Long-run Stability in Dynamic Scheduling

Daria Terekhov^a, Tony T. Tran^a, Douglas G. Down^b, J. Christopher Beck^a
^{a}Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario, Canada
{dterekho,tran,jcb}@mie.utoronto.ca
^{b}Department of Computing and Software, McMaster University, Canada
downd@mcmaster.ca

Abstract

Stability analysis consists of identifying conditions under which the number of jobs in a system is guaranteed to remain bounded over time. To date, such long-run performance guarantees have not been available for periodic approaches to dynamic scheduling problems. However, stability has been extensively studied in queueing theory. In this paper, we introduce stability to the dynamic scheduling literature and demonstrate that stability guarantees can be obtained for methods that build the schedule for a dynamic problem by periodically solving static deterministic sub-problems. Specifically, we analyze the stability of two dynamic environments: a two-machine flow shop, which has received significant attention in scheduling research, and a polling system with a flow-shop server, an extension of systems typically considered in queueing. We demonstrate that, among stable policies, methods based on periodic optimization of static schedules may achieve better mean flow times than traditional queueing approaches.

Introduction

In a typical real-world scheduling problem, the set of jobs changes dynamically over time and their processing times 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. The definition of "good" or optimal performance is dependent on the time horizon considered. In particular, managers of production facilities and supply chains are interested in achieving long-run, strategic goals as well as short-run, operational ones.

Many methods for dynamic scheduling developed within the scheduling community are based on the idea of periodically reviewing the status of a system and solving a static scheduling problem for the jobs present at that time (Bidot et al. 2009). The advantage of this approach is that it can utilize the abundance of methods developed for static deterministic scheduling, and can therefore effectively deal with short-run objectives. The question of whether such an approach is able to achieve long-run goals is generally not addressed in the dynamic scheduling literature. Queueing theory, on the other hand, focuses on analysis of long-run performance of dispatching-type scheduling policies. In this paper, we utilize ideas from queueing theory in order to gain an understanding of how well periodic scheduling approaches perform with respect to long-run objectives.

One of the most important measures of long-run performance considered in queueing theory is *stability*. Informally, a stable system is one in which the number of jobs is guaranteed to be bounded over time.[1] Knowledge of whether a system is stable for a given job arrival rate, processing rate and scheduling policy is essential for practical applications. For instance, processes in semi-conductor manufacturing are frequently modelled as reentrant lines (Kumar 1994), and Dai and Weiss (1996, p. 27) state that "stability is a first issue one needs to address if one wishes to study optimal or near-optimal scheduling of a reentrant line." Thus, a proof of stability can be viewed as a necessary, but not a sufficient, step for obtaining an accurate understanding of a scheduling policy's long-run performance. After a policy is proven stable, it is important to evaluate its effectiveness with respect to more detailed long-run performance metrics, such as mean flow time. While stability can be formally proven, evaluation of other measures usually requires simulation.

In this paper, we introduce long-run stability to the combinatorial scheduling literature. We do this by looking at two related systems: a novel polling system that is an extension of systems traditionally examined in queueing theory and a dynamic two-machine flow shop, which is important in scheduling research. In the dynamic flow shop, stability of the first-come, first-served policy can be proven using the fluid model methodology of Dai (1995), while stability of a periodic scheduling approach is shown using a sample path argument. In the polling system, stability of both policies is proven using fluid model methods. We further experimentally evaluate the performance of these policies with respect to mean flow time. Our results suggest that the periodic approach that optimizes makespan at each decision time point provides the best mean flow time for the polling system but

[1] In the scheduling literature, a predictive schedule is called *stable* if the schedule executed online is close to the planned schedule (Bidot et al. 2009). We do not use this definition here, instead adopting the meaning of stability from queueing theory.

not for the dynamic flow shop. These results also provide several insights into the behaviour of periodic scheduling policies in dynamic environments.

Our paper has three main contributions. Firstly, we introduce long-run stability to combinatorial scheduling and prove that a periodic scheduling approach based on makespan minimization is stable in the dynamic flow shop environment. Secondly, we empirically evaluate the performance of various scheduling methods with respect to mean flow time. This analysis demonstrates that periodic scheduling approaches may perform better than traditional queueing methods in optimizing long-run performance. Finally, this work develops a new link between queueing and scheduling, showing that combining problem settings and ideas from these two areas can lead to novel insights about scheduling in dynamic environments.

This paper is organized as follows. The first section talks about the motivations for combining queueing and scheduling ideas, followed by an introduction to the concept of stability. Next, problem settings and scheduling methods are presented. In the subsequent two sections, stability and mean flow time of these methods are analyzed both theoretically and empirically. We conclude with a discussion of related research and ideas for future work.

Motivations

The problem of scheduling in a dynamic environment involves a long time horizon and has to somehow deal with all the possible realizations of the job arrival process and of the job characteristics. The ultimate goal in solving this problem is to construct a schedule that would be optimal for the specific realization that actually occurs. The quality 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 use only the information that is known with certainty at that particular decision point and the stochastic properties of scenarios that can occur in the future. In addition, the effect of the decision on both short-run and long-run performance has to be considered. To deal with such problems, queueing theory and scheduling have adopted different approaches.

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 how start time decisions should be made each time a new job arrives or a job is completed. However, the schedule resulting from such a policy, while being of good quality in expectation, may 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. This viewpoint implies that methods developed for static scheduling problems become directly applicable to dynamic ones. Such methods can effectively deal with complex combinatorics and can optimize the quality of the schedules for each static sub-problem. However, they tend to overlook the long-run performance and the stochastic properties of the system, with notable exceptions being the work on anticipatory scheduling (Branke and Mattfeld 2002) and online stochastic combinatorial optimization (Van Hentenryck and Bent 2006).

Thus, queueing theory and scheduling have differing views on dynamic problems. In this work, we leverage these differences in order to gain a better understanding of the long-run performance of dynamic scheduling methods.

Stability

Informally, a system is *stable* if its queues remain bounded over time. In queueing theory, establishing the stability of a system has always been considered a precursor to more detailed analysis (Kumar and Meyn 1995). In early queueing work, it was believed that, to ensure a stable system, it was necessary and sufficient to have the load of each machine, defined as the ratio of the arrival rate to the processing rate, be strictly less than 1 (Dai and Weiss 1996). However, a series of papers (Lu and Kumar 1991; Bramson 1994; Seidman 1994) provided counter-examples which demonstrated that this load condition is not always sufficient for stability (Dai and Weiss 1996). In particular, the stability of a system is dependent on the scheduling policy it employs: for a given set of problem parameters (arrival and processing rates), one policy may stabilize the system while another might not.

Formally, a system operating under a particular queueing discipline is stable if the Markov process that describes the dynamics of the system is *positive Harris recurrent* (Dai 1995). Positive Harris recurrence implies the existence of a unique stationary distribution. Due to space constraints and the considerable notation required, we do not formally define positive Harris recurrence here, but instead refer the reader to the papers by Dai (1995), Dai and Meyn (1995) and Bramson (2008). In the case when the state space of the Markov process is countable[2] and all states communicate, positive Harris recurrence is equivalent to the more widely-known concept of positive recurrence (Bramson 2008). The Markov chain is positive recurrent if every state i is positive recurrent: the probability that the process starting in state i will ever return to i is 1, and the expected time to return to this state is finite (Ross 2003). In particular, this property guarantees that the system will empty.

The next section provides detailed descriptions of two problem settings. We subsequently analyze the stability of the two systems under different scheduling methods.

Problem Settings

Consider two dynamic environments: a two-machine flow shop and a polling system with a flow shop-like server.

[2] This process is referred to as a *continuous-parameter Markov chain* in the book by (Gross and Harris 1998), to which the reader is referred for an introduction to queueing theory.

Dynamic Flow Shop

In a two-machine dynamic flow shop, jobs arrive according to some stochastic process over time and must be processed first on machine 1 and then on machine 2. We assume that the inter-arrival distribution is general with rate λ, while the processing time distributions for machine 1 and 2 are general with rates μ_1 and μ_2, respectively. Thus, the load for machine 1 is $\rho_1 = \frac{\lambda}{\mu_1}$ and the load for machine 2 is $\rho_2 = \frac{\lambda}{\mu_2}$. Both ρ_1 and ρ_2 are assumed to be less than 1, as these are known necessary conditions for stability.

Processing times of a job on both machines become known at the instant of its arrival to the system. Both machines are of unary capacity. Preemptions are not allowed. The queues in front of machine 1 and machine 2 are both assumed to be of infinite size. In the queueing literature, this system is known as a tandem queue.

Polling 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 become known upon their arrival. Polling systems usually consist of a single server and multiple job types. Each arriving job has to wait for processing in the queue corresponding to its specific type. The server visits these queues in a particular order and serves a subset of the jobs during each visit. A variety of policies for the order in which the queues should be visited and for deciding the number of jobs that should be served on each visit have been considered in the literature (Levy and Sidi 1990; Takagi 2000). We assume that the server visits the queues in a cyclic manner. During each visit, the server employs a gated discipline: it processes all jobs that are present at the time of the server's arrival to the given queue. Preemptions are not allowed.

Unlike standard polling models, ours assumes that the server consists of two machines in series (a two-machine flow shop). This is one of the simplest ways to introduce combinatorial scheduling into a queueing model, and it provides us with a setting in which ideas from the two areas can be explored. Browne and Weiss (1992) considered a polling system with a server consisting of coupled parallel machines, yet polling models with coupled machines in series have not been studied. Coupling occurs in, e.g., polling systems where each queue belongs to a customer who gets exclusive rights to the resources once its service is started.

We assume that the inter-arrival distribution for each queue b is general with rate λ_b. The processing time distributions are general with rates μ_{1b} and μ_{2b} for machines 1 and 2, respectively. We assume that $\rho_{1b} = \frac{\lambda_b}{\mu_{1b}} < 1$ and $\rho_{2b} = \frac{\lambda_b}{\mu_{2b}} < 1$ for all b, as these assumptions are necessary for stability. Due to the assumption of a gated policy, upon arrival to a queue, the server is faced with a static two-machine flow shop scheduling problem.

The goal, in both systems, is to assign start times to all jobs in a way that ensures stability and minimizes the mean job flow time over a long time horizon. 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.

Scheduling in Polling Systems and Dynamic Flow Shops

We can schedule jobs in the systems 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, however, 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 time of every new sub-problem is equal to the completion time of the last job in the previous sub-problem, as shown in Example 1 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 by Example 2 in Figure 2. This difference may seem minor, but as we show below it has a significant impact on the analysis of the two systems.

Methods for Solving Static Sub-problems

To our knowledge, policies for the polling system discussed in this paper have not been examined in the queueing literature. We are also unaware of any queueing policy that has been proven to be optimal for the flow time objective, even in the expected sense, for a dynamic two-machine flow shop under our assumptions. Thus, we consider two queueing approaches for which theoretical results are available for systems related to ours. Specifically, the two queueing policies 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). Wierman et al. (2007) show that such a policy can outperform *FCFS* by as much as 15% in a gated, cyclic polling system.

From the perspective of scheduling, each static queue sub-problem presents an opportunity for optimization. Since

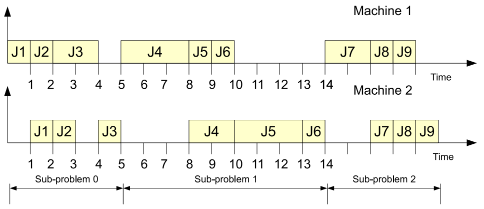

Figure 1: Polling system with 3 sub-problems (each corresponding to a queue visit) and 3 jobs per sub-problem. (Example 1)

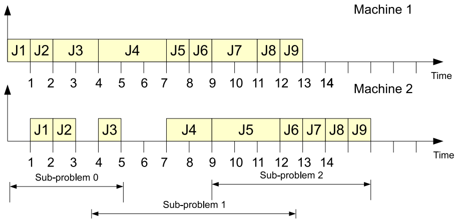

Figure 2: Dynamic flow shop with 3 sub-problems and 3 jobs per sub-problem. The start of a sub-problem is the start of a set of jobs on machine 1, the end of a sub-problem is the end of a set of jobs on machine 2. (Example 2)

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, prior to the work presented in this paper, it was unclear how this method would perform with respect to *long-run* objectives. 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 earlier than under the *completionTime* approach. Earlier start times for all jobs would imply lower total completion times for all 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).

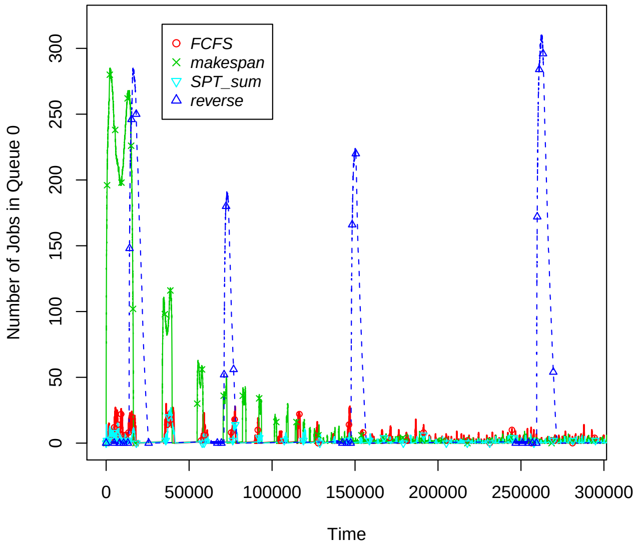

Figure 3: Number of Jobs at Machine 2 in Queue 0 Over Time for Example 3.

Johnson's rule divides jobs into two sets: set I consists of all jobs whose processing time on machine 1 is less than or equal to its processing time on machine 2, and set II consists of all the remaining jobs. Set I is processed before set II. Johnson's rule creates permutation schedules, that is, schedules in which the order of jobs is the same on both machines. Within set I, jobs are sequenced in non-decreasing order of the processing time on machine 1, while within set II, jobs are sequenced in non-increasing order of the processing times on machine 2. The scheduling approach that uses Johnson's rule to solve each sub-problem will be referred to as the *makespan* approach.

Stability of Dynamic Flow Shop Environments

Consider an instance of the polling system with 5 queues: at time 0, there are 1000 jobs in queue 0, and no jobs in the other queues. Jobs arrive to each queue according to (independent) Poisson processes with rate 1. Processing times are exponentially distributed with rate 6. Therefore, for each machine and a given queue, the load is $\frac{1}{6} < 1$, and the load on the system is $5(\frac{1}{6}) = 0.833 < 1$. We refer to this instance as Example 3. In Figure 3, we show the number of jobs in queue 0 over time for four policies: *FCFS*, SPT_{sum}, *makespan* and *reverse*. The *reverse* policy is a modification of Johnson's rule: set II is scheduled before set I (i.e., in reverse order as compared to Johnson's rule). In Figure 3, we see that all policies initially have a large number of jobs at machine 2 in queue 0 – this corresponds to the sub-problem containing the initial 1000 jobs. Under *FCFS*,

SPT_{sum} and *makespan*, there is little variation in the number of jobs at machine 2 after approximately 150,000 time units: the three policies keep the number of jobs below 25. For the *reverse* policy, on the other hand, the number of jobs in sub-problems following the first one grows over time. After slightly over 250,000 time units, the number of jobs in queue 0 is greater than in the initial sub-problem. The figure therefore suggests that the number of jobs in queue 0 for *FCFS*, SPT_{sum} and *makespan* will remain bounded, corresponding to stable behaviour; for the *reverse* policy, the increasing number of jobs in queue 0 suggests instability.

Figure 3 shows the behaviour of the policies on one particular realization of the evolution of the system. In order to understand whether the same behaviour will be observed for other realizations, formal analysis of stability is necessary.

Except for the stability of *FCFS* in a two-machine flow shop, the stability problems we address are different from those in the literature for two reasons. Firstly, our polling system with a two-machine flow-shop server can be viewed as a hybrid of two well-studied systems, a tandem queue and a polling system with a single-machine server. Secondly, for both the dynamic flow shop and the polling system, we consider scheduling policies which assume knowledge of the processing times and are based on the notion of periodic scheduling and makespan minimization. As far as we are aware, neither the stability of the hybrid system nor the stability of the periodic scheduling method with makespan minimization have been addressed in the queueing literature. In fact, stability analysis of scheduling policies based on processing time information is rare.

Additional Assumptions on the Distributions

We make additional assumptions on the inter-arrival and processing time distributions that are standard in the queueing literature dealing with stability analysis (Dai 1995):

- For each queue, the sequences of processing times for machine 1 and machine 2, and the sequence of inter-arrival times are independent and identically distributed sequences. They are also mutually independent.
- Mean processing times and mean inter-arrival times for all queues are finite.
- The inter-arrival times are unbounded and continuous.

These assumptions are satisfied by commonly used distributions such as the exponential distribution.

Dynamic Flow Shop

In the dynamic flow shop, the following proposition holds:

Proposition 1. *If* $\frac{\lambda}{\min\{\mu_1,\mu_2\}} < 1$*, then the tandem queue with the periodic* FCFS *policy is stable.*

This result follows from the fact that for the dynamic flow shop, the periodic *FCFS* policy is equivalent to the "standard", non-periodic implementation of *FCFS*, and the fact that, under our assumptions, the tandem queue is a generalized Jackson network. Stability of such networks under the condition that the load of each machine is strictly less than 1 has been proven previously by various methods, such as the fluid model methodology of Dai (1995).

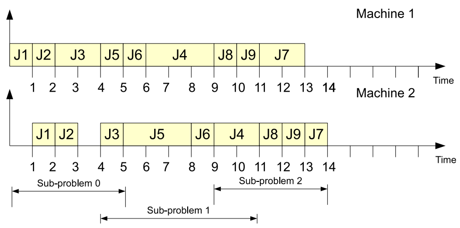

Figure 4: Schedule for *makespan* approach for Example 2 (the same problem instance as in Figure 2).

The same methodology can be used to show the stability of *FCFS* in an m-machine flow shop under the condition that $\frac{\lambda}{\min_{i\in\{1,\ldots,m\}}\{\mu_i\}} < 1$.

For the *makespan* policy, we can prove a result which holds for every sample path in the evolution of the system. Let t_n^* be the time point at which sub-problem n completes (on machine 2) under the *makespan* approach. Let v_n and v_n^π be the total processing time of all jobs completed by time t_n^* under the *makespan* policy and an arbitrary policy π, respectively. Following the queueing literature, we further refer to v_n and v_n^π as the *work* completed by time t_n^*.

Proposition 2. *The amount of work completed by t_n^* is maximized by the* makespan *policy. That is, $v_n \geq v_n^\pi$ for all n and all non-idling π.*

Proof. For any arbitrary non-idling policy π, v_n^π can be written as $v_n^{1,\pi} + v_n^{2,\pi}$, where $v_n^{1,\pi}$ is the amount of work completed by t_n^* on machine 1 and $v_n^{2,\pi}$ is the amount of work completed by t_n^* on machine 2. (In Figure 2, t_0^* is 5, $v_0^{1,\pi} = 5$ and $v_0^{2,\pi} = 3$.) For the *makespan* approach, we use the notation without the superscript π, i.e., $v_n = v_n^1 + v_n^2$. In the dynamic flow shop, the amount of work completed by t_n^* on machine 1 is the same for any non-idling policy. Thus, it remains to prove that $v_n^2 \geq v_n^{2,\pi}$. We prove this by induction.

Base Case: $v_0^2 \geq v_0^{2,\pi}$ since any policy other than *makespan* can complete the set of initial jobs only at the same time as *makespan* (t_0^*) or later.

Inductive Hypothesis: Assume the property is true for t_n^*, that is, $v_n^2 \geq v_n^{2,\pi}$.

Inductive Step: We need to show the same property for t_{n+1}^*, i.e., that $v_{n+1}^2 \geq v_{n+1}^{2,\pi}$. For the *makespan* approach, $v_{n+1}^2 = v_n^2 + \gamma((t_{n-1}^*, t_n^*])$, where $\gamma((t_{n-1}^*, t_n^*])$ is the total machine 2 workload that arrives in the time period $(t_{n-1}^*, t_n^*]$ and, therefore, is the workload that is processed in the sub-problem starting at t_n^* and ending at t_{n+1}^*.

By the induction hypothesis, we know that at t_n^*, $v_n^2 \geq v_n^{2,\pi}$. Thus, the amount of work processed on machine 2 by time t_{n+1}^* by policy π, $v_{n+1}^{2,\pi}$, equals the amount of work processed by π by time t_n^* plus some fraction of the difference in the amount of work completed by π and *makespan* by t_n^* plus some fraction of the amount of machine 2 work that arrives in $(t_{n-1}^*, t_n^*]$. Thus, $v_{n+1}^{2,\pi} \leq v_n^{2,\pi} + (v_n^2 - v_n^{2,\pi}) + \gamma((t_{n-1}^*, t_n^*]) = v_n^2 + \gamma((t_{n-1}^*, t_n^*]) = v_{n+1}^2$. □

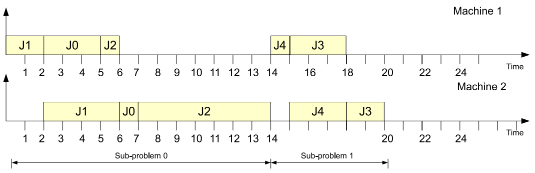

Figure 5: *completionTime* schedule for Example 4.

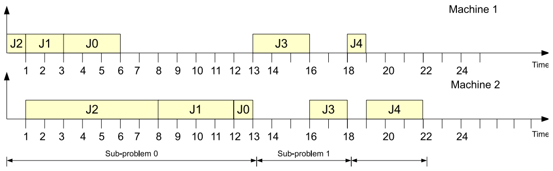

Figure 6: *makespan* schedule for Example 4.

For example, consider the schedules in Figures 2 and 4: $t_0^* = 5$, $t_1^* = 11$, $t_2^* = 14$, $v_2^2 = v_1^2 + \gamma((t_0^*, t_1^*]) = 9 + 3 = 12$, $v_2^{2,\pi} = 7 + (9 - 7) + 1 \leq 12$.

Corollary 1. *If* $\frac{\lambda}{\min\{\mu_1, \mu_2\}} < 1$ *then the tandem queue with the periodic* makespan *policy is stable.*

The corollary holds since we know that *FCFS* is stable and that, at every sub-problem completion time, the *makespan* approach has finished at least as much work as *FCFS*. Proposition 2 does not hold for an m-machine flow shop when $m > 2$. However, we conjecture that the corollary can be shown true for such an environment via a fluid model approach.

Polling System

The sample path result of Proposition 2 does not hold in the polling system. This can be illustrated by Figures 5 and 6, which show schedules for the *completionTime* and the *makespan* approaches for an example instance with jobs 0, 1, 2 available at time 0, job 3 arriving at time 13 and job 4 arriving at time 14 (Example 4). Under the *makespan* approach, sub-problem 0 completes at time 13 and so sub-problem 1 consists of job 3 only; when the *completionTime* method is used, sub-problem 0 completes at time 14, implying that sub-problem 1 consists of both jobs 3 and 4. Thus, by time $t_1^* = 18$, the total amount of work completed is 23 for the *makespan* model and 25 for the *completionTime* approach. However, the following proposition can be proven by applying the fluid model methodology of Dai (1995):

Proposition 3. *The given polling system is stable under both* FCFS *and* makespan *if* $\sum_{b=1}^{B} \frac{\lambda_b}{\min\{\mu_{1b}, \mu_{2b}\}} < 1$.

Proposition 3 also holds for a more general system in which the server is an m-machine flow shop, under the condition $\sum_{b=1}^{B} \frac{\lambda_b}{\min\{\mu_{1b}, \mu_{2b}, \ldots \mu_{mb}\}} < 1$. Similarly, the proposition and the proof methodology can be extended to a d-stage flexible flow-shop server with m machines at each stage. The reader is referred to the paper by Terekhov, Down, and Beck (2012) for details of the polling system stability proofs. We leave the investigation of the stability of SPT_{sum} and *completionTime* for future work.

In this section, we have considered one measure of long-run performance – stability. By using a counter-example, we have shown that, in the polling system, not all non-idling policies are stable. We have established the stability conditions for *FCFS* and *makespan* in both the dynamic flow shop and the polling system with a two-machine flow shop server. We now empirically examine the policies with respect to a more detailed long-run performance metric, mean flow time.

Mean Flow Time in Dynamic Flow Shop Environments

Proving stability for a scheduling policy provides a high-level performance guarantee: the queue lengths will remain bounded if the conditions on the parameter values necessary for the proof hold. However, different stable policies may perform differently with respect to more detailed objectives. Specifically, we would like to determine whether replacing a queueing policy by an optimization approach will result in any long-run benefits.

Below we present experiments comparing the performance of *FCFS*, SPT_{sum}, *makespan* and *completionTime* models for minimizing the mean flow time over a long time horizon in our two problem settings. The *completionTime* model was implemented via constraint programming in Ilog Scheduler 6.5 and uses the *completion* global constraint (Kovács and Beck 2011). The remaining methods were implemented using C++. In these experiments, the *completionTime* model was run with a time limit of 1 second per sub-problem in order to ensure reasonable run-times for our experiments. With a time limit, this approach does not always find the sub-problem schedule with the optimal sum of completion times. We do not take algorithm run-time into account in any of our results. Preliminary experiments with a time limit of 5 seconds as well as with uniformly-distributed processing times showed identical performance.

The experiments below demonstrate that for some problems, such as the dynamic flow shop, there is little difference in the performance of different stable policies at a finer level of detail, which in this case is evaluated in terms of the mean flow time. In other problems, such as the polling system with a flow-shop server, different stable policies lead to different performance with respect to more detailed metrics. More importantly, the polling system results demonstrate that periodic scheduling methods can perform better than queueing-based dispatching rules for optimization of long-run performance. However, the choice of objective function for each sub-problem, which is a proxy measure for the long-run objective, may not always be obvious. In the polling system, optimizing the mean flow time within each static sub-problem does not lead to the best mean flow time in the long-run.

Dynamic Flow Shop

To evaluate the performance of our four methods in a dynamic flow shop, we considered a system with exponentially distributed inter-arrival times and exponentially distributed

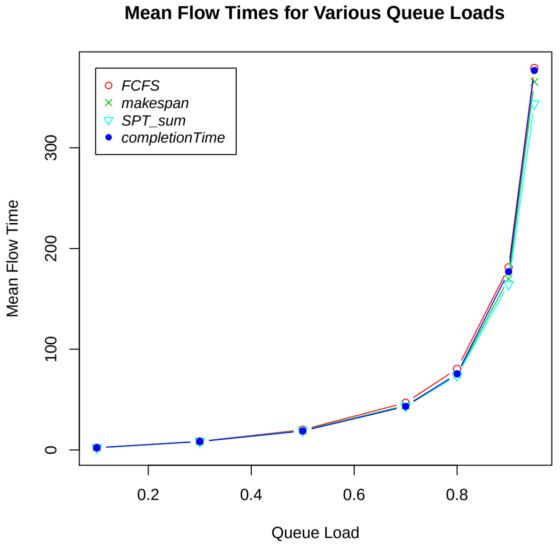

Figure 7: Mean flow times in a dynamic two-machine flow shop for *FCFS*, SPT_{sum}, *completionTime* and *makespan* models as the system load varies.

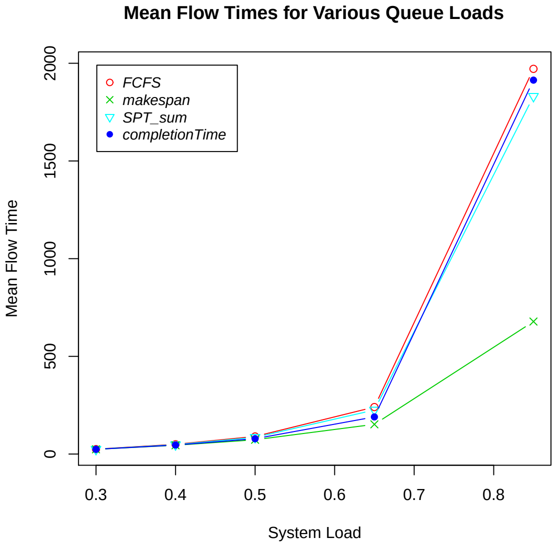

Figure 8: Mean flow times in a polling system with a two-machine flow shop server for *FCFS*, SPT_{sum}, *completionTime* and *makespan* models as the system load varies.

processing times with the same means on both machines. We fixed the 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 7. Each point in the figure represents the mean flow time over 100 instances of 55,000 jobs each.

Figure 7 shows that there is no significant difference between the methods. *completionTime* has a slight advantage over the others for loads of 0.7 and less, while SPT_{sum} is slightly better for loads of 0.8 and greater. The *makespan* model is marginally better than *FCFS* and *completionTime* at loads above 0.8.

SPT_{sum} is the best-performing model for loads above 0.8, when the static sub-problems become large. These observations are supported by the results of Xia, Shanthikumar, and Glynn (2000), who have shown that SPT_{sum} is asymptotically optimal for the static average completion time objective as the number of jobs in a two-machine flow shop increases. It was not clear, a-priori, that applying this method to each sub-problem would result in the best long-run behaviour for high loads. *FCFS*, on the other hand, is the worst performer over all loads, due to the fact that it is the only method that does not use processing time information. Further investigation has shown that *FCFS* finds sub-problem solutions that are of lower quality than those of the other methods, both in terms of their makespan and their total completion time values.

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 $B = 5$ queues and with a fixed arrival rate of $\lambda_b = 1$ for all b. In each system, the processing times are exponentially distributed with the same means on both machines for all queues (i.e., $\mu_{1b} = \mu_{2b} = \mu$ for all b). As the mean processing times increase in the different experimental conditions, the load on the system, defined as $\sum_{b=1}^{B} \max\{\rho_{1b}, \rho_{2b}\} = 5\rho_{1b}$, increases. Thus, in order to observe the variation in performance as the load changes, we considered systems with $\mu \in \{16, 12, 10, 8, 6\}$.

Figure 8 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 25,000 jobs (5000 per queue). The figure shows that, for loads of 0.5 or less, the performance of the four methods is almost identical, although *makespan* has a slight advantage over the remaining three approaches. 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, and SPT_{sum} performs better than *completionTime* for a load of 0.85. For an asymmetrical system consisting of five queues with different loads, the results matched the pattern of Figure 8.

Polling System vs. Dynamic Flow Shop

We can view the global system schedule for each of the two dynamic flow shop settings as a sequence of linked sub-schedules. In this global schedule, every job j completes at time C_j. C_j^0 denotes 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 (taking into account any unfinished machine 2 jobs from the previous sub-problem in the dynamic flow shop). C_j is equal to C_j^0 *shifted* forward in time by an amount that is a function of the previous sub-problems. This function explains the differences in the relative performance of the methods shown in Figures 7 and 8.

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, in contrast, the value of the shift is also a function of the sum of machine 1 processing times, which are independent of the schedule. Unlike in the polling model, the shifts of jobs resulting from the *makespan* approach are not much smaller than from the other methods. This analysis suggests that, 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. Thus, in our experiments, *completionTime* is the best for low and medium loads; SPT_{sum} is the best for high loads since it finds better quality schedules than *completionTime* due to the 1-second run-time limit. Additional analysis under the assumption that, in the dynamic flow shop, C_j^0 is calculated without taking into account unfinished machine 2 jobs is given in the workshop paper by 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 flow time performance for the current sub-problem, but leads to poor flow time performance in the long-run. Minimization of the makespan, in contrast, leads to sub-optimal flow 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.

Our results indicate that, in a polling system with a flow shop-like server, an approach that would find, for each sub-problem, the minimum makespan schedule with the best total completion time value would outperform the methods we presented here. This hybrid method would have the long-run focus of the *makespan* model, but would also be better than *makespan* in the short-run. In a dynamic flow shop, an approach that finds the optimal completion time schedule with the smallest makespan is of interest. However, our results imply that this approach would provide only marginal improvements over the *completionTime* model. Investigation of these hybrid methods is part of our future work.

Discussion

Our initial motivation for studying the integration of scheduling and queueing was that the two areas address similar problems in different ways. Specifically, the advantages of using queueing policies for scheduling in dynamic environments include guarantees of stability and good expected long-run performance, and the fact that there is no need for processing time information. In contrast, periodic scheduling approaches are able to optimize short-run objectives and provide a framework in which short-run constraints (e.g., due dates) can be easily incorporated. In this paper, we have shown that it is possible to establish stability of periodic scheduling approaches, and thus enhance the periodic scheduling framework with a guarantee of stability that has traditionally been available for queueing approaches only. Moreover, we have shown that in some systems, such as the polling system with a flow-shop server, periodic scheduling approaches may perform better than queueing-based approaches with respect to secondary long-run objectives.

In some applications it may be unreasonable to assume knowledge of job processing times, and queueing policies may be the only feasible approach to scheduling. However, if processing time information is available, it is worthwhile to use it to construct better schedules via periodic methods. If this information is stochastic, then a periodic approach may still be useful, though every sub-problem would become a stochastic problem instead of a deterministic one.

Related and Future Work

From the queueing perspective, our paper is related to the literature on tandem queues (Towsley and Baccelli 1991), polling systems (Takagi 2000; Levy and Sidi 1990; Browne and Weiss 1992) and stability (Dai 1995; Meyn and Down 1994; Down 1998; Georgiadis and Szpankowski 1992), while from the scheduling perspective, it is related to research on flow shop scheduling (Hejazi and Saghafian 2005; Della Croce, Narayan, and Tadei 1996; Xia, Shanthikumar, and Glynn 2000) and scheduling in dynamic and uncertain environments (Aytug et al. 2005; Bidot et al. 2009).

We see a variety of ways to integrate ideas from these areas in the future. For example, we would like to: look at the optimization of polling order from a scheduling viewpoint; extend our analysis to more general dynamic scheduling environments, such as job shops; compare the methods discussed in this paper with more complex queueing approaches; and develop hybrid queueing/scheduling methods.

Conclusion

We have considered two measures of long-run performance in dynamic flow shop environments. Motivated by the queueing theory literature, we have discussed stability of a traditional queueing policy, *FCFS*, and proved stability of a periodic scheduling approach based on optimizing makespan within each sub-problem. We have therefore demonstrated that theoretical long-run performance guarantees can be obtained for periodic scheduling methods. Secondly, we have considered a more detailed measure of long-run performance, mean flow time. We have shown that, among the stable policies, methods based on periodic optimization can outperform traditional queueing disciplines in some systems. 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.

References

Aytug, H.; Lawley, M.; McKay, K.; Mohan, S.; and Uzsoy, R. 2005. Executing production schedules in the face of uncertainties: A review and future directions. *European Journal of Operational Research* 161:86–110.

Bidot, J.; Vidal, T.; Laborie, P.; and Beck, J. C. 2009. A theoretic and practical framework for scheduling in a stochastic environment. *Journal of Scheduling* 12(3):315–344.

Bramson, M. 1994. Instability of FIFO queueing networks. *The Annals of Applied Probability* 414–431.

Bramson, M. 2008. Stability of queueing networks. *Probability Surveys* 5(169-345):1.

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

Browne, S., and Weiss, G. 1992. Dynamic priority rules when polling with multiple parallel servers. *Operations Research Letters* 12(3):129–137.

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

Dai, J. G., and Meyn, S. P. 1995. Stability and convergence of moments for multiclass queueing networks via fluid limit models. *IEEE Transactions on Automatic Control* 40(11):1889–1904.

Dai, J. G., and Weiss, G. 1996. Stability and instability of fluid models for reentrant lines. *Mathematics of Operations Research* 21(1):115–134.

Dai, J. G. 1995. On positive Harris recurrence of multiclass queueing networks: A unified approach via fluid limit models. *The Annals of Applied Probability* 5(1):49–77.

Della Croce, F.; Narayan, V.; and Tadei, R. 1996. The two-machine total completion time flow shop problem. *European Journal of Operational Research* 90(2):227–237.

Down, D. 1998. On the stability of polling models with multiple servers. *Journal of Applied Probability* 35(4):925–935.

Georgiadis, L., and Szpankowski, W. 1992. Stability of token passing rings. *Queueing systems* 11(1):7–33.

Gross, D., and Harris, C. 1998. *Fundamentals of Queueing Theory*. John Wiley & Sons, Inc.

Hejazi, S. R., and Saghafian, S. 2005. Flowshop scheduling problems with makespan criterion: a review. *International Journal of Production Research* 43:2895–2929.

Kovács, A., and Beck, J. C. 2011. A global constraint for total weighted completion time for unary resources. *Constraints* 16(1):100–123.

Kumar, P. R., and Meyn, S. P. 1995. Stability of queueing networks and scheduling policies. *IEEE Transactions on Automatic Control* 40(2):251–260.

Kumar, P. R. 1994. Scheduling semiconductor manufacturing plants. *IEEE Control Systems Magazine* 14(6):33–40.

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

Lu, S. H., and Kumar, P. R. 1991. Distributed scheduling based on due dates and buffer priorities. *IEEE Transactions on Automatic Control* 36(12):1406–1416.

Meyn, S. P., and Down, D. 1994. Stability of generalized Jackson networks. *The Annals of Applied Probability* 4(1):124–148.

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

Ross, S. M. 2003. *Introduction to Probability Models*. Academic Press. chapter 6 – Continuous-Time Markov Chains, 349–399.

Seidman, T. 1994. "first come, first served" can be unstable! *IEEE Transactions on Automatic Control* 39(10):2166–2171.

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.; Down, D. G.; and Beck, J. C. 2012. Stability of a polling system with a flow-shop server. Technical Report MIE-OR-TR2012-01, Department of Mechanical and Industrial Engineering, University of Toronto. Available from http://www.mie.utoronto.ca/labs/ORTechReps/.

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

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.

Van Hentenryck, P., and Bent, R. 2006. *Online Stochastic Combinatorial Optimization*. MIT Press.

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

Xia, C. H.; Shanthikumar, J. G.; and Glynn, P. W. 2000. On the asymptotic optimality of the SPT rule for the flow shop average completion time problem. *Operations Research* 48(4):615–622.