Chapter 1

SCHEDULING MANUFACTURING SYSTEMS OF RE–ENTRANT LINES

P. R. Kumar

Abstract

Re–entrant lines are manufacturing systems where parts may return more than once to the same machine, for repeated stages of processing. Examples of such systems are semiconductor manufacturing plants. We consider the problems of scheduling such systems to reduce manufacturing lead times, variations in the manufacturing lead times, or holding costs.

We assume a deterministic model, which allows for bursty arrivals. We show how one may design scheduling policies to help in meeting these objectives. To reduce the mean or variance of manufacturing lead time, we design a class of scheduling policies called Fluctuation Smoothing policies. To reduce the holding costs in systems with set–up times, we introduce the class of Clear–A–Fraction scheduling policies.

We study the stability and performance of these scheduling policies. We illustrate how scheduling policies can be unstable in that the levels of the buffers become unbounded. However, we show that all Least Slack policies, including the well known Earliest Due Date policy and all the Fluctuation Smoothing policies, are stable.

1.1 Introduction

Manufacturing systems consist of one or many machines, processing many parts. Arriving parts may require processing by specific machines, usually in a fixed order (a route). They may come with pre-determined processing times at each machine, or they could be random. The arrival of parts into the system may be either at our control, or exogenously determined. Machines may fail at random repair times, and be subject to random repair times. If a machine can cater to different types of parts, or the same part-type at different stages, it may incur a set-up or change-over time when switching between part-types.

While many of these contingencies are outside our control, there are several decisions that are at our disposal. We can decide which of several parts, waiting for processing at a machine, is processed next when the machine becomes available. This can be executed by a machine *scheduling policy*. In some cases we may even be allowed to regulate the release of new parts to the system, subject to certain constraints such as maintaining an average throughput. This can be governed by a part *release policy*.

The goal of scheduling is to choose such policies to provide good performance with respect to performance measures of interest. Such performance measures may be the cost of the total work-in-process, the mean manufacturing lead-time, or even the variance of the manufacturing lead time.

Depending on the particular issues being addressed, one should adopt a different model of the system that captures the relevant features. In turn, depending on the model adopted and the problem being addressed, one may adopt different strategies to design scheduling policies and analyze them. The choice of the model or framework adopted to address the relevant issues is usually a compromise between faithfulness to the situation, and tractability of subsequent design and analysis.

The model of a manufacturing system can be based on either discrete or fluid flows, and the various unpredictable events can be viewed as either deterministic but uncertain, or stochastic. The design procedure can range between pure intuition based, for example, on hypotheses about large scale system behavior, or derived from an optimal control formulation. In some cases, analysis can provide proofs of important properties of scheduling policies, or bounds on their performance, while in other cases recourse has to be made to carefully designed simulation experiments.

In this chapter, we shall study the design and analysis of scheduling policies for two models of systems. Together they illustrate some of the choices outlined above. Our goal here is to not only show new strategies for handling specific problems in manufacturing systems, but also to exhibit the flexibility available in modeling, design and analysis. By exploiting this richness of problems, solutions and analyses, a more comprehensive and applicable theory of manufacturing systems may emerge.

1.2 Re-Entrant Lines: The Models

Consider a manufacturing system as in Figure 1. It consists of four machines labelled M1 through M4. Parts arrive to the system at machine 1. It is convenient to suppose that while they are waiting for service at M1 on their first visit to it, they are stored in a (real or virtual) buffer called b_1. After processing is completed at M1 they visit machine M2 where they are stored in a buffer b_2, and so on. The last buffer visited is b_9 at machine M4, from which they then exit the system.

A very important feature of the system in Figure 1, is that parts return more than once to some of the machines, for repeated stages of processing. For example, machine M1 is visited thrice by parts during their route, at which time they are stored in buffers b_1, b_4 or b_7. We shall call such systems as *re-entrant lines*. A fundamental consequence is that when a machine, say M1, becomes idle, there may be parts waiting in all of the buffers b_1, b_4 and b_7. These are parts at different stages of their manufacturing lives, but all clamouring for attention from machine M1. The machine M1 then has to decide which part from which buffer it will process next. We therefore have the problem of resolving the competition for machines, by parts at different stages of production. We thus need to choose some machine *scheduling policy* to resolve which part to serve next at machine M1. Our choice of a scheduling policy will affect the performance of the overall system.

These systems of re-entrant lines can be contrasted with flow shops which produce automobiles. No machine or worker is ever revisited, and each worker simply serves the next car in line[1]. Thus, the issue of machine scheduling to resolve competition, as posed here, does not arise.

To complete the specification of the system, it is necessary to specify how new arrivals to the system are generated, and also the service requirements for parts in different buffers.

For example, a Poisson process may faithfully model the arrival process. Clearly, in that case, the arrivals are outside our control. Alternatively, the inter-arrival time between parts may be a constant. We shall label such periodic arrivals as *deterministic*. It may however be the case that arrivals are not so regular as to be periodic. The arrivals may simply be *bursty*. One may model such arrivals by requiring that

$$\text{Number of arrivals in } [t,T] \leq \lambda(T-t) + \delta \text{ for every } 0 \leq t \leq T. \tag{1.1}$$

The constant δ above allows some burstiness, and λ is an upper bound on the long term rate of arrivals. We shall simply call λ as the *arrival rate*, and refer to such arrival processes as *bursty arrivals*.

[1]Strictly speaking, this is not true. To address problems associated with quality control, a car may on occasion return to the same worker. We are grateful to Semyon Meerkov for pointing this out.

If the arrivals, or more properly, releases of raw parts into the system, are fully at our disposal, then we have several possibilities for regulating them. For example, one could try to maintain the total number of parts in the system at a constant level N. Thus, whenever one part exits the system, another raw part is released into the system. Such a policy will be called a *closed-loop release policy*. It can be simply viewed as creating a direct link between the exit buffer b_9 and the entry buffer b_1, in Figure 1. One can simply envision the system as a closed queueing network which contains N trapped parts that are constantly circulating round the network. Another possibility is to try to release new parts into the system whenever the number of parts, or work, destined for a certain machine in the system, drops below a threshold. Such a release policy is often used to alleviate starvation of certain critical bottleneck machines. It is called a *workload regulation* policy.

Finally, one needs to specify the service requirements for parts. At one extreme, service requirements, or processing times, may be deterministic. In that case, we simply suppose that parts in buffer b_i require τ_i time units of processing, from the machine serving b_i. We call $\mu_i := \frac{1}{\tau_i}$ the processing rate. Alternatively, the service times may be random variables, with a probability distribution that is possibly different for each buffer, together with some independence assumptions.

Let us turn next to examining what performance measures may be of possible interest. The most common performance measure of interest is the *manufacturing lead time* (MLT). This is the time elapsed from the moment of entry of a raw part, to its exit as a finished product, i.e., the time spent by a part in the system. It is called the sojourn time in queueing networks, the delay in communication networks, and the *cycle-time* in semiconductor manufacturing parlance. Since the manufacturing lead times of different parts may be different, one may want to take as the objective to minimize, or reduce, the *mean* manufacturing lead time. It is worth noting that by Little's Law, under some stability assumptions,

$$\begin{aligned} \text{Mean Number of Parts in System} &= \text{(Mean Arrival Rate of Parts)} \\ &\times \text{(Mean Manufacturing Lead Time)}. \end{aligned}$$

Thus, for a fixed mean arrival rate of parts, reducing the mean manufacturing lead time is equivalent to reducing the mean total number of parts in the system. The parts in the system are called the *work-in-process (WIP)*. Hence, minimizing the mean manufacturing lead time minimizes the cost associated with the WIP.

The manufacturing lead time of parts may vary dramatically, and in that case one may want to reduce the *variance* of the manufacturing lead time. A small variance ensures that the exit time of a part may be well predicted, important in make to order situations, since we can reliably predict when the product will be available. This allows better planning of releases into the system, as well as better coordination with respect to further operations (such as assembly) on

the output of finished parts produced by the plant.

Another performance measure of interest is the *throughput* of the plant, i.e., the rate at which finished parts leave the plant. If the manufacturing system is *stably* operated, by which we mean that there is no accumulation of parts in the system, then the rate of departure of parts from the system is the same as the rate of arrivals. If arrivals are outside our control, then this is not a performance measure that we can alter. On the other hand, for some release policies, such as the closed-loop release policy, the number of parts in the system is maintained at N. However the rate of circulation of parts, i.e., the throughput, may depend on the scheduling policy. Therefore, maximizing throughput is usually the objective of interest in such situations.

1.3 Fluctuation Smoothing Scheduling Policies to Reduce Variance of Lateness, Variance of Cycle–Time, and Mean Cycle–Time

In this section we will consider the problem of designing good scheduling policies for re-entrant lines. Our goal is to reduce both the mean and variance of manufacturing lead time. Of course, it is unlikely that two performance measures can be simultaneously optimized. However, in what follows, we will design policies that do well with respect to both criteria, compared to many other policies. First, we need to be a bit more specific about the model.

Let us suppose initially that the arrivals are exogenously determined, say deterministic or Poisson. We shall suppose that all processing times are independent, and that the processing times for parts in buffer b_i are random variables with a certain probability distribution, say F_i.

Our design will be based on intuitive considerations, using some intuition gleaned from queueing theory. It is convenient to start by supposing that each part arriving to the system has a *due-date*, and that the goal is to minimize the variance of *lateness*. By lateness, we mean the difference between the *exit time* and the due date. If π denotes a part, let $\alpha(\pi)$, $\delta(\pi)$ and $e(\pi)$ denote, respectively, its arrival time, due-date, and exit time.

Suppose that ζ_i is the mean, or an estimate of the mean, of the time required by parts to move from b_i to the exit. Consider a time t at which a part π is in buffer b_i. Then $(\delta(\pi) - t)$ is the time remaining till the due-date, while ζ_i is an estimate of the remaining sojourn time in the system. Thus, $(\delta(\pi) - t - \zeta_i)$ is an estimate of the lateness of the part π. Let us call

$$s(\pi) \quad := \quad \delta(\pi) - t - \zeta_i$$

the *slack* of part π. Since the current time t is the same for all parts, we could simply redefine $s(\pi)$ as,

$$s(\pi) \quad := \quad \delta(\pi) - \zeta_i.$$

It seems "fair" that a machine should process that part π, waiting in one of its buffers, that has the *least* slack. We shall say that this is a *Least Slack* (LS) policy. (Note that if there are waiting parts, the machine is not allowed to stay idle, but must choose between them based on their slacks). Now let us ponder over what such "fairness" amounts to. First, such a policy will tend to make all parts equally late or equally early. Thus it tends to reduce the *variance of lateness*.

Now let us turn to the issue of reducing the *variance of manufacturing lead-time*. Suppose that the *mean* manufacturing lead time is M. If we now set the due-date as $\delta(\pi) := \alpha(\pi)+M$, then the lateness is $e(\pi)-\delta(\pi) = e(\pi)-\alpha(\pi)-M$. Since M is a constant, the variance of the lateness is also the variance of the manufacturing lead time $e(\pi) - \alpha(\pi)$. Thus, the policy which chooses the part with least slack,

$$s(\pi) \quad := \quad \alpha(\pi) + M - \zeta_i,$$

tends to reduce the variance of manufacturing lead time. One should note that since M does not depend on π, it does not play any role in selecting the part π with the least slack. One can therefore simply redefine the slack as $s(\pi) := \alpha(\pi) - \zeta_i$. This is also a Least Slack policy, though with a different definition for slack. We shall identify it as the *Fluctuation Smoothing Policy for Variance of Cycle-Time* (FSVCT).

One should note that the above policies are parametrized by the estimates of the mean remaining time, the ζ_i's. However, these estimates are themselves determined by the scheduling policy employed. Thus we have a mutual dependence. To resolve this we use a fixed point iteration scheme. First, we set all the estimates ζ_i to zero, and simulate the resulting policy. Next, we use the results of the simulation to refine the estimates. Then we conduct a new simulation using the same seeds (to fix the behavior of extraneous events) but employing the refined estimates. This process is repeated, and typically converges in just a few iterations, less than ten. We can also average over the results of several such simulations using different seeds.

A simulation study of a model of a semiconductor plant, which is a large re-entrant line that is also subject to machine failures and repairs, shows that the FSVCT policy is effective. The result is shown in Figure 2, which compares several policies. The strategy appears to lead to a sizable reduction in the variance of the manufacturing lead time.

Now let us address the problem of minimizing the *mean* manufacturing lead time. It is convenient to start with the simplest case, where arrivals are deterministic, i.e., periodic.

We shall take our cue from queueing theory for single-server queues. Consider first an M/G/1 queue. It is known from the Pollaczek-Khintchine formula that the mean delay is given by $\frac{2\rho-\rho^2+\lambda^2\sigma_s^2}{2\lambda(1-\rho)}$, where λ is the arrival rate, ρ is the mean server utilization, and σ_s^2 is the variance of the service times. We see

that when σ_s^2 is reduced, the mean delay is also reduced. One can paraphrase this by saying that the smaller the *burstiness* in service times, the smaller the delay. More generally, consider a GI/GI/1 queue, where the mean interarrival time is $\frac{1}{\lambda}$, and the variance of the interarrival times is σ_a^2, while ρ and σ_s^2 are as above. Kingman has shown that a bound on the mean delay is $\frac{2\rho - 2\rho^2 + \lambda^2\sigma_a^2 + \lambda^2\sigma_s^2}{2\lambda[1-\rho]}$. Again, we see that large values of σ_s^2 produce large bounds. Additionally, we see that large values of σ_a^2 also produce large bounds. We thus see that the larger the burstiness in interarrival times, the larger is the upper bound on mean delay. This is in fact a general maxim in queueing theory. Burstiness increases delay.

We will now attempt to reduce delays *throughout* the manufacturing system by *simultaneously* attempting to reduce the burstiness of arrivals to all nodes. Since the arrivals to the system, i.e., arrivals to b_1, have been assumed to be deterministic, they have no burstiness at all.

Hence, to reduce the burstiness of arrivals to a certain buffer b_k, it is reasonable to attempt to reduce the variance of the delay incurred by parts in going from b_1 to b_k. But this is the same as the problem of reducing the variance of the manufacturing lead time, except that one truncates and excises the system after b_{k-1}. Thus, one would form estimates of mean time to go from every b_i, with $i < k$, to b_k; call it ζ_i^k, and define the slack as $s(\pi) = \alpha(\pi) - \zeta_i^k$, for a part in buffer b_i. (We will not fully ignore all buffers b_i for $i \geq k$, since that would be unrealistic.)

So far we have argued that this policy reduces the burstiness of arrivals to a particular buffer, namely b_k. Now note that $\zeta_i = \zeta_i^k + \zeta_k$, and since k was fixed in our argument above, we could redefine the slack as $s(\pi) := \alpha(\pi) - \zeta_i$, for a part in b_i. However, now we have a policy which is *uniform* in k. Thus, the Least Slack policy with slack defined as $s(\pi) := \alpha(\pi) - \zeta_i$, for a part in b_i, should be a reasonable policy to reduce the burstiness of *all* flows in the network simultaneously, i.e., to all buffers, when the arrivals of raw parts to the system are periodic. It should therefore (one hopes) lead to a small mean manufacturing lead time.

Above, we have only considered deterministic, i.e., periodic, arrivals. Now let us consider more general arrivals. First note that to reduce the burstiness of exits from the system, one can simply set periodic due dates, and then reduce the variance of lateness. Define $\delta(\pi) = n/\lambda$, if π is the n-th release into the system, and $\frac{1}{\lambda}$ is the mean interarrival (equivalently, interdeparture) time. This gives rise to the Least Slack policy with slack $s(\pi)$ defined as

$$s(\pi) \quad := \quad \frac{n}{\lambda} - \zeta_i,$$

if π is the n-th part released into the system, and it is located in buffer b_i. By the same argument as above, this scheduling policy can be used to reduce burstiness of flows *throughout* the network.

Thus, we arrive at the Least Slack policy with $s(\pi)$ defined as $s(\pi) = \frac{n}{\lambda} - \zeta_i$

for reducing the mean manufacturing lead time. Let us call this the *Fluctuation Smoothing policy for Mean Cycle-Time* (FSMCT).

One may note that it agrees with the policy derived earlier for deterministic, i.e., periodic releases, since there $\alpha(\pi) = \frac{n}{\lambda}$. In fact, for such arrivals, FSMCT = FSVCT, and so the *same* policy reduces both mean and variance of cycle-time.

There is a very appealing interpretation of the FSMCT policy as trying to equalize and reduce the total *downstream* shortfall from each buffer. To see this, let $x_i(t)$ be the number of parts in buffer b_i at time t, and let $\bar{x}_i$ be its mean value in steady state. Consider the quantity ($\lambda s(\pi)$ - Total Number of Exits from the System). This is a monotone increasing function of $s(\pi)$, and so the FSMCT policy simply chooses that part π for which this quantity is smallest. However, this quantity has a nice interpretation, which can be seen as follows:

$$\begin{aligned}
\lambda s(\pi) \quad &- \quad \text{Total Number of Exits from the System} \\
&= \quad \lambda(n/\lambda - \zeta_i) - \text{ Total Number of Exits from the System} \\
&= \quad (n - \text{ Total Number of Exits from the System}) \ - \lambda\zeta_i \\
&= \quad \text{Number of Parts Downstream from } b_i \text{ in the System} \\
&\quad\ - \text{ Mean Number of Parts Downstream from } b_i \text{ in Steady State} \\
&= \quad \sum_{j=i}^{L} x_j(t) - \sum_{j=i}^{L} \bar{x}_j .
\end{aligned}$$

Above, the second to last equality is true because the number of parts currently downstream from b_i is (n - Total Number of Exits from the System), since the n-th release is at the head of buffer b_i, and parts leave the system in the order that they arrive. (We are implicitly assuming here that the system is started empty). Also, by Little's Theorem, $\lambda\zeta_i$ = Mean Number of Parts Downstream from b_i. Thus we see that the FSMCT policy has the very appealing property of trying to equalize and reduce the *total downstream shortfall* from every buffer.

The above equivalent representation also shows that FSMCT is a *stationary* policy, in the sense of Markov Decision Processes. By this, we mean that its actions are based only on the numbers of parts in each buffer.

Figure 3 shows a comparison of several policies for a model of a semiconductor manufacturing plant. We see that our FSMCT strategy seems very effective.

1.4 Stability of LBFS, SRPTS, EA, EDD and All Least Slack Scheduling Policies

In Section 3, we have motivated the use of various Least Slack policies. All of them define the slack of a part π in buffer b_i as $s(\pi) := \gamma(\pi) - \zeta_i$. Here $\gamma(\pi)$ is an attribute of the part, while ζ_i is an attribute of the buffer b_i in which the part π is located. The choices $\gamma(\pi) = \delta(\pi)$, the due-date, $\gamma(\pi) = \alpha(\pi)$, the arrival time, and $\gamma(\pi) = \frac{n}{\lambda}$, were all shown to be useful.

Also, ζ_i was a number, chosen for each buffer b_i. While ζ_i was in all cases taken to be an "estimate of remaining delay from b_i," let us allow ζ_i here to be any real number. We can then regard $(\zeta_1, \ldots, \zeta_L)$ as a vector which *parametrizes* the least slack policies.

We now wish to analyze the behavior of such Least Slack policies. To do so for general stochastic models is currently infeasible. Let us simplify the situation, and assume a bursty model for arrivals, and a deterministic model for service times. The latter could be relaxed somewhat.

Let us suppose that arrivals to the system satisfy (1.1). Also, suppose that every part in buffer b_i requires τ_i time units of processing from the machine serving b_i. Regarding the definition of slacks, we will only assume that for every part π, the part attribute $\gamma(\pi)$ satisfies,

$$|\gamma(\pi) - \alpha(\pi)| \quad \leq \quad M \text{ for all } \pi, \tag{1.2}$$

for some $M < +\infty$. It clearly covers the situation $\gamma(\pi) = \alpha(\pi)$. It covers $\gamma(\pi) = \frac{n}{\lambda}$, if π is the n-th part released into the system too, by virtue of (1.1). Finally, it will cover $\gamma(\pi) = \delta(\pi)$, if one assumes that

$$|\delta(\pi) - \alpha(\pi)| \quad \leq \quad M' \text{ for all } \pi, \tag{1.3}$$

for some $M' < +\infty$, i.e., if the due-date is set to within some bounded time of the arrival time – a reasonable assumption.

One would like to quantify the performance of such Least Slack policies. However, an even more fundamental issue needs to be resolved first. Can the performance be so bad that the delay of parts is unbounded? This is the problem of *stability* of a scheduling policy. Let $e(\pi)$ denote the time that part π exits from the system. We shall say that a scheduling policy is *stable* if

$$e(\pi) - \alpha(\pi) \quad \leq \quad M'' \text{ for all } \pi,$$

for some $M'' < +\infty$, i.e., the manufacturing lead time is bounded. Note that when the due-date setting satisfies (1.3), stability is equivalent to

$$| \, \delta(\pi) - e(\pi) \, | \quad \leq \quad M''' \text{ for all } \pi,$$

for some $M''' < +\infty$, i.e., boundedness of lateness of parts.

Clearly, for *any* scheduling policy, a necessary condition for its stability is that

$$\sum_{\{i : b_i \text{ is served by } \sigma\}} \lambda \tau_i \quad < \quad 1 \text{ for all machines } \sigma. \tag{1.4}$$

This is simply the requirement that each machine be fast enough to serve the needs of arriving parts, if they arrive at rate λ. Let us call this the *capacity condition*.

Is it true that all scheduling policies are stable whenever the arrival rate is within the capacity of the system? Clearly not, since a scheduling policy that insists on keeping all machines always idle, is unstable.

Let us say that a scheduling policy is *non-idling* if no machine is ever allowed to stay idle whenever any of its buffers is non-empty. It is clear that some policies which are *not* non-idling are unstable; an extreme example is the always idling policy above. However, a policy which is not non-idling need not necessarily be unstable, or even unreasonable, for that matter.

Let us restrict our attention to non-idling policies. There are several examples of such scheduling policies. One example is the well known First Come First Serve (FCFS) policy, where a machine provides service to the part which arrived first to that machine. (We should note here that by the arrival time of a part to the machine we mean the arrival time for the current visit to the machine, since a part may revisit a machine more than once.) Also, the class of Least Slack policies is non-idling, as noted earlier.

Another class of non–idling policies is the class of *buffer priority* policies. These are policies where a priority order is chosen for the buffers at each machine, and the machine then provides service according to the ordering. For choosing parts from within a buffer, the service priority could, for example, choose from the head of the buffer.

As an example, consider ordering buffers in a re-entrant line according to the order visited, i.e., $\{b_1, \ldots, b_L\}$. We shall call this the *First Buffer First Serve* policy (FBFS). Thus, if b_i and b_j, with $i < j$, are in contention for the same machine, then b_i has higher priority. A part in b_j gets taken up for processing only when all buffers b_k with $k < j$, which are located at the same machine as b_j, are empty.

In all cases, we could let the service priority be pre-emptive resume or non-preemptive. Under a *pre-emptive resume* priority, a part arriving to a higher priority buffer pre-empts the part currently undergoing processing, which then has to wait till the next time that there are no other higher priority parts at the machine, for the resumption of its remaining processing. Under a *non-preemptive* priority policy, a new part is taken up for processing only when the part currently undergoing processing completes its service. For many manufacturing operations, a non-preemptive priority is more realistic[2].

Diametrically opposed to the FBFS policy is the *Last Buffer First Serve* policy (LBFS), where the priority ordering of the buffers is $\{b_L, b_{L-1}, \ldots, b_1\}$. Thus, buffers visited later in the route are given higher priority.

The LBFS policy can also be regarded as a *Shortest Remaining Processing Time in System* (SRPTS) policy. Assuming deterministic processing times, a part closer to exit has a lesser total of remaining processing times left, and an SRPTS policy gives priority to such parts – just as the LBFS policy does.

[2]However, in the operation of computer systems, a pre-emptive resume priority is often used.

A little thought should convince the reader that the LBFS policy is the *system-wide* analog of the FCFS policy, where the arrival time *to the system* dictates the priority. Clearly, if every machine gives priority to parts which arrived first to the system, then parts which arrived first would be located in buffers nearer the exiting end of the system. Thus the LBFS policy which gives priority to buffers nearer the exiting end of the system also gives priority to the parts which arrived first to the system. Therefore, it could also be called an *Earliest Arrival* (EA) policy.

In turn, the EA policy, and thus the LBFS policy too, is a special case of a Least Slack policy, which is obtained by defining the slack as $s(\pi) = \alpha(\pi)$, the arrival time to the system, and setting all $\zeta_i \equiv 0$.

Finally, the EA policy, is the same as the well known *Earliest Due Date* (EDD) policy, provided the due-dates assigned to parts are monotone increasing in the arrival times of the parts to the system, i.e., if $\alpha(\pi) < \alpha(\pi') \Rightarrow \delta(\pi) < \delta(\pi')$. Another way to state this is to say that parts arrive in the order of their due dates.

One therefore notes that the LBFS policy coincides with EA, SRPTS, and EDD policies. For these reasons, the LBFS policy is a useful policy to analyze. Another perhaps more important reason is that since it is a special case of the family of Least Slack policies, one may hope to learn something about this more general class of policies. We have already seen in Section 3 that such Least Slack policies can yield useful results.

To show that stability of non-idling scheduling policies cannot be taken for granted, we now give an example of an unstable buffer priority policy.

Example: An Unstable Buffer Priority Policy.

Consider the re-entrant line of Figure 4. Parts arrive periodically to buffer b_1, one every hour. Each part requires 0 hrs, 2/3 hrs, 0 hrs and 2/3 hrs of processing, at buffers b_1, b_2, b_3 and b_4, respectively. The requirement of 0 hrs means that the processing time is very small; however a part may still have to wait for processing due to the nature of the scheduling policy. Clearly, the capacity condition (1.4) is satisfied at each machine, since each part requires 2/3 hrs of total processing at each machine, and parts only arrive 1 per hour.

Consider now the buffer priority ordering $\{b_4, b_2, b_3, b_1\}$. This is slightly different from LBFS. It gives priority to b_4 over b_1 at machine M1, and priority to b_2 over b_3 at machine M2. Consider now the initial condition $(x, 0, 0, 0)$ at time $t = 0$, which denotes that buffer b_1 has x parts in it, and all other buffers are empty.

Let us now trace the evolution of the system forward from time $t = 0$. Since b_4 is empty, machine M1 works on b_1, and immediately transfers all x parts to b_2. Machine M2, which gives priority to b_2 over b_3, then goes into a busy period working on buffer b_2. While it is working on buffer b_2, new parts are arriving to buffer b_1, one every hour. Since b_4 is still empty, machine M1 can work on

these new arrivals, and so they are promptly transferred to b_2. The buffer b_2 finally empties for the first time at $t = 2x$. To see this, note that at $t = 2x$, the x original parts, plus the $2x$ new parts which arrived to b_1, a total of $3x$ parts, have been sent to b_2. These $3x$ parts take up $2x$ hours of processing time (2/3 hours per part). Thus, at $t = 2x$, buffers b_1, b_2 and b_4 are empty, while b_3 has $3x$ parts. Machine M2 can then work on b_3 since b_2 is empty, and so it promptly dispatches all $3x$ parts to b_4. At M1, b_4 has higher priority than b_1, and so M2 goes into a busy period while processing the $3x$ parts in b_4. During this busy period, new parts arriving to b_1 are held up, and thus machine M2 undergoes an enforced *starvation*. The busy period to process $3x$ parts at b_4 lasts $2x$ hours. During this busy period, $2x$ new parts have arrived to b_1.

Thus, we see that at the end of this busy period, the new state of the system is $(2x, 0, 0, 0)$.

Hence the number of parts in the system has doubled. This process continues indefinitely, and so the buffer levels are all unbounded. □

The key feature to notice in the above example is that when working on parts in one buffer, the scheduling policy may cause starvation of some machine. Due to the re–entrant nature of the line, and the consequent flow of material in both directions between machines, the build-up of parts in buffers can undergo back and forth amplification, which leads to instability.

In fact, such instability can be latent in any scheduling policy, and it is of interest to see which scheduling policies are unstable. In particular, we would like to determine whether the class of Least Slack policies developed in Section 3 is stable.

We shall show below that the entire class of Least Slack policies, i.e., for all choices of parameters $\{\zeta_i\}$, is stable for all re-entrant lines with arrival rate within the capacity of the system.

It is convenient to begin however with the LBFS policy. The key result in establishing the stability of LBFS is a bound on the delay experienced by a new part, as a function of the number of parts already in the system when it arrives.

Theorem: Bound on Delay of a New Part. Consider a re-entrant line with service times τ_i at buffer b_i, operating under the pre-emptive resume LBFS policy. For every $\epsilon > 0$, there exists a constant $c(\epsilon)$ with the following property. If a new part π arrives to the system to find x parts already in the system, then

$$\text{Manufacturing Lead Time of } \pi \leq (\bar{w} + \epsilon)x + c(\epsilon) \text{ for every } \epsilon > 0, \text{ and all } x.$$

Above, $\bar{w}$ is the maximum work brought to a machine by an incoming part, i.e.,

$$\bar{w} \quad := \quad \max_{\{\sigma : \sigma \text{ is a machine}\}} \sum_{\{i : b_i \text{ is served by } \sigma\}} \tau_i.$$

Ignoring the ϵ, the above result says that the delay to clear x parts from the system is essentially only the delay experienced by the x parts in traversing the *bottleneck* machine. Thus, it rules out excessive enforced starvation.

Proof of Theorem. Let us define by $B_i := \{b_i, b_{i+1}, \ldots, b_L\}$ the *head section* of the manufacturing system. Thus B_1 is the entire system.

Let

$$\bar{w}_i \quad := \quad \max_{\{\sigma : \sigma \text{ is a machine }\}} \sum_{\{j : b_j \text{ is served by } \sigma \text{ and } j \geq i\}} \tau_j$$

be the maximum work contributed to a machine by a part in the head section B_i. Note that $\bar{w} = \bar{w}_1 \geq \bar{w}_2 \geq \cdots \geq \bar{w}_L = \tau_L$.

Consider a part π which arrives to the head section B_L, i.e., to b_L, to find x parts in B_L. Clearly, since the highest priority under pre-emptive LBFS is given to parts in b_L, the delay suffered by π is

$$\text{Delay} \quad \leq \quad x\tau_L + \tau_L.$$

Noting that $\tau_L = \bar{w}_L$, we have proved,

$$\begin{pmatrix}\text{Delay experienced by a part } \pi \text{ arriving to} \\ B_L \text{ to find } x \text{ parts in } B_L\end{pmatrix} \quad \leq \quad (\bar{w}_L + \epsilon)x + c_L(\epsilon)$$

where $c_L(\epsilon) := \tau_L$ for all $\epsilon > 0$.

Our proof is by induction on i, starting with $i = L$, that there is a constant $c_i(\epsilon)$, for every $\epsilon > 0$, so that

$$\begin{pmatrix}\text{Delay experienced by a part } \pi \text{ arriving to} \\ B_i \text{ to find } x \text{ parts in } B_i\end{pmatrix} \\ \leq (\bar{w}_i + \epsilon)x + c_i(\epsilon) \text{ for every } \epsilon > 0.$$

Now suppose that the induction hypothesis is true for $i+1, i+2, \ldots, L$. Consider a part π arriving to B_i, to find x parts already in B_i.

When part π arrives to B_i, i.e., to b_i, say at time 0, it may find some number n of parts already in b_i. The value of n may range between 0 and x.

Consider $n = 0$. Let us denote by π' the part that arrived just prior to π. Alternatively, π' is the part which has second lowest priority in B_i; its priority being higher only than that of π. Since $n = 0$, π' has to be in B_{i+1}. Moreover, π' has $(x-1)$ parts ahead of it in B_{i+1}. Hence, using the validity of our induction hypothesis for the head section B_{i+1}, we have

$$\begin{aligned}\text{Exit time of } \pi' \quad &\leq \quad (\bar{w}_{i+1} + \epsilon)(x-1) + c_{i+1}(\epsilon) \\ &= \quad (\bar{w}_{i+1} + \epsilon)x + (c_{i+1}(\epsilon) - \bar{w}_{i+1} - \epsilon) \\ &\leq \quad (\bar{w}_i + \epsilon)x + (c_{i+1}(\epsilon) - \bar{w}_{i+1} - \epsilon), \text{ for every } \epsilon > 0.\end{aligned}$$

Above, we have used the fact that $\bar{w}_{i+1} \leq \bar{w}_i$. Note now that after π' has exited the system, the part π has a clear run through the empty buffers ahead of it. Hence, it exits no more than $\sum_{j=i}^{L} \tau_j$ time units after π'. Thus,

$$\text{Delay of } \pi \quad \leq \quad (\bar{w}_i + \epsilon)x + \left(c_{i+1}(\epsilon) - \bar{w}_{i+1} - \epsilon + \sum_{j=i}^{L} \tau_j \right).$$

Now we perform a second layer of induction on n starting with $n = 0$. We will show that there exist constants $c_i^{(n)}(\epsilon)$ for every $\epsilon > 0$, so that

$$\begin{pmatrix}\text{Delay experienced by a part } \pi \text{ which ar-} \\ \text{rives to find } n \text{ parts in } b_i \text{ and } x \text{ parts in} \\ B_i\end{pmatrix} \quad \leq \quad (\bar{w}_i + \epsilon)x + c_i^{(n)}(\epsilon)$$

for every $\epsilon > 0$ and all x.

Note that this induction hypothesis has been shown above to be true for $n = 0$ if we choose

$$c_i^{(0)}(\epsilon) \quad \geq \quad c_{i+1}(\epsilon) - \bar{w}_{i+1} - \epsilon + \sum_{j=i}^{L} \tau_j.$$

Now suppose the induction hypothesis has been established for $0, 1, 2, \ldots, n-1$. Consider a part π arriving to B_i, at time 0, to find n parts in b_i, and $(x-n)$ parts in B_{i+1} – making for a total of x parts in B_i.

We need to introduce a bit of notation. Let T denote the time at which π moves from b_i to b_{i+1}. (It is easy to see that $T < +\infty$; in fact every part entering the system eventually leaves it). Also, let y be the number of parts in B_{i+1} when π enters b_{i+1}.

We will fix $\epsilon > 0$, and consider three cases.

Case 1: Suppose $y \leq \Gamma(\epsilon)$, where $\Gamma(\epsilon) := \frac{2c_{i+1}\left(\frac{\epsilon}{2}\right)}{\epsilon}$.

Note that π can wait no more than $\bar{w}_i x$ time units in b_i, before its processing starts. This is the worst case even if all x parts require attention from the machine serving b_i. Hence

$$T \quad \leq \quad \bar{w}_i x + \tau_i.$$

After entering b_{i+1}, π finds $y \leq \Gamma(\epsilon)$ parts ahead of it, and its further delay is therefore no more than $(\bar{w}_{i+1} + \epsilon)\Gamma(\epsilon) + c_{i+1}(\epsilon)$. Hence, the total delay incurred by π is no more than $\bar{w}_i x + \tau_i + (\bar{w}_{i+1} + \epsilon)\Gamma(\epsilon) + c_{i+1}(\epsilon)$. Therefore, the choice of $c_i^{(n)}(\epsilon) \geq (\bar{w}_{i+1} + \epsilon)\Gamma(\epsilon) + c_{i+1}(\epsilon) + \tau_i$ will suffice for this case.

Case 2: Suppose $\Gamma(\epsilon) < y \leq n$.

Note that when π enters b_i, there are n parts already in b_i. When it enters b_{i+1}, the assumption in this case is that the number of parts in B_{i+1} is $y \leq n$. Together, these imply that the part with highest priority in B_{i+1}, at the time that π enters B_{i+1}, was originally in buffer b_i with $(n-y)$ parts ahead of it in b_i, at time 0. Call this part π''.

Hence, at time 0, the part π'' has $(x-y)$ parts ahead of it in B_i, and $(n-y)$ parts ahead of it in b_i. Thus, its delay is no more than $(\bar{w}_i+\epsilon)(x-y)+c_i^{(n-y)}(\epsilon)$, since the induction hypothesis is true for $(n-y)$. Since at time T, π'' still has not exited the system, we see that

$$T \quad \leq \quad (\bar{w}_i + \epsilon)(x-y) + c_i^{(n-y)}(\epsilon).$$

After π enters B_{i+1}, it sees y parts ahead of it. Hence the further delay experienced by π is no more than $\left(\bar{w}_{i+1} + \frac{\epsilon}{2}\right) y + c_{i+1}\left(\frac{\epsilon}{2}\right)$. Here we have used the crucial fact that the induction hypothesis is true for all $\epsilon > 0$, and in fact true for $\frac{\epsilon}{2}$.

Taking the sum of the delay to enter b_{i+1}, and the delay to thereafter exit B_{i+1}, we see that

$$\begin{aligned}
\text{Delay of } \pi &\leq (\bar{w}_i + \epsilon)(x-y) + c_i^{(n-y)}(\epsilon) + \left(\bar{w}_{i+1} + \frac{\epsilon}{2}\right) y + c_{i+1}\left(\frac{\epsilon}{2}\right) \\
&= (\bar{w}_i + \epsilon)x + \left(\bar{w}_{i+1} - \bar{w}_i - \frac{\epsilon}{2}\right) y + c_{i+1}\left(\frac{\epsilon}{2}\right) + c_i^{(n-y)}(\epsilon) \\
&\leq (\bar{w}_i + \epsilon)x - \frac{\epsilon}{2} y + c_{i+1}\left(\frac{\epsilon}{2}\right) + c_i^{(n-y)}(\epsilon) \quad (\text{using } \bar{w}_{i+1} \leq \bar{w}_i) \\
&\leq (\bar{w}_i + \epsilon)x + c_i^{(n-y)}(\epsilon) \quad \left(\text{using } y > \Gamma(\epsilon) = \frac{2c_{i+1}\left(\frac{\epsilon}{2}\right)}{\epsilon}\right).
\end{aligned}$$

Thus the induction hypothesis is true for this case if we choose $c_i^{(n)}(\epsilon) \geq c_i^{(j)}(\epsilon)$ for all $0 \leq j \leq n-1$.

Case 3: Suppose $y > \max\{n, \Gamma(\epsilon)\}$.

Again, consider the part π'' which has highest priority in B_{i+1}, at the time T that π enters b_{i+1}. Note that at time 0, π had n parts ahead of it in b_i, while at time T it has y parts ahead of it in B_{i+1}. Since $y > n$, it follows that the part π'' was originally already in B_{i+1} at time 0. Hence the exit time of π'' from B_{i+1} is no more than $(\bar{w}_{i+1}+\epsilon)(x-y)+c_{i+1}(\epsilon)$. Since such exit has not yet occurred at time T, it follows that

$$T \quad \leq \quad (\bar{w}_{i+1} + \epsilon)(x-y) + c_{i+1}(\epsilon).$$

At time T, part π enters B_{i+1} to find y parts ahead of it. Hence its further delay is no more than $(\bar{w}_{i+1}+\epsilon)y + c_{i+1}(\epsilon)$.

Summing the delay incurred by π to enter B_{i+1}, and its further delay to exit the system, we see that

$$\begin{aligned} \text{Delay of } \pi &\leq (\bar{w}_{i+1}+\epsilon)(x-y) + c_{i+1}(\epsilon) + (\bar{w}_{i+1}+\epsilon)y + c_{i+1}(\epsilon) \\ &= (\bar{w}_{i+1}+\epsilon)x + 2c_{i+1}(\epsilon). \end{aligned}$$

Thus, the induction hypothesis is satisfied for this case if we take $c_i^{(n)}(\epsilon) \geq 2c_{i+1}(\epsilon)$.

Combining all three cases together, we see that the requirements on the constants $c_i^{(n)}(\epsilon)$ are

$$c_i^{(0)} \geq c_{i+1}(\epsilon) - \bar{w}_{i+1} - \epsilon + \sum_{j=1}^{L} \tau_j$$

and

$$c_i^{(n)} \geq \max\{(\bar{w}_{i+1}+\epsilon)\Gamma(\epsilon) + c_{i+1}(\epsilon) + \tau_i, c_i^{(0)}(\epsilon), c_i^{(1)}(\epsilon), \ldots, c_i^{(n-1)}(\epsilon), 2c_{i+1}(\epsilon)\}.$$

They are met by simply choosing,

$$\begin{aligned} c_i^{(n)}(\epsilon) := \max\{0, c_{i+1}(\epsilon) - \bar{w}_{i+1} - \epsilon + \sum_{j=i}^{L} \tau_j, \frac{2(\bar{w}_{i+1}+\epsilon)c_{i+1}\left(\frac{\epsilon}{2}\right)}{\epsilon} \\ + c_{i+1}(\epsilon) + \tau_i, 2c_{i+1}(\epsilon)\}. \end{aligned}$$

Thus the induction on n is complete, and that also completes the induction on i. □

From the proof of the preceding Theorem, by taking the explicit values of the constants into account, we have the following more explicit result.

$$\begin{pmatrix} \text{Delay experienced by a part } \pi \text{ that arrives to } b_i \text{ to find } x \text{ parts already in} \\ \{b_i, b_{i+1}, \ldots, b_L\} \end{pmatrix} \leq (\bar{w}_i + \epsilon)x + c_i(\epsilon) \text{ for every } \epsilon > 0,$$

where

$$\begin{aligned} \bar{w}_i &:= \max_{\{\sigma : \sigma \text{ is a machine}\}} \sum_{\{j : b_j \text{ is served by } \sigma \text{ and } j \geq i\}} \tau_j \\ c_L(\epsilon) &:= \tau_L \\ c_i(\epsilon) &:= 2c_{i+1}(\epsilon) + \frac{2(\bar{w}_{i+1}+\epsilon)c_{i+1}\left(\frac{\epsilon}{2}\right)}{\epsilon} + \sum_{j=i}^{L} \tau_j. \end{aligned}$$

Keeping track of the powers of ϵ so generated shows that,

$$c_i(\epsilon) \quad = \quad \sum_{j=0}^{L-i} \frac{a_j}{\epsilon^j}, \text{ for some system dependent constants } a_j, \ldots, a_{L-1}.$$

A particular corollary is the following result, which shows that even re-entrant lines, possessing cycles, satisfy a "pipeline" property, when operated under an LBFS policy.

Corollary: The Pipeline Property of LBFS.

$$\begin{pmatrix} \textit{Delay of part } \pi \textit{ which arrives to} \\ \textit{find } x \textit{ parts ahead of it in system} \end{pmatrix} \quad \leq \quad \bar{w}x + o(x).$$

Proof Note that,

$$\frac{\textit{Delay of } \pi - \bar{w}x}{x} \quad \leq \quad \epsilon + \frac{c(\epsilon)}{x} \text{ for every } \epsilon > 0.$$

Hence

$$\limsup_{x \to +\infty} \frac{\textit{Delay of } \pi - \bar{w}x}{x} \quad \leq \quad \epsilon \text{ for every } \epsilon > 0.$$

Thus,

$$\limsup_{x \to +\infty} \frac{\textit{Delay of } \pi - \bar{w}x}{x} \quad \leq \quad 0.$$

□

Using the above result on the delay experienced by an incoming part, one can prove the stability of the LBFS policy for *all* arrival rates within the capacity of the system.

Theorem: Stability of the LBFS Policy. Consider any bursty arrival process satisfying (1.1), with arrival rate λ. Suppose that the arrival rate is within the capacity of the system, i.e., (1.4) is satisfied. Assume that the system is operating under the LBFS policy. Then the system is stable, i.e., the number of parts in the system is bounded. If $x(t)$ denotes the number of parts in the system at time t, then the asymptotic number of parts in the system is bounded by,

$$\limsup_{t \to +\infty} x(t) \quad \leq \quad \frac{\lambda c(\epsilon) + \delta}{1 - \rho - \lambda\epsilon},$$

where $\rho := \lambda\bar{w}$ is the utilization ratio of the bottleneck machine, and $\epsilon > 0$ is any number with $\rho + \lambda\epsilon < 1$.

Proof Let $t_0 := 0$ and denote by $\pi^{(0)}$ the part which has the lowest priority among the $x(0)$ parts in the system at t_0, i.e., it is in the tail of some buffer b_j with buffers $b_1, \ldots, b_{j-1}$ behind it being empty. Recursively define for $n = 1, 2, \ldots$

$$t_n := \text{ time at which } \pi^{(n-1)} \text{ exits the system,}$$

where,

$$\pi^{(n)} := \text{ part with lowest priority in system at time } t_n.$$

We will analyze the system in the time intervals $[t_{n-1}, t_n]$ for $n \geq 1$. Note that

$$t_n - t_{n-1} = \text{ remaining manufacturing lead time, after } t_{n-1}, \text{ of } \pi^{(n-1)}.$$

However, at time t_{n-1}, the part $\pi^{(n-1)}$ has $x(t_{n-1}) - 1$ parts ahead of it in the system. Hence

$$t_n - t_{n-1} \quad \leq \quad (\bar{w} + \epsilon) x(t_{n-1}) + c(\epsilon) \text{ for every } \epsilon > 0.$$

Now note that at time t_{n-1}, there are no parts *behind* $\pi^{(n-1)}$. Hence, when it exits at time t_n, it leaves behind it in the system only those parts which arrived in the time interval $[t_{n-1}, t_n]$. Therefore, by the assumption (1.1) on the arrival process,

$$x(t_n) \quad \leq \quad \lambda(t_n - t_{n-1}) + \delta.$$

Combining the two inequalities above, we see that

$$\begin{aligned} x(t_n) \quad &\leq \quad \lambda(\bar{w} + \epsilon) x(t_{n-1}) + \lambda c(\epsilon) + \delta \\ &= \quad (\rho + \lambda\epsilon) x(t_{n-1}) + \lambda c(\epsilon) + \delta \text{ for every } \epsilon > 0. \end{aligned} \tag{1.5}$$

Since the arrival rate is within capacity, $\rho < 1$. Hence, one can choose $\epsilon > 0$ so small that $\rho + \lambda\epsilon < 1$. Consider any such ϵ. Then, the above difference equation is stable, and one obtains,

$$\limsup_{n \to \infty} x(t_n) \quad \leq \quad \frac{\lambda c(\epsilon) + \delta}{1 - \rho - \lambda\epsilon}.$$

This proves that the number of parts in the system is bounded. It also shows that the bound on the asymptotic number of parts in the system is valid if we sample the system at the times $\{t_n\}$. The demonstration that it is valid for all large t involves a slight extension of the above argument, and is omitted. □

So far, we have analyzed only the pre-emptive resume version of the LBFS policy. However, the arguments above are easily extended to the non-preemptive case, and in fact to the class of all Least Slack policies.

Let us directly turn to non-preemptive Least Slack policies, and see how one may prove a bound on the manufacturing lead time of a part which arrives to find x parts in the system. The heart of the matter is as follows. Consider a part π at a buffer b_i. If the system is operated under a pre-emptive resume LBFS policy, then the part π is delayed only by parts ahead of it. Now turn to a non–pre-emptive Least Slack policy. The part π may have to wait τ_i time units for a part to complete processing, because we are now operating in a non-preemptive mode. After this, it has to defer, at most, to those parts π' which satisfy

$$\gamma(\pi') - \zeta_k \quad < \quad \gamma(\pi) - \zeta_i,$$

where b_i and b_k share the same machine. Such a part π' must satisfy

$$\gamma(\pi') \quad < \quad \gamma(\pi) - \zeta_i + \zeta_k .$$

Recall now our assumption that the attribute $\gamma(\pi')$ for Least Slack policies satisfies (1.2). Thus, we see that a necessary condition for a part π' to delay π at b_i is that the arrival time $\alpha(\pi')$ of the part π' must satisfy,

$$\alpha(\pi') < \quad \alpha(\pi) - \zeta_i + \zeta_k + 2M \leq \quad \alpha(\pi) + 2M + 2\overline{\zeta},$$

where $\overline{\zeta} := \max_j |\zeta_j|$.

Under LBFS, a part π can be delayed at a buffer b_i only by parts that arrived in $(-\infty, \alpha(\pi)]$, i.e., by the parts that arrived before it. In comparison, under LS, a part may also be delayed by parts that arrived in $(\alpha(\pi), \alpha(\pi) + 2M + 2\overline{\zeta}]$. The number of such additional parts is at most $\lambda(2M + 2\overline{\zeta}) + \delta$. Together, these parts can delay π by at most $[\lambda(2M + 2\overline{\zeta}) + \delta]\,(\max_j \tau_j)$. Adding up the non-preemptive delay, and the delay that can possibly be caused by parts arriving after π, we see that their effect is bounded.

Such a bound can be propagated through the arguments of the stability Theorem above. We can thus prove the stability of all Least Slack policies.

Theorem: Stability of all Least Slack policies. Consider a re-entrant line with processing times τ_i at b_i, with arrivals satisfying the burstiness constraint (1.1), and the capacity condition (1.4). Consider any Least Slack policy, where the part attributes $\gamma(\pi)$ satisfy (1.2), and $\{\zeta_i, \zeta_2, \ldots, \zeta_L\}$ are any real numbers associated with the buffers $\{b_1, b_2, \ldots, b_L\}$, respectively. Then the system is stable, i.e., the number of parts in the system is bounded.

1.5 Dynamic Scheduling of a Single Machine with Set-Up Times: A Push Model

In this section we will address the problem of scheduling a machine producing several types of parts. The critical new issue we examine is how to schedule

the machine if it incurs set-up times whenever it changes the part-type it is processing. For notational convenience, it is useful to adopt a *fluid* model.

Consider the single machine system shown in Figure 5. There are P types of parts, labeled $1, 2, \ldots, P$ which arrive to the machine at the rates λ_1 parts/hr, λ_2 parts/hr ,..., and λ_P parts/hr, respectively.

Parts of type i take τ_i units of processing time; equivalently the processing *rate* for parts of type i is $\frac{1}{\tau_i}$. Moreover, the machine incurs δ hours of *set-up time* whenever it switches between part-types.

Let us suppose that parts of type i are held in buffer b_i, while awaiting processing. We denote by $x_i(t)$ the number of such parts in buffer b_i at time t, and suppose that there is a holding cost of c_i units per part of type i, per unit time. Thus, over a time interval $[0, T]$, the total cost incurred is

$$\int_0^T \sum_{i=1}^{P} c_i x_i(t) dt.$$

Let us suppose that our goal is to choose a scheduling policy which reduces the long-term average cost per unit time,

$$\limsup_{T \to \infty} \frac{1}{T} \int_0^T \sum_{i=1}^{P} c_i x_i(t) dt. \tag{1.6}$$

As in the previous section, we shall say that a scheduling policy is *stable* if $x_i(t)$ is bounded over $t \geq 0$ for $i = 1, 2, \ldots P$. In addition to stability, we will also examine how to attain good *performance*, i.e., a small value of the cost (1.6).

1.6 Clear-A-Fraction Policies

Let us first examine the issue of stability. Clearly, it is necessary that the arrival rate vector $(\lambda_1, \lambda_2, \ldots, \lambda_L)$ be within the *capacity* of the machine to handle. This translates to the requirement that

$$\rho \quad := \quad \sum_{i=1}^{P} \lambda_i \tau_i < 1. \tag{1.7}$$

Note that ρ is the average fraction of time that the machine needs to be busy processing parts, if it is to keep up with the incoming flow of parts. Thus, it can only afford to spend a fraction $(1 - \rho)$ of its time, at most, not processing any parts. This includes the time lost to set-up changes. Since each set-up incurs δ hours, the average time between set-ups can be no less than $\frac{\delta}{1-\rho}$.

This argument shows the danger of making too frequent set-up changes. In fact, a policy which changes set-ups too frequently will be unstable. As an

example, a policy which produces that part-type i for which $x_i(t)$ is largest, may be unstable.

Thus, once a part-type is taken up for production, it should be continued for a sufficient length of time, before the set-up is changed. However, there is also a disadvantage to having production runs which are too long; the buffer levels can reach unacceptably high levels resulting in too large a value of the cost (1.6). This gives rise to a need to maintain long, but not excessively long, production runs.

We will now study the dynamics of scheduling policies which attempt to regulate the lengths of production runs.

First, let us call a scheduling policy *clearing* if it always continues to work on a part-type until its buffer level hit zero. We shall say that a policy is *non-idling*, if no fraction of its capacity is unutilized whenever it is not changing a set-up. Thus, we exclude here policies which work on a part-type i at rate λ_i, rather than at the maximum possible rate $\frac{1}{\tau_i}$. Hence, as soon as a buffer level hits zero, a non-idling policy necessarily forces a change in set-up.

We shall say that a policy has the "clear-a-fraction" property if it is clearing, non-idling, and if there is an $\epsilon > 0$ such that whenever it begins a set up to a new part type p at time τ, then that part type's buffer level $x_p(\tau)$ is at least a fraction ϵ of the total of the buffer levels of all part-types, i.e.,

$$x_p(\tau) \quad \geq \quad \epsilon \sum_{i=1}^{P} x_i(\tau). \tag{1.8}$$

The first fundamental fact about such *Clear-A-Fraction* (CAF) policies is that they are always stable.

Theorem: Stability of Clear-A-Fraction Policies. Consider a single machine system as in Figure 5, satisfying the capacity condition (1.7). If a CAF policy is used, then the system is stable. Moreover,

$$\limsup_{t\to\infty} \sum_{i=1}^{P} x_i(t) \quad \leq \quad \frac{\delta\rho}{\underline{\tau}} + \frac{\delta\overline{\tau}}{\epsilon\underline{\tau}(1-\rho)} \max_i(\frac{\rho-\rho_i}{\tau_i}).$$

Above,

$$\begin{aligned} \underline{\tau} &:= \min_i \tau_i \\ \overline{\tau} &:= \max_i \tau_i \\ \rho_i &:= \lambda\tau_i, \text{ and } \rho = \sum_{i=1}^{P} \rho_i. \end{aligned}$$

Proof. Let $w(t) := \sum_{i=1}^{P} \tau_i x_i(t)$ denote the *work* in the system. Let $\{\tau_n\}$ denote the sequence of times at which set-ups are commenced, with $\tau_0 := 0$, and let p_n be the corresponding sequence of part-types chosen for the set-ups. Note that

$$t_{n+1} - t_n = \frac{x_{p_n}(t_n) + \delta\lambda_{p_n}}{\frac{1}{\tau_{p_n}} - \lambda_{p_n}} + \delta.$$

The reasoning here is that the machine is actively working at rate $\frac{1}{\tau_{p_n}}$ on part-type p_n during the entire time interval $[t_n + \delta, t_{n+1}]$. During this period it produces exactly as many parts as needed to clear both the $x_p(t_n) + \delta\lambda_{p_n}$ parts originally there at time $(t_n + \delta)$, plus the $\lambda_{p_n}(t_{n+1} - t_n - \delta)$ new parts that have arrived.

Hence, the net work in the system satisfies,

$$\begin{aligned} w(t_{n+1}) &= w(t_n) + \rho(t_{n+1} - t_n) - (t_{n+1} - t_n - \delta) \\ &= w(t_n) - (1-\rho)(t_{n+1} - t_n) + \delta. \end{aligned}$$

Clearly, if we can guarantee that there is an ϵ' such that $(t_{n+1} - t_n) \geq \epsilon' w(t_n)$, then the above equation will be stable, guaranteeing bounded buffer levels $\{w(t_n)\}$. Moreover, using the at most linear growth between t_n and t_{n+1}, one also obtains that $w(t)$ is bounded.

The rest of the proof simply consists of noting that the CAF condition (1.8) guarantees the existence of such an ϵ'. □

We have seen earlier that the class of Least Slack policies is a large class containing some useful policies for scheduling re-entrant lines. So also, for single machine systems, the class of CAF policies is a large class, which also contains policies which yield good performance.

1.7 A Lower Bound on Optimal Cost

In this section, we obtain a lower bound on the long-term average cost of any non-idling stable policy.

Consider any stable policy, i.e., one which ensures that $\sup_t x_i(t) \leq c$ for all i. Let T be a large time. We shall consider the trajectories of the system over the time interval $[0, T]$. Let n_i be the total number of production runs of part-type i in $[0, T]$, and let T_i be the total time of all the production runs for part-type i. Clearly, $x_i(T) = x_i(0) + \lambda_i T - \frac{1}{\tau_i} T_i$. Since $\sup_t x_i(t) \leq c$, we obtain $\sum_{i=1}^{P} T_i \geq \rho T - \hat{c}$ for

$$\hat{c} := c \sum_{i=1}^{P} \tau_i. \tag{1.9}$$

Note that $\hat{c}$ is bounded independently of the value of T.

Moreover, $\sum_{i=1}^{P} T_i + \delta \left(\sum_{i=1}^{P} n_i - 1 \right) \leq T$, since each set-up consumes δ time units. Hence

$$\delta \sum_{i=1}^{P} n_i \leq (1-\rho)T + \delta + \hat{c}. \tag{1.10}$$

Now we shall consider a single part-type "relaxation." We shall determine the smallest (there is a minimizer) possible value of

$$\frac{1}{T} \int_0^T c_i y_i(t) dt, \tag{1.11}$$

subject to just four requirements:

i) $y_i(t)$ is piecewise linear, and has a slope of either $\left(\lambda_i - \frac{1}{\tau_i} \right) < 0$ or $\lambda_i > 0$,

ii) there are at most n_i segments of slope $\left(\lambda_i - \frac{1}{\tau_i} \right)$,

iii) the total length of all segments with slope $\left(\lambda_i - \frac{1}{\tau_i} \right)$ is no more than T_i, and

iv) $0 \leq y_i(t) \leq c$ for all t.

Clearly, the actual trajectory $x_i(t)$ of part-type i satisfies these requirements. It however also satisfies the additional constraints induced by the fact that the machine cannot work on more than one part-type at a given time. This implies that two part types i and j cannot both have decreasing slopes at the same time. However, this last constraint is *relaxed* for $y_i(t)$.

It can be shown by a series of swapping arguments that the trajectory with smallest value of (1.11) has the form shown in Figure 6. Except for an initial and a final segment, the optimal trajectory consists of number m_i of alternating non-production and production runs, with all production runs being of equal lengths, and all non-production runs also being of equal length.

Let $a, b, u, v, u, v, \ldots, u, v, c$ be the corresponding lengths of the line segments, as in Figure 6. Then

$$\lambda_i(u+v) = \frac{1}{\tau_i} v$$

and

$$m_i(u+v) = T - (a+b+c).$$

Together these yield $u + v = \frac{v}{\rho_i} = \frac{(T-\bar{c})}{m_i - 1}$, where $\bar{c} := a + b + c$. Earlier, in (1.9), we saw that $\hat{c}$ was bounded independently of the value of T. Similarly,

due to the boundedness constraint $\sup_t x_i(t) \leq c$, the constant $\bar{c}$ is also bounded independently of the value of T.

Utilizing the values for u and v, the sum of the areas of the m_i triangles in Figure 6 is $\frac{1}{2}\frac{(T-\bar{c})^2\lambda_i(1-\rho_i)}{m_i}$. Thus, we obtain

$$\int_0^T x_i(t)dt \quad \geq \quad \int_0^T y_i(t)dt \geq \frac{1}{2}\frac{(T-\bar{c})^2\lambda_i(1-\rho_i)}{(m_i-1)}.$$

Hence we see that a lower bound on the cost $\frac{1}{T}\int_0^T \sum_{i=1}^P c_i x_i(t)dt$ is,

$$\min \sum_{i=1}^P \frac{1}{2}\frac{c_i(T-\bar{c})^2\lambda_i(1-\rho_i)}{Tm_i}$$

subject to the constraints

$$\begin{aligned} \delta \sum_{i=1}^P m_i &\leq (1-\rho)T + \delta + \hat{c} \\ m_i &\geq 0. \end{aligned}$$

Above, we have used the constraint (1.10). We have also relaxed the constraint that m_i be an integer.

This convex program can be explicitly solved, to yield

$$m_i \quad = \quad k\sqrt{c_i\lambda_i(1-\rho_i)}$$

where the constant k satisfies

$$k \quad = \quad \frac{(1-\rho)T + \delta + \hat{c}}{\delta \sum_{i=1}^P \sqrt{c_i\lambda_i(1-\rho_i)}}.$$

Hence

$$\frac{1}{T}\int_0^T \sum_{i=1}^P c_i x_i(t)dt \quad \geq \quad \frac{(T-\bar{c})^2\delta\left[\sum_{i=1}^P \sqrt{c_i\lambda_i(1-\rho_i)}\right]^2}{2T\left[(1-\rho)T+\delta+\hat{c}\right]}.$$

The constants $\hat{c}$ and $\bar{c}$ are bounded as $T \to \infty$. Taking the limit, we obtain the following lower bound on long-term average cost.

Theorem: Lower Bound on Long-Term Average Cost of Any Non-Idling Stable Policy. Consider a single machine system, satisfying the capacity condition (1.7). The long term average cost of any stable, non-idling scheduling policy is lower bounded as follows:

$$\liminf_{T\to+\infty} \frac{1}{T}\int_0^T \sum_{i=1}^P c_i x_i(t)dt \quad \geq \quad \frac{\delta\left[\sum_{i=1}^P \sqrt{c_i\lambda_i(1-\rho_i)}\right]^2}{2(1-\rho)}$$

1.8 A Good CAF Policy

In our definition of CAF policies, let us loosen the requirement (1.8), to

$$x_{p_n}(\tau_n) \quad \geq \quad \epsilon \sum_{i=1}^{P} x_i(\tau_n) - c \tag{1.12}$$

where $c \geq 0$ is any constant. Earlier, we took $c = 0$. Let us continue to call policies which satisfy (1.12) as CAF policies. Even with this extension, the CAF policies are stable. Moreover, as earlier, an upper bound on buffer levels can be obtained.

Within the class of CAF policies, one can design good scheduling policies. For our intuition, we turn to the proof of the lower bound on the optimal cost. There, we saw that the peak value of the triangles for part-type i in Figure 6 is $v\left(\frac{1}{\tau_i} - \lambda_i\right)$, where $v = \frac{\rho_i(T-\bar{c})}{m_i - 1}$. Moreover, $m_i = k\sqrt{c_i\lambda_i(1-\rho_i)}$. These relations suggest that the peak value of $x_i(t)$ is roughly proportional to $\frac{\rho_i\left(\frac{1}{\tau_i}-\lambda_i\right)}{\sqrt{c_i\lambda_i(1-\rho_i)}} = \sqrt{\frac{\lambda_i(1-\rho_i)}{c_i}}$. This shows that the "superoptimal" policy, shown in Figure 6, that attains the lower bound, attempts to equalize peak values of $\frac{x_i\sqrt{c_i}}{\sqrt{\lambda_i(1-\rho_i)}}$ for all i. Thus, it demonstrates what are desirable "scalings" for the buffers.

Since the peak value of a buffer is obtained just after a set-up to it, the following CAF policy suggests itself. At the end of a clearing run, choose a part-type i, for which $(x_i + \lambda_i\delta)\frac{\sqrt{c_i}}{\sqrt{\lambda_i(1-\rho_i)}}$ is maximal. Since the above expression can be written as $\sqrt{\frac{c_i}{\lambda_i(1-\rho_i)}}x_i(\tau_n) + \frac{\delta\sqrt{c_i\lambda_i}}{\sqrt{(1-\rho_i)}}$, we see that it is a CAF policy.

One may simulate this CAF policy, and compare its performance with respect to the lower bound.

Example:

Consider a system producing 10 part types with $\lambda_1 = 0.24$, $\lambda_2 = 0.16$, $\lambda_3 = 0.12$, $\lambda_4 = 0.09$, $\lambda_5 = 0.08$, $\lambda_6 = 0.06$, $\lambda_7 = 0.04$, $\lambda_8 = 0.03$, $\lambda_9 = 0.02$, $\lambda_{10} = 0.01$, and $\tau_1 \equiv 1$ for all i. Let $c_i \equiv 1$ for all i, and $\delta = 1$. Then

$$\frac{\text{Average cost in simulation of } 10,000 \text{ production runs}}{\text{Lower bound on average cost}} = 1.0225$$

was observed. Thus, the CAF policy designed by us yielded a cost no more than 2.25% away from the optimal.

1.9 Non-Acyclic Manufacturing Systems with Set-Up Times

Let us now turn to the fairly general model of a manufacturing system shown in Figure 7. There are M machines labeled $1, 2, \ldots, M$ in the system which produces P types of parts labeled $1, 2, \ldots, P$. Parts of type i arrive at rate λ_i. They visit the machines $\mu_{i,1}, \mu_{i,2}, \ldots, \mu_{i,n_i}$ in order, where $1 \leq \mu_{i,j} \leq M$ for each j. They exit from μ_{i,n_i} as a finished product. We suppose that at machine $\mu_{i,j}$, parts of type i require $\tau_{i,j}$ units of processing time, and that they await processing in a buffer labeled $b_{i,j}$. Since the directed graph obtained from the union of the routes of all the part–types may contain cycles, we say that such a manufacturing system is *non–acyclic*.

We will allow two features. First, we suppose that a machine incurs a set-up time $\delta_{b,b'}$ when it switches from processing parts in buffer b to parts in buffer b'.

We will also allow a transportation delay when parts move between buffers. We will simply assume that the transportation delay is bounded by some τ.

Our goal now is to schedule this entire system in such a way that all buffer levels are bounded. Such a scheduling policy will be called *stable*. Clearly, the vector of arrival rates $(\lambda_1, \ldots \lambda_P)$ should be within the capacity of every machine to handle, if such a goal is to be attained. Hence we suppose that the *capacity* condition,

$$\rho_m \quad := \sum_{\{(i,j):b_{i,j} \text{ is served by } m\}} \lambda_i \tau_{i,j} < 1 \text{ for every machine } m. \quad (1.13)$$

There are some further attributes that we may desire of the scheduling policy. First, it should not be stymied by the computational complexity, even for large systems. Second, we may also wish it to be robust; for example tolerant to the bound on the transportation delay.

Below we will provide examples of such policies, which still yield bounded buffer levels. Our scheduling policies will be "distributed." They will neither require any information exchange between machines, nor any coordination of action across machines.

First we show the intricacies involved in establishing stability of non–acyclic systems.

Example: A System Where All Clearing Policies are Unstable.

Consider the system shown in Figure 8. There are two machines, labeled M1 and M2. There is only type of part, which arrives at rate 1. The parts visit the four buffers b_1, b_2, b_3 and b_4 in that order, located at machines M1, M2, M2 and M1. Parts in buffer b_i take τ_i units of processing time. For simplicity we will assume that set-up times are zero. We assume that $\tau_1 + \tau_4 < 1$ and

$\tau_2 + \tau_3 < 1$, which imply that the capacity condition (1.13) is satisfied. However, we suppose that $\tau_2 + \tau_4 > 1$.

Let us consider perhaps the simplest class of distributed scheduling policies. Suppose that each machine simply adopts a clearing policy, which continues to work on a buffer until it is empty, and another buffer is non-empty. (In this case, there can only be one other non-empty buffer).

We will now show that such a policy is unstable. Consider the initial state $x(t) = (x, 0, 0, 0)$, where $x_i(t)$ is the number of parts in buffer i. At time $t_1 := \frac{x\tau_1}{1-\tau_1}$, buffer b_1 clears, and $x(t_1) = \left(0, x - \left(\frac{1}{\tau_2} - 1\right) t_1, \frac{t_1}{\tau_2}, 0\right)$. After t_1, b_4 is empty and so M1 continues to work on b_1, but at the reduced rate of 1, since it is *starved* of parts. At time $t_2 := t_1 + \frac{x_2(t_1)\tau_2}{1-\tau_2}$, buffer b_2 is cleared, and $x(t_2) = (0, 0, x + t_2, 0)$. At time $t_3 := t_2 + \tau_3 x_3(t_2)$, buffer b_3 clears, and $x(t_3) = \left(t_3 - t_2, 0, 0, \left(\frac{1}{\tau_3} - \frac{1}{\tau_4}\right)(t_3 - t_2)\right)$. Finally (for our analysis), at time $t_4 := t_3 + \tau_4 x_4(t_3)$, buffer b_4 also clears, and the state is $x(t_4) = \left(\frac{\tau_4}{1-\tau_2} x, 0, 0, 0\right)$.

Thus, after some time the system has returned to an amplified version of its original state; it is an amplification since $\frac{\tau_4}{1-\tau_2} > 1$. This process continues, and so we see that the buffer levels are unbounded. □

The cause of instability is the enforced starvation of M2 during intervals such as $[t_1, t_2]$. Thus, the scheduling policy will have to somehow prevent excessive starvation.

There are many strategies for doing so. We will now describe a class of scheduling policies, that we call *Universally Stabilizing Supervisors* (USS), which guarantee stability.

The key idea is to enforce some sort of high level mechanism which provides some "discipline." We will assume that there is some "lower level" scheduling policy, which is in operation. However, at certain times, a supervisor steps in to intervene. What we show is that by implementing a few simple rules the supervisor can guarantee stability, irrespective of the details of the "lower level" scheduling policy. Moreover, the supervisor is really a set of *distributed* supervisors, one for each machine.

For each machine m, let γ_m be a number which is large enough to satisfy

$$\gamma_m(1 - \rho_m) \quad > \quad \sum_{\{b : b \text{ is served by } m\}} \max_{\{b' : b' \text{ is a buffer served by } m\}} \delta_{b'b}.$$

Also, for each buffer b_{ij}, we choose an arbitrary nonnegative number z_{ij}.

The operation of the supervisor is governed by the following five rules.

The Truncation Rule: The processing time for buffer b_{ij} is truncated at machine m after $\gamma_m \lambda_i \tau_{ij}$ time units.

Rule for entering an FCFS Queue: Each machine m maintains an FCFS queue Q_m of *buffers* that it is serving. A buffer b_{ij} located at machine m enters Q_m when i) its buffer level exceeds z_{ij} *and* ii) it is not being processed or set-up.

Serving the FCFS Queue of Buffers: If the queue Q_m of buffers is non-empty, then it is served in the FCFS order, i.e., the order in which buffers enter it.

Rule for leaving the FCFS queue A buffer leaves the queue Q_m when it is taken up for processing.

The Processing Time for buffers in Q_m When a buffer b_{ij} in Q_m is taken up for processing, it is given exactly $\gamma_m \lambda_i \tau_{ij}$ time units of processing, unless it is cleared even before this.

One may note the following features. As long as Q_m is empty, the lower level scheduling policy is free to proceed as it pleases. The supervisor only intervenes when Q_m is non-empty. The numbers z_{ij} determine the frequency with which the supervisor intervenes. If the z_{ij}'s are small, the supervisor intervenes frequently. In the extreme case, if the z_{ij}'s and γ_m's are chosen large, and the lower level scheduling policy already stabilizes the system, then the supervisor never intervenes.

Irrespective of the lower level policy, the USS guarantees stability. What is not known, however, is how to obtain good performance, i.e., small values of buffer levels.

1.10 Concluding Remarks

We have shown how one may design dynamic scheduling policies through a combination of intuition and analysis. We have also analyzed their properties. In all cases, our scheduling policies are easy to implement. We believe that more work is clearly necessary to further develop our understanding about the dynamics of manufacturing systems, and also to provide a more definitive analysis, especially concerning the performance of scheduling policies. Instead of adopting the traditional deductive mode, where one attempts to deduce the optimal policy from, say, the theory of optimal control – which usually is intractable – more attention should be paid to what are usually called "heuristics." However, that term has come to acquire a pejorative meaning because it is frequently not backed up either by any analysis or a careful comparative evaluation. Such heuristics need more careful justification and analysis than has been traditional in manufacturing systems. Finally, careful comparative analysis of heuristics is necessary.

Notes

We have avoided all references to the literature in this Chapter. We make no attempt to properly attribute results, and apologize to authors in advance. A few pertinent references, which could be used to begin delving into the literature, are provided below:

1. An excellent comparative analysis of various scheduling policies, and a model of a semiconductor manufacturing plant is provided in Wein [1]. The Fluctuation Smoothing policies of Section 3, and the simulation results are from Lu, Ramaswamy and Kumar [2]. A statistical analysis can also be found there. The argument that reducing the variations in the interarrival times leads to smaller delays can be made precise by considering the convexity properties of Lindley's equation; see Ross [3]. The appealing interpretation of FSMCT, as attempting to equalize and reduce the total downstream shortfall from each buffer, is from Lu and Kumar [4]. This is different from an alternative policy of making each buffer's value tend to its own mean value, proposed by Li [5, 6].

2. The stability analysis of Least Slack scheduling policies in Section 4 is from Lu and Kumar [7]. There, the FBFS policy is also shown to be stable. Also some stable buffer priority policies are provided for systems consisting of multiple re-entrant lines. A broader account of re-entrant lines can be found in Kumar [8]. Recently, Kumar and Meyn [9] have developed linear programming tests to establish the stability of queueing networks and scheduling policies. These tests have been extended more recently in Kumar and Meyn [10]. Also, Rybko and Stolyar [11], Dai [12], and Chen [13], have shown that one may establish the stability of systems by establishing the stability of associated fluid models. The stability of such fluid models, and their connection to the linear program tests, is also examined in Kumar and Meyn [10].

3. Recently Bertsimas, Paschalidis and Tsitsiklis [14, 15] and Kumar and Kumar [16] have shown how one may obtain performance bounds for queueing networks and scheduling policies. There, bounds can be found for the performance of any policy, as well as buffer priority policies. More recently, Kumar and Meyn [10] show that the stability and performance problems are related by duality.

4. The treatment of set-up times in Sections 5 and 6 is from Perkins and Kumar [17] and Kumar and Seidman [18]. Chase and Ramadge [19] have shown that sometimes one can obtain a lower cost by selectively idling, and obtain a performance bound, as well as a good policy. In Perkins and Kumar [17] and Kumar and Seidman [18], other stable distributed scheduling policies are also developed. Also, sufficient conditions for the

stability of distributed CAF policies are presented in [18]. Further recent results on stability and performance can be found in Perkins, Humes and Kumar [20].

The research reported here has been supported by the National Science Foundation under Grant No. NSF-ECS-90-25007, and the Joint Services Electronics Program under Contract No. N00014–90–J1270.

Bibliography

[1] L. M. Wein. Scheduling semiconductor wafer fabrication. *IEEE Transactions on Semiconductor Manufacturing*, 1(3):115–130, August 1988.

[2] Steve C. H. Lu, Deepa Ramaswamy, and P. R. Kumar. Efficient scheduling policies to reduce mean and variance of cycle–time in semiconductor manufacturing plants. Technical report, University of Illinois, Urbana, IL, 1992.

[3] Sheldon M. Ross. *Stochastic Processes*. John Wiley, New York, 1982.

[4] Steve H. Lu and P. R. Kumar. Fluctuation smoothing scheduling policies for queueing systems. To appear in *Proceedings of the Silver Jubilee Workshop on Computing and Intelligent Systems*, Bangalore, India, December 1993.

[5] S. Li. Private Communication. 1992.

[6] S. Li. An introduction to basic principles of production. In *Semiconductor Manufacturing Technology Workshop*, pages 3–20, Hsinchu, Taiwan, R.O.C., March 1993.

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

[8] P. R. Kumar. Re–entrant lines. *Queueing Systems: Theory and Applications: Special Issue on Queueing Networks*, 13(1–3):87–110, May 1993.

[9] P. R. Kumar and Sean Meyn. Stability of queueing networks and scheduling policies. Technical report, C. S. L., University of Illinois, 1993.

[10] P. R. Kumar and Sean Meyn. Duality and linear programs for stability and performance analysis of queueing networks and scheduling policies. Technical report, C. S. L., University of Illinois, 1993.

[11] A. N. Rybko and A. L. Stolyar. On the ergodicity of stochastic processes describing open queueing networks. *Problemy Peredachi Informatsii*, 28:2–26, 1991.

[12] J. G. Dai. On the positive Harris recurrence for multiclass queueing networks: A unified approach via fluid models. Technical report, Georgia Institute of Technology, 1993.

[13] Hong Chen. Fluid approximations and stability of multiclass queueing networks I: Work conserving disciplines. Technical report, University of British Columbia, 1993.

[14] D. Bertsimas, I. Ch. Paschalidis and J. N. Tsitsiklis. Scheduling of multiclass queueing networks: Bounds on achievable performance. In *Workshop on Hierarchical Control for Real–Time Scheduling of Manufacturing Systems*, Lincoln, New Hampshire, October 16–18, 1992.

[15] D. Bertsimas, I. Ch. Paschalidis and J. N. Tsitsiklis. Optimization of multiclass queueing networks: Polyhedral and nonlinear characterizations of achievable performance. Laboratory for Information and Decision Systems and Operations Research Center, M. I. T., December 1992.

[16] S. Kumar and P. R. Kumar. Performance bounds for queueing networks and scheduling policies. Technical report, Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1992. To appear in *IEEE Transactions on Automatic Control*, August 1994.

[17] J. R. Perkins and P. R. Kumar. Stable distributed real-time scheduling of flexible manufacturing/assembly/disassembly systems. *IEEE Trans. Automat. Control*, AC-34(2):139–148, February 1989.

[18] P. R. Kumar and T. I. Seidman. Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Trans. Automat. Control*, AC-35(3):289–298, March 1990.

[19] C. J. Chase and P. J. Ramadge. On the real time control of flexible manufacturing systems. In *Proc. IEEE 28th Conf. on Decision and Control*, pages 2026–2027, Tampa, FL, 1989.

[20] J. R. Perkins, C. Humes, Jr., and P. R. Kumar. Distributed control of flexible manufacturing systems: Stability and performance. Technical report, University of Illinois, Urbana, IL, 1993. To appear in *IEEE Transactions on Robotics and Automation*, 1994.