Quality vs. Performance in Lookahead Scheduling

Ben Coleman[1]
[1]Department of Mathematics and Computer Science, Moravian College, Bethlehem, PA 18018, USA

Abstract
We consider the use of lookahead to minimize the average wait time when scheduling jobs on a single machine. We compare the relationship between quantity of lookahead and quality of schedule. Our results indicate that while we do gain further improvement with more lookahead, the amount of improvement drastically decreases beyond Lookahead-1. Further, the time required for each scheduling decision increases with the size of the lookahead. We conclude that Lookahead-1 offers the best balance between quality of schedule and performance of the scheduler.
Keywords. Lookahead, Real-Time Scheduling

1 Problem Description

In the single machine job scheduling problem, there are n jobs, and associated with each job, J_j, is a processing time, p_j, and arrival time, r_j. We assume that, through a lookahead queue, we always know the arrival time and processing time of one or more of the next arriving jobs, called the lookahead jobs, and that jobs that will arrive later than the lookahead jobs are not known [3]. The goal is to minimize the sum of completion times, $\sum c_j$, equivalent to the average wait time $\frac{1}{n}\sum w_j$ because $c_j = w_j + p_j$.

Given that not all jobs are known when decisions must be made, it is not possible to create a scheduling algorithm that produces the optimal schedule in all cases. Instead, we use a greedy heuristic to incrementally create a schedule that is an approximation of the optimal. When a job arrives and a machine is idle, the algorithm decides whether to schedule that job or to hold it until another job arrives. Likewise, when a job completes and there are jobs waiting, the algorithm must decide whether to execute the first job in the wait queue or hold the machine idle for the next job to arrive.

To construct a complete schedule for n jobs, the decision whether to wait or schedule is made multiple times. It is important to point out that the algorithm we present executes once for each of these decisions. This is a significant departure from a traditional algorithm that executes outside of time and creates the entire schedule in one execution. Our algorithm executes in real-time and must complete execution before any job may begin. Therefore, each decision must be made quickly.

There is a strong relationship between the size of the lookahead and the time necessary to determine the optimal schedule. At one extreme, we have no lookahead, and the "best" decision is always to schedule the shortest waiting job, which can be completed in constant time (assuming the wait queue is sorted). At the other extreme we have a system where all future jobs are known. In this case, we have a well-studied single machine scheduling problem that can produce the optimal schedule but is known to be NP-hard [4].

In this paper, we extend previous results [1, 2] to search for a balance between quantity of lookahead and time required to make decisions. We formally consider the Shortest Job First (SJF), Lookahead-1 (LA1), and Lookahead-2 (LA2) algorithms and generalize these results to consider a Lookahead-k algorithm. Through algorithm analysis and simulation, we find that LA1 produces the appropriate balance.

2 Model Overview

We have a single machine capable of executing jobs. As these jobs arrive, they are either scheduled immediately, or enter a wait queue, Q_W. The lookahead jobs are "held" in their own queue, Q_{LA} until they arrive. Figure 1 shows a representative case for our scheduling model where jobs J_1 through J_l are in the wait queue and two lookahead jobs, J_{l+1} with arrival time r_{l+1} and J_{l+2} with arrival time r_{l+2}, are in the lookahead queue.

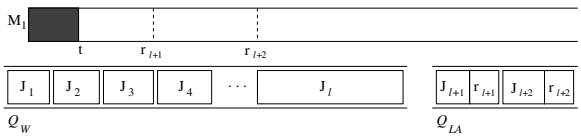

Figure 1: A typical situation for LA2 scheduling

In a general scheduling decision, the algorithm has four possible choices:

- W/W – wait-wait (Figure 2) – The algorithm decides to wait for the first lookahead job at time t and then wait again for the second lookahead job at the appropriate time.

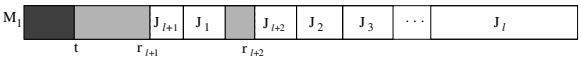

Figure 2: A Typical W/W schedule

- $W/\overline{W}$ – wait-nowait (Figure 3) – The algorithm

waits for the first lookahead job but then does not wait for the second job. Instead, the second job arrives during the execution of another job and is added to the wait queue to be scheduled later.

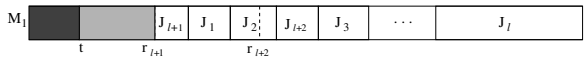

Figure 3: A Typical $W/\overline{W}$ schedule

- $\overline{W}/W$ – nowait-wait (Figure 4) – The algorithm does not wait for the first job, but rather schedules J_1. When J_{l+1} arrives, it is added to the wait queue. Jobs are scheduled in sorted order until immediately before J_{l+2} would arrive and then the scheduler waits for this job to arrive.

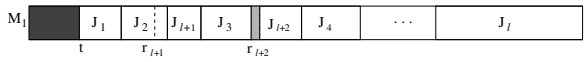

Figure 4: A Typical $\overline{W}/W$ schedule

- $\overline{W}/\overline{W}$ – nowait-nowait (Figure 5) – For both jobs, the algorithm does not wait and simply adds the jobs to the wait queue when they arrive.

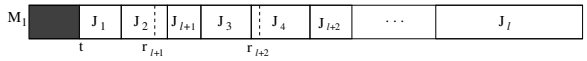

Figure 5: A Typical $\overline{W}/\overline{W}$ schedule

In many cases, we can eliminate one or more of these possibilities by arguing that they always produce a worse schedule than other decisions. In the analysis that follows, we use the following notation:

- t (the current time)
- $p_1, \ldots, p_l$ (processing times of the jobs in Q_W)
- p_{l+1} and r_{l+1} (attributes of the first lookahead job)
- p_{l+2} and r_{l+2} (attributes of the second lookahead job)
- For notational simplicity, define $\Delta = r_{l+1} - t$ and $\Delta' = r_{l+2} - t$

3 Algorithm Analysis

In this section, we demonstrate that the Lookahead-2 algorithm can make local optimal decisions. That is, we show that the evaluation of various simple conditions is sufficient for the algorithm to conclude whether to wait or schedule.

The following six lemmas divide the decision-making process based on the lengths of the jobs and the arrival times for the lookahead jobs. For instance, Lemmas 1 and 2 consider the case when J_1 is shorter than both J_{l+1} and J_{l+2} and all the cases when J_1 is longer than Δ'. The final two lemmas consider the most complex cases and include the notation S_W and $S_{\overline{W}}$, representing the set of jobs that execute between the current time and the moment the second lookahead job arrives.

For each Lemma, the "if and only if" condition also implies that if the inequality evaluates to false the algorithm chooses to execute J_1 at time t.

Lemma 1. *For any of the following cases,*

- $p_1 \leq \Delta$,
- $p_1 > \Delta$ *and* $r_{l+1} = r_{l+2}$,
- $p_1 \geq \Delta'$ *and* $r_{l+2} > r_{l+1}$ *and* $p_1 \leq p_{l+1}$,
- $p_1 \geq \Delta'$ *and* $r_{l+2} > r_{l+1}$ *and* $p_1 > p_{l+1}$ *and* $(l+2)(r_{l+2} - r_{l+1}) \leq p_{l+1} - p_{l+2}$, *or*
- $\Delta < p_1 < \Delta'$ *and* $r_{l+2} > r_{l+1}$ *and* $p_1 \leq p_{l+1}$,

the algorithm chooses to wait until r_{l+1} if and only if

$$(l+2)\Delta' \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}.$$

Lemma 2. *When $p_1 \geq \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+2)(r_{l+2} - r_{l+1}) > p_{l+1} - p_{l+2}$, the algorithm chooses to wait until r_{l+1} if and only if $(l+2)\Delta + \min\{\max\{0, p_1 - p_{l+2}\}, (l+1)\max\{0, r_{l+2} - r_{l+1} - p_{l+1}\}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\} - \max\{0, p_{l+1} - p_{l+2}\}$.*

Lemma 3. *When $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) \leq p_{l+1} - p_{l+2}$, the algorithm chooses to wait until r_{l+1} if and only if $(l+2)\Delta - (l+1)(\Delta' - p_1) \leq \max\{0, p_1 - p_{l+1}\} - \max\{0, p_{l+1} - p_{l+2}\}$.*

Lemma 4. *When $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ and $p_{l+1} \geq \Delta' - p_1$, the algorithm chooses to wait until r_{l+1} if and only if $(l+2)\Delta + \min\{\max\{0, p_1 - p_{l+2}\}, (l+1)\max\{0, r_{l+2} - r_{l+1} - p_{l+1}\}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_1 - p_{l+2}\}$.*

Lemma 5. *When $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ and $p_{l+1} < \Delta' - p_1$ and $g = h+1$, the algorithm chooses to wait until r_{l+1} if and only if $(h+2)\Delta + (l-h)(\Delta + p_{l+1} + \sum_{j=1}^{h+1} p_j - \Delta') + \max\{(l-h)(\Delta' - p_{l+1} - \sum_{j=1}^{h+1} p_j), p_{h+2} - p_{l+2}\} \leq \max\{0, p_1 - p_{l+1}\} + \max\{0, p_{h+2} - p_{l+2}\}$, where h and g are as defined based on the contents in sets S_W and $S_{\overline{W}}$.*

Lemma 6. *When $\Delta < p_1 < \Delta'$ and $r_{l+2} > r_{l+1}$ and $p_1 > p_{l+1}$ and $(l+1)(\Delta' - p_1) > p_{l+1} - p_{l+2}$ and $p_{l+1} < \Delta' - p_1$ and $g = h$, the algorithm chooses to wait until r_{l+1} if and only if $(l+2)\Delta + \min\{(l-h+1)(\Delta' - \Delta - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\} - \min\{(l-h+1)(\Delta' - p_{l+1} - \sum_{j=1}^{h} p_j), p_{h+1} - p_{l+2}\} \leq \max\{0, p_1 - p_{l+1}\}$, where h and g are as defined based on the contents in sets S_W and $S_{\overline{W}}$.*

For Lemmas 1 through 4, the running time is constant because all values used are known. However, in Lemmas 5 and 6, the running time is linear. This occurs because the algorithm must know when r_{l+2} occurs relative to the previous completion time. If

the arrival occurs close to a completion time, waiting may be beneficial. However, given the sets of waiting jobs and lookahead jobs, this fact can only be determined by direct calculation, considering the jobs in order. Therefore, although there are many cases where the LA2 algorithm makes decisions in constant time, there exists instances requiring linear time.

Mao and Kincaid [5] present a LA1 algorithm that always executes in constant time, and we have shown that the LA2 algorithm requires at most linear time. Obviously, the quality of the resulting schedule will improve as more lookahead jobs are considered, but so does the running time. In fact, we can produce the optimal schedule if we know of all the future jobs, but this problem is strongly NP-hard [4] meaning that it will require exponential time (unless $\mathcal{P} = \mathcal{NP}$).

Thus, there is a tension between quality of schedule and running time. If we use an algorithm with a large lookahead, it creates very good schedules, but the execution time of the algorithm is large. On the other hand, if we use a small amount of lookahead, the algorithm runs quickly, but the quality of the schedule is reduced.

4 Simulation Results

To further study this trade-off, we used simulation to compare the quality of schedules produced by the SJF, LA1, and LA2 algorithms. In particular, we were interested in the amount of improvement the various lookahead algorithms produced over the SJF algorithm.

In our simulation, we assumed the system started empty and idle, there were no jobs in the single wait queue at time zero, and no jobs were executing. For each simulation, a fixed number of jobs arrived over time, and the simulations finished when all jobs had completed.

In our first simulation, a stationary arrival process was used with inter-arrival times drawn from an Exponential distribution. We found that improvement drastically decreased as the number of jobs increased, especially when the arrival rate closely matched the rate at which jobs could be processed. This occurs because the resulting schedule contains very long time periods where the machine is busy. Consequently, waiting for a lookahead job is almost never beneficial because so many other jobs will have increased wait times. Using a smaller number of jobs, however, we found that the LA1 and LA2 algorithms both produced significant improvements over the SJF algorithm. Figure 6 shows a typical percent improvement graph when 25 jobs were simulated.

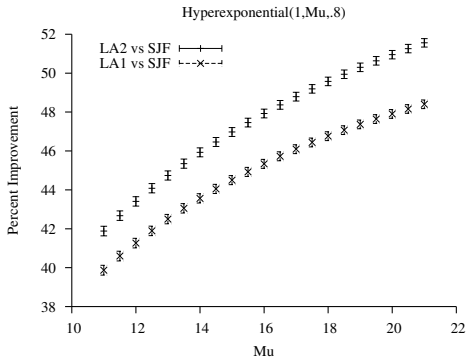

Figure 6: Improvement of LA Algorithms over SJF

Through other simulations using 25 jobs, we found that the amount of improvement was closely related the the variance of the distribution used to generate service times. Intuitively, the variance is a rough measure of the likelihood that a short job arrives after a long job. For lookahead to be beneficial, the algorithm should hold long jobs in the queue to wait for short jobs to arrive.

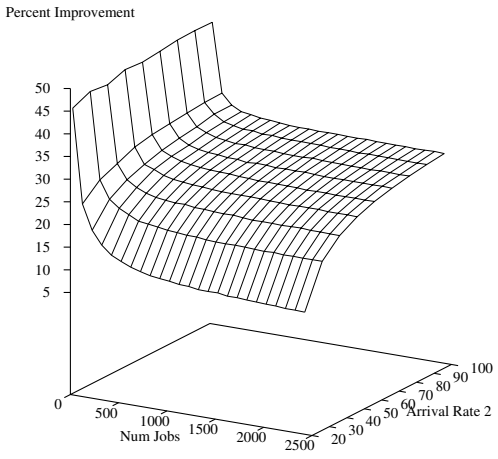

Figure 7: Improvement of LA1 over SJF

Since using only 25 jobs does not give a feel for the long-term behavior of these algorithms, we also considered arrival times drawn from a Markovian Modulated Poisson Process (MMPP), where the arrival rate toggles between two rates. In essence, we use the same arrival rate as the first simulations to allow a "burst" of jobs to arrive. Then the arrival rate slows so that jobs only arrive intermittently, and the machine has time to process waiting jobs. Therefore, we achieve alternating periods of congestion and idleness. As Figures 7 and 8 show, the resulting improvement over the SJF algorithm is consis-

tent with our short simulation results from Figure 6.

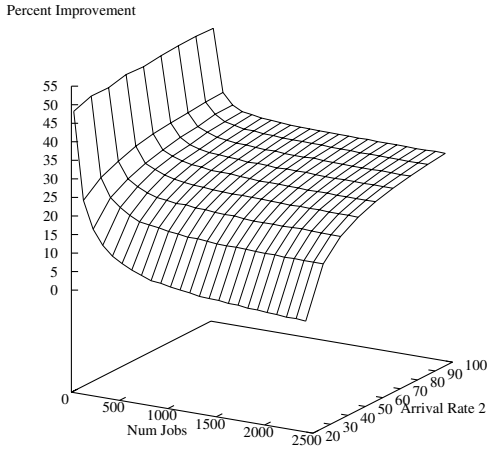

Figure 8: Improvement of LA2 over SJF

One other observation concerns the relative performance of the LA1 and LA2 algorithms. As expected, we see that the LA2 algorithm produces better schedules than the LA1 algorithm, and both lookahead algorithms perform better than the SJF algorithm. However, the amount of improvement of the LA2 algorithm over the LA1 algorithm is not significant. That is, most of the improvement comes from LA1 over SJF and the inclusion of a second lookahead job only adds a small amount of additional improvement.

5 Conclusions

From these experiments, we gained significant insight into the behavior of lookahead algorithms. As expected, lookahead algorithms produce superior schedules in many situations. We found that one requirement for improvement is a mixture of short and long jobs. In general, a high variance of the service times indicates that this mixture is present.

A second requirement for improvement is small to medium-sized blocks of jobs. When a large number of jobs arrive such that they can be scheduled without any idle periods, using lookahead is detrimental. This is especially true when decisions to wait are made at the beginning of these long blocks. Because the lookahead algorithm makes a local decision by considering only the known jobs, it makes a poor decision that delays not only the known jobs, but also many jobs that arrive in the future. Therefore, what appears to be a good decision immediately turns out to be a very poor decision in the future.

Even when these requirements for improvement are met, we were surprised to discover that while algorithms with more lookahead produce better results, the amount of improvement is not significantly larger than a Lookahead-1 algorithm. This is important when we consider the effort required to develop these algorithms. In addition, algorithms with more than a single lookahead job have linear running times, or worse, making them poor candidates for real-time systems.

Taking our design results together with our performance results, we conclude that lookahead is an important factor when trying to minimize the average wait time. However, because of the limited increase in schedule quality, the increased running time, and the additional design-time required, we conclude that Lookahead-1 algorithms should be utilized when possible, but Lookahead-k algorithms for $k > 1$ should not.

6 Future Work

Although our results indicate that the variance of service times indicates the amount of improvement, it is not a perfect indicator. We are curious whether there is a distribution statistic that more accurately predicts the amount of improvement. To study this idea, we plan to consider the skewness of each distribution, defined as $\frac{\mu_3}{\mu_2^{3/2}}$, where μ_2 and μ_3 are the second and third moments of the distribution.

In addition, we plan to consider other optimality criteria for scheduling. All of our results are based on minimizing the average wait time or total completion time. We plan to consider other criteria, such as average lateness. With this metric, each job has a deadline, and we attempt to minimize average lateness.

References

[1] B. Coleman and W. Mao. Lookahead scheduling in a real-time context. In *Proceedings of the 6th International Conference on Computer Science and Informatics*, 2002.

[2] B. Coleman and W. Mao. Lookahead scheduling of unrelated machines. In *Proceedings of the 7th International Conference On Computer Science and Informatics*, 2003.

[3] P. Keskinocak and W. Mao. Online algorithms: How much is it worth to know the futre? A survey in preparation.

[4] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[5] W. Mao and R. K. Kincaid. A look-ahead heuristic for scheduling jobs with release dates on a single machine. *Computers Operations Research*, 21(10):1041–1050, 1994.