The Destabilizing Effect of Blocking due to Finite Buffering Capacity in Multi-class Queueing Networks*

Spyros A. Reveliotis
School of Industrial & Systems Engineering
Georgia Institute of Technology
765 Ferst Drive
Atlanta, GA 30332

Tel #: (404)-894-6608
Fax #: (404)-894-2301
email: *spyros@isye.gatech.edu*

Abstract

This paper demonstrates that blocking due to finite buffering capacity of the system workstations can be a destabilizing factor for distributed scheduling policies applied in multi-class queueing networks, in addition to the currently identified sources of instability. The issue is important since, in practice, all systems modeled by such networks operate under buffering capacity constraints.

Keywords: Multi-class Queueing Networks, Stability, Distributed Scheduling Policies, Reentrant Lines

*The paper is currently under review for publication in the IEEE Trans. on Automatic. Control. Please, do not cite or reproduce without the author's permission.

1 Introduction

This paper revisits the problem of stability of distributed scheduling policies in multi-class queueing networks (QN's). The issue is of considerable practical importance, especially in the semiconductor manufacturing industry where the complexity and the dynamic nature of the operational environment makes *dispatching rules* [PI77] the most practical and convenient scheduling approach [LK91, Kum94b]. In that context, stability essentially characterizes the efficiency of the different distributed scheduling schemes with respect to throughput maximization, an objective of strategic importance for the expensive and competitive semiconductor fab line. Specifically, it has been recently shown that a number of distributed scheduling schemes, including the well-known and widely-used First-Come-First-Serve (FCFS) dispatching rule, might fail to deliver the maximal production capacity of the system, as defined by its bottleneck workstations [KS90, Sei94]. The logical explanation of the instability of these policies is provided in [DVV96]. Furthermore, on the positive side, the work presented in [LK91] and later in [Kum94a] has shown that some classical dispatching rules – namely, a family known as *Generalized Least Slack* policies which includes the Earliest Due Date (EDD), Shortest Remaining Processing Time (SRPT), Last Buffer First Serve (LBFS), and Earliest Arrival (EA) – are stable.

In this technical note, we show, through an example, that Generalized Least Slack policies can become unstable (i.e., inefficient) if the finite buffering capacity of the system workstations and/or material handling is taken into account. Indeed, in the theoretical analysis undertaken in [LK91, Kum94a], it is assumed that the operation of the system workstations is *non-idling*, i.e., every workstation is allowed to idle only if there is no additional Work-In-Process (WIP) inventory waiting for service. Hence, *starvation* of the bottleneck resources is the main cause of instability. Under finite buffering capacity, a workstation can be also forced to idleness due to *blocking*, i.e., due to the inability of its currently finished parts to advance to their next stage and release this resource.

The rest of the paper is as follows: Section 2 describes the structure of system to be considered in the example development, details its real-time operation, and the job release mechanism. It also provides a formal characterization of the policy stability concept. Section 3 provides a specific unstable cycle which can be developed under operation by the LBFS policy, and establishes its inefficiency. Finally, Section 4 concludes the paper presentation, and raises issues for further investigation.

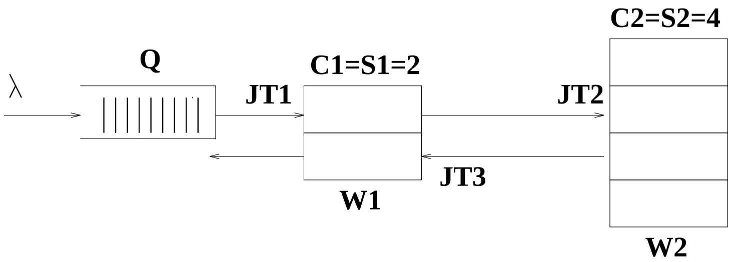

Figure 1: The considered re-entrant line

2 Preliminaries

The system The system considered in this paper is the *re-entrant line* of Figure 1. It consists of two workstations, W_1 and W_2, with workstation W_1 being a pool of two identically configured machines, and workstation W_2 being a pool of four identically configured machines. Each of these machines can accommodate one job at a time, while no additional buffering capacity is provided at any of these stations. Hence, the buffering and processing capacity for these two stations are: $C_1 = S_1 = 2$, and $C_2 = S_2 = 4$. In its current configuration, this re-entrant line supports the production of a job with three processing stages, JT_1, JT_2 and JT_3; stages JT_1 and JT_3 are executed on workstation W_1, while stage JT_2 is executed on workstation W_2. Processing times, τ_i, $i = 1, 2, 3$, for these stages are *deterministic*, with $\tau_1 = \tau_3 = 1.0$ and $\tau_2 = 4.0$.[1]

The system *state* is defined by *the number of job instances executing the different processing stages, plus their remaining processing times*. Specifically, in the following we shall use the notation $JT_i : n(r)$ to denote that n jobs are currently executing stage JT_i, with a remaining processing time equal to r. Also, let $x_i(t)$, $i = 1, 2, 3$ denote the collective number of jobs executing stage JT_i at time t (i.e., irrespective of their remaining processing time).

Deadlocks and Deadlock Avoidance: An immediate implication of the definition of x_i's and the finite buffering capacity of the system workstations is that:

$$\begin{aligned} x_1(t) + x_3(t) &\leq C_1 = 2 \\ x_2(t) &\leq C_2 = 4 \end{aligned} \tag{1}$$

However, notice that operating the system under the naturally arising constraints of Equation 1 is *too* permissive, since it does not guard against situations similar to that depicted in

[1] Alternatively, in a more realistic context, one can think of the two units of workstation W_1 as the vehicles of a single-loop AGV system, transferring parts between the entry buffer and the processing units of workstation W_2.

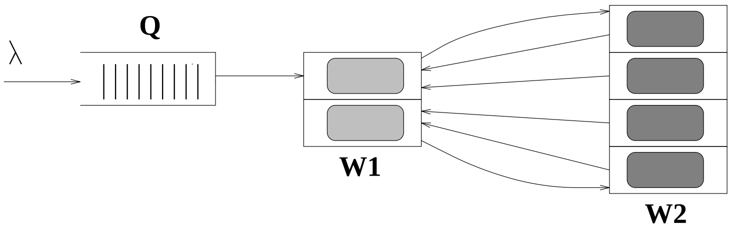

Figure 2: A system deadlock

Figure 2. The problem with the resource allocation status depicted in Figure 2 is that the system workstations are filled to capacity with jobs requiring a free unit of capacity on the same stations in order to advance to their next processing stage. Therefore, the system is permanently stuck, a situation characterized as a manufacturing system *deadlock* [RLF98].

To guard against the occurrence of deadlocks, additional restrictions, in terms of the system resource allocation status x_i, $i = 1, 2, 3$, must be imposed on the system operation; these restrictions are collectively known as *Deadlock Avoidance Policies (DAP's)*, and their effective analysis and design is the topic of an emerging research area known as the *Structural Control (SC)* of (automated) production systems [RLF98]. Although in the general case the analytical characterization of the optimal DAP for a given configuration is an intractable problem [RLF98], for the small system of Figure 1, it is easy to see, by inspection, that the only arising deadlock is the one depicted in Figure 2. Furthermore, the occurrence of this deadlock can be prevented in the *least restrictive* manner, by imposing the additional constraint during the system operation:

$$x_1(t) + x_2(t) \;\leq\; C_1 + C_2 - 1 \;=\; 5 \tag{2}$$

In other words, Equation 2 is the analytical characterization of the optimal DAP for the small system of Figure 1; a more theoretical justification for this claim can be found in [RLF98].

The job-release mechanism Following the assumptions made in [LK91], we consider that jobs are released into the system at an *average* rate λ, while the actual release process might be characterized by some bounded *burstiness*, γ. Analytically, if $u(t)$ denotes the total number of parts released into the system over the interval $[0, t]$, it must hold [LK91]:

$$u(t) - u(s) \leq \lambda(t - s) + \gamma \qquad \text{for all } 0 \leq s \leq t \tag{3}$$

From a more practical perspective, the "burstiness" introduced in the job-release process can model either fluctuations in the overall supply chain, or situations in which the entire production requirements are released at the shop-floor at the beginning of the production period (e.g., the beginning of the shift). For the system of Figure 1, it is assumed that released jobs that have not been loaded yet into the system, are accommodated in a large – theoretically infinite-capacity – storage area, denoted by buffer Q. Hence, buffer Q expresses the *backlog* between the job releases and the actual system production, in terms of number of unprocessed parts.

Stability Considerations: As it was stated in the Introduction, the issue of the stability of a scheduling policy $\mathcal{P}$ for the re-entrant line environment essentially considers the relative performance of policy $\mathcal{P}$ to that of the *optimal* scheduling policy, with the performance measured in terms of *throughput*. In the analysis undertaken in [LK91, Kum94a], the maximal attainable throughput is characterized by the maximal production rate sustained at the *bottleneck* workstations, i.e., those workstations for which the applied job-loading pattern poses the maximal processing requirements. In the case of structurally controlled systems, the constraints imposed by the applied DAP introduce additional but unavoidable limitations to the maximal attainable production rates. In fact, these constraints are conceptually equivalent to the *virtual bottlenecks*, introduced in [DVV96] in order to explain the instability of certain distributed scheduling policies. Hence, in the light of these remarks, an upper bound to the maximal attainable throughput for the re-entrant line of the system of Figure 1 is established by Constraints 1 and 2, as:

$$\begin{aligned}\bar{\lambda} &= \min\{\frac{C_1}{\tau_1+\tau_3}, \frac{C_2}{\tau_2}, \frac{C_1+C_2-1}{\tau_1+\tau_2}\} \\ &= \min\{\frac{2}{1+1}, \frac{4}{4}, \frac{5}{1+4}\} \;=\; 1.0 \qquad (4)\end{aligned}$$

In the next section we show that although it is possible to define a scheduling policy able to sustain the maximal production rate of 1.0, computed above, the system operation under the LBFS dispatching rule can enter a "limit cycle" which is characterized by a considerably lower production rate, and therefore, under this scheduling scheme, the system might not be able to deliver its maximal performance. To formally characterize this phenomenon, we provide the following definition of *(in-)stability*; this definition extends naturally the definition of stability provided in [LK91] to structurally controlled re-entrant lines, and it is conceptually equivalent to the stability definition provided in [Sha97].

Definition 1 *Let the* sojourn time *of a given part denote the time it spends into the system, from its release, till it has finished processing.*

(i) The system operation under policy $\mathcal{P}$ and for feeding rate λ is stable*, if the sojourn time of every part processed over an* infinite *horizon is* bounded*. Clearly, under stable system operation, the content of the released part-accommodating buffer Q, considered over an* infinite *horizon, remains bounded.*

(ii) The processing capacity *of a considered re-entrant line is defined by the* maximal *feeding rate for which there exists a scheduling policy $\mathcal{P}^*$, able to control the line in a stable mode.*

(iii) A scheduling policy $\mathcal{P}$ *is* stable *for a given re-entrant line, if it can provide the line processing capacity in a stable fashion.*

(iv) A scheduling policy $\mathcal{P}$ *is* stable *for the entire class of re-entrant line production systems, if its instantiation on every re-entrant line is stable.*

3 The Instability of the LBFS Policy

To obtain the main result of this paper, consider the system of Figure 1 at time t_0 (to be taken as time 0, in the following, w.l.o.g.), where the two workstations are idle and empty of any jobs, but a sufficiently large backlog has accumulated in the receiving buffer Q. Furthermore, buffer Q is constantly fed with rate $\bar{\lambda} - \epsilon$, where $\epsilon \to 0$. Applying the notation $n(r)$ defined in Section 2, it is easy to see that application of the LBFS policy results to the following state sequence (operational trace):

t	J_1	J_2	J_3
0	2(1)	0	0
1	2(1)	2(4)	0
2	1(1)	2(4) 2(3)	0
3	1(0)	2(3) 2(2)	0
5	0	1(4) 2(1)	2(1)
6	0	1(3)	2(1)
7	2(1)	1(2)	0
8	2(1)	2(4) 1(1)	0
9	1(1)	2(4) 2(3)	1(1)
10	1(0)	2(3) 2(2)	0

Notice that the state reached at time-step 10 is identical to that of time-step 3. Hence, while processing the backlog of buffer Q, the system runs through the cycle defined by states 3–9, above. This cycle has a period of 7 time units, during which only 5 jobs are processed. Therefore, the throughput attained over this period is $TH = 5/7 < 1 = \bar{\lambda}$. Assuming that the initial backlog is large enough, and that after time t_0 the system is fed with a relatively constant rate λ such that $5/7 < \lambda \leq 1.0$, – which is consistent with the assumed job-release mechanism – the content of buffer Q will keep constantly increasing, maintaining the system operation in the afore-stated cycle. Hence, the long-run production rate of the system will be $TH(LBFS) = 5/7$, significantly less than the estimated upper bound of 1.0.

To complete the example, however, we must establish that the estimated maximal throughput $\bar{\lambda} = 1.0$ is, indeed, attainable by some optimal scheduling policy P^*. For the re-entrant line of Figure 1, it is easy to see that such an optimal policy is the scheduling policy that drives the system to the "limit cycle":

t	J_1	J_2	J_3
t_i	1(1)	1(4) 1(3) 1(2) 1(1)	1(1)
t_{i+1}	1(1)	1(4) 1(3) 1(2) 1(1)	1(1)

This cycle finishes one part at each time unit, and furthermore, the resource allocation status, $\mathbf{x}^*$, maintained by this cycle is easily reachable from state $\mathbf{x}^0$, in which the system is idle and empty of any jobs.

4 Discussion

Notice that the ability of the optimal scheduling policy, $\mathcal{P}^*$, to provide the maximal possible throughput, $\bar{\lambda}$, is based on the establishment of a delicate flow that constantly keeps all the system bottlenecks busy to their capacity. On the other hand, LBFS policy, in its effort to push running jobs out of the system, eventually blocks the operation of workstation W_1, and the virtual station defined by Equation 2 (Period 3–5). This subsequently leads to starvation of the workstation W_2, and additional blocking of the virtual station established by Eq. 2 (Periods 5-9), until we reach again the cycle-entering status of Period 10. In this way, the underutilization of the system resources perpetuates itself, leading eventually to reduced throughput.

Furthermore, the stability of the LBFS scheduling policy is used in [LK91, Kum94a] as the stepping-stone for establishing the stability of the broader class of the Generalized Least Slack policies. Hence, the instability example of Section 3 implies that the stability of all these dispatching rules must be re-considered in the context of the capacitated, structurally controlled, re-entrant lines. At this point, it is conjectured that, to the extent that these policies ignore the interaction effects between the system workstations arising by the mechanism of *blocking*, they will be found to be unstable. In that case, one issue to be considered is the degree of the policy instability, i.e., how much the throughput eventually attained by these policies varies from the actual system capacity. Structural system features that might affect and characterize this variation is an additional research issue. Finally, it must be noticed that stabilizing these policies is a non-trivial issue, since, as it is shown in ([SH96], Example B), simply increasing the workstation buffering capacity does not guarantee improved stability; on the contrary, it might destabilize the system even further.

On the basis of these remarks, we believe that *distributed tracking schemes*, similar to those developed in [Sha95], seem to hold more promise. The formulation of the appropriate continuous-flow (relaxation) models, and the stability of the applied tracking mechanism(s) are issues that need further investigation; currently, they constitute another part of our current research agenda.

References

[DVV96] J. G. Dai and J. H. Vande Vate. Virtual stations and the capacity of two-station queueing networks. Technical report, ISyE, Georgia Tech, 1996.

[KS90] P. R. Kumar and T. I. Seidman. Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Trans. Autom. Control*, 35:289–298, 1990.

[Kum94a] P. R. Kumar. Scheduling manufacturing systems of re-entrant lines. In D. D. Yao, editor, *Stochastic Modeling and Analysis of Manufacturing Systems*, pages 325–360. Springer-Verlag, 1994.

[Kum94b] P. R. Kumar. Scheduling semiconductor manufacturing plants. *IEEE Control Systems Magazine*, 14–6:33–40, 1994.

[LK91] S. H. Lu and P. R. Kumar. Distributed scheduling based on due dates and buffer priorities. *IEEE Trans. on Aut. Control*, 36:1406–1416, 1991.

[PI77] S. S. Panwalkar and W. Iskander. A survey of scheduling rules. *Oper. Research*, 25:45–61, 1977.

[RLF98] S. A. Reveliotis, M. A. Lawley, and P. M. Ferreira. Structural control of large-scale flexibly automated manufacturing systems. In C. T. Leondes, editor, *Computer Aided and Integrated Manufacturing Systems: Techniques and Applications*. GORDON & BREACH, 1998.

[Sei94] T. I. Seidman. "first come first serve" can be unstable! *IEEE Trans. on Automatic Control*, 39:2166–2171, 1994.

[SH96] T. I. Seidman and C. Jr. Humes. Some kanban-controlled manufacturing systems: A first stability analysis. *IEEE Trans. on Aut. Control*, 41:1013–1018, 1996.

[Sha95] A. Sharifnia. Stability and performance of a simple distributed tracking policy for production control of manufacturing systems. *IEEE Trans Autom. Control*, 40:1109–1113, 1995.

[Sha97] A. Sharifnia. Instability of the join-the-shortest-queue and fcfs policies in queueing systems and their stabilization. *Operations Research*, 45:309–314, 1997.