the amount of work assigned to individual slaves as well as the postprocessing work (which may now also invlove worrying about partition boundary effects).

Certain semiconductor testing operations also utilize the master-slave paradigm. In the case of burn-in operations, chips are subject to thermal stress for an extended period of time in order to bring out latent defects leading to infant mortality that might otherwise surface in the operating environment. The thermal stressing is accomplished by maintaining the oven at a constant temperature while powering up the chip. The burn-in times for each chip are specified by the customer for whom it is made and it is thus fixed apriori. After the initial burn-in operation each chip cools off for a specified amount of time that depends on the length and intensity of the initial burn-in period. After cooling, each chip is subject to a final burn-in operation" (see [13] for a more detailed description of semiconductor burn-in operations). In this application the burn-in oven corresponds to the master processor, the two burn-in tasks correspond to pre- and post-processing and the cooling period corresponds to the slave task. Since the burn-in operations are near the end of the production process, scheduling is critical in determining on-time delivery and output performance for the entire company.

Industrial applications of the master-slave paradigm include the case of consol-

idators that receive orders to manufacture quantities of various items. The actual manufacturing is done by a collection of slave agencies. The consolidator needs to assemble the raw material (from his/her inventory) needed for each task, load the trucks that will deliver this material to the slave processors, and perform an inspection before the consignment leaves. All of these are part of the task preprocessing done by the master processor (i.e., the consolidator). The slave processors need to wait for the arrival of the raw material, inspect the received goods, perform the manufacture, load the goods on to the trucks for delivery, perform an inspection as the trucks are leaving. These activities together with the delay involved in getting the trucks to their destination (i.e., the consolidator) represent the slave work. When the finished goods arrive at the consolidator, they are inspected and inventoried. This represents the postprocessing.

In certain maintenance/repair environments, the maintenance manager examines the maintenance tasks to be performed and writes up a formal work order for each and prepares the task for maintenance; the work orders are executed by different maintenance crews that are dispatched following the receipt of the work order; upon completion, the maintenance manager inspects the completed work and signs an acceptance document.

It is easy to see that the examples cited earlier for single master systems generalize to multiple master systems. For example, we may have a computational resource that is comprised of a large number of processors. This resource is shared by several host computers whose function is to obtain the data and code for each job (say from a disk) and to store the results on a disk or to print the results out. For each job, the actual computation is done on a single processor of the shared computational resource. Each job has a preprocessing task (gather the data and code needed), a postprocessing task (output the results), and a slave task (computation). Assuming that the total number of jobs is no more than the number of processors in the shared computational resource, the problem of scheduling the jobs can be modeled as a multiple master scheduling problem. In this application, it is required that for each job, the pre- and post-processing tasks be done by the same master. This is referred to as *restricted multiple master* scheduling.

If the consolidator example is generalized to include several consolidators, then

the resulting scheduling problem may be modeled as a restricted multiple master system. On the other hand if there is a single consolidator with multiple trucks and each truck has its own crew for loading, inspecting, etc., then the scheduling problem can be modeled as a multiple master system (each truck and crew define one master) in which the master that pre-processes job i (i.e., the truck that delivers the raw material for the job) need not be the same as the one that post-processes job i (i.e., the truck that brings back the finished goods corresponding to this job).

While the problem of scheduling multiprocessor computer systems has received considerable attention [3], [4], [10], [12], [14], [15], [18], [21], it appears that the master-slave model has not been studied prior to the work of Sahni [19]. It is interesting to note that the master-slave scheduling model may be regarded as a variant of the job shop (see [1], [2] for a definition of a job shop as well as for elementary terminology concerning scheduling) as described below:

1. the job shop has two classes of machines: master and slave
2. there is exactly one master machine and the number of slave machines equals the number of jobs
3. each job has three tasks to be done in order; the first and third on the master and the second on a slave

The two machine flowshop model with transfer lags (2FTL) is a close relative to the master-slave model. In this model the preprocessing task has to be processed by the upstream machine, followed by a waiting period known as transfer lag, followed by the postprocessing task at the downstream machine. Special cases of this model are among the first problems considered in scheduling theory; see [8], [16], [20]. In [7], the problem of finding minimum makespan schedules for 2FTL was shown to be strongly NP-hard. Further results on 2FTL may be found in [5]. The problem of scheduling single machines with time lags and two tasks per job is identical to the single-master master-slave model. Since the former problem is strongly NP-hard [9], the single master problem is also strongly NP-hard.

In [19], the problem of finding minimum makespan no-wait-in-process schedules is shown to be NP-hard for the case of a single master. This remains true even when the pre- and post-processing tasks are required to be done in the same order. When

the order in which the post-processing tasks is done is required to be reverse of the pre-processing order, the minimum makespan schedule can be found in $O(n \log n)$ time. Fast polynomial time algorithms to obtain minimum makespan schedules in which the pre- and post-processing orders are the same (or reverse) and a job may wait between the completion of one task and the start of the next are also developed in [19].

For no-wait scheduling, the single-master master-slave model and the coupled-task model of [17] are identical. Orman and Potts [17] show that many versions of this latter problem are strongly NP-hard. These results carry over to the no-wait master-slave model.

The outline of the rest of this paper is as follows. In Section 2 we define the problems to be considered and present some basic results. In Section 3, we develop fast approximate algorithms for problems on a single master processor. In Section 4, we consider the problem of obtaining minimum finish time schedules for multiple master systems. We conclude with future research directions in Section 5.

2 Notation and Basic Results

Figure 1 (a) shows a possible schedule for the case when $n = 2$, (a_1, b_1, c_1)= (2, 6, 1), and $(a_2, b_2, c_2) = (1, 2, 3)$. In this schedule, the preprocessing of job 1 is handled first by the master; all other tasks begin at the earliest possible time. M denotes the master processor and S_1 and S_2 denote the slaves. The finish time is 9. The schedule that results when the master pre-processes job 2 first and all other tasks begin at the earliest possible time is shown in Figure 1 (b). This has a finish time of 10.

Let us examine the schedules of Figure 1. Notice that in both schedules, once the processing of a job begins, the job is processed continuously until completion. Schedules with this property are said to have *no-wait-in-process*. In industrial applications, one may impose this requirement on a schedule. Another interesting feature of the schedules of Figure 1 is that in one the postprocessing is done in the reverse order of the preprocessing while in the other the pre- and post-processing orders are the same. In some settings, we may require that schedules satisfy one

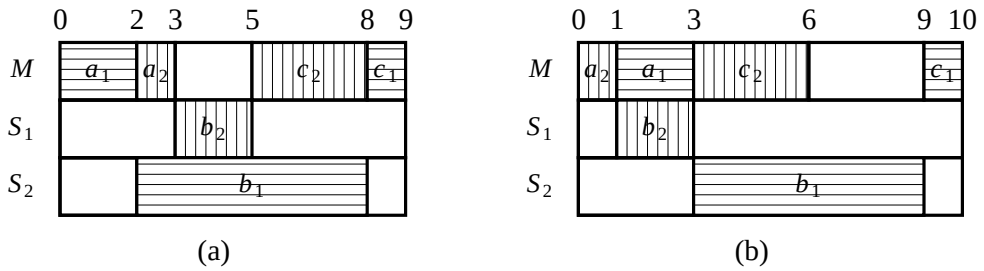

Figure 1: Example schedules

order or the other. For example, this could simplify the postprocessing if a stack is used, by the master, to maintain a record of jobs in process. Similarly, if the master uses a queue to maintain this information, we might require that the postprocessing be done in the same relative order as the preprocessing. Another discipline that might be imposed on the master is to complete all the preprocessing tasks before beginning the first postprocessing task. Both of the schedules of Figure 1 obey this discipline.

Similar requirements may be imposed in our consolidator example. This time suppose that all the raw material is loaded on a single truck and that the slaves are uniformly spaced. Whenever the truck stops, it has to wait at the slave location while the material for that location is unloaded and checked. This constitutes the preprocessing. When the truck returns to pick up the finished goods, it must again wait to load and check. This constitutes the postprocessing. If the truck route is circular, then the pre- and post-processing orders are the same. If the route is linear, then the postprocessing is done when the truck is returning to its point of origin and so is done in the reverse order of preprocessing. In both cases, all preprocessing tasks are done before the first postprocessing task.

For the case of a single master processor, Sahni [19] has considered order preserving sequencing (**OPS(1)**) and reverse order sequencing (**ROS(1)**). In the former case the pre- and post-processing tasks must be processed in the same order while in the latter these orders should be in reverse order. Optimal algorithms with complexity $\mathcal{O}(n \log n)$ have been developed for both of these cases. To facilitate later developments we provide a description of these algorithms denoted by **OOPS(1)** and **OROS(1)** respectively.

OOPS(1)

Step 1. Jobs with $c_j > a_j$ come first in nondecreasing order of $a_j + b_j$

Step 2. Jobs with $c_j = a_j$ come next in any order

Step 3. Jobs with $c_j < a_j$ come last in nonincreasing order of $b_j + c_j$

Step 4. Generate the order preserving schedule whose preprocessing tasks are ordered according to steps 1-3

OROS(1)

Step 1. Order the jobs according to nonincreasing order of b_j

Step 2. Generate the reverse order schedule whose preprocessing tasks are ordered according to step 1

The single master problem to minimize makespan with no restriction on the relative ordering of tasks of different jobs has not been considered before. We refer to this problem as *unconstrained minimum finish time* or **UMFT**. In light of the strong $\mathcal{NP}$-completeness of the UMFT problem, we develop an approximation algorithm in Section 3.

For master-slave systems with multiple master processors we can distinguish two classes of problems. In the first class we require both pre- and post-processing tasks to be processed by the same processor; we shall refer to such systems as *restricted multiple master systems*. In the second class we allow the pre- and post-processing task of each job to be processed by different processors; we shall refer to such systems as *unrestricted multiple master systems*. For instance, both modes of operation are applicable in semiconductor testing in the presence of multiple burn-in ovens.

For unrestricted multiple master systems we need to be careful about the definition of order-preserving and reverse-order schedules as the pre- and post-processing tasks of a job may be done by different master processors.

Definition 1 *For multiple master processor systems we shall say that a schedule is order preserving iff for every pair of jobs i and j such that the preprocessing of i begins before the preprocessing of j, the postprocessing of i completes before or at the same time as the postprocessing of j.*

Definition 2 *For multiple master processor systems we shall say that a schedule is a reverse order schedule iff for every pair of jobs i and j such that the preprocessing of i begins before the preprocessing of j, the postprocessing of i completes after or at the same time as the postprocessing of j.*

In Section 4 we will develop unconstrained, order preserving and reverse order schedules for both restricted and unrestricted multiple master systems.

3 Approximation Algorithms for Unconstrained MFT

In light of the complexity status of UMFT we are motivated to investigate heuristic algorithms that have good worst case performance. If S is an unconstrained schedule, then a straightforward interchange argument shows that we may rearrange the master tasks so that all preprocessing tasks complete before any postprocessing task starts. Such a rearrangement can be done without increasing the makespan of the schedule. Further, the rearranged schedule has no preemptions. We may shift the a tasks in the rearranged schedule left so as to start at time 0 and complete at time $\sum a_i$ and the b tasks may be shifted left so as to begin as soon as their corresponding a tasks complete. The c tasks may be ordered to begin in the same order as the b tasks complete. None of these rearrangement operations affects the makespan of S. With this as motivation, we define a *canonical schedule* to be one which satisfies the following properties:

1. There are no preemptions.
2. The a tasks begin on the master at time 0 and complete at time $\sum a_i$.
3. The b tasks begin as soon as their corresponding a tasks complete.
4. The c tasks are done in the same order as the b tasks complete and as soon as possible.

It is evident that for every unconstrained schedule S, there is a corresponding canonical schedule with better or the same makespan. So, in the remainder of this section we limit ourselves to canonical schedules. Note that a canonical schedule is

completely specified by giving the relative order in which the preprocessing tasks are done. As a result, such a schedule is defined by a permutation that gives the relative order in which the preprocessing tasks are done. We will use the terminology i follows (precedes) j to mean i comes after (before) j in the permutation that defines the schedule.

The next theorem finds the worst case performance of an arbitrary canonical schedule S. Let C^S be the makespan of the canonical schedule S and $C^\star$ the optimal makespan of UMFT.

Theorem 1 *For any canonical schedule S, $\frac{C^S}{C^\star} \leq 2$ and the bound is tight.*

Proof: If $C^S = \sum_i (a_i + c_i)$ then S is optimal and the error bound of 2 is valid. Else, $C^S > \sum_i (a_i + c_i)$ in which case there exists idle time on the master processor. Since S is canonical, this idle time will have to precede one or more postprocessing tasks. Let c_{i_0} be the last postprocessing task in S that starts immediately after its corresponding slave task b_{i_0}. Since there is idle time on the master, such an i_0 exists. Then,

$$C^S = \sum_{i \ precedes \ i_0} a_i + (a_{i_0} + b_{i_0} + c_{i_0}) + \sum_{i \ follows \ i_0} c_i \leq 2C^\star$$

since $a_{i_0} + b_{i_0} + c_{i_0} \leq C^\star$ and $\sum_i (a_i + c_i) \leq C^\star$.

To see that the error bound is tight consider an instance with $k+1$ jobs where k is an arbitrary positive integer. The first k jobs have processing requirements $(1, \epsilon, \epsilon)$ while the $(k+1)$-st job has requirements (ϵ, k, ϵ), $\epsilon < 1/k$. The schedule S that processes $a_{k+1} = \epsilon$ last among all preprocessing tasks has makespan $C^S = 2k + 2\epsilon$. The schedule $S^\star$ that processes a_{k+1} first among all preprocessing tasks has makespan $C^\star = k + (k+2)\epsilon$ and hence $\frac{C^S}{C^\star} \to 2$ as $\epsilon \to 0$. □

In what follows we present a heuristic whose error bound is $\frac{3}{2}$.

Heuristic H

Step 1. Let $S_1 = \{i : a_i \leq c_i\}$ and $S_2 = \{i : a_i > c_i\}$.

Step 2. Reorder the jobs in S_1 according to nondecreasing order of b_i

Step 3. Reorder the jobs in S_2 according to nonincreasing order of b_i

Step 4. Generate the canonical schedule in which the a tasks of S_1 precede those of S_2

The complexity of heuristic H is readily seen to be $O(n \log n)$. Let C^H be the makespan of the schedule generated by the above heuristic. Then,

Theorem 2 $\frac{C^H}{C^\star} \leq \frac{3}{2}$ *and the bound is tight.*

Proof: Let $S^\star$ be an optimal schedule for UMFT with makespan $C^\star$. Based on the processing requirements (a_i, b_i, c_i) of job i, we define an auxiliary problem P' with processing requirements (a'_i, b'_i, c'_i) defined as follows:

$$a'_i = \begin{cases} 0 & \text{if } a_i \leq c_i; \\ a_i & \text{otherwise} \end{cases}, \quad b'_i = b_i, \quad c'_i = \begin{cases} 0 & \text{if } c_i < a_i; \\ c_i & \text{otherwise} \end{cases}.$$

Note that P' isn't a legal instance of UMFT as it contains tasks whose processing requirement is zero. However, this doesn't affect the validity of our proof.

In P', all preprocessing tasks in S_1 are zero and hence they can precede all non-zero preprocessing tasks (i.e. the preprocessing tasks of S_2). Similarly, all post-processing tasks in S_2 are zero and hence they can follow all non-zero postprocessing tasks (i.e. the postprocessing tasks of S_1). Also, in P' every job has either $a'_i = 0$ or $c'_i = 0$.

A straightforward interchange argument shows that there exists an optimal schedule for P' where all postprocessing tasks for which $a'_i = 0$ are ordered in nondecreasing order of b_i. Similarly, all preprocessing tasks with $c'_i = 0$ are ordered in nonincreasing order of b_i. Therefore, an optimal sequence S' for P' looks like:

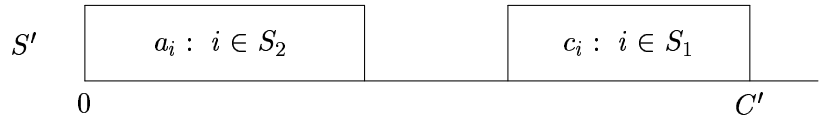

Figure 2: An optimal sequence for P'

Note that S' is the schedule generated by Step 4 of H if applied on P'. Let C' be the makespan of S'. By optimality of S' we have that $C' \leq C^\star$. From the schedule S' for P' we generate a schedule S^H for the original problem (where the processing requirements are (a_i, b_i, c_i)) by appending the tasks a_i; $i \in S_1$ in the

beginning of S' and the tasks c_i; $i \in S_2$ at the end of S'. Note that the resulting schedule S^H is feasible for the original data because S' is feasible for the modified data and $b' = b_i$. It is easy to check that S^H is the schedule generated by H for the input data (a_i, b_i, c_i) $i = 1, 2, \ldots, n$.

Let C^H be the makespan of S^H. Then, by construction

$$C^H = C' + \sum_{i \in S_1} a_i + \sum_{i \in S_2} c_i \leq C^\star + \frac{1}{2} \sum_{i \in S_1} (a_i + c_i) + \frac{1}{2} \sum_{i \in S_2} (a_i + c_i) =$$

$$= C^\star + \frac{1}{2} \sum_i (a_i + c_i) \leq \frac{3}{2} C^\star$$

since $\sum_i (a_i + c_i) \leq C^\star$.

To see that the bound of $\frac{3}{2}$ is tight consider an instance that consists of $k+1$ jobs where k is an arbitrary positive integer. The first k jobs have processing requirements $(1, \epsilon, 1)$ while the $(k+1)$-st job has requirements $(\epsilon, 2k, \epsilon)$. For this instance we have $S_2 = \emptyset$ and H produces the canonical schedule of Figure 3 a). In this the preprocessing tasks of jobs 1 through k are done first, in any order, and then that of job $k+1$ is done. The makespan is $3k + 2\epsilon$.

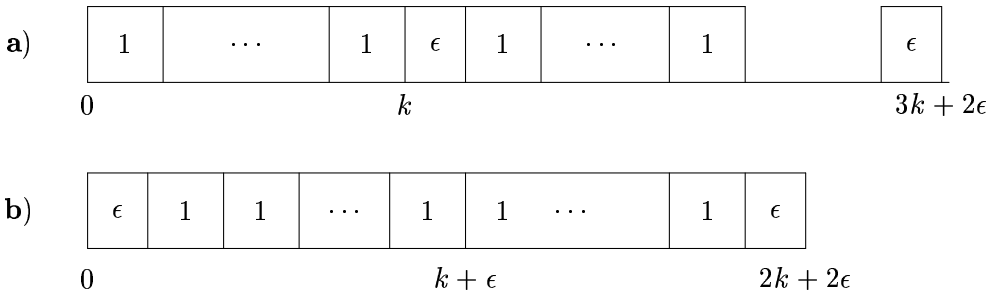

Figure 3: The bound of $\frac{3}{2}$ is tight

An optimal solution with makespan $2k + 2\epsilon$ is depicted in Figure 3 b) and hence

$$\frac{C^H}{C^\star} = \frac{3k + 2\epsilon}{2k + 2\epsilon} \to \frac{3}{2}$$

as $\epsilon \to 0$. This completes the proof of the theorem. □

4 Multiple Master Systems

A versatile heuristic, *general*, that obtains multimaster schedules with an error bound of at most 2 is developed in Section 4.1. For the case of reversed order

sequencing a heuristic with worst case error bound $2 - \frac{1}{m}$ (m is the number of master processors) is presented in Section 4.2.

4.1 A General Heuristic

The heuristic *general* may be used for both restricted and unrestricted systems as well as when constraints are placed between the orders in which the pre- and post-processing tasks are executed. Before presenting this heuristic, we define the *first available machine* (FAM) rule. In this, jobs are assigned to master processors one-at-a-time. Each job has a time t_i associated with it and the jobs are considered in a given order σ. When a job is considered, it is assigned to the master on which the sum of the times of already assigned jobs is the least (ties are broken arbitrarily).

Heuristic general(m)

Step 1. For each job, let $t_i = a_i + c_i$. Sort the jobs so that $t_1 \geq t_2 \geq \cdots \geq t_n$.

Step 2. Consider the jobs in this order and use the FAM rule to assign jobs to masters.

Step 3. On each master, schedule the preprocessing tasks in any order from time 0 to time T where T is the sum of the preprocessing tasks of the jobs assigned to this master. The slave tasks are scheduled to begin as soon as their corresponding preprocessing tasks are complete. The postprocessing tasks are scheduled to begin as soon after the completion of their slave tasks as is feasible.

The heuristic *general*(m) constructs schedules with the property that each job's pre- and post-processing tasks are done by the same master. Hence the schedules are feasible for both the restricted and unrestricted master models. The complexity of the heuristic is readily seen to be $O(n \log n)$.

Let $C^{general}$ be the makespan of the schedule generated by heuristic *general*. Let C^*_{UMFT} and C^*_{RMFT}, respectively, be the makespans of the optimal unrestricted and restricted master system schedules.

Theorem 3 *$C^{general}/C^*_{UMFT} \leq 2$ and $C^{general}/C^*_{RMFT} \leq 2$.*

Proof: Since $C^*_{UMFT} \leq C^*_{RMFT}$, it is sufficient to show that $C^{general}/C^*_{UMFT} \leq 2$. Assume that on the k'th master the last postprocessing task completes at time $C^{general}$. If there is no idle time on this master, then from step 2 it follows that

$$C^{general} \leq \frac{1}{m}\sum_{i=1}^{l-1}(a_i + c_i) + (a_l + c_l) \leq \frac{1}{m}\sum_{i=1}^{n}(a_i + c_i) + \frac{m-1}{m}(a_l + c_l)$$

where l is the last job assigned to master k by the FAM rule. Since, $C^*_{UMFT} \geq \frac{1}{m}\sum_{i=1}^{n}(a_i + c_i)$ and $C^*_{UMFT} \geq a_l + c_l$, we get

$$C^{general} \leq C^*_{UMFT} + \frac{m-1}{m}C^*_{UMFT} = (2 - \frac{1}{m})C^*_{UMFT}$$

or

$C^{general}/C^*_{UMFT} \leq 2 - \frac{1}{m}$

If the k'th master has idle time, then from step 3 it follows that there is a job q scheduled on this master such that the master is busy from time 0 to the start of b_q and again from the finish of b_q to time $C^{general}$. Let Q be the set of jobs assigned to this master in step 2.

$$C^{general} \leq \sum_{i \in Q}(a_i + c_i) + b_q = \sum_{i \in Q}(a_i + c_i) - (a_q + c_q) + (a_q + b_q + c_q)$$

From step 2, it follows that $\sum_{i \in Q}(a_i + c_i) \leq \frac{1}{m}\sum_{i=1}^{n}(a_i + c_i) + \frac{m-1}{m}(a_l + c_l)$ where l is the last job assigned to the master in step 2. Because of the ordering of step 1, $a_l + c_l \leq a_q + c_q$. Hence,

$$C^{general} \leq \frac{1}{m}\sum_{i=1}^{n}(a_i + c_i) + (a_q + b_q + c_q)$$

Each term on the right hand side of the above inequality is easily seen to be no more than C^*_{UMFT}. Hence, $C^{general} \leq 2C^*_{UMFT}$.

Combining the bounds for the two cases, we get $C^{general}/C^*_{UMFT} \leq 2$. □

To see that the bound of 2 is a tight one, consider the $n(m-1)+2$ job instance in which the first job's pre-, slave, and post-processing tasks are given by $(n-\epsilon, \epsilon, \epsilon/2)$, the next $n(m-1)$ job task times are $(1/2, \epsilon, 1/2)$ and the last job has times (ϵ, n, ϵ). Here, $0 < \epsilon < 1/2$. The jobs have been given in the order produced in step 1. The heuristic assigns jobs 1 and $n(m-1)+2$ to master 1. The remaining jobs are distributed evenly across the remaining masters. If in step 3, the first master is scheduled to process a_1 first, then $C^{general} = 2n + \epsilon$. However, $C^*_{UMFT} = C^*_{RMFT} = n + 2.5\epsilon$. The ratio approaches 2 as $\epsilon \to 0$.

Heuristic *general* may be used to obtain order preserving and reverse order schedules by modifying step 3 to produce such schedules. In fact, since optimal single master order preserving and reverse order schedules can be obtained in polynomial time ([19]), step 3 can generate optimal schedules using the jobs assigned to each master. Since the proof of Theorem 3 does not rely on how the schedule is constructed in step 3, the error bound of 2 applies even for the case of order preserving and reverse order schedules.

4.2 Restricted Reverse Order Schedules

In this subsection we develop an approximation algorithm for restricted multiple master systems in which each master processor is required to process its postprocessing tasks in an order that is the reverse of the order in which it processes its preprocessing tasks. This problem is abbreviated as $ROS(m)$ (reverse order scheduling with m masters). The $OROS(1)$ algorithm provided in Section 2 solves optimally the $ROS(1)$ problem.

The approximation algorithm, *Heuristic* $ROS(m)$, given below obtains schedules with an error bound no more than $2 - 1/m$.

Heuristic $ROS(m)$

Step 1. Sort the jobs so that $b_1 \geq b_2 \geq \cdots \geq b_n$.

Step 2. Consider the jobs in this order and use the FAM rule to assign jobs to masters using $t_i = a_i + c_i$.

Step 3. On each master, schedule the preprocessing tasks in the order the jobs were assigned to the master. Schedule the postprocessing tasks in the reverse order and to begin as soon as possible after all preprocessing tasks complete.

Note that in step 1, we obtain the ordering needed to construct an $OROS(1)$ for the n jobs and that in step 3 the jobs assigned to each master are scheduled to form an $OROS(1)$ for that master. The complexity of $ROS(m)$ is easily seen to be $O(n \log n)$. To establish the error bound, we need to first establish two other results. This is done in Lemmata 1 and 2. The error bound itself is established in Theorem 4.

Let I, I', and I'' be three sets of jobs. $I = \{(a_i, b_i, c_i) | 1 \leq i \leq n\}$, I' has n jobs defined by $a'_i = c'_i = (a_i + c_i)/2$ and $b'_i = b_i$, and I'' has n jobs defined by $a''_i = c''_i = (a_i + c_i)/(2m)$ and $b'_i = b_i$. Let $C^*_I(m)$, $C^*_{I'}(m)$, and $C^*_{I''}(m)$, respectively, denote the makespans of the $OROS(m)$ for I, I', and I''.

Lemma 1 *$C^*_I(m) = C^*_{I'}(m)$ for all m.*

Proof: Let the optimal schedules for I and I' be $S^*_I(m)$ and $S^*_{I'}(m)$, respectively. In $S^*_I(m)$, consider a job k with $a_k \neq c_k$. Let p be the master processor on which job k is scheduled in $S^*_I(m)$. If $a_k < c_k$, then increase the time for which the preprocessing of k is scheduled to $a'_k = (a_k + c_k)/2$ and reduce the time for which its postprocessing is scheduled to $c'_k = (a_k + c_k)/2$. This will require us to shift right by $a'_k - a_k$ all tasks of jobs whose preprocessing is scheduled after the preprocessing of job k on master p and also the slave and postprocessing tasks of job k. This transformation does not increase the schedule length. A similar transformation can be made when $a_k > c_k$. By applying this transformation to all jobs with $a_i \neq c_i$, we transform $S^*_I(m)$ into a feasible reverse order m master schedule for I' without increasing the schedule length. So, $C^*_{I'}(m) \leq C^*_I(m)$.

Using a reverse transformation, we can transform $S^*_{I'}(m)$ into a feasible reverse order schedule for I without increasing the schedule length. So, $C^*_I(m) \leq C^*_{I'}(m)$. Hence, $C^*_I(m) = C^*_{I'}(m)$. □

Lemma 2 *$C^*_{I''}(1) \leq C^*_I(m)$ for all m.*

Proof: From Lemma 1, it follows that it is sufficient to show that $C^*_{I''}(1) \leq C^*_{I'}(m)$. In $S^*_{I'}(m)$, we may assume that the preprocessing tasks on each master are scheduled continuously (i.e., with no idle time) from time zero to the time the last preprocessing task on that master completes (this may require us to shift some preprocessing tasks to the left). Also, we may assume that the postprocessing tasks are scheduled continuously from the start of the first postprocessing task on the master to time $C^*_{I'}(m)$ (this may require us to shift some postprocessing tasks to the right). Let F_i and B_i, respectively, denote the finish time of a'_i and the start time of c'_i in $S^*_{I'}(m)$. Since $a'_i = c'_i = \frac{1}{2}(a_i + c_i)$, it follows that $F_i = C^*_{I'}(m) - B_i$ for all i. Assume that the jobs are numbered so that $F_i \leq F_{i+1}$, $1 \leq i < n$. Hence, $B_i \geq B_{i+1}$, $1 \leq i < n$.

Since $F_i \leq F_{i+1}$, $1 \leq i < n$, $mF_i \geq \sum_{j=1}^{i} a'_j$ or $F_i \geq (\sum_{j=1}^{i} a'_j)/m$. So, $C^*_{I'}(m) - B_i \geq (\sum_{j=1}^{i} a'_j)/m = (\sum_{j=1}^{i} c'_j)/m$. Since, $C^*_{I'}(m) \geq F_i + b'_i + (C^*_{I'}(m) - B_i) \geq (\sum_{j=1}^{i}(a'_j + c'_j))/m + b'_i$, for all i, we get

$$C^*_{I'}(m) \geq \max_i\{(\sum_{j=1}^{i}(a'_j + c'_j))/m + b'_i\}$$

Now, consider the reverse order schedule $S_{I''}(1)$ obtained by scheduling the preprocessing tasks of I'' in the order $1, 2, \cdots, n$. In this schedule, we may assume that the preprocessing tasks are scheduled continuously and the slave and postprocessing tasks are scheduled as early as is feasible. Let $C_{I''}(1)$ be its makespan. Clearly, $C^*_{I''}(1) \leq C_{I''}(1)$. If there is no idle time on the master, then $C_{I''} = \sum_{j=1}^{n}(a''_j + c''_j)$ $= (\sum_{j=1}^{n}(a'_j + c'_j))/m < \sum_{j=1}^{n}(a'_j + c'_j)/m + b'_n \leq \max_i\{(\sum_{j=1}^{i}(a'_j + c'_j))/m + b'_i\}$ $\leq C^*_{I'}(m)$.

If there is idle time on the master, then there is a k such that $C_{I''} = \sum_{j=1}^{k}(a''_j + c''_j) + b''_k = (\sum_{j=1}^{k}(a'_j + c'_j))/m + b'_k \leq \max_i\{(\sum_{j=1}^{i}(a'_j + c'_j))/m + b'_i\} \leq C^*_{I'}(m)$.

So, $C^*_{I''}(1) \leq C_{I''}(1) \leq C^*_{I'}(m) = C^*_I(m)$. □

Effectively, Lemmata 1 and 2 show that the makespan of the schedule produced by $OROS(1)$ on I'' is a lower bound on the optimal makespan value for $ROS(m)$ which is denoted by $C^\star_I(m)$. The following theorem makes use of this result.

Theorem 4 *Let $C^{ROS}_I(m)$ be the makespan of the schedule generated by $ROS(m)$ on instance I. $C^{ROS}_I(m)/C^*_I(m) \leq 2 - 1/m$ and this bound is tight.*

Proof: Assume that the jobs are numbered so that $b_i \geq b_{i+1}$, $1 \leq i < n$. Using the transformations of Lemma 1, we can transform $S^{ROS}_I(m)$ into a schedule for I' that has the same makespan. Let this schedule be $S_{I'}(m)$. In this schedule, we may assume that the preprocessing tasks on each master are scheduled continuously from time zero to the time the last preprocessing task on that master completes and that the postprocessing tasks are scheduled continuously from the start of the first postprocessing task on the master to time $C^{ROS}_I(m)$. Let F_i and B_i, respectively, denote the finish time of a'_i and the start time of c'_i in $S^{ROS}_{I'}(m)$. From the application of the FAM rule in step 2 of $ROS(m)$, it follows that

$$F_i \leq \frac{1}{m}\sum_{j=1}^{i-1} a'_j + a'_i = \frac{1}{m}\sum_{j=1}^{i} a'_j + \frac{m-1}{m}a'_i$$

and

$$B_i \geq C_I^{ROS}(m) - [\frac{1}{m}\sum_{j=1}^{i} c'_j + \frac{m-1}{m}c'_i]$$

If there is no idle time on at least one of the masters, then let Q be the set of jobs processed by any master, say k, with no idle time. If l is the last job preprocessed on the k-th master, it follows that $C_I^{ROS}(m) = \sum_{i\in Q}(a'_i + c'_i) \leq \frac{1}{m}\sum_{i=1}^{n} a'_j + \frac{m-1}{m}a'_l + \frac{1}{m}\sum_{i=1}^{n} c'_j + \frac{m-1}{m}c'_l = \frac{1}{m}\sum_{i=1}^{n}(a'_j + c'_j) + \frac{m-1}{m}(a'_l + c'_l) \leq C^*_{I'}(m) + \frac{m-1}{m}C^*_{I'}(m) = C^*_I(m) + \frac{m-1}{m}C^*_I(m)$.

If all masters have idle time, then $C_I^{ROS}(m) = \max_i\{F_i + b'_i + C_I^{ROS}(m) - B_i\} \leq \max_i\{\frac{1}{m}\sum_{j=1}^{i}(a'_j + c'_j) + \frac{m-1}{m}(a'_i + c'_i) + b'_i\}$. Now consider the instance I''. In $OROS(1)$, the preprocessing tasks of I'' are scheduled in the order 1, 2, $\cdots$, n because $b_i \geq b_{i+1}$. As a result, $C^*_{I''}(1) \geq \max_i\{\frac{1}{m}\sum_{j=1}^{i}(a'_j + c'_j) + b_i\}$. From Lemma 2, we know that $C^*_I(m) \geq C^*_{I''}(1)$. So, $C_I^{ROS}(m) \leq \max_i\{\frac{1}{m}\sum_{j=1}^{i}(a'_j + c'_j) + \frac{m-1}{m}(a'_i + c'_i) + b'_i\} \leq C^*_{I''}(m) + \frac{m-1}{m}C^*_{I''}(m) \leq C^*_I(m) + \frac{m-1}{m}C^*_I(m)$.

Hence, $C_I^{ROS}(m)/C^*_I(m) \leq 2 - 1/m$.

To see that this error bound is tight, consider the m master instance I with $n = m(m-1)+1$ jobs. The first $m(m-1)$ of these jobs have $(a_i, b_i, c_i) = (1, \epsilon, \epsilon)$, where ϵ is a small number. The last job has $(a_n, b_n, c_n) = (m, \epsilon/2, \epsilon)$. The job numbering corresponds to that produced in step 1 of $ROS(m)$. In step 2, each master is assigned $m-1$ of the first $m(m-1)$ jobs and one of them gets job n in addition. The optimal schedule for this master has makespan $2m - 1 + \epsilon/2 + m\epsilon$. The $OROS(m)$ schedule assigns job n alone to one of the m masters and distributes the remaining jobs equally among the remaining $m-1$ masters. So, $C^*_I(m) = m + (m+1)\epsilon$. So, $C_I^{ROS}(m)/C^*_I(m) = (2m - 1 + \epsilon/2 + m\epsilon)/(m + (m+1)\epsilon)$ which tends to $2 - 1/m$ (from below) as $\epsilon \to 0$. □

Note that the above result has some similarities with the problem of minimizing makespan in a two-stage hybrid flowshop considered by Lee and Vairaktarakis [11]. In this problem preprocessing tasks are executed on the machines of stage 1, postprocessing tasks are executed on the machines of stage 2, and the slave tasks are null. A heuristic with bound $2 - \frac{1}{m}$ was developed for that problem as well.

5 Conclusion

In this paper we introduced various applications of the master-slave paradigm. We proposed fast bounded performance approximation algorithms to schedule both single master and multiple master systems. Our future research direction will be towards new variations of the master-slave model that are of practical relevance.

References

[1] K. Baker, *Introduction to Sequencing and Scheduling*, John Wiley, New York, 1974.

[2] E. Coffman, *Computer & Job/Shop Scheduling Theory*, John Wiley, New York, 1976.

[3] G. Chen and T. Lai, Preemptive scheduling of independent jobs on a hypercube, *Information Processing Letters*, 28, 201-206, 1988.

[4] G. Chen and T. Lai, Scheduling independent jobs on partitionable hypercubes, *Jr. of Parallel & Distributed Computing*, 12, 74-78, 1991.

[5] M. Dell'Amico, Shop problems with two machine and time lags, *Operation Research*, to appear.

[6] M. Garey and D. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Co., New York, 1979.

[7] R. Graham, E. Lawler, J. Lenstra, and A. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, *Annals of Discrete Mathematics*, 5, 287-326, 1979.

[8] S.M. Johnson, Discussion: Sequencing n jobs on two machines with arbitrary time lags, *Management Science*, 5, 299-303, 1959.

[9] W. Kern and W. Nawijn, Scheduling multi-operation jobs with time lags on a single machine, University of Twente, 1993.

[10] P. Krueger, T. Lai, and V. Dixit-Radiya, Job scheduling is more important than processor allocation for hypercube computers, *IEEE Trans. on Parallel & Distributed Systems*, 5, 5, 488-497, 1994.

[11] C.-Y. Lee and G.L. Vairaktarakis, Minimizing makespan in hybrid flowshops, *Operations Research Letters*, 16, 149-158, 1994.

[12] S. Leutenegger and M. Vernon, The performance of multiprogrammed multiprocessor scheduling policies, *Proc. 1990 ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, 226-236, 1990.

[13] C.-Y. Lee, R. Uzsoy and L.A.M. Vega, Efficient algorithms for scheduling semiconductor burn-in operations, *Operations Research*, 40, 4, 764-775, 1992.

[14] S. Majumdar, D. Eager, and R. Bunt, Scheduling in multiprogrammed parallel systems, *Proc. 1988 ACM SIGMETRICS*, 104-113, 1988.

[15] C. McCreary, A. Khan, J. Thompson, and M. McArdle, A comparison of heuristics for scheduling DAGS on multiprocessors, *8th International Parallel Processing Symposium*, 446-451, 1994.

[16] L.G. Mitten, Sequencing n jobs on two machines with arbitrary time lags, *Management Science*, 5, 293-298, 1959.

[17] A. Orman and C. Potts On the complexity of coupled-task scheduling, *Discrete Applied Mathematics*, To appear.

[18] S. Sahni, Scheduling multipipeline and multiprocessor computers, *IEEE Trans on Computers*, C-33, 7, 637-645, 1984.

[19] S. Sahni, Scheduling master-slave multiprocessor systems, *Proceedings, First International EURO-PAR Conference*, Lecture Notes In Computer Science, Vol. 966, Springer, 1995, pp 611-622.

[20] W. Szwarc, On some sequencing problems, *Naval Research Logistics Quarterly*, 15, 127-155, 1968.

[21] Y. Zhu and M. Ahuja, Preemptive job scheduling on a hypercube, *Proc. 1990 International Conference on Parallel Processing*, 301-304, 1990.