Chapter 45
Non-Clairvoyant Scheduling

Rajeev Motwani* Steven Phillips* Eric Torng*†

Abstract

Virtually all research in scheduling theory has been concerned with *clairvoyant scheduling* where it is assumed that the characteristics of a job (in particular, its execution time, release time and dependence on other jobs) are known a priori. This assumption is invalid for scheduling problems that arise in time-sharing operating systems where the scheduler must provide fast turnaround for processes being generated by the users without any knowledge of the future behavior of these processes.

We study *preemptive, non-clairvoyant* scheduling schemes where the scheduler has no knowledge of the jobs' characteristics. We develop a model for evaluating scheduling strategies for single and multi-processor systems. This model compares the non-clairvoyant scheduler against the optimal clairvoyant scheduler, and it takes into account various issues such as release times, execution time, preemption cost, and the interdependence between jobs. Within this model we study some standard scheduling algorithms described in the systems literature, and we provide some theoretical justification for their effectiveness in practice by presenting some randomized and deterministic upper and lower bounds.

1 Introduction

We consider the problem of preemptively scheduling jobs with unspecified resource requirements so as to minimize the *average waiting time* of jobs in the system. This *non-clairvoyant* scheduling problem is the task typically faced by an operating system in a time-sharing environment where as many users as possible should get fast responses from the system. (A *clairvoyant* scheduling algorithm needs to know the jobs' characteristics in advance, and this is typically the class of algorithms considered in classical scheduling theory [13].) While we focus on non-clairvoyance, we also address other issues relevant to operating system scheduling such as the cost of preemption, scheduling for multiple processors, and inter-dependence of jobs.

Even a cursory examination of the literature on operating system scheduling [5, 7, 12, 14, 19, 25, 26] reveals that the designers of operating systems find the standard scheduling theory to be of little relevance to this problem. In fact, systems researchers have often complained about the inapplicability of results in classical scheduling theory to their problems [26]. They have resorted to analyzing scheduling strategies using empirical testing or probabilistic analysis [12]. We provide a theoretical evaluation of scheduling strategies for operating systems without the use of such probabilistic assumptions.

Standard worst-case analysis provides little insight since for each deterministic scheduling strategy, one can construct a scheduling problem on which that strategy performs poorly. We use the method of *competitive analysis* pioneered by Sleator and Tarjan [22] to study *preemptive non-clairvoyant scheduling strategies* using the *average waiting/idle time* of a job as the performance measure. The performance of a non-clairvoyant scheduler is compared to that of an optimal clairvoyant scheduler for each set of input jobs. We provide lower and upper bounds for randomized and deterministic schedulers, and our analysis may help explain the superiority of some commonly used scheduling strategies such as *Round Robin* and *multi-level adaptive feedback policy*.

There has been some recent work on non-clairvoyant scheduling [10, 21] that has focused on non-preemptive batch processing where the task is to minimize the *makespan* (i.e. the length) of the schedule. However, preemptions and the waiting-time measure are essential for modeling *interactive time-sharing systems* which rely on preemptive scheduling to give each user the view that they have sole access to a fast "virtual" machine. In addition non-clairvoyant scheduling should not be confused with the form of online scheduling stud-

*Department of Computer Science, Stanford University, Stanford, CA 94305-2140. Supported by NSF Grant CCR-9010517, and grants from Mitsubishi and OTL.

†Supported by NDSEG Fellowship.

ied by Graham [11] and Bartal *et al.* [2] where release times are unknown but the running time of a job is provided as soon as it appears in the system.

1.1 The Model and Preliminaries. We first define a generic scheduling problem; refer to the survey article by Lawler, Lenstra, Kan and Shmoys [13] for further details. A scheduling problem of *size n* consists of a collection of n independent jobs $J = \{J_1, J_2, \ldots, J_n\}$. Each job J_i has an *execution time* x_i and a *release time* r_i, the time at which job J_i is first available for processing. We are interested in *preemptive scheduling* where the execution of any job can be suspended at any time and resumed later from the point of preemption. In general, the jobs are to be processed on a collection of p identical processors $P = \{P_1, P_2, \ldots, P_p\}$.

A *schedule* is an assignment to each job of one or more time intervals on one or more processors with the total time assigned to job J_i equaling x_i. No two time intervals may overlap on any processor, and there should be no overlap between any two time intervals assigned to any job. Our goal is to find schedules which minimize the following objective functions.

DEFINITION 1.1. *The* **completion time** c_i *of a job* J_i *is the time at which it completes its execution. The* **total completion time** *is* $C = \sum_{i=1}^{n} c_i$, *and the* **average completion time** *is* $\widehat{C} = \frac{C}{n}$.

DEFINITION 1.2. *The* **waiting time** w_i *of a job* J_i *is the amount of time which elapses between its release and completion, i.e.* $w_i = c_i - r_i$. *The* **total waiting time** *is* $W = \sum_{i=1}^{n} w_i$, *and the* **average waiting time** *is* $\widehat{W} = W/n$.

DEFINITION 1.3. *The* **idle time** v_i *of a job* J_i *is the amount of time which elapses between its release and completion when it is not being processed, i.e.* $v_i = w_i - x_i$. *The* **total idle time** *is* $V = \sum_{i=1}^{n} v_i$, *and the* **average idle time** *is* $\widehat{V} = V/n$.

We classify scheduling problems into two types. A scheduling problem is said to be **offline** if all the release times are 0, i.e. all the jobs are available for execution at the start of the schedule. An **online** scheduling problem allows arbitrary (non-negative) release times. For offline scheduling, there is no difference between the completion time and the waiting time of a job.

A **clairvoyant scheduling algorithm** may use knowledge of the jobs' execution times to assign time intervals to jobs. A **non-clairvoyant scheduling algorithm** assigns time intervals to jobs without knowing the execution times of jobs that have not yet terminated. We will show that the optimal clairvoyant algorithm only needs to know the processing times of jobs currently in the system, even in the case of online scheduling; no knowledge of the existence, release time, or length of jobs to be released in the future is necessary.

For a scheduling problem J of size n, let $\widehat{W}_A(J)$ denote the average waiting time with respect to the schedule produced by an algorithm A. The **optimal clairvoyant algorithm** OPT minimizes $\widehat{W}(J)$ for each J. The **performance ratio** of a non-clairvoyant algorithm A is defined as

$$R_A(n) = \sup_{|J|=n} \frac{\widehat{W}_A(J)}{\widehat{W}_{OPT}(J)}$$

A non-clairvoyant scheduling algorithm is said to $f(n)$-**competitive** if $R_A(n) \leq f(n)$. The algorithm is said to be **competitive** if there exists a constant k for which it is k-competitive. If we measure the performance ratio in terms of the average idle time, we will refer to it as the **idle-performance ratio**, denoted $R_A^{\text{idle}}(n)$.

FACT 1.1. *For any scheduling algorithm* A, $R_A(n) \leq R_A^{\text{idle}}(n)$

We assume that the smallest possible quantum of time that can be assigned by a processor to a job is negligibly small compared to the length of the jobs being scheduled. This allows us to work with the notion of continuous time. In particular, the definition of Round Robin will make use of this feature. Clearly, such an assumption can affect our bounds by only a negligibly small amount.

1.2 Main Results. In Section 2, we consider offline scheduling on a single processor. We show that *Round Robin* (one of the most commonly used algorithms in practice) has a performance ratio of $2 - \frac{2}{n+1}$ which is optimal for deterministic, non-clairvoyant algorithms. For randomized algorithms, the lower bound on the performance ratio changes only in the lower order term, and a randomized variant of Round Robin achieves this lower bound. These results easily extend to the measure of idle-performance ratio. Finally, we show that the lower bounds remain unchanged even when the jobs are of bounded sizes (i.e. the ratio of the largest to the smallest execution time is bounded by some small constant). This is an interesting case since, in practice, fast turnaround is essential only for relatively small jobs. It is surprising that a non-clairvoyant scheduler cannot use either randomness or the restriction on the job sizes to obtain a better performance ratio. In Section 3, we extend some of these results to multi-processor scheduling.

We study the issue of minimizing the number of preemptions in Section 4. This is given a great deal of importance in the systems literature because preemptions can significantly slow down the execution of jobs due

to the "context switch". In addition, for many computationally intensive jobs such as scientific applications, significant time is required to rebuild the cache after a context switch.

A fundamental question is: what is the trade-off between the performance ratio and the number of preemptions? We show that any scheduling strategy that performs a sub-logarithmic number of preemptions per job has performance ratio $\Omega(n)$. Next we observe that Round Robin performs $\Omega(x)$ preemptions for a job of length x. In practice, the number of preemptions can be reduced by using variants of a scheme called the *multi-level adaptive feedback policy* [7, 19, 25]. We prove that a very simple version of this algorithm, the *geometric* algorithm, has a near-optimal performance ratio while using only a logarithmic number of preemptions. A randomized version of the geometric algorithm is shown to have essentially the same performance ratio as Round Robin. This may help explain why the multi-level adaptive feedback policy performs well in practice.

Section 5 is devoted to online scheduling. We show that clairvoyance can make a huge difference in a scheduler's ability to keep a system responsive. Any deterministic, non-clairvoyant algorithm has performance ratio $\Omega(n^{1/3})$ while any randomized, non-clairvoyant algorithm has performance ratio $\Omega(\log n)$. The proofs of these lower bounds use jobs of unbounded length. We also study the performance of algorithms for the natural case where the job lengths are bounded. If the ratio of the longest to the shortest job is bounded by k, then any deterministic, non-clairvoyant algorithm has performance ratio at least k (for large n), and a trivial algorithm achieves this bound.

Finally, in Section 6, we consider a version of the dining/drinking philosophers problem where some pairs of jobs cannot be executed simultaneously due to a conflict in their resource requirements. We demonstrate a competitive strategy for minimizing the makespan and exhibit nearly tight lower bounds. We leave open the problem of finding a good scheduler for minimizing average completion time for jobs with dependencies.

Some proofs have been omitted in this abstract and will appear in the final version.

2 Offline scheduling for single processors

In this section we analyze the situation where all jobs are released at time 0. In this case, total completion time, average completion time, total waiting time, and average waiting time are all equivalent measures for comparing scheduling algorithms, i.e. the performance ratio of a scheduling algorithm is identical with respect to all four performance measures. For ease of description, we will use the total completion time performance measure instead of the average waiting time performance measure until we study online scheduling.

In the first subsection, we describe the optimal clairvoyant algorithm OPT for offline scheduling. In the next two subsections, we prove matching upper and lower bounds on the performance ratios of deterministic and randomized non-clairvoyant algorithms for this problem. In the final subsection, we prove surprising lower bounds for the case where job sizes are bounded.

2.1 The Optimal clairvoyant algorithm OPT.

Before we can determine the performance ratios of non-clairvoyant algorithms, we first must describe the optimal clairvoyant algorithm OPT.

The completion time of a specific job can be broken down into two components: the running time of the individual job itself and the time it is delayed by the execution of other jobs before it finishes. To make the second term more tractable, we define the pairwise delay of two jobs J_i and J_j with respect to an algorithm A as follows:

DEFINITION 2.1. *$D^A_{i,j}$ is the time allocated by algorithm A to job J_i before job J_j completes. In other words, this is the amount of time by which job J_i delays job J_j.*

DEFINITION 2.2. *The* pairwise *delay $P^A_{i,j} = D^A_{i,j} + D^A_{j,i}$ is the amount by which two jobs J_i and J_j delay each other.*

We can now write the total completion time for an algorithm A on an instance J as

$$C_A(J) = \sum_{i=1}^{n} x_i \quad + \quad \sum_{1 \le i < j \le n} P^A_{i,j}$$

Since the first term is identical for all algorithms, OPT must minimize the second term. The minimum value of $P^A_{i,j}$ for any pair of jobs J_i and J_j is $\min(x_i, x_j)$, and this is achieved if job J_i is executed before job J_j. Hence OPT runs the jobs sequentially in non-decreasing order of job length. This is a special case of Smith's rule [23] for scheduling weighted jobs, and it is sometimes referred to as the SPT (shortest processing time first) algorithm [13].

2.2 Deterministic Algorithms.

We now show that the best possible performance ratio achievable by a *deterministic* algorithm is $2 - \frac{2}{n+1}$ and that it is achieved by the well-known Round Robin (RR) algorithm.

THEOREM 2.1. *For the offline scheduling problem, every deterministic, non-clairvoyant algorithm has $R_A(n) \ge 2 - \frac{2}{n+1}$ and $R^{idle}_A(n) \ge 2$.*

Proof. Fix any deterministic, non-clairvoyant algorithm A and execute the schedule produced by A for

1 time unit assuming no job finishes. Let e_i equal the amount of time A spent executing job J_i. The instance J is defined by choosing x_i to be $e_i + \epsilon$ for vanishingly small ϵ. We have

$$C_A(J) = n,$$

since A finishes all jobs immediately after time 1. Let us now compare this to the completion time of these jobs when scheduled by OPT. Rename the jobs so that $x_1 \leq x_2 \leq \cdots \leq x_n$; OPT will execute the jobs in this order. Clearly $P_{i,j}^{OPT} = x_i$ for $i < j$, so the pairwise delay sum for OPT is $\sum_{i=1}^{n}(n-i)x_i$. Adding in the total running time term, we get

$$C_{OPT}(J) = (n+1)\sum_{i=1}^{n} x_i \quad - \quad \sum_{i=1}^{n} i x_i.$$

This expression is maximized (subject to the constraints that $\Sigma_{i=1}^{n} x_i = 1$ and that the x_i are in non-decreasing order) when $x_1 = x_2 = \ldots = x_n = \frac{1}{n}$. Hence $C_{OPT}(J) \geq \frac{n+1}{2}$, so $R_A(J) \geq \frac{n}{\frac{n+1}{2}} = 2 - \frac{2}{n+1}$. The same proof ignoring the sum of execution times yields the idle-competitive lower bound. □

Notice that worst case for OPT in the above proof is when the algorithm A divides each unit of processing time equally between all currently uncompleted jobs. This is exactly the algorithm which is referred to as the Round Robin algorithm in the literature on operating systems scheduling [5, 7, 19, 25].

DEFINITION 2.3. *The Round Robin (RR) algorithm is a non-clairvoyant scheduling algorithm which at all times ensures that each uncompleted job has received an equal amount of processing time.*

Recall that in practice the RR algorithm cycles through the list of jobs, giving each job τ units of processing time in turn. For small enough τ, this is equivalent to the above definition.

We have seen that for the case where all the jobs have the same running time, RR achieves the optimal performance ratio of $2 - \frac{2}{n+1}$. In fact, this is the worst case scenario for RR which gives us a tight bound for the deterministic performance ratio.

THEOREM 2.2. *For the offline scheduling problem, Round Robin (RR) is exactly 2-idle-competitive and $(2 - \frac{2}{n+1})$-competitive.*

2.3 Randomized Algorithms. A natural question is whether randomization allows us to improve upon the lower bound of 2 on the performance ratio. We show that there is a simple randomized algorithm which improves very slightly upon RR, but its performance ratio is still asymptotically 2. We then prove a matching lower bound on the performance ratio of any randomized algorithm.

DEFINITION 2.4. *The algorithm $RAND$ behaves as follows on an instance of size n. With probability $1 - \frac{2}{n+3}$, it acts like RR. With probability $\frac{2}{n+3}$, it randomly orders the jobs and runs each job to completion in that order.*

We will refer to the run-to-completion algorithm as RTC.

THEOREM 2.3. *$RAND$ is $(2 - \frac{4}{n+3})$-competitive.*

Proof. Fix a scheduling instance J with job sizes $x_1, \ldots, x_n$. As shown earlier, the total completion time for RR is the following.

$$C_{RR}(J) = \sum_{i=1}^{n} x_i \quad + \quad 2 \sum_{1 \leq i < j \leq n} \min\{x_i, x_j\}$$

For randomly ordered jobs, the run-to-completion algorithm has the following expected total completion time. This is because $P_{i,j}^{RTC}$ is equally likely to be x_i or x_j.

$$E[C_{RTC}(J)] = \sum_{i=1}^{n} x_i + \sum_{1 \leq i < j \leq n} \frac{x_i + x_j}{2} = \frac{n+1}{2} \sum_{i=1}^{n} x_i$$

Thus when $RAND$ runs RR with probability $1 - \frac{2}{n+3}$ and run to completion with the remaining probability, the expected total completion time $C_{RAND}(J)$ is

$$\begin{aligned} &= \left(1 - \frac{2}{n+3}\right) C_{RR}(J) \quad + \quad \frac{2}{n+3} C_{RTC}(J) \\ &= \left(2 - \frac{4}{n+3}\right) \left(\sum_{i=1}^{n} x_i + \sum_{1 \leq i < j \leq n} \min\{x_i, x_j\}\right) \\ &= \left(2 - \frac{4}{n+3}\right) C_{OPT}(J) \end{aligned}$$

□

We now prove a matching lower bound for randomized algorithms.

THEOREM 2.4. *Any randomized algorithm has $R_A(n) \geq 2 - \frac{4}{n+3}$ and $R_A^{idle}(n) \geq 2$.*

Proof. We use Yao's [28] method (a variant of the von-Neumann minimax principle) by first proving a lower bound on the expected performance ratio of any deterministic algorithm on problem instances chosen from a specific probability distribution and then showing that the same lower bound holds for all randomized algorithms.

Let the n jobs have sizes which are independently chosen from the exponential distribution. The probability that a job has size greater than t is $G[t] = e^{-t}$. Fix a non-idling, deterministic algorithm A and consider two

jobs J_i and J_j. Let $i(t)$ denote the amount of time A has spent on J_i after a total of t time units have been spent on jobs J_i and J_j assuming neither J_i nor J_j has completed yet. Clearly $0 \leq i(t) \leq t$.

The probability that both jobs J_i and J_j are still running after exactly t time units have been allocated to the two jobs is exactly $G[i(t)]G[t-i(t)] = e^{-t}$. We conclude that

$$E[P_{i,j}^A] = \int_0^\infty G[i(t)]G[t-i(t)]\mathrm{d}t = \int_0^\infty e^{-t}\mathrm{d}t = 1$$

Thus for any algorithm A, the expected total completion time can be computed as follows.

$$E[C_A] = \sum_{i=1}^{n} E[x_i] \quad + \sum_{1\leq i<j\leq n} E[P_{i,j}^A] \;=\; n + \binom{n}{2}$$

Note that A's total completion time could be greater if A ever idles the processor while unfinished jobs are available.

For the optimal clairvoyant algorithm OPT, the pairwise delay of two jobs J_i and J_j can be computed as follows.

$$\begin{aligned} E[P_{i,j}^{OPT}] &= E[\min\{x_i, x_j\}] \\ &= \int_0^\infty G[t]^2\mathrm{d}t = \int_0^\infty e^{-2t}\mathrm{d}t = \frac{1}{2} \end{aligned}$$

Hence by linearity of expectation,

$$E[C_{OPT}] = \sum_{i=1}^{n} E[x_i] + \sum_{1\leq i<j\leq n} E[P_{i,j}^{OPT}] = n + \frac{1}{2}\binom{n}{2}$$

Therefore

$$\frac{E[C_A]}{E[C_{OPT}]} \geq \frac{n+\binom{n}{2}}{n+\frac{1}{2}\binom{n}{2}} = 2 - \frac{4}{n+3}$$

To see that this implies the lower bound for all randomized algorithms, consider a randomized algorithm B, which can be defined as a probability distribution on deterministic algorithms A. Let the random variable I be drawn from the distribution on scheduling instances described above. We have

$$\frac{E_{A,I}[C_A]}{E_I[C_{OPT}]} \geq 2 - \frac{4}{n+3}$$

Therefore

$$\frac{E_I[E_A[C_A]]}{E_I[C_{OPT}]} \geq 2 - \frac{4}{n+3}$$

so there must be some I for which

$$\frac{E_A[C_A]}{C_{OPT}} \geq 2 - \frac{4}{n+3}$$

and hence $R_B \geq 2 - \frac{4}{n+3}$. Again, the same proof holds for the idle-competitive lower bound when the sum of the execution times term is ignored. □

2.4 Bounded job sizes. The lower bounds in the previous section use instances with arbitrary large jobs. In some practical situations we might know in advance that the size of all jobs is within a specific range. Surprisingly, a non-clairvoyant scheduler cannot use this extra information to get a better performance ratio. Theorem 2.1 and a slightly weaker version of Theorem 2.4 still hold even when the job sizes are restricted to be within fixed bounds.

THEOREM 2.5. *For offline scheduling, any deterministic, non-clairvoyant algorithm A has $R_A(n) \geq 2 - \frac{2}{n+1}$ and $R_A^{\text{idle}}(n) \geq 2$, even when all jobs have length between 1 and 5.*

Note that $k = 5$ was chosen to make the proof simpler; it is not the smallest possible value for which the above theorem holds. However, the technique used in the proof breaks down for $k = 4$.

We can also show that when the maximum ratio between job sizes is bounded, no randomized algorithm can have a competitive ratio much better than 2.

THEOREM 2.6. *There is a function f such that for $\epsilon > 0$, any randomized algorithm A has $R_A(n) \geq 2 - \epsilon - o(n)$ when the maximum job ratio is at most $f(\epsilon)$.*

3 Offline multi-processor scheduling

We now consider offline scheduling for multiple identical processors. In practice, there may be communication costs associated with transferring a job from one processor to another, but we assume that any job can be assigned to any processor instantaneously. We define the performance ratio for a scheduling problem involving p processors as follows; the notion of competitiveness generalizes in the obvious manner.

DEFINITION 3.1. *Let $C_A(p, J)$ denote the total completion time of an instance J when scheduled by the algorithm A on p processors. The performance ratio of a non-clairvoyant algorithm for offline scheduling is*

$$R_A(p, n) = \sup_{|J|=n} \frac{C_A(p, J)}{C_{OPT}(p, J)}$$

The results and algorithms from the single processor offline model generalize to the multi-processor model. McNaughton's [18] results imply that OPT still obeys the SPT (shortest processing time) rule of Conway, Maxwell and Miller [6]; this rule is to assign the jobs in order of nondecreasing processing time to the earliest available processor. (For online scheduling on multiple processors, Du, Leung and Young [9] have shown that the problem of finding an optimal clairvoyant schedule is NP-hard, even in the case of 2 processors.) The Round Robin algorithm for multiple processors still

ensures that at all times each uncompleted job has received an equal amount of processing. Again Round Robin is the best possible deterministic, non-clairvoyant algorithm for multi-processor scheduling. Observe that the performance ratios we derive are identical to the single processor case when $p = 1$. We will assume that $n > p$, since otherwise RR and OPT are identical.

THEOREM 3.1. *For all $1 \leq p \leq n$, the Round Robin algorithm for offline scheduling on p processors is $\left(2 - \frac{2p}{n+p}\right)$-competitive and 2-idle-competitive.*

A generalization of the lower bound argument for single processor scheduling now shows that RR is the optimal non-clairvoyant algorithm.

THEOREM 3.2. *For offline scheduling on p processors, any deterministic, non-clairvoyant algorithm A has $R_A(p,n) \geq 2 - \frac{2p}{n+p}$ and $R_A^{\text{idle}}(p,n) \geq 2$.*

4 Minimizing Preemptions

We have ignored the cost of performing preemptions in the preceding analysis. However preemptions can significantly slow down the execution of jobs because there is a "context switch" requiring the access of secondary memory. Moreover, when a job has been preempted, the contents of the cache are not restored in a context switch. For computationally intensive jobs, such as many scientific applications, the time required to rebuild the cache is significant, and while the cache is being rebuilt, job execution can be slowed significantly.

The fundamental question we study in this section is: what is the trade-off between the performance ratio of an algorithm and the number of preemptions performed by the algorithm? We show that any scheduling strategy that performs a sublogarithmic number of preemptions per job has performance ratio $\Omega(n)$. While RR has an optimal performance ratio, it also performs an expensive $\Omega(x)$ preemptions on a job of length x. Hence we introduce the class of *geometric algorithms* which have low performance ratios (a randomized version has performance ratio very close to RR) while limiting the number of preemptions per job J_i to $O(\log x_i)$, the best possible for a competitive algorithm.

This is particularly interesting because the geometric algorithm is closely related to a commonly used scheduling strategy, *multi-level adaptive feedback policy* (MLAF) [7, 19, 25]. Although we don't consider some issues (such as the difference between I/O bound and computationally intensive jobs) relevant to the efficiency of MLAF, when a set of computationally intensive jobs is released simultaneously, MLAF reduces to the geometric algorithm. This helps explain why MLAF performs so well in practice.

To study the effect of preemptions we must assume a minimum job length (we choose 1 for convenience); otherwise, an adversary could allow the algorithm to run until the first preemption is made and then state that all other jobs have infinitesimal length compared to this first job which would imply no deterministic algorithm could have a performance ratio $o(n)$.

4.1 Preemptions versus Performance Ratio.

The following theorem shows that no algorithm that performs a sub-logarithmic number of preemptions can have a constant performance ratio.

THEOREM 4.1. *Let the function f satisfy $f(x) = o(\log x)$. Any deterministic scheduling strategy that performs at most $f(x)$ preemptions on a job of length x has performance ratio $\Omega(n)$.*

Proof. Fix any deterministic scheduling strategy A as described in the theorem statement. Suppose there are n jobs in the system. For a large enough x, we can upper bound the number of times A preempts a job of length x by $\frac{\log x}{3n \log n}$. Therefore, when some job first runs for x time, we can upper bound the total number of preemptions for all jobs by $\frac{\log x}{3 \log n}$. Thus, at some time t, some job must be at least n^3 greater than all other jobs and the minimum job size 1.

Let e_i be the amount of time A spent executing job J_i before time t. Choose x_i to be $e_i + \epsilon$ with ϵ vanishingly small. Let s be the length of the second longest job, and let b be the length of the longest job. The total completion time for these jobs as scheduled by A is at least nb while the total completion time for these jobs is at most $n^2 s + b$ when scheduled by OPT. The dominant term in both expressions is the b term, so the performance ratio is asymptotic to n. □

Now consider the RR algorithm. Although it achieves an optimal performance ratio, it performs a worst-case $\Omega(x)$ preemptions for a job of length x if all jobs have equal length. Assuming constant cost per preemption, performing this many preemptions multiplies the makespan by a constant greater than 1 and increases the performance ratio above 2.

We can reduce the number of preemptions performed by increasing the time slice offered to each job in each successive round. An additive increase only reduces the constant factor. However, a geometric increase in time slice reduces the number of preemptions of a job of length x from $\Omega(x)$ to $O(\log x)$. Thus, assuming constant cost per preemption, the performance ratio derived ignoring preemptions is accurate in the high order term, and the makespan increases by less than a constant factor. We call any algorithm with a geometrically increasing time slice a *geometric algorithm.*

4.2 Deterministic Geometric Algorithms. We first study the version of geometric algorithms which are used in practice – deterministic geometric algorithms. The family of geometric algorithms G_r is defined as follows for any $r > 1$.

DEFINITION 4.1. *G_r works in rounds. In the k'th round ($k \geq 0$), G_r cycles through the jobs giving each job a time slice of t_k units of processing time. The sum of time slices till the end of the k'th round is r^k. If a job completes within a time slice, G_r immediately moves on to the next job.*

Note that the time slices satisfy $t_0 = 1, t_1 = r - 1$, and $t_k = rt_{k-1}$ for $k > 1$.

RR and *RTC* fit naturally around the family of geometric algorithms G_r with *RR* corresponding to G_1 and *RTC* corresponding to G_∞.

THEOREM 4.2. *For any fixed ratio $r > 1$, $R^{idle}_{G_r}(n) = (1 + r)$. Therefore G_r is $(1 + r)$-competitive.*

Proof. We consider only the pairwise delay component of total completion time. Consider of pair of jobs J_i and J_j with $x_i \leq x_j$. Since G_r never spends more than rx_i time executing job J_j before finishing job J_i, we have $P^{G_r}_{i,j} \leq (1 + r)x_i = (1 + r)\min\{x_i, x_j\} = (1 + r)P^{OPT}_{i,j}$. It follows that $R^{idle}_{G_r}(n) \leq (1 + r)$.

To see that this upper bound is tight, consider the instance where the n job sizes are $1+\epsilon$, $r+\epsilon$, $r^2+\epsilon$, ..., $r^{n-1}+\epsilon$ with ϵ vanishingly small. If, in each round, G_r executes the remaining jobs in non-increasing order of execution time, the pairwise delay for each pair of jobs J_i and J_j, $x_i < x_j$, will be $(1 + r)x_i$ whereas for *OPT* the pairwise delay will only be x_i. The upper bound on the performance ratio of G_r follows from Fact 1.1. □

The upper bound on the performance ratio may not be tight because we have ignored the sum of execution times term in the total completion time. In the example in the proof, the sum of the execution times is significant when compared to the pairwise delays. To find a good lower bound on the performance ratio, we need to find an instance which maximizes the pairwise delays between jobs of different sizes relative to the pairwise delays between jobs of roughly equal size while keeping the sum of the execution times insignificant.

The instances we have found that provide the best lower bound for $R_{G_r}(n)$ are the following:

DEFINITION 4.2. *Choose $m \gg k$ with both m and k very large. Define the set of jobs $J_{r,s}$ to be $\frac{m}{s^i}$ jobs of size $r^i + \epsilon$ for $0 \leq i \leq k-1$, $s > 1$, and some vanishingly small $\epsilon > 0$.*

We can prove that G_r is $2\sqrt{r}$-competitive for $J_{r,\sqrt{r}}$ and that $J_{r,\sqrt{r}}$ is the worst example for G_r within the class $J_{r,s}$.

THEOREM 4.3. *For any fixed ratio $r > 1$, $C_{G_r}(J_{r,\sqrt{r}}) \geq 2\sqrt{r}C_{OPT}(J_{r,\sqrt{r}})$, so $R_{G_r} \geq 2\sqrt{r}$.*

The following theorem shows that the choice $s = \sqrt{r}$ is optimal in the above theorem.

THEOREM 4.4. *$C_{G_r}(J_{r,s})/C_{OPT}(J_{r,s})$ is maximized by choosing $s = \sqrt{r}$.*

4.3 Randomized Geometric Algorithms. We now describe a randomized version of the geometric algorithm where the initial time slice is randomly chosen. This randomization avoids the worst-case behavior of G_r which is achieved by having the job's execution times synchronized with the time allocated to them during the various rounds. We feel the analysis of this algorithm provides real insight into why MLAF works well in practice as it accurately models real life situations where there is not an aggressive adversary choosing job lengths synchronized with the machine's quantum.

DEFINITION 4.3. *For any $r > 1$, the randomized geometric algorithm RG_r works in rounds, giving each uncompleted job an equal time slice. The jobs are randomly permuted once, and this ordering is used in each round. The time slice of each round is determined as follows; the value t is chosen at random using some probability distribution on $[1, r]$. In Round 0, each job gets $t/(r-1)$ time units; in Round i, each job gets tr^{i-1} time units ($i \geq 1$).*

Round 0 exists only to simplify analysis by ensuring that the sum of the lengths of rounds $0 \ldots i$ is $\frac{t}{r-1}r^i$ for each $i \geq 0$. We assume that each job has an execution time of at least $\frac{1}{r-1}$ time units.

We need to specify the distribution used to choose t. The following lemma shows that there is a unique choice of a best possible distribution.

LEMMA 4.1. *For the algorithm RG_r, let $m(x, y)$ denote the supremum over job lengths x and y of $\frac{E[P_{x,y}]}{\min\{x,y\}}$. Then $m(x, y)$ is minimized by the choice of the probability distribution given by the density function $f(t) = \frac{1}{t \log r}$.*

The next theorem determines the idle-performance ratio of this "best possible" randomized geometric algorithm.

THEOREM 4.5. *For any $r > 1$, the randomized algorithm RG_r using the density function $f(t) = \frac{1}{t \log r}$ has idle performance ratio $c(r) = 1 + \frac{(r+1)(r-1)}{2r \log r}$, and for no other distribution does RG_r have a lower idle-performance ratio.*

Notice that there is a tradeoff between performance ratio and the number of preemptions. As r increases, the performance ratio increases but the number of preemptions decreases. Notice also that for small values of r, $c(r)$ is close to the optimal value of 2 (see

Theorem 2.4, Section 2.3). For instance, when $r = 2$, $c(r) = 2.08$.

5 Online scheduling

We now turn our attention to the problem of *online* non-clairvoyant scheduling. The n jobs can be released at arbitrary times, and the non-clairvoyant algorithm is not aware of the existence of a job till it is released. The measure of performance for the online situation is the average waiting time of the jobs (note that this is no longer the same as the average completion time). The online scheduling problem can model the practical situation where jobs can be interrupted for I/O. In our context, I/O interrupts merely divide a job into a set of smaller jobs with varying release times.

Clearly, the offline model we explored in the previous sections is a special case of the online model. It is reasonable to hope that the algorithms which work well in the offline model will carry over with little or no loss of competitiveness. Unfortunately, any deterministic, non-clairvoyant algorithm must perform very poorly when compared to the optimal clairvoyant algorithm. The situation may be better for randomized non-clairvoyant algorithms, but it is still not possible to be competitive. These facts (and their proof) show that clairvoyance can make a very large difference for a scheduler trying to keep a system responsive under heavy loads.

The clairvoyant algorithm OPT obeys the following generalization of Smith's rule [13]: *the job being executed at any time is the one which requires the smallest amount of processing to complete.* Note that OPT requires no knowledge about future jobs.

We now show that the non-clairvoyant algorithms of the previous sections are no longer competitive in this more general setting. We start with RR; in the online setting, RR ensures that during any time interval where there are no completions or releases, each active job receives an equal share of the processing time.

THEOREM 5.1. *For the online scheduling problem of size n, RR has performance ratio $\Omega(\frac{n}{\log n})$.*

Proof. Let H_k denote the k'th harmonic number. Consider the set of n jobs with the following release and execution times.

- At time 0, two jobs of length 1 are released.
- For $1 \leq k \leq n-2$, at time H_k, a job of length $\frac{1}{k+1}$ is released.

First consider the performance of OPT on this instance. OPT sets aside one of the first two jobs and schedules the remaining jobs in a non-preemptive manner. In other words, each job (except the one set aside) is scheduled as soon as it is released and finishes just as the next job is released. Finally, the initially set aside job is scheduled. This one job has waiting time $1 + H_{n-1}$ while every other job has waiting time equal to its length. Therefore the total waiting time for the n jobs is $2H_{n-1} + 1$.

RR, on the other hand, does not complete any job until time $H_{n-1} + 1$. To see this, observe that when the job of size $\frac{1}{k}$ enters the system, the k jobs that have already been released require exactly the same amount of processing time, i.e. $\frac{1}{k}$, to complete. A straightforward calculation shows that RR has a total waiting time of $2n + H_{n-1} - 1$. Thus the performance ratio of RR on this instance is $\Omega(\frac{n}{\log n})$. □

While Round Robin has a very large performance ratio (no algorithm which does not idle the processor can have a performance ratio worse than n), the following theorem shows that every deterministic, non-clairvoyant algorithm has a large performance ratio.

THEOREM 5.2. *For the online scheduling problem of size n, any deterministic non-clairvoyant algorithm has performance ratio $\Omega(n^{\frac{1}{3}})$.*

Proof. We present an adversary strategy which works in two stages. At time 0, the adversary releases k jobs and lets the algorithm A schedule them for k time units. The adversary ensures that the length of each job is large enough so that no job is completed during that period. Let e_i be the amount of processing time allocated to job i during the first k time units. Without loss of generality, assume that the jobs are ordered so that $e_i \geq e_j$ if $i < j$; clearly, $e_1 \geq 1$. The adversary now chooses the execution times of the k jobs such that $x_i = e_i + \frac{1}{k-1}$. Observe that by this time OPT would have completed all these jobs except the largest one, viz. job 1, while the algorithm A has completed no jobs, and each job requires $\frac{1}{k-1}$ time units of processing to complete.

After the first k time units, the adversary releases a job of length $\frac{1}{k-1}$ every $\frac{1}{k-1}$ time units apart for a total of k^2 time units. A total of $k^3 - k^2$ jobs are released in this phase. It is clear that for both algorithms, OPT and A, the best possible strategy is to immediately run to completion each job that is released during this second phase. Finally, both algorithms will finish off all the remaining jobs from the first phase.

During the first phase, the waiting time for both algorithms is $O(k^2)$ because there are only k jobs around for k time. For the algorithm A, $k+1$ jobs are present in the system at any time during the second phase while OPT only has two unfinished jobs at any time during the second phase. Thus OPT has total waiting time $O(k^2)$, while the algorithm A has total waiting time $\Omega(k^3)$. This implies a lower bound of $\Omega(k)$ on the performance ratio of A. Since $n = k^3 - k^2 + k < k^3$,

$k > n^{\frac{1}{3}}$ completing the proof. □

The assumption that A is deterministic is crucial to the above proof. The adversary must decide upon the lengths of the jobs after observing the amount of processing received by each job. While we hold out hope for defeating such an adversary and obtaining a reasonably small performance ratio by the use of randomness, the following theorem shows that even randomized algorithms cannot be competitive.

THEOREM 5.3. *For the online scheduling problem of size n, any (randomized or deterministic) non-clairvoyant algorithm has performance ratio* $\Omega(\log n)$.

The above lower bounds use collections of jobs where the ratio between the shortest and longest job is unbounded. The following theorems give tight bounds when the ratio between the smallest and largest job is bounded by k, for some constant k. These theorems should be contrasted with Theorem 2.5 where it is shown that knowledge of a small bound on the job sizes does not improve the competitive ratio of deterministic, non-clairvoyant algorithms for offline scheduling.

THEOREM 5.4. *No deterministic, non-clairvoyant algorithm has performance ratio bounded away from k when the maximum ratio is k.*

THEOREM 5.5. *Run-to-completion is k-competitive if the largest job is at most k times as large as the smallest job.*

Proof. At any time, OPT and RTC have spent the same total time processing the set of jobs. In addition, RTC never sets aside a job that has received some processing time. Hence at any time, RTC has no more than k times as many jobs as OPT that are not currently being processed. Note that the total waiting time is the sum of the job execution times plus the integral over time of the number of jobs not currently being processed. Therefore RTC is k-competitive. □

6 Scheduling dependent jobs

We now consider scheduling jobs on multiple machines when there is some dependence between the jobs. Any multi-processor system will have certain resources which can only be made available to one job at a time. Two jobs which demand such a resource are constrained to have non-overlapping executions. Any scheduler for such a system must guarantee two properties: 1) *safety*: no two conflicting jobs may be executed simultaneously; and, 2) *liveness*: there is progress being made on completing the jobs. This problem has been abstracted into the well-known *dining philosophers* and *drinking philosophers* problems [8, 20, 3, 16]. Several interesting algorithms have been devised [17, 24, 3, 16, 1] to solve such problems in an "online" setting. Typically, the goal is to minimize the response time for a job in terms of the parameters of the job set.

We have obtained some preliminary results in the following model. An instance of the *distributed scheduling problem* is a graph where the vertices represent jobs, and two vertices are adjacent when the corresponding jobs are in conflict. Each job has an execution time and a release time. The processing environment consists of an infinite number of processors, and a centralized scheduler assigns jobs to processors. A clairvoyant scheduler has complete knowledge of the jobs which will appear in the future including their conflict relations, execution times, and release times. A non-clairvoyant scheduler has the complete information only about the jobs which have already been released; in particular it knows the conflict relations between the currently available jobs.

To focus on the issue of clairvoyance, we assume that the number of processors is infinite. Having a bounded number of processors can only make the non-clairvoyant algorithms more competitive. Unfortunately, our results apply only to the makespan measure of performance. The goal of a scheduler is to minimize the time till all the jobs have been completed. An interesting open problem is to study the performance of non-clairvoyant algorithms when the performance measure is the average waiting time. Given that the measure of performance is the makespan, there is no essential difference between the preemptive and the non-preemptive version of the problem, and our results apply to both cases.

In the *uniform distributed scheduling* problem each job has unit execution time, and jobs are released at integral times. We start by considering this special case which is closely related to the problem of online coloring of graphs [15, 27]. For this problem, the optimal non-clairvoyant algorithm, even with randomization, has a performance ratio asymptotic to 2. An algorithm that achieves this bound is the *CHROMATIC* algorithm.

THEOREM 6.1. *For uniform distributed scheduling, the performance ratio of any deterministic, non-clairvoyant algorithm is at least* $2 - \frac{2}{n+1}$.

This argument can be extended to allow for randomized schedulers.

THEOREM 6.2. *For uniform distributed scheduling, no randomized, non-clairvoyant algorithm is c-competitive for any constant* $c < 2$.

We show that there exists a clairvoyant algorithm which achieves the competitive ratio of 2. We refer to this algorithm as the *CHROMATIC* algorithm – at each time step it optimally colors the vertices of the residual task graph and schedules any one color class in parallel.

THEOREM 6.3. *The CHROMATIC algorithm is*

2-competitive for the uniform distributed scheduling problem.

Proof. Suppose that the last job to be released is released at time T. Let the chromatic number of the task graph be χ. Then, regardless of the jobs scheduled before time T, the residual graph at time T has chromatic number at most χ. It follows that the makespan of the *CHROMATIC* algorithm is at most $T+\chi$. On the other hand, the optimal clairvoyant algorithm cannot finish all the jobs before time $T+1$. Moreover, the chromatic number of the entire task graph is a lower bound on the number of time units needed by *OPT*. Thus, we have that

$$C_{CHROMATIC} = T + \chi \leq (C_{OPT} - 1) + C_{OPT}. \quad \square$$

Consider now the general distributed scheduling problem where the jobs are allowed to have arbitrary execution and release times. The algorithm we analyze is a generalization of the *CHROMATIC* algorithm and is referred to as the *GREEDY* algorithm. At any time when no job is currently being executed, the *GREEDY* algorithm considers the residual job graph and initiates the execution of those jobs which would have been executed first by an optimal algorithm.

THEOREM 6.4. *The GREEDY algorithm is 3-competitive.*

The question of determining the optimal ratio for the general problem remains open.

Acknowledgements

This work originated in discussions the first author had with Howard Karloff and Seffi Naor. We would like to thank them for their contributions. The second author is grateful for conversations with Anna Karlin. We would also like to thank Barun Chandra and Sundar Vishwanathan for sharing their observations with us. Finally, we are grateful to Amos Fiat for his help and encouragement.

References

[1] B. Awerbuch and M. Saks, A dining philosophers algorithm with polynomial response time, *FOCS* (1990) 65–74.

[2] Y. Bartal, A. Fiat, H. Karloff and R. Vohra, New Algorithms for an Ancient Scheduling Problem, *STOC* (1992) 51–58.

[3] K. Chandy and J. Misra, The drinking philosophers problem, *ACM TOPLAS* **6** (1984) 632–646.

[4] H. Chernoff, A Measure of Asymptotic Efficiency for Tests Based on the Sum of Observations, *Ann. Math. Stat.* **23** (1952) 493–509.

[5] E.G. Coffman, Jr., and P.J. Denning, *Operating Systems Theory* (Prentice-Hall, Englewood Cliffs, 1973).

[6] R.W. Conway, W.L. Maxwell and L.W. Miller, *Theory of Scheduling* (Addison-Wesley, Reading, 1967).

[7] H.M. Deitel, *An Introduction to Operating Systems* (Addison-Wesley, Reading, 1990).

[8] E.W. Dijkstra, Hierarchical ordering of sequential processes, *Acta Informatica* (1971) 115–138.

[9] J. Du, J.Y.-T. Leung and G.H. Young, Minimizing Mean Flow Time with Release Time Constraints, Technical Report, Comp. Sc., Univ. Texas at Dallas, 1988.

[10] A. Feldmann, J. Sgall and S-H. Teng, Dynamic Scheduling on Parallel Machines, *FOCS* (1991) 111–120.

[11] R.L. Graham, Bounds on multiprocessing anomalies, *SIAM J. Appl. Math.* **17** (1969) 263–269.

[12] L. Kleinrock, *Queueing Systems, Vols. I and II* (John Wiley & Sons, New York, 1976).

[13] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, Sequencing and Scheduling: Algorithms and Complexity, *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory* (1990).

[14] E.D. Lazowska, J. Zahorjan, G.C. Graham and K.C. Sevcik, *Quantitative System Performance* (Prentice-Hall, 1984).

[15] L. Lovasz, M. Saks and W.T. Trotter, An online graph coloring algorithm with sublinear performance ratio, *Disc. Math.* **75** (1989) 319–325.

[16] J. Lundelius and N. Lynch, Synthesis of efficient drinking philosophers algorithms, Unpublished manuscript, 1988.

[17] N. Lynch, Upper bounds for static resource allocation in a distributed system, *JCSS* **23** (1981) 254–278.

[18] R. McNaughton, Scheduling with deadlines and loss functions, *Manag. Sci.* **6** (1959) 1–12.

[19] J.L. Peterson and A. Silberschatz, *Operating Systems Concepts* (Addison-Wesley, 1985).

[20] M. Rabin and D. Lehmann, On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem, *POPL* (1981) 133–138.

[21] D.B. Shmoys, J. Wein and D.P. Williamson, Scheduling Parallel Machines On-line, *FOCS* (1991) 131–140.

[22] D.D. Sleator and R.E. Tarjan, Amortized efficiency of list update and paging rules, *CACM* **28** (1985) 202–208.

[23] W.E. Smith, Various optimizers for single-stage production, *Naval Res. Logist. Quart.* **3** (1956) 59–66.

[24] E. Styer and G. Peterson, Improved algorithms for distributed resource allocation, *PODC* (1988) 105–116.

[25] A.S. Tanenbaum, *Modern Operating Systems* (Prentice-Hall, Englewood Cliffs, 1992).

[26] A.S. Tanenbaum and R. van Renesse, Distributed Operating Systems, *ACM Comp. Surveys* **17** (1985) 419–470.

[27] S. Vishwanathan, Randomized online graph coloring, *FOCS* (1990) 464–469.

[28] A.C.-C. Yao, Probabilistic computations: Towards a unified measure of complexity, *FOCS* (1977) 222–227.