Integrating Scheduling and Queueing For Dynamic Scheduling Problems

Student: Daria Terekhov   Supervisor: J. Christopher Beck   Collaborator: Tony T. Tran
Department of Mechanical & Industrial Engineering
University of Toronto, Toronto, Ontario, Canada
{dterekho,jcb,tran}@mie.utoronto.ca

Abstract

Within the scheduling community, there has recently been an increasing interest in modelling and solving of scheduling problems in dynamic and uncertain environments. Such research does not utilize the developments of queueing theory, an area that has examined problems of similar nature. This paper is initial work towards the goal of integrating queueing theory and scheduling. Specifically, by combining concepts and problem settings from queueing theory and scheduling, we obtain new insights about dynamic scheduling.

Introduction

In a typical real-world scheduling problem, the set of jobs changes dynamically over time and the durations of jobs are affected by various types of uncertainty. The goal is to determine how the available machine processing time is to be allocated among competing requests with the objective of optimizing the performance of the system. Methods for solving dynamic scheduling problems, in general, must address the combinatorial structure inherent in most interesting scheduling problems, the uncertainty about the evolution of the system, and the effect of current decisions on the future. The quality of the resulting schedule should be close to that of the schedule that could have been constructed if all of the uncertainty had been revealed *a priori*. Clearly, this is a difficult task, because to make a decision, one can only use the information that is known with certainty at that particular decision point and the stochastic properties of future events. The effect of the decision on both short-run and long-run system performance also has to be considered.

Queueing theory has taken the viewpoint that, since it is impossible to create an optimal schedule for every single sample path in the evolution of the system, one should aim to achieve optimal performance in some *probabilistic* sense (e.g., in expectation) over a long time horizon. This goal could be attained by construction of a policy based on the global stochastic properties of the system. For example, a policy could specify the start time assignment decisions to be made each time a new job arrives. However, the schedule obtained from such a policy, while being of good quality probabilistically, may in reality be far from optimal for the particular realization of uncertainty that occurs. Moreover, queueing theory generally studies systems with simple combinatorics, as such systems are more amenable to rigorous analysis of their stochastic properties.

In the scheduling community, a dynamic scheduling problem is generally viewed as a collection of linked static problems. Adopting this view makes the abundance of algorithms developed for static scheduling problems directly applicable as these algorithms can deal with complex combinatorics and can optimize the quality of each static subproblem schedule. However, these approaches tend to overlook the long-run performance and the stochastic properties of the system, with a notable exception being the work of Branke and Mattfeld (2002).

Thus, queueing theory and scheduling have differing views of dynamic problems. Our goal is to investigate whether these two viewpoints can be combined in way that would be beneficial for dynamic scheduling. This paper is an initial step towards this goal: we combine problem settings and ideas from queueing theory and scheduling and, as a result, obtain novel insights about dynamic scheduling.

Problem Formulations

We study two related dynamic scheduling environments: a two-machine flow shop and a polling system with a flow shop-like server.

In a two-machine dynamic flow shop, jobs arrive stochastically over time and must be processed first on machine 1 and then on machine 2. The time of a job arrival and the processing times on both machines are not known until the instant the arrival occurs. Both machines are of unary capacity, and preemptions are not allowed. The buffers in front of machine 1 and machine 2 are assumed to be of infinite size. The goal of the problem is to assign start times to all jobs on the two machines so that the mean job flow time is minimized. A job's flow time is defined as the difference between the job's completion time on the second machine and its arrival time to the system.

Polling systems are also dynamic environments: jobs arrive at random points in time and, under one set of queueing assumptions, the processing times of these jobs are not known until their arrival. Polling systems usually consist of a single server and multiple job classes. Each arriving job has to wait for processing in the queue corresponding to its specific class. As in much of polling system research (Levy

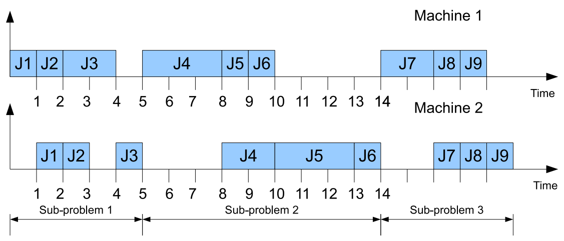

Figure 1: Polling system with 3 sub-problems (each corresponding to a queue visit) and $n = 3$ jobs per sub-problem.

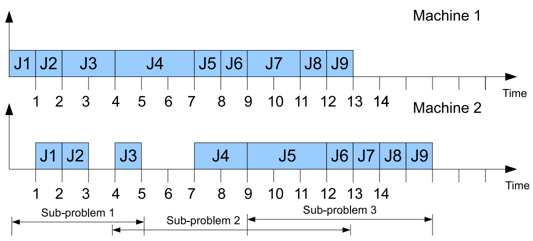

Figure 2: Dynamic flow shop with 3 sub-problems and $n = 3$ jobs per sub-problem.

and Sidi 1990; Takagi 2000), we assume that the server visits (i.e., *polls*) the queues in a cyclic manner and, upon each visit to a queue, employs a gated discipline, processing all jobs that are present at the time of its arrival. Preemptions are not allowed.

Unlike in standard queueing models, we assume that the server of the polling model consists of two machines in series (a two-machine flow shop). We consider the problem of scheduling jobs for processing by the two-machine server with the objective of optimizing the mean flow time. Due to the gated policy assumption, upon arrival to a queue, the server is faced with a static two-machine flow shop scheduling problem.

Scheduling in Polling Systems and Dynamic Flow Shops

We solve the two problems described above via periodic scheduling strategies: at a given point in time, we review the jobs present in the system or a particular queue of the system, create a schedule for these jobs, and once this schedule is executed, review the status of the system again. We examine two queueing-based and two scheduling-based methods for creating each sub-schedule in both polling systems and dynamic flow shops. The time at which scheduling happens is different in the two environments. In the polling system, the server switches from one queue to the next only after all of the jobs present upon its arrival to the queue have been processed on *both* machines. The start of every new sub-problem is equal to the completion time of the last job in the previous sub-problem, as shown in Figure 1. In a dynamic flow shop without a polling structure, such an assumption is unreasonable since it would create unnecessary idle time on machine 1. Thus, in a dynamic flow shop, scheduling is performed once all jobs of the previous sub-problem have been processed on the *first* machine, as illustrated in Figure 2. This difference plays a key role in further analysis.

Methods for Solving Static Sub-problems

To our knowledge, policies for the polling system discussed in this paper have not been examined in queueing theory. We are also unaware of any queueing policy that has been proven to be optimal, even in the expected sense, for a dynamic two-machine flow shop under the assumptions we make above. Thus, we consider two queueing approaches for which theoretical results are available for systems related to ours. Specifically, the two queueing policies that we use to solve each sub-problem are first-come, first-served (*FCFS*) and shortest total processing time first (SPT_{sum}).

Under *FCFS*, the jobs are processed in non-decreasing order of their arrival times to the queue. Towsley and Baccelli (1991) show that *FCFS* achieves the smallest expected flow time in a two-machine dynamic flow shop in the class of work-conserving, non-preemptive policies that do not use processing time information.

Employing SPT_{sum} means that all jobs present in the queue at the time when the schedule is constructed are processed in non-decreasing order of the *sum* of their durations on machine 1 and machine 2. This policy choice is motivated by the fact that, in the case when the server is a single unary resource, shortest processing time first minimizes the expected flow time in queueing systems with a single queue or with a polling structure under a cyclic, gated service discipline (Wierman, Winands, and Boxma 2007).

From the perspective of scheduling, each static queue sub-problem presents an opportunity for optimization. Since minimizing flow time under the above assumptions is equivalent to minimizing the total completion time, a natural choice of objective to be optimized in a sub-problem is the sum of completion times of activities on the second machine. We refer to this model as the *completionTime* model. Optimizing the total completion time will lead to the best *short-run* performance but, given the dynamics of the system, may not result in the best mean flow time in the *long-run*. The *completionTime* model also has a computational disadvantage: minimizing the sum of completion times in a two-machine flow shop is NP-hard (Pinedo 2003).

The fourth method we employ is motivated by a combination of queueing-based reasoning and the fact that scheduling methods can be used to optimize local queue performance. Specifically, suppose that we minimize the makespan for the set of jobs present in the queue, with makespan being defined as the difference between the end time of the job that is scheduled in the last position on machine 2 and the start time of the job that is scheduled in the first position on machine 1. Minimizing makespan may lead to a schedule with a sub-optimal mean flow time for the sub-problem, since minimizing mean flow time is not equivalent to minimizing makespan. However, the minimum makespan schedule may allow jobs in subsequent sub-problems to start

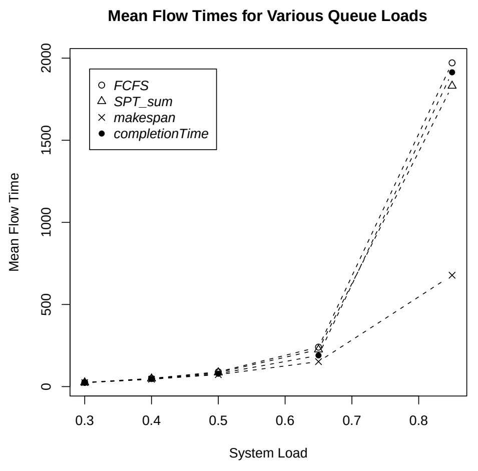

Figure 3: Mean flow times in a polling system with a two-machine flow shop server for *FCFS*, SPT_{sum}, *completionTime* and *makespan* models as the system load varies.

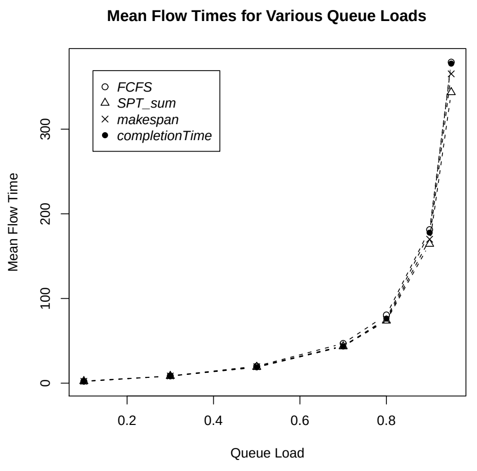

Figure 4: Mean flow times in a dynamic two-machine flow shop for *FCFS*, SPT_{sum}, *completionTime* and *makespan* models as the system load varies.

earlier than under the *completionTime* approach. Earlier start times for all jobs would imply lower total completion times for future sub-problems and, therefore, better *long-run* performance. Moreover, the optimal makespan schedule for a static two-machine flow shop can be found using a polynomial-time algorithm – Johnson's rule (Conway, Maxwell, and Miller 1967). The scheduling approach that uses Johnson's rule to solve each sub-problem will be referred to as the *makespan* approach.

Polling System

In order to understand the performance of the four methods in a polling system with a flow shop server, we simulated five symmetrical systems with $N = 5$ queues and with an arrival rate of $\lambda = 1$ for every queue. In each system, the processing times are exponentially distributed with the same means on both machines. As the mean processing times increase in the different experimental conditions, the load on the system, defined as $N\rho$, where $\rho = \lambda/\mu$ and $1/\mu$ is the mean processing time both on machine 1 and 2, increases. Thus, to observe the variation in performance as the load changes, we considered systems with $\mu \in \{16, 12, 10, 8, 6\}$.

Figure 3 shows the mean flow times for the *completionTime* model with a 1-second time limit, *FCFS*, SPT_{sum} and the *makespan* model as the system load increases. Every point in this figure represents the mean flow time over 100 problem instances, each consisting of 25000 jobs (5000 per queue). The figure shows that, for loads of 0.5 or less, the performance of the four methods is almost identical. For loads greater than 0.5, *makespan* achieves the lowest mean flow times. Moreover, the difference in performance between *makespan* and the other methods grows as the load increases. *FCFS* results in the highest flow times, while the relative performance of *completionTime* and SPT_{sum} changes as the loads change. For loads of 0.65 or less, the *completionTime* model has a slight advantage over SPT_{sum}; when the load increases to 0.85, the reverse is true.

Dynamic Flow Shop

To evaluate the performance of our four methods in a dynamic flow shop, we considered a system with exponential inter-arrival times and exponential processing times with the same means on both machines, with the arrival distribution and processing time distributions being independent. We fixed the inter-arrival rate, λ, to 10, and varied the load on the system by changing the rates of the processing time distributions from 100 to 10.53. The results of these experiments are shown in Figure 4. Each point in the figure represents the mean flow time over 100 instances of 55000 jobs each.

Figure 4 shows that the relative performance of the four methods is different without a polling structure. Although there is no significant difference between the methods, SPT_{sum} is slightly better than the rest, especially when the load of the system is at or above 0.9. The mean flow times obtained by the *completionTime* model are comparable to those of *FCFS*, with the *makespan* model being only marginally better.

Polling System vs. Dynamic Flow Shop

We can view the global system schedule for each environment as a sequence of linked sub-schedules. In this global schedule, every job j has completion time C_j. We denote by C_j^0 the end time of job j under the assumption that the first job of the sub-problem to which j belongs starts at time 0. C_j is equal to C_j^0 *shifted* forward in time by an amount that is a function of the characteristics of previous sub-problems. This function explains the differences in the relative performance of the four methods shown in Figures 3 and 4.

Specifically, in a polling system, C_j^0 is always shifted by the sum of the makespans of the preceding sub-problems. Thus, as the number of time periods in the overall problem grows, so does the saving obtained from applying to each sub-problem the schedule with the minimum makespan rather than with the minimum total completion time. In a dynamic flow shop, the value of the shift is dependent on the sum of machine 1 processing times and the idle times occurring before each job in addition to the makespans of sub-problems. Unlike in the polling model, the shifts of jobs resulting from the *makespan* approach are not much smaller than from the other methods, while the total completion times could be significantly greater. In fact, in a dynamic flow shop, the model that finds the schedule with the best total completion time for every sub-problem will achieve the smallest overall mean flow time. In the results of Figure 4, the model that finds the best total completion time schedules for the sub-problems is SPT_{sum}: the *completionTime* model cannot find good quality solutions due to the 1-second run-time limit. Formal support for these results is given in the work of Terekhov, Tran and Beck (2010).

In the polling system, a conflict exists between short-run and long-run objectives. That is, minimization of the sum of completion times at each scheduling point results in the best performance for the current sub-problem, but leads to poor performance in the long-run. Minimization of the makespan, on the other hand, leads to sub-optimal sum of completion time values for each sub-problem, but results in significant overall mean flow time improvements. In the dynamic two-machine flow shop there is no conflict between the ways in which we have attempted to optimize long-run and short-run objectives: minimizing the total completion time of each sub-problem also leads to better mean flow time in the long-run than does minimizing makespan.

Future Work

We have shown that integrating queueing theory and scheduling on a conceptual level can lead to a better understanding of dynamic scheduling. In subsequent work, we would like to combine queueing and scheduling on an algorithmic level, with the goal being the development of more effective methods for scheduling in dynamic environments. The polling system described above and a multi-class dynamic flow shop can serve as a basis for such research.

The queueing theory methods that we would like to investigate are based on fluid models, which treat jobs as a continuous fluid rather than as discrete entities and focus on system dynamics. These models, expressed as linear programs and solved periodically, can give the proportion of time that should be allocated to each class within the next scheduling period (Atkins and Chen 1995). Fluid models have a global view of the system: they are based on properties of classes rather than individual jobs and provide high level guidance of how machine time should be used rather than exact sequencing decisions. As a consequence, they are likely to optimize long-run objectives but may perform poorly for a given short time horizon.

Since scheduling methods, in contrast, achieve good short-run performance but may be myopic, combining a scheduling method with a fluid model-type approach appears promising. For example, we can use a fluid model to determine the percentage of machine time within the next period that should be allocated to each class, then select and schedule the first l_k jobs from each class k such that the sum of the processing times of these jobs would be smaller than or equal to the amount of time that is allocated. We would like to empirically show whether such hybrids outperform pure queueing or pure scheduling approaches as well as determine the reasons for differences in the relative perfomance of these methods.

Conclusion

In this paper, we have considered the problem of minimizing the mean flow time in two-machine dynamic flow shop environments from the perspectives of queueing theory and scheduling. Queueing theory approaches tend to optimize the long-run expected performance of the system, while scheduling focuses on actual short-run objectives. Taking both viewpoints into account, we have found that, in a polling system, a method which incrementally constructs the schedule by minimizing the makespan of each subsequent set of jobs results in a better mean flow time than does a method that minimizes the mean flow time itself. In a dynamic flow shop, the reverse is true. Our work shows the importance of considering both short-run and long-run objectives in dynamic scheduling, and demonstrates that combining ideas from scheduling and queueing theory can lead to new insights about dynamic scheduling.

Acknowledgments This work was supported by the Natural Sciences and Engineering Research Council of Canada.

References

Atkins, D., and Chen, H. 1995. Performance evaluation of scheduling control of queueing networks: Fluid model heuristics. *Queueing Systems* 21:391–413.

Branke, J., and Mattfeld, D. C. 2002. Anticipatory scheduling for dynamic job shop problems. In *Proceedings of the ICAPS'02 Workshop on Online Planning and Scheduling*, 3–10.

Conway, R. W.; Maxwell, W. L.; and Miller, L. W. 1967. *Theory of Scheduling*. Addison-Wesley.

Levy, H., and Sidi, M. 1990. Polling systems: Applications, modeling, and optimization. *IEEE Transactions on Communications* 38(10):150–1760.

Pinedo, M. 2003. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, 2 edition.

Takagi, H. 2000. Analysis and application of polling models. In *Performance Evaluation: Origins and Directions*, Lecture Notes in Computer Science. Springer. 423–442.

Terekhov, D.; Tran, T. T.; and Beck, J. 2010. Investigating two-machine dynamic flow shops based on queueing and scheduling. In *Proceedings of ICAPS'10 Workshop on Planning and Scheduling Under Uncertainty*. To Appear.

Towsley, D., and Baccelli, F. 1991. Comparisons of service disciplines in a tandem queueing network with real time constraints. *Operations Research Letters* 10(1):49–55.

Wierman, A.; Winands, E.; and Boxma, O. 2007. Scheduling in polling systems. *Performance Evaluation* 64:1009–1028.