ONLINE SCHEDULING OF MALLEABLE PARALLEL JOBS

Richard A. Dutton and Weizhen Mao
Department of Computer Science
The College of William and Mary
P.O. Box 8795
Williamsburg, VA 23187-8795, USA
email: {radutt,wm}@cs.wm.edu

ABSTRACT
In this paper, we study a parallel job scheduling model which takes into account both computation time and the overhead from communication between processors. Assuming that a job J_j has a processing requirement p_j and is assigned to k_j processors for parallel execution, then the execution time will be modeled by $t_j = p_j/k_j + (k_j - 1) \cdot c$, where c is the constant overhead cost associated with each processor other than the master processor. In this model, $(k_j - 1) \cdot c$ represents the cost for communication and coordination among the processors. This model attempts to accurately portray the actual execution time for jobs running in parallel on multiple processors. Using this model, we will study the online algorithm Earliest Completion Time (**ECT**) and show a lower bound for the competitive ratio of ECT for $m \geq 2$ processors. For $m \leq 4$, we show the matching upper bound to complete the competitive analysis for $m = 2, 3, 4$. For large m, we conjecture that the ratio approaches $30/13 \approx 2.30769$.

KEY WORDS
parallel job scheduling, online algorithms, malleable jobs, competitive ratio, resource allocation

1 Introduction

Along with the power of parallelization on supercomputers and massively parallel processors (MPPs) comes the problem of deciding how to use this capability efficiently to utilize the available processing power. In this world of parallel computing, multiple processors are available for concurrently completing submitted jobs. Therefore, the task of job scheduling in a parallel environment becomes more difficult than in the traditional world of computers with a single processor. It must be determined how to schedule these jobs in a manner that most effectively makes use of the multiple available processors while taking into account the added overhead due to communication and coordination among the processors. In an ideal situation, the computation time for a job would scale linearly with the number of processors assigned to the job. This means that a job J_j with processing requirement p_j would execute in time p_j/k_j when assigned to k_j processors. However, there is an inherent cost of communication between the processors executing the job due to thread creation, data movement, and message passing for coordination.

In a system with m identical processors, the input is a sequence of n independent jobs $J_1, J_2, \ldots, J_n$, each of which must be scheduled on some or all of the m potential processors. Our goal is to schedule all n jobs in the order they are submitted to minimize the makespan of the entire schedule. We study the scheduling of a group of jobs that are *malleable*. Malleable jobs can run on any number of processors available. There are other groups of jobs such as *moldable* jobs, which can be executed on any number of processors within certain constraints, and *rigid* jobs [1], where the number of processors is specified as input. The way a program is written will determine the type of job it is. While it may be more difficult to write an application that can be executed on a varying number of processors, a scheduler is able to make better use of moldable or, in the best case, malleable jobs to create an efficient schedule of jobs [2].

The model chosen to represent the execution times of jobs can have an important effect on the competitive ratio of a scheduling algorithm. It is necessary to have a simple model that mirrors the performance of parallel jobs in reality [3]. The simple aspect is desirable, if not necessary, for theoretical analysis, and the realistic aspect is necessary to give validity to the model. To this end, we use the model

$$t_j = p_j/k_j + (k_j - 1) \cdot c$$

for the actual execution time t_j of a job J_j, given that J_j has processing requirement p_j and is assigned to k_j processors, and c is the constant representing communication overhead. This model accurately accounts for the communication and coordination costs present when a job is run in parallel on multiple processors. Due to the fact that we are judging the quality of the algorithm based on the makespan of its schedules, we are concerned with the processing requirement p_j, execution time t_j, and completion time C_j of jobs. Completion time is defined as

$$C_j = s_j + t_j,$$

where s_j is the start time of the job in the schedule. Makespan can be defined as the maximum completion time of any job J_j.

In this paper, we will study the performance of the Earliest Completion Time (ECT) algorithm. This is an online algorithm that requires malleable jobs with known or estimated processing requirements in order to determine its schedule. Because ECT is an online algorithm that schedules jobs in the order of input, we will investigate the competitive ratio of the algorithm with regards to the makespan of the schedule it produces. This technique effectively bounds the performance of the algorithm when compared to an optimal schedule. We will show that for any $m \geq 2$ processors the competitive ratio of ECT is at least $30/13 \approx 2.30769$. We will prove that this bound is tight for $m = 2, 3, 4$. A tight bound of $30/13$ means that the makespan of the online schedule produced by the ECT algorithm is never more than $30/13$ times that of the optimal (offline) schedule.

The rest of the paper is structured as follows. Section 2 formally defines our model for the parallel job scheduling problem and provides a detailed description of the ECT algorithm. Section 3 describes previous research related to the parallel job scheduling problem. Section 4 outlines the analysis, including theoretical bounds. Section 5 concludes the paper.

2 Problem Formulation

In this section, we formally define the parallel job scheduling problem we study. We consider a model for the execution time of jobs that accounts for both computation and communication among processors. This model $t_j = p_j/k_j + (k_j - 1)c$ shows the linear speedup from parallelizing the job on k_j processors in addition to the overhead costs from coordinating the job on the k_j processors. In the model, c is a constant overhead cost associated with each processor other than the processor where the job is originated. It should be noted that when a job is run on one processor, i.e. $k_j = 1$, the execution time is $t_j = p_j$ as would be expected from a job executing on a single processor.

We study a parallel system with m identical processors available. The n independent jobs $J_1, J_2, \ldots, J_n$ arrive and are scheduled in order. Each job J_j is malleable, i.e. can be executed on any subset of the m available processors, and has a known processing requirement p_j.

To evaluate the quality of the schedule produced, we study the makespan. The makespan can be defined mathematically as the maximum completion time among all jobs or

$$C = \max_j\{C_j\}.$$

As discussed in Section 1, the competitive ratio is the best measure of an online algorithm. For an arbitrary instance, we will denote the makespan of the ECT schedule as C and the makespan of the optimal schedule as C^*. Therefore, the competitive ratio r can be defined as the smallest possible value such that

$$C \leq r \cdot C^* + b,$$

where b is an additive term that may be a function of m. In the case when $b = 0$, the competitive ratio becomes $\max\{\frac{C}{C^*}\}$ for all instances.

2.1 ECT Algorithm Description

An online algorithm for our problem will schedule jobs in the order of arrival without any knowledge of future jobs. It is assumed that all jobs are available at time 0 but are still scheduled in order in an online manner. We study the Earliest Completion Time (ECT) algorithm.

The ECT algorithm will use the previously defined model of job execution time to determine the number of processors $1 \leq k_j \leq m$ with $k_j \in \mathbb{N}$. Once k_j is known, the algorithm will identify the start time s_j to be the earliest time when k_j processors are available (idle) to execute the job.

As suggested by the name, the ECT algorithm will choose k_j for a job J_j such that k_j processors will give the earliest time that J_j can possibly complete. This can be expressed as choosing k_j such that

$$C_j = \min_{1 \leq k \leq m}\{p_j/k + (k-1)c + s_j\}.$$

From the expression, we see that the minimum completion time is dependent on both the execution time $p_j/k + (k-1)c$ and the start time s_j of the job on k processors. The these two factors, execution time and start time, tell the algorithm the size of the job and the status of the system, respectively. The current status of the system, reflected through the time when k processors become available, manifests itself through the start time s_j of a job. A job J_j will start as soon as the necessary number of processors become available.

It should be obvious there may be a tradeoff between beginning the execution of a job quickly and waiting for more processors to become available. A job may complete more quickly if it uses fewer processors but begins executing immediately. The main idea of choosing a subset k_j of processors that minimizes the completion time is the most important aspect of the algorithm.

The practical details of the algorithm are not altogether specified. One way of finding the earliest completion time for a job would be to maintain a vector of the times when each processor is next available. For each $k_j = 1, 2, \ldots, m$ calculate the possible start time for the job on that number of machines, i.e. the k_j^{th} earliest available time is the possible start time on that number of processors. Then it is simple to compute the execution time using the model $t_j = p_j/k_j + (k_j - 1)c$. Adding together t_j and s_j, we arrive at the completion time C_j. After scheduling the job on the specific k_j processors, we can update the available time of those k_j processors to be C_j, or the time when job J_j will finish.

Since this is an online algorithm, the scheduling of the jobs will be done in order without knowledge of jobs

coming in the future. This means that the earliest completion time for a job J_j will be computed and the job will be scheduled before any consideration is given to job J_{j+1}.

3 Related Work

Scheduling is a prevalent aspect of most areas of computer science. Job scheduling, and more specifically parallel job scheduling, has been the target of more research as parallel machines have become more ubiquitous. Du and Leung first broke from the assumption of single processor execution to propose a system of parallel job scheduling, where tasks can be executed by more than one processor concurrently, also showing this new problem to be strongly NP-hard for certain variants [4]. Within this area, many variants of the problem arise. There are other algorithms and approaches for solving our parallel job scheduling problem, some of which are similar to our ECT approach and some of which are very different. One such similar approach is the Shortest Execution Time (SET) algorithm [5]. SET first computes the number of processors k_j which minimizes a job's execution time. It then determines the earliest time k_j processors will be available simultaneously and schedules the job for that time. SET was shown to have a competitive ratio of $4(m-1)/m$ for even $m \geq 2$ and $4m/(m-1)$ for odd $m \geq 3$. This bound approaches 4 for large m. A similar approach to the problem can also be found in [6, 7]. algorithm. Another approach, the offline variant, has also garnered attention from the research community. The offline approach leads to approximation algorithms with corresponding approximation ratios [8, 9, 10]. In the offline problem, all jobs to be scheduled are known prior to making the scheduling decisions. However, the online approach must schedule the jobs in order of arrival to the system without any knowledge of future jobs [11].

Other variations of the problem include job preemption [12] and precedence constraints among jobs [13]. Metrics such as sum of completion times, sum of response times, and average .stretch have also been studied to analyze the quality of scheduling algorithms [14, 15]. Feitelson et al. [2] and Sgall [16] provide surveys of both the theoretical and practical challenges involved in parallel job scheduling. In this paper, we focus on analysis of independent jobs without preemption by using the makespan as a metric.

4 Analysis and Results

We next study the competitive ratio of the ECT algorithm. We first present a lower bound on the competitive ratio for any $m \geq 2$. We then give a matching upper bound for $m \leq 4$ to complete the competitive analysis for $m = 2, 3, 4$.

4.1 Lower Bound for arbitrary $m \geq 2$

We show that for an arbitrary number of processors $m \geq 2$, the competitive ratio of the ECT algorithm is at least

$$\frac{2m((m-\lfloor m/3 \rfloor - 1) + (m-1))}{(m-\lfloor m/3 \rfloor)(m-\lfloor m/3 \rfloor - 1) + m(m-1)}$$

which approaches $30/13$ for large m.

Before moving to the lower bound, we first present the following two lemmas.

Lemma 1. *When choosing from m processors and all m processors are idle (available) at the same time, a job J_j with processing requirement $p_j = (m - \lfloor m/3 \rfloor)(m - \lfloor m/3 \rfloor - 1) \cdot c + \epsilon$, with arbitrarily small $\epsilon > 0$, will be assigned $k_j = m - \lfloor m/3 \rfloor$ processors in the ECT schedule.*

Lemma 2. *When choosing from m processors, $\lfloor m/3 \rfloor$ processors are idle (available) at time T and the remaining processors are available at time $T + \left(2((m-\lfloor m/3 \rfloor)-1)c + \frac{\epsilon}{m-\lfloor m/3 \rfloor}\right)$, a job J_j with processing requirement $p_j = m(m-1) \cdot c + \epsilon$, with arbitrarily small $\epsilon > 0$, will be assigned $k_j = m$ processors at time $T + 2((m-\lfloor m/3 \rfloor)-1)c + \frac{\epsilon}{m-\lfloor m/3 \rfloor}$ in the ECT schedule.*

Proof. The proofs for Lemmas 1 and 2 are omitted. □

With these lemmas, we can now proceed to the lower bound for the competitive ratio of the ECT algorithm.

Theorem 1. *The competitive ratio of ECT for $m \geq 2$ is at least*

$$\frac{2m((m-\lfloor m/3 \rfloor - 1) + (m-1))}{(m-\lfloor m/3 \rfloor)(m-\lfloor m/3 \rfloor - 1) + m(m-1)}.$$

Proof. Our goal is to construct an instance with a shorter job J_j for all odd j where $k_j \leq m$ and a longer job J_j for all even j where $k_j = m$. (Note: $k_j = m$ for odd jobs only for the case of $m = 2$. For other m, $k_j < m$ for odd jobs, meaning these jobs leave at least 1 processor idle.) For this instance, let there be $n = 2 \cdot m \cdot \alpha$ jobs, for some $\alpha > 0, \alpha \in \mathbb{Z}$. Let the processing requirement for odd jobs be

$$p_j = (m - \lfloor m/3 \rfloor)(m - \lfloor m/3 \rfloor - 1) \cdot c + \epsilon$$

and the processing requirement for even jobs be

$$p_j = (m)(m-1) \cdot c + \epsilon,$$

where $\epsilon > 0$ is arbitrarily small. From Lemma 1, we know that all odd jobs will be assigned to $k_j = m - \lfloor m/3 \rfloor$ processors in the ECT schedule. From Lemma 2, we get that all even jobs will be assigned to $k_j = m$ processors in the ECT algorithm.

Using the execution time model $t_j = p_j/k_j + (k_j - 1)c$, we find that the odd jobs have execution time

$$t_j \quad = \quad 2((m-\lfloor m/3 \rfloor)-1)c + \frac{\epsilon}{m-\lfloor m/3 \rfloor}.$$

The even jobs have execution time

$$t_j \quad = \quad 2(m-1)c + \frac{\epsilon}{m}.$$

Now that the execution times and processor partitions have been established for each job, the next step is determining the makespan of the ECT schedule. Since no two jobs are being run concurrently, the completion time of the last job will simply be the sum of the execution times of all jobs scheduled. Therefore, the makespan C will be

$$\begin{aligned} C &= \sum_{\text{odd } j} t_j + \sum_{\text{even } j} t_j \\ &= \frac{n}{2}\Big(2((m - \lfloor \frac{m}{3} \rfloor) - 1)c + 2(m-1)c + \\ &\quad \frac{\epsilon}{m} + \frac{\epsilon}{m - \lfloor m/3 \rfloor}\Big). \end{aligned}$$

In the optimal schedule, all jobs will be assigned to a single processor with no idle time with all odd jobs scheduled first followed by all even jobs. Each job will have $t_j = p_j$ in the optimal schedule. The resulting makespan C^* is

$$\begin{aligned} C^* &= \frac{1}{m} \cdot \sum_{\text{odd } j} t_j + \frac{1}{m} \cdot \sum_{\text{even } j} t_j \\ &= \frac{1}{m} \cdot \frac{n}{2}\Big((m - \lfloor \frac{m}{3} \rfloor)(m - \lfloor \frac{m}{3} \rfloor - 1) \cdot c \\ &\quad + m(m-1)c + 2\epsilon\Big). \end{aligned}$$

Given this instance with C and C^*, the competitive ratio of ECT is at least

$$\frac{C}{C^*} \quad \rightarrow \quad \frac{2m((m - \lfloor m/3 \rfloor - 1) + (m-1))}{(m - \lfloor m/3 \rfloor)(m - \lfloor m/3 \rfloor - 1) + m(m-1)}$$

as $\epsilon \rightarrow 0$.

□

Theorem 1 provides the lower bound for any $m \geq 2$. Given the $\lfloor m/3 \rfloor$ term in the lower bound, it is apparent that there are 3 distinct cases to consider when m is a multiple of 3, $m \bmod 3 = 1$, and $m \bmod 3 = 2$. multiple of 3 . Upon inspection of these cases, we find three different monotonically increasing functions that approach the same bound for large m. More specifically, we see that as $m \rightarrow +\infty$, $C/C^* \rightarrow 30/13$. The cases match the bounds previously proved by Havill and Mao [7] for the cases of $m = 2, 3$.

4.2 Upper Bound

We next present the matching upper bound proofs to complete the competitive analysis for $m = 2, 3, 4$. Havill and Mao [7] previously proved tight bounds for the cases $m = 2, 3$ for ECT with a different method from ours. We adapted the proof construction and have now proved a tight bound for the $m = 4$ case. Due to the length and complexity of the $m = 4$ proof, we show the entire $m = 3$ case with the same proof method as the $m = 4$ proof and then present an outline for $m = 4$. The lower bounds of 2, $\frac{9}{4}$, and $\frac{20}{9}$ for $m = 2, 3,$ and 4, respectively, can be calculated from the bound shown in Theorem 1.

We turn to a *block method* where the ECT schedule will be partitioned into blocks according to job start times and analyzed accordingly. We define a block B_i as the maximum time interval $[b_i, b_{i+1})$ during which no new job begins executing. Let the work of a job J_j be the total amount of time that all assigned processors are dedicated to J_j, or $w_j = k_j \cdot t_j$. W_i is the work done in block B_i and I_i is the idle area from B_i. Let W and I be the total work and idle area in the ECT schedule, respectively. To prove an upper bound of X_m , we wish to show

$$I \leq \frac{X_m - 2}{2} W + \frac{X_m}{2} \sum (p_j - k_j(k_j - 1)c) + mb \quad (1)$$

for some b that is independent of the input but may be related to m. Or alternatively

$$\begin{aligned} \sum_{r=i}^{q} I_r \quad \leq \quad & \sum_{r=i}^{q} \Bigg(\frac{X_m - 2}{2} W_r + \\ & \frac{X_m}{2} \sum_{j \in F_r} \alpha_{jr}(p_j - k_j(k_j - 1)c) + mb \Bigg) \quad (2) \end{aligned}$$

for each block sequence $B_i, \ldots, B_q$, starting from B_1.

The objective is to show the inequality in (1) holds for the entire schedule. We will prove this by showing inequality (2) must hold for each block or group of consecutive blocks, thereby implying that inequality (1) must hold. Equation (2) shows the variation of the inequality when considering a group of blocks $[i, q]$ where $i \leq q$. From (2), F_r is defined as the set of jobs executing at any point during the time interval of block B_r. α_{jr} defines the fraction of the job's processing that occurs during block B_r.

As defined above, the work of a job J_j is $w_j = k_j \cdot t_j$. This can also be written as $w_j = p_j + k_j(k_j - 1) \cdot c$. The term W denotes the total work of all jobs scheduled. The idle area I is defined as all area of a schedule where processors are not dedicated to executing jobs. It is also helpful to know that $(1/m) \sum p_j$ is a lower bound to C^*. The following derivation shows that proving the inequality in (1) proves the desired upper bound, where X_m is the competitive ratio we are trying to prove for m:

$$\begin{aligned} C &= \frac{1}{m}(W + I) \\ &\leq \frac{X_m}{2m} W + \frac{X_m}{2m} \sum (p_j - k_j(k_j - 1)c) + b \\ &\leq X_m \cdot \frac{1}{m} \sum p_j + b \\ &\leq X_m \cdot C^* + b \end{aligned}$$

By substituting the competitive ratio of $X_m = 9/4$

for $m = 3$, we arrive at the inequality

$$I \leq \frac{1}{8}W + \frac{9}{8}\sum_{j\in J}(p_j - k_j(k_j-1)c) + 3b. \qquad (3)$$

Theorem 2. *The competitive ratio of the ECT algorithm for $m = 3$ is $9/4$.*

Proof. The lower bound of $9/4$ for $m = 3$ was shown in Theorem 1. Now we must prove the matching upper bound to complete the competitive ratio. We wish to prove the inequality from (3) for an entire schedule. More specifically, we wish to show a version of the inequality similar to (2) holds for each block or group of consecutive blocks. Let us consider a block B_i starting from B_1. Because of the definition of the partitioning scheme for the blocks, we encounter 3 cases to consider.

Case 1: There is no idle time in B_i.
Because there is no idle time in B_i, $I_i = 0$ and inequality (3) holds true.

Case 2: Exactly one processor is idle in B_i.

Subcase 2a: Block B_{i+1} contains a job J_l on 3 processors (a **3-job**). Because this group of blocks shares properties with Case 3, we combine their proofs below.

Subcase 2b: Block B_{i+1} contains a job J_h on 2 processors (a **2-job**) and block B_{i+2} has a job J_l on 3 processors. We need to show

$$\begin{aligned}\sum_{r=i}^{i+2} I_r &\leq \frac{1}{8}W_i + \frac{9}{8}\sum_{j\in F_i}\alpha_{ji}(p_j - k_j(k_j-1)c) \\ &\quad + \frac{5}{4}p_h + \frac{5}{4}p_l - 8c.\end{aligned}$$

In this inequality, the work terms W_{i+1} and W_{i+2} have been incorporated into the last three terms of the inequality. Assume the length of the time interval of B_i is z and the length of the (partial) job finishing before the end of B_i is z_1. Since J_h is a 2-job and J_l is a 3-job, we know that $2c \leq p_h \leq 6c$ and $p_l > 6c$. From the rules of ECT, we also know

$$z - z_1 < \frac{2}{3}p_l - \frac{1}{2}p_h - 3c.$$

Therefore,

$$\begin{aligned}\sum_{r=i}^{i+2} I_r &= (z - z_1) + (p_h/2 + c) \\ &= \frac{1}{8}W_i + \frac{3}{4}(z - z_1) - \frac{3}{8}z_1 + p_h/2 + c \\ &< \frac{1}{8}W_i + \frac{5}{4}p_h + \frac{5}{4}p_l - 8c \\ &\leq \frac{1}{8}W_i + \frac{9}{8}\sum_{j\in F_i}\alpha_{ji}(p_j - k_j(k_j-1)c) \\ &\quad + \frac{5}{4}p_h + \frac{5}{4}p_l - 8c\end{aligned}$$

Case 3: Exactly two processors are idle in B_i.
In this case, we know that block B_{i+1} contains only a job J_l on 3 processors.

Consider cases 2a and 3 together. Let us assume again that the time interval forB_i has length z. Let z_1 and z_2 be the length of (partial) jobs on the two processors with idle time during B_i where $z > z_1$, $z \geq z_2$, and $z_2 \geq z_1$. Because J_l is a 3-job, we know that $p_l > 6c$. We can also quantify the lengths of the idle periods in B_i in terms of p_l,

$$z - z_1 < \frac{2}{3}p_l - 2c \quad \text{and}$$

$$z - z_2 < \frac{p_l}{6} - 2.$$

We wish to prove the inequality

$$\begin{aligned}I_i + I_{i+1} &\leq \frac{1}{8}W_i + \frac{9}{8}\sum_{j\in F_i}\alpha_{ji}(p_j - k_j(k_j-1)c) \\ &\quad + \frac{5}{4}p_l - 6c.\end{aligned}$$

As with Case 2b, the equation has been simplified by incorporating the work term W_{i+1} with the terms on the second line of the inequality. We know that

$$\begin{aligned}I_i + I_{i+1} &= (z - z_2) + (z - z_1) \\ &= \frac{1}{8}W_i + \frac{9}{8}(z - z_2) - \frac{6}{8}(z - z_1) - \frac{3}{8}z_1 \\ &< \frac{1}{8}W_i + \frac{9}{8}(\frac{p_l}{6} - c) - \frac{6}{8}(\frac{2}{3}p_l - 2c) - \frac{3}{8}z_1 \\ &< \frac{1}{8}W_i + \frac{5}{4}p_l - 6c \\ &\leq \frac{1}{8}W_i + \frac{9}{8}\sum_{j\in F_i}\alpha_{ji}(p_j - k_j(k_j-1)c) \\ &\quad + \frac{5}{4}p_l - 6c\end{aligned}$$

□

Theorem 3. *The competitive ratio of the ECT algorithm for $m = 4$ is $20/9$.*

Proof. The lower bound for the competitive ratio was shown in Theorem 1. To complete the proof, we must show the matching upper bound by proving the inequality

$$I \leq \frac{1}{9}W + \frac{10}{9}\sum_{j\in J}(p_j - k_j(k_j-1)c) + 4b. \qquad (4)$$

Let us consider a block B_i (starting from B_1). Because of the definition of the partitioning scheme for the blocks, we encounter 4 cases to consider. We simply list the cases below due to space considerations.

- **Case 1:** There is no idle time in B_i.
- **Case 2:** Exactly 1 processor is idle in B_i.
- **Case 3:** Exactly 2 processors are idle in B_i.

- **Case 4:** Exactly 3 processors are idle in B_i.

In the upper bound proof for $m = 4$, we need to examine up to 4 consecutive blocks to prove inequality (4). Because the proof for $m = 3$ requires examining up to 3 consecutive blocks, this method appears to require examining at least m consecutive blocks to prove the inequality for an arbitrary m. If the number of blocks to examine grows linearly with the number of processors, the proof method would prove prohibitive for larger values of m.

□

5 Conclusions

In this paper, we examined the parallel job scheduling problem using a model for execution time which accounts for both computational speedup and communication slowdown with the goal to minimize makespan. The scheduling problem for a sequence of submitted jobs studied involves the determination of how many processors k_j to assign to a job and a start time s_j for beginning execution. We investigated a simple yet powerful algorithm, Earliest Completion Time. This online algorithm minimizes the completion time for a job given the current status of the system after all previous jobs have been scheduled. We showed a lower bound on the competitive ratio of

$$\frac{2m((m - \lfloor m/3 \rfloor - 1) + (m - 1))}{(m - \lfloor m/3 \rfloor)(m - \lfloor m/3 \rfloor - 1) + m(m - 1)}$$

for any $m \geq 2$ processors. For $m = 2, 3, 4$, the competitive ratio is exactly 2, $\frac{9}{4}$, and $\frac{20}{9}$, respectively. However, as m becomes large, the competitive ratio is at least $30/13 \approx 2.30769$. Due to the simple yet efficient nature of ECT, the algorithm is a good candidate for practical use once it can be shown to be competitive from a theoretical standpoint.

References

[1] B. Johannes, Scheduling parallel jobs to minimize the makespan, *Journal of Scheduling*, 9(5), 2006, 433-452.

[2] D. Feitelson, L. Rudolph, U. Schwiegelshohn, K. Sevcik, and P. Wong, Theory and practice in parallel job scheduling, *Job Scheduling Strategies for Parallel Processing*, Geneva, Switerland, 1997, 1-34.

[3] A.C. Laroy, Parallel run time models for real machines, *CSci710 M.S. Project: The College of William and Mary*, 2001.

[4] J. Du and J.T. Leung, Complexity of scheduling parallel task systems, *SIAM Journal on Discrete Mathematics*, 2(4), 1989, 473-487.

[5] J. Havill and W. Mao, Competitive online scheduling of perfectly malleable jobs with setup times, *European Journal of Operational Research*, to appear 2007.

[6] W. Mao, J. Chen, and W. Watson, On-line algorithms for parallel job scheduling problem, *Proceedings of the IASTED International Conference on Parallel and Distributed Computing Systems*, Cambridge, MA, USA, 1999, 753-757.

[7] J. Havill, W. Mao, and V. Dimitrov, Improved parallel job scheduling with overhead, *Proceedings of the Seventh Joint Conference on Information Sciences*, Cary, North Carolina, USA, 2003, 393-396.

[8] J. Turek, J. Wolf, and P. Yu, Approximate algorithms for scheduling parallelizable tasks, *Proc. of the ACM Symposium on Parallel Algorithms and Architectures*, San Diego, CA, USA, 1992, 323-332.

[9] W. Ludwig and P. Tiwari, Scheduling malleable and nonmalleable parallel tasks, *Proc. of the ACM-SIAM Symposium on Discrete Algorithms*, Arlington, Virginia, USA, 1994, 167-176.

[10] J. Blazewicz, M. Machowiak, G. Mounie, and D. Trystam, Approximation algorithms for scheduling independent malleable tasks, *Proc. of Euro-Par 2001*, Manchester, United Kingdom, 2001, 191-197.

[11] R. Karp, On-line algorithms versus off-line algorithms: How much is it worth to know the future?, *Tech. Report TR-92-044*, International Computer Science Institute at Berkeley, 1992.

[12] J. Blazewicz, M. Kovalyov, M. Machowiak, D. Trystram, and J. Weglarz, Preemptable malleable task scheduling problem, *IEEE Transactions on Computers*, 55(4), 2006, 486-490.

[13] K. Jansen and H. Zhang, Scheduling malleable tasks with precedence constraints, *Proc. of the ACM Symposium on Parallel Algorithms and Architectures*, Las Vegas, Nevada, USA, 2005, 86-95.

[14] U. Schwiegelshohn and R. Yahyapour. Fairness in parallel job scheduling, *Journal of Scheduling*, 3(5), 2000, 297-320.

[15] S. Muthukrishnan, R. Rajaraman, A. Shaheen and J. Gehrke, Online scheduling to minimize average stretch, *Proc. IEEE Symposium on Foundations of Computer Science*, New York City, NY, USA, 1999, 433-442.

[16] J. Sgall, On-line scheduling – a survey, in A. Fiat and G. Woeginger (Ed.) *On-Line Algorithms*, (Berlin: Springer-Verlag, 1997) .