Performance of a serial-batch processor system with incompatible job families under simple control policies

John Benedict C. TAJAN[1], **Appa Iyer SIVAKUMAR**[2], and **Stanley B. GERSHWIN**[3]

[1] Singapore-MIT Alliance, Innovations in Manufacturing Systems and Technology,
(phone: (65)-9194-9944; e-mail: taja0001@ ntu.edu.sg)
[2] Mechanical and Aerospace Engineering Department, Nanyang Technological University, Singapore,
(e-mail: msiva@ntu.edu.sg)
[3] Mechanical Engineering Department, Massachusetts Institute of Technology, MA, USA, (e-mail: sbgershwin@mit.edu)

Abstract—**A typical example of a batch processor is the diffusion furnace used in wafer fabrication facilities. In diffusion, silicon wafers are placed inside the furnace, and dopant is flown through the wafers via nitrogen gas. Then, a thin layer of silicon dioxide is grown, to help the dopant diffuse into the silicon. This operation can take 10 hours or more to finish processing, as compared to one or two hours for other wafer fab operations. Diffusion furnaces typically can process six to eight lots concurrently; we call the lots processed concurrently a batch. The quantity of lots loaded into the furnace does not affect the processing time. Only lots that require the same chemical recipe and temperature may be batched together at the diffusion furnace.**

We wish to control the production of a manufacturing system, comprised of a serial processor feeding the batch processor. The system produces different job types, and each job can only be batched together with jobs of the same type. More specifically, we explore the idea of controlling the production of the serial processor, based on the wip found in front of the batch processor. We evaluate the performance of our manufacturing system under several simple control policies under a range of loading conditions and determine that the concept of using the wip levels found in front of the batch processor to control the serial processor can reduce the mean cycle time. It is hoped that the results obtained from this small system could be extended to larger systems involving a batch processor, with particular emphasis placed on the applicability of such policies in wafer fabrication.

***Index* —Batch processor, control policy, multiple job types.**

I. Introduction

Consider the following problem: milk and ice cream arrive into a packaging facility and wait for an operator for final packaging. The packaged items are then transferred to a loading dock to wait for refrigerated trucks to distribute the items to various supermarkets. Each truck can carry up to Q items. A truck cannot transport both items at the same time, since milk and ice cream require different temperatures. Each time a truck is available, the driver has to decide which product to bring, and, if the number of items waiting at the dock is less than Q, the driver also has to decide whether to wait for more items, or to leave immediately. We wish to reduce the amount of time the items spend in the packaging facility.

The system described above consists of a serial processor (processor with unit capacity) feeding a batch processor (processor with capacity greater than one). Jobs of different types cannot be processed together in a batch processor. Batch processors are used in several industries, most notably in the metal works industry (Mathirajan, et al. [1]) and semiconductor manufacturing (Bhatnagar, et al. [2]) A survey on batch processor scheduling for the semiconductor industry by Mathirajan and Sivakumar [3] noted an increasing trend in research related to batch processor control for the semiconductor industry. This may be attributed to the extent batch processors are used in semiconductor manufacturing.

A typical example of a batch processor is the diffusion furnace used in wafer fabs. This operation can take 10 hours or more to finish processing, as compared to one or two hours for other wafer fab operations, according to Uzsoy [4]. Diffusion furnaces typically can process six to eight lots concurrently as a batch; and the batch quantity does not affect the processing time. Only lots that require the same processing parameters may be batched together.

To obtain intuition on system behavior, we characterize the performance of the system as system parameters are varied. We also explore the idea of controlling the production of the serial processor, based on the wip found in front of the batch processor. We compare the performance of our manufacturing system under several

Manuscript received November 21, 2005. This work is supported by the Singapore-MIT Alliance, Singapore.

J. C. Tajan is a PhD student with the Singapore-MIT Alliance under the Innovations in Manufacturing Systems and Technology program (phone: (65)-9194-9944; e-mail: taja0001@ ntu.edu.sg)

A. I. Sivakumar is an Associate Professor with the Mechanical and Aerospace Engineering Department, Nanyang Technological University, Singapore. (e-mail: msiva@ntu.edu.sg)

S. B. Gershwin is a Senior Research Scientist with the Mechanical Engineering Department, Massachusetts Institute of Technology, MA, USA. (e-mail: sbgershwin@mit.edu).

simple control policies to determine under what conditions certain concepts improve system performance.

A. Literature Review

There are two different types of batch processors. For the burn-in oven in semiconductor test, there are no job type restrictions in creating a batch. However, the processing time for the burn-in oven is equal to the maximum required processing time of all the jobs included in the batch. This is not covered in this review, nor will it cover the case where all future arrivals are known. The purpose of this review is to give readers an idea as to what has been done. A more exhaustive review on controlling batch processors in the context of semiconductor manufacturing is [3].

We divide the research involving batch processors with incompatible jobs into three classes. The first class considers the batch processor as a single entity. The second class refers to look-ahead methods, which assume a limited knowledge of future job arrivals. The third class considers multi-stage models involving a batch processor.

1. Control of batch processor as a single entity

Uzsoy [4] developed efficient algorithms for minimizing the makespan (CMAX), maximum lateness, and the weighted sum of completion times (SUMCI) of a single processor, when all jobs are already in front of the batch processor. It has been difficult to obtain efficient exact algorithms to extensions to this basic model; non-polynomial algorithms or heuristics have been proposed to optimize total tardiness (Mehta and Uszoy[5]), total weighted tardiness (Carlyle, et al.[6]), weighted SUMCI (Azizoglu and Webster[7]), and the number of tardy jobs (Jolai[8]), among others. Heuristics have also been proposed by Uszoy [4] and Balasubramaniam [9] to extend the results when there are *m* processors in parallel.

When future job arrivals are unknown, Deb and Serfozo [10] used a dynamic programming (DP) formulation to obtain the optimal minimum batch size (MBS) policy when all jobs are compatible at the batch processor; at any instance the batch processor is available, only load a batch if there are at least x jobs in the queue. Later extensions to this model include compound Poisson job arrivals (Deb [11]), limited buffer sizes (Aalto[12]), and limited buffer sizes combined with compound Poisson arrivals (Aalto[13]).

Duenyas and Neale [14] extended [10] to the case where jobs are incompatible, and suggested a heuristic that considered system stability and the estimated time for the next compatible job type arrival. Reference [15] proposed threshold policies for reducing the cycle time when the arrivals follow a Poisson process and the processing time is independent and identically distributed.

2. Batch processor look-ahead methods

Look-ahead methods are based on the premise that knowledge of future arrivals will result in a better batching decision at the batch processor. The advent of automated manufacturing execution systems have encouraged increasingly sophisticated look-ahead methods. Reviews of some look-ahead strategies can be found in [16] and [17].

Glassey and Weng [18] proposed a predicting the next *n* arrivals, and calculating the best time to load the batch to minimize the average waiting time. Although it assumed that all jobs are compatible at the batch processor, this has served as the prototype for all look-ahead methods. Fowler, et al. [19] proposed using only the next arrival information, and extended the results to handle incompatible jobs. Variations of this include using different objective functions (Weng and Leachman[20]) and taking the set-up requirements of a downstream machine into consideration, (Robinson, et al.[21], Van Der Zee [22], Solomon, et al.[23]). Van Der Zee [24] considered the case where there are *m* batch processors in parallel, each with different capacities and processing times.

3. Multi-stage models involving a batch processor

To facilitate discussion, β refers to a batch processor with capacity Q, δ refers to a discrete processor with unit capacity, and → refers to a precedence relation. Unless mentioned, it is assumed all jobs are waiting in front of the first processor, and all jobs are compatible.

Ahmadi, et al. [25] looked into reducing *CMAX* and SUMCI for the following systems: $\beta \rightarrow \delta$, $\beta_1 \rightarrow \beta_2$, $\delta \rightarrow \beta$. Kim and Kim [26] used genetic algorithms to improve on the heuristics suggested by [25] for a $\beta \rightarrow \delta$ system. Sung and Min [27] looked into earliness/tardiness measures for the same systems as [25]. Sung and Kim [28] developed polynomial algorithms for minimizing tardiness measures for the $\beta_1 \rightarrow \beta_2$ system configuration. This was later expanded into a flow line of *m* batch processors by Sung, et al. [29].

When future job arrivals are unknown, Gurnani, et al. [30] suggested threshold policies for a $\delta \rightarrow \beta$ system where there are *m* unreliable serial processors feeding the batch processor. Neale and Duenyas [31] developed DP formulations for minimizing SUMCI for the $\delta \rightarrow \beta$, $\beta \rightarrow \delta$ and $\delta \rightarrow \delta \rightarrow \beta$ systems. They determine that information from upstream processors is more valuable compared to information from downstream processors, although there is a large decay in the incremental improvements for each additional stage considered.

Due to the complexity of evaluating the control of multiple stage systems involving multiple job types, simulation has become the dominant method in handling incompatible job types at the batch processor. Rulkens, et al. [32] used simulation to determine the respective optimal minimum batch sizes for each lot priority dispatching rule, while Akcali, et al. [33] used simulation to evaluate which set of dispatch rules improve fab cycle time.

B. Research Significance

The semiconductor manufacturing industry has quickly become one of the main drivers of the global economy; worldwide semiconductor revenues reached $100.8 billion in the first half of 2004 [34]. However, capacity expansion in the semiconductor industry is both expensive and timeconsuming. This is particularly true for wafer fabs. High utilization rates leave little room for error in terms of production control; it is estimated that capacity utilization for wafer fabs in the second quarter of 2005 is 88.8% [34]. This magnifies the importance of production control in keeping a company competitive.

A typical lot passes through several batch processors before exiting the wafer fab. The control of these batch processors has significant implications to the performance of wafer fabs, for several reasons. These include:

- Batch processors typically have long processing times, compared to other operations. Yield is inversely proportional to the cycle time of the lot; this delay not only increases the probability of not meeting the customer due date, but can also result in yield loss.
- The throughput of batch processors is dependent on the control policies used. A control policy that loads only two lots when the maximum capacity is four lots effectively halves the maximum throughput of the batch processor. Although photolithography is generally acknowledged to be the bottleneck in wafer fabrication, a poor control policy at the batch processors can cause the furnace to act as a situational bottleneck and limit the throughput of the entire system, with a corresponding increase in wip.
- Batch processors are bulk servers; they induce large wip decrease and increase. This introduces more variability into the system and can cause system-wide performance degradation if not properly controlled.

Few researchers have designed wafer fab production control policies that place an emphasis on batch processor; the control of the batch processor has often been considered an isolated sub-problem to the problem of controlling wafer fabs. This is despite what Fowler and Mason [35] observed: good wafer fab schedules tend to have good batch processor schedules. We believe there is a significant opportunity for improvement by setting the control of the serial processors as a function of the batch processor's status.

Look-ahead control policies for the batch processor that use information from a limited number of upstream and downstream processors to control the batch processor have become increasingly popular. These methods have a potentially significant drawback: the batch processor cannot influence the arrival patterns of its wip; it can only predict its arrival pattern. A hypothetical control policy that can influence the job type being processed by upstream processors should, in theory, be at least as good as a look-ahead policy, with the potential for significant improvement.

The long-term goal of this research is a manner of controlling the wafer fab that bridges the divide between the control of batch processors as a single entity and the control of the entire wafer fab. The idea of driving the correct type wip into batch processors to improve system-wide performance is a fundamental shift from what has been previously proposed. As a first step to accomplishing this goal, we characterize the performance of a small two-stage system and evaluate the performance of this system under some simple control policies. Since we are concerned mainly about building intuition about the system in this early stage of the research, we neither emphasize numbers nor prove statements; but instead concentrate on interpreting the overall trends observed.

C. *Paper Outline*

Section two describes the problem in greater detail and presents the manner in which the problem was modeled. Section three describes the control policies for which the manufacturing system is ran. Section four characterizes the gross system performance and compares the system performance under several control policies. Section five summarizes the results, and briefly discusses future research directions.

II. Problem Statement and Methodology

A. *Problem Statement*

A manufacturing system is comprised of two perfectly reliable processors that process *n* different job types. All job types visit the processors in the same order. The first processor M_1 is a serial processor, and the second processor M_2 is a batch processor. The batch processor M_2 can process up to Q jobs of the same type at an instance, and processing time is independent of the number of jobs loaded into the processor. Job preemption is not allowed. No set up times are required on either processor.

In front of each processor M_i are *n* different buffers B_{ij} of finite size C_{ij}; each buffer B_{ij} serves as a temporary holding cell for jobs of type *j* waiting to be processed by processor M_i. When the current buffer level b_{ij} of buffer B_{ij} is 0, processor M_i is starved of job type *j*, and M_i cannot process job type *j*; When $b_{ij} = C_{ij}$, incoming arrivals into buffer B_{ij} are blocked, and these arrivals are lost.

A typical sequence of events for a job *J* of type *j* would be: job *J* attempts to enter the system and checks if $b_{1j} > C_{1j}$. If $b_{1j} < C_{1j}$, then job *J* enters B_{1j} and waits for M_1 to process it. If $B_{1j} \geq C_{1j}$, then job *J* is rejected and lost forever. It is assumed that any control policy at M_1 ensures that job *J* can be accommodated by B_{2j} before processing *J*. This ensures that the rate jobs enter the system is equal to the rate jobs exit the system. After being processed by M_1, job *J* enters B_{2j} and waits for M_2 to process it. Job *J* exits the manufacturing system after being processed by M_2. Figure 1 illustrates the manufacturing system when the number of job types $n = 2$.

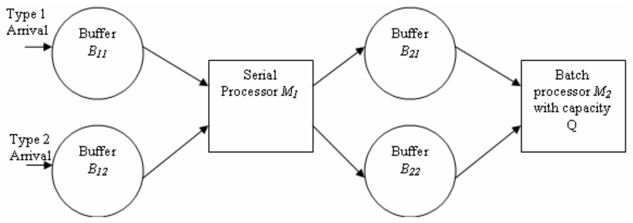

Fig. 1: A two-stage manufacturing system with a serial-batch configuration. In this example, the system is processing two job types, and would have two buffers, one for each job type, in front of every processor.

B. Methodology

Our preliminary research has two main goals: to characterize the performance of the system under a set of parameters and to determine which conditions a particular control policy would perform better than others.

We assume two different control policies for the batch processor. The full batch policy will not load any partial batches into the batch processor; the batch processor will wait for additional arrivals, if a full batch is unavailable. The no idling policy will load batches into the batch processor as long as wip is present in the buffers B_{21} and B_{22}. Both policies will load the job type with more wip in front of the batch processor; in case of a tie, job type one is selected. These two control policies can be considered the extremes of every control policy at the batch processor; one policy never waits for additional arrivals, while another policy always waits until the batch processor is full.

We test three different serial processor control policies, as combined with the two batch processor policies, to determine relative performance of the manufacturing system under each serial-batch control policy combination. All three control policies will not process a job if there is no space to store this job in front of the batch processor. Furthermore, all three policies are non-idling: as long as a job of any type can be processed, the processor is kept busy. The three policies are:

- Myopic control policy - The operator prefers to process job type one if $b_{11} \geq b_{12}$, and prefers to process job type two if $b_{12} > b_{11}$. The intuition behind this policy is simple: by processing the product type with more wip, we reduce the probability of any external arrivals getting rejected by the system.
- Greedy look-ahead control policy - The operator prefers to process job type one if $b_{21} \geq b_{22}$, and prefers to process job type two if $b_{22} > b_{21}$. Since the batch processor will process the job type j with the most wip in front of the batch processor, we want to either hasten the formation of a full batch of type j, or include more jobs of type j into the batch.
- Balanced look-ahead control policy - The operator prefers to process job type one if $b_{21} \leq b_{22}$, and prefers to process job type two if $b_{22} < b_{21}$. This policy is essentially the reverse of the greedy look-ahead policy. By aiming for a balanced wip level between two job types in front of the batch processor, we guard against prematurely blocking the production of a job type. This happens when the serial processor keeps on processing a single job type.

The two batch processor and three control processors combine to form six possible two-stage control policies. We model the two-stage manufacturing system running under one of these six policies as a continuous time-discrete state Markov chain, with time-invariant transition probability rates. Due to the exponential growth of the state space with respect to the number of job types n, we limit n to two.

We make the following assumptions:

- The time between external arrivals of for any job type j is exponentially distributed with mean $1/\lambda_j$. If the buffer B_{1j} is full, arrivals are rejected.
- The time required to process a job of type j at the serial processor is exponentially distributed with mean $1/\mu_{sj}$. The serial processor will not process job type j if buffer B_{2j} is full.
- The time required to process a batch at the batch processor is exponentially distributed with mean $1/\mu_B$, regardless of the type currently being processed. The common mean processing time allows us to obtain the mean cycle time from the steady-state probabilities without keeping track of the amount being processed at the batch processor.

For brevity the details of the Markov model is omitted. The reader is invited to contact the authors for the implementation details. We use the steady state probabilities to obtain the throughput and the mean cycle time of the system subjected to a specific control policy.

III. Discussion of Numerical Results

A. Experimental Setup

A total of six two-stage control policies are evaluated: We have two broad experimental set ups. In the first setup, the buffer size is kept constant, and the batch processor capacity is varied. For each two-stage control policy, batch capacity and buffer capacity setting, 25 data points are collected. Each data point corresponds to a particular combination of traffic intensities on the two processors. The second setup has a fixed batch capacity and the buffer capacity is incremented. Table 1 summarizes the settings for which the experiments performed. We use traffic intensity to refer to the ratio between the arrival rate and the maximum processing rate. When the processor has unit capacity, the traffic intensity is equivalent to the utilization.

Traffic intensity TI = (mean arrival rate) / (processor capacity * mean processing rate).

For each experiment, the mean processing rate of the serial processor M_1 is held constant at one unit per unit time, and the nominal traffic intensities are used to determine the mean arrival rate of both job types, as well as the mean processing time at the batch processor M_2. For all experiments, both job types are assumed to have equal arrival rates. Let $1/\lambda_1$ be the mean arrival rate for job type 1, $1/\lambda_2$ be the mean arrival rate for job type 2, and μ_B be the mean time to finish processing a batch in the batch processor. Then,

$1/\lambda_1 = 1/\lambda_2 =$ traffic intensity TI of M_1/ 2 and

$\mu_B = (TI$ of M_2 * capacity Q) / TI of M_1.

This equates utilization of the serial processor with average throughput rate. For fixed serial processor TI and fixed batch processor capacity, TI at the batch processor is proportional to batch processor processing times. For fixed batch processor *TI* and capacity, when the serial processor *TI* is low, the arrival rate is low and the mean time to process a batch is long. When the serial processor *TI* is high, the arrival rate is high and the mean processing time is short.

Due to space constraints, it would not be possible to show all the results obtained. We will be concentrating on the case where the serial processor *TI* is high. When the serial processor *TI* is low, the job arrival rate is low. When this happens, the processing at the serial processor is dictated by the availability of wip, and the differences between the control policies are muted.

Table 1: Experimental setup summary

	No. of control policies	Buffer Size	*Q*	M_1 *TI*	M_2 *TI*
Evaluate effect of batch capacity	6	6	2, 3, 4, 5	0.2, 0.4, 0.6, 0.8, 1.0	0.2, 0.4, 0.6, 0.8, 1.0
Evaluate effect of buffer size	6	2, 3, 4, 5, 6, 7	2	0.2, 0.4, 0.6, 0.8, 1.0	0.2, 0.4, 0.6, 0.8, 1.0

B. Performance characterization of system model

We have observed that the performance of the system is generally invariant of the problem parameters. Only one representative graph is provided.

1. Effect of increasing buffer size

Figure two illustrates the evolution of the system's throughput as a function of the batch processor traffic intensity for several buffer sizes, when the serial processor traffic intensity is 0.8. This is a strong bound on the throughput of the system. By increasing the batch processor *TI* while keeping the serial processor *TI*, the average processing time of the batch processor lengthens.

We expect that the throughput of the system will decrease as the batch processor *TI* increases; as the mean processing time of the batch processor increases, the probability of wip accumulating in front of the batch processor increases, and this can block the production of the serial processor. This intuition is corroborated by Figure 2. Furthermore, the larger the buffer sizes, the less likely this chain of events will occur. Judging from the concavity and the negative slopes of the curves, the throughput loss increases as the batch processor *TI* increases. Furthermore, the incremental throughput increase diminishes as the buffer sizes increase.

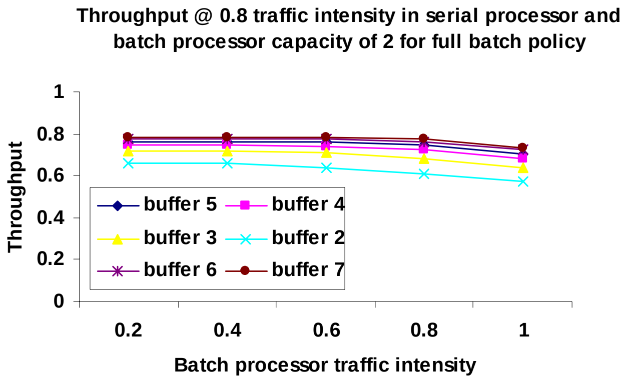

Fig. 2: Throughput as a function of the batch processor *TI*. As the average processing time of the batch processor increases, the throughput of the system goes down. Increasing the size of the buffers can help mitigate this effect.

Figure 3 illustrates the mean cycle time of products of the system as a function of the batch processor *TI.* The mean cycle time increases as the mean processing time of the batch processor increases. Increased buffer sizes lead to higher cycle times. This is because more jobs enter the system, which increases the probability of congesting the processors. This is a trade-off between throughput and cycle time in deciding buffer sizes.

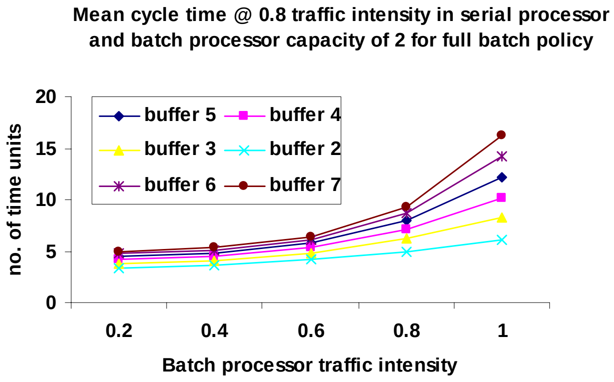

Fig. 3: Cycle time as a function of the batch processor *TI*. The cycle time increases as the processing time of the batch processor increases. Larger buffers allow more jobs to enter the system, causing the mean cycle time to also increase.

While majority of the results we present agree with what we would expect if we had a transfer line composed of serial processors, when the batch processor is under full batch policy, the mean cycle time can be reduced when the mean arrival rate is increased. When the mean arrival rate is very low, jobs in front of the batch processor need to wait for a long time before a full batch can be formed, which causes the mean cycle time to go up. When the mean arrival rate is increased, the waiting time of a job in front of the serial processor increases, but this is offset by the reduction in the time needed to form a full batch, to a certain extent. Increasing buffer sizes cause both cycle time and throughput to increase.

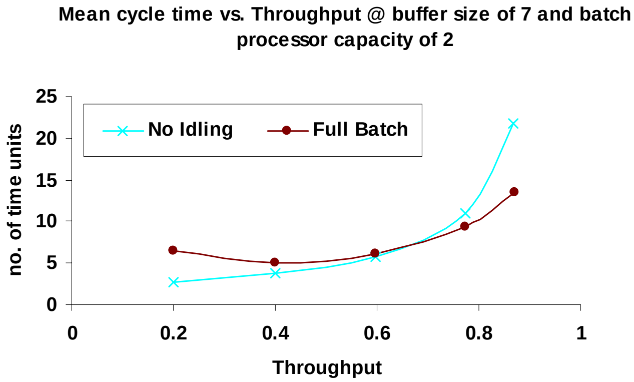

Fig. 4: Evolution of cycle time as the arrival rate is increased. Notice that when the batch processor is under full batch policy, it is possible to increase throughput and decrease mean cycle time.

2. Effect of increasing batch processor capacity

In general, increasing batch processor capacity has the same effect on throughput as reducing buffer sizes. When we increase the batch processor capacities, the average wip in front of the batch processor also increases. This reduces the buffering ability of the system, and increases the probability of external job arrivals being rejected. Thus, increasing batch processor capacities reduce throughput. Figure 5 supports our analysis: increasing batch processor capacities result in reduced throughput.

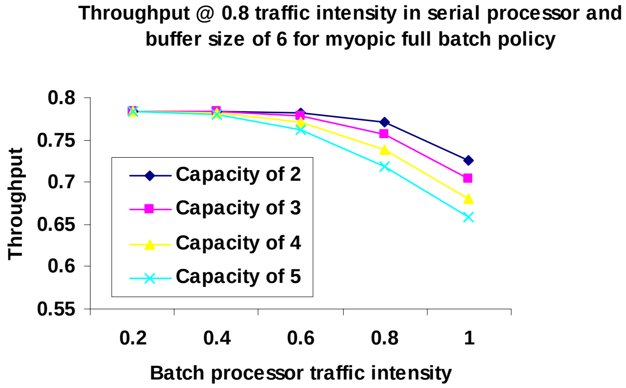

Fig. 5: Throughput as a function of the batch processor *TI*. Increasing the batch processor capacity (while maintaining constant batch processor *TI*) reduces the throughput.

As batch processor capacities increase, it takes a longer time to form full batches, if we use a full batch policy. Furthermore, as the mean processing time increases, the number of jobs that arrive in front of the batch processor to find the batch processor busy also increases. These jobs will also have to wait for a longer period of time before the batch processor will become available again. This causes cycle time to increase even if we are using no idling policy at the batch processor. Figure 6 illustrates the evolution of the mean cycle time of jobs in the system as the batch processor *TI* is varied. As the batch processor capacity increases, the cycle time increases.

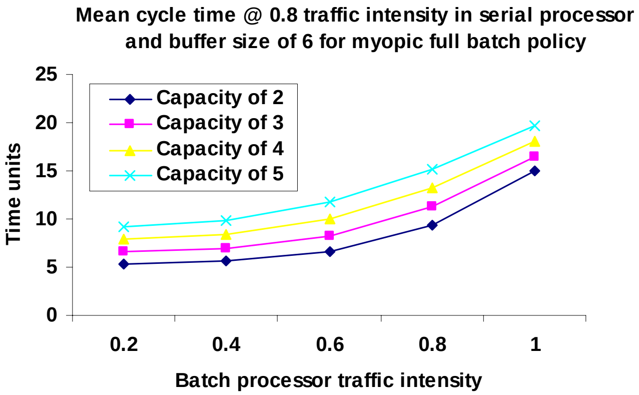

Fig. 6: Mean cycle time of the system as a function of the batch processor *TI*. As the batch processor capacity increases, cycle time increases. Furthermore, the incremental increase in cycle time is non-decreasing with respect to the *TI*.

C. Performance comparison between batch processor control policies

When arrivals into the batch processor's buffers are frequent compared to the mean processing time, building a full batch takes a shorter time. Thus we expect that no idling policies would outperform full batch policies when the batch processors have low traffic intensities, while full batch policies would outperform no idling policies. The figures in this section illustrate the difference between the two policies; positive values mean that the no idling policy generated larger values. We observe that the shape of the curves generated is generally invariant of the serial processor control policy selected.

1. Effect of increasing buffer size

Figure 7 illustrates the difference in throughput as the buffer sizes are changed, when the serial processor has high *TI*. When the serial processor *TI* is high and batch processor *TI* is very low, the no idling policy may have greater throughput than the full batch policy, when it is combined with balanced look-ahead control policy at the serial processor. When the serial processor is always busy, it is important to keep the buffers feeding the batch processor as empty as possible. This prevents the buildup of wip in front of the batch processor that could lead to job arrivals being rejected. When the batch processor *TI* is low, the potential throughput loss due to partial batches is unimportant, since the batch processor is rarely busy. As the batch processor's *TI* increases, the full batch policy eventually outperforms the no idling policy in terms of production. This is because the potential throughput loss due to partial batches produced by the no idling policy increases as the batch processor *TI* increases. When the buffer sizes are small; the probability of job arrivals getting rejected due to a full batch at the batch processor is higher when the buffer sizes are small. The no idling policy increases the probability of a job going in front of the batch processor and finding the processor busy, which increases the probability of buffers being full.

Figure 8 charts the mean cycle time as the batch processor *TI* is increased. When the batch processor *TI* is low, the time required to wait for full batches to form is saved by the no idling policy. However, as the batch processor *TI* increases, wip builds up in front of the batch processor, as the no idling policy finds difficulty processing all the jobs. This causes the mean cycle time to increase relative to the full batch policy.

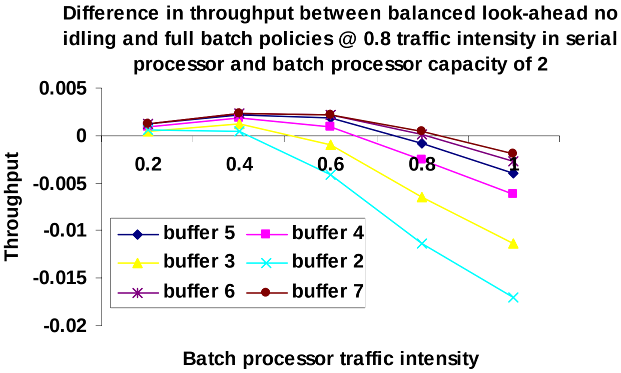

Figure 7: Throughput difference between no-idling and full batch policy when the serial processor *TI* is high. Positive values mean the no-idling policy had larger throughput. As the batch processor *TI* increases, the full batch policy outperforms the no idling policy.

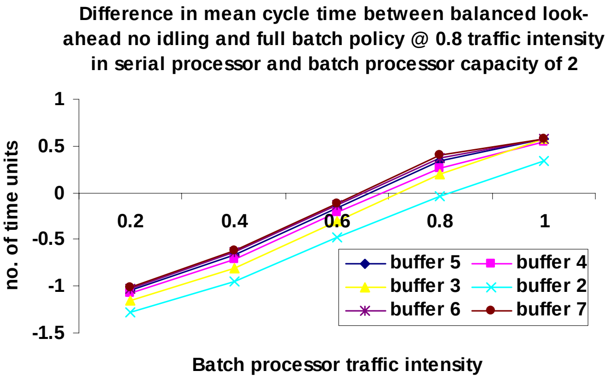

Fig. 8: Difference in mean cycle time between no-idling and full batch policy. When the batch processor *TI* is low, the no idling policy has lower cycle time than the full batch policy.

2. Effect of increasing batch processor capacity

The use of the no idling policy at the batch processor carries with it an implicit assumption that future batches will be able to compensate for the potential throughput loss incurred in the partial batch currently loaded. When the batch processor capacities are increased and the batch processor *TI* is low, the no idling policy is able to achieve lower cycle times than the full batch policy. Larger batch processor capacities force batch processors under full batch policy to stay idle for long periods waiting for a full batch to be formed. However, when the batch processor *TI* is high, it becomes difficult for future batches to compensate for any potential throughput loss incurred by the no idling policy, as shown in Figure 9. Wip accumulates in front of the batch processor and causes the no idling policy to generate larger mean cycle time compared to the full batch policy, as shown in Figure 10.

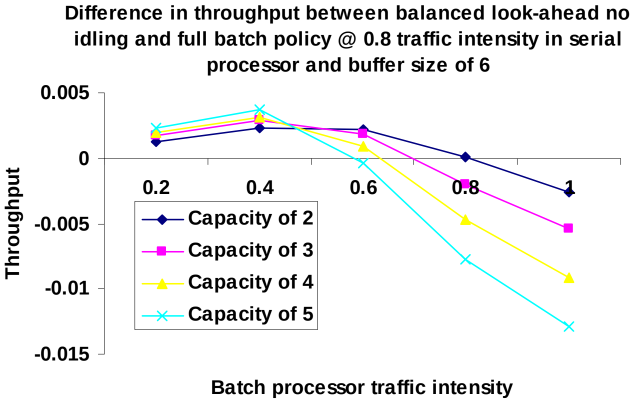

Fig. 9: Difference in throughput between no-idling and full batch policy. When the batch processor capacity is increased, the throughput difference observed increases.

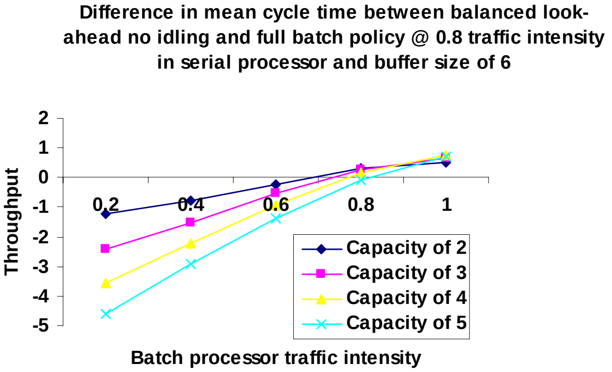

Fig. 10: Difference in mean cycle time between no-idling and full batch policy. When the batch processor TI is low, it takes a long time to form a full batch. This is why no idling policy has lower mean cycle time than full batch policy.

D. Performance comparison between serial processor control policies

1. Effect of increasing buffer sizes

Figure 11 shows the mean cycle time vs. throughput plot for the three serial processor control policies when the batch processor is under full batch policy and both processors have equal traffic intensities, while Figure 12 shows the same plot when the batch processor is under no idling policy. Both figures show that, given the same throughput, using the greedy look-ahead policy results in reduced cycle time, compared to the other policies. The cycle time reduction is magnified when the buffer sizes are increased.

Are there conditions in which the myopic policy would be preferred? For a given set of processor traffic intensities, the system throughput is generally maximized when the myopic policy is used. Since the serial processor is upstream of the downstream processor, it is easier for congestion at the serial processor to cause external arrivals to be rejected, compared to congestion at the batch processor. The difference in throughput is magnified when the serial processor has high traffic intensity, compared to the batch processor. These observations are not unexpected; a transfer line composed of two serial processors will behave similarly.

Similarly, when the batch processor uses full batch policy, the balanced look-ahead policy has slightly higher throughput than the greedy look-ahead full batch policy when the buffer sizes are large and the batch processor *TI* is low. This is most noticeable when the serial processor *TI* is high. When the serial processor *TI* is high, the job arrival rate to the front of the batch processor is high, and sufficient space must be provided, or the serial processor will get blocked. Since the balanced look-ahead policy attempts to balance the wip levels in front of the batch processor, it maximizes the storage space found in front of the batch processor, and reduces throughput loss.

On the other hand, when the batch processor uses no idling policy, the balanced look-ahead policy has slightly higher throughput than the greedy look-ahead policy under the opposite conditions: when the buffer sizes are small and the batch processor *TI* is high, and is most noticeable when the serial processor *TI* is low. When the serial processor *TI* is very low and the batch processor *TI* is very high, there would be very low wip level in front of the serial processor and there would be very high wip level in front of the batch processor. Consequently, if buffer sizes were small, jobs would be blocked due to jobs accumulating in front of the batch processor. The greedy look-ahead policy accelerates the blocking of a certain job type, whereas the balanced look-ahead policy would maximize the buffers in front of the batch processor, delaying the rejection of the job types.

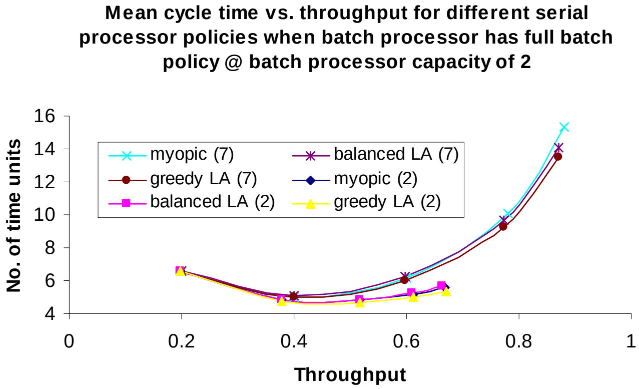

Fig. 11: Comparing the performance of three different serial processor control policies when the batch processor uses full batch policy. The number in parenthesis is the buffer size. When the job arrival rate is low, there is little difference between the performances of the serial processor control policies. The greedy look-ahead policy generally has lower cycle time for the same throughput. However, the myopic policy tends to have greater throughput for a given traffic intensity.

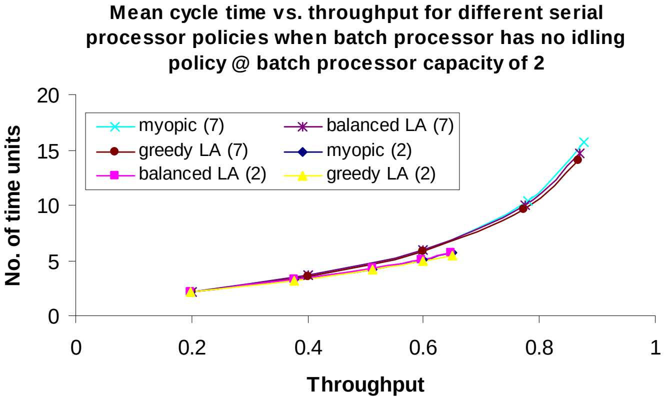

Fig. 12: Comparing the performance of three different serial processor control policies when the batch processor uses no idling policy. In this case, the myopic policy always has greater throughput.

2. Effect of increasing batch processor capacity

Figure 13 shows the mean cycle time vs. throughput plot for the three serial processor control policies when the batch processor is under full batch policy and both processors have equal traffic intensities, while Figure 14 shows the same plot when the batch processor is under no idling policy. For both Figures 13 and 14, the batch processor capacities are changed. Both figures show that, given the same throughput, using the greedy look-ahead policy results in reduced cycle time, compared to both the myopic policy and the balanced look-ahead policy. This is encouraging: despite changing the buffer size and the batch processor capacities, the greedy look-ahead policy still performs consistently better than the other two candidate policies. Furthermore, as the batch processor capacities are increased, the reduction in cycle time is also increased.

From Figures 13 and 14, the balanced look-ahead policy has lower mean cycle time compared to the myopic policy, when the batch processor capacity is low. However, when the batch processor capacity is high, the balanced look-ahead policy has higher mean cycle time compared to the myopic policy. The balanced look-ahead policy is designed specifically to redistribute wip in front of the batch processor; not only does this maximize the storage area available for the output of the serial processor, but it also prevents the batch processor from having too much wip of one type and none of the other. Since the first *Q-1* slots of the buffers are used to store the jobs to be processed, we would expect that the balanced look-ahead policy performs better when the difference between the batch processor capacity Q and the buffer sizes are larger. As the batch processor capacity increases, the effectiveness of the balanced look-ahead policy diminishes.

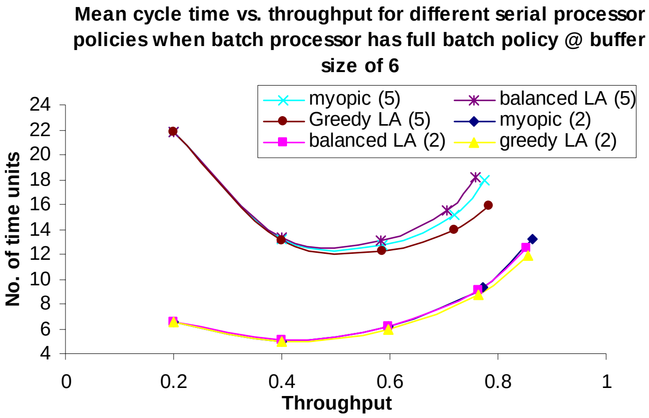

Fig. 13: Comparing the performance of three different serial processor control policies as the batch processor capacity is varied, when the batch processor is under full batch policy. The number in parenthesis is the batch processor capacity. The greedy look-ahead policy has lower cycle times for a particular throughput. Furthermore, when batch processor capacities are high, the greedy look-ahead policy can also have the highest throughput for a given traffic intensity.

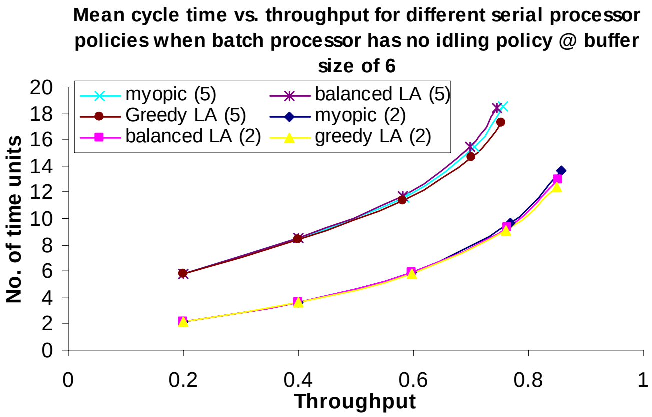

Fig. 14: Comparing the performance of three different serial processor control policies as the batch processor capacity is varied, when the batch processor is under no idling policy. When the batch processor capacity is low, the balanced look-ahead policy has lower mean cycle times than the myopic policy. When the capacity is increased, the balanced look-ahead policy has the highest cycle times of all three serial processor control policies.

IV. Conclusion

The relationship between the performance of the system and the parameters is generally invariant of the control policy used on either processor. This allows us to use the intuition obtained from these set of experiments and apply them to more complicated control policies with high degree of confidence. Furthermore, the behavior of the system can often be inferred by thinking of the behavior of a two–stage system involving serial processors. There are, however, some important exceptions: when the batch processor is operated under full batch policy, it takes a long time to form full batches when job arrivals are infrequent. Consequently, increasing the job arrival rate increases the throughput and decreases the mean cycle time. This example illustrates one of the possible pitfalls of not explicitly considering the presence of the batch processor in controlling a system.

We confirmed the intuition that the no idling policy would be preferred if the average wip level in front of the batch processor is low, while the full batch policy would be preferred if the average wip level in front of the batch processor is high. When majority of the system's wip can be found in front of the batch processor, it does not take long to form a full batch. This allows the full batch policy to have greater throughput and shorter cycle time than the no idling policy.

When throughput loss is to be minimized, it is generally more important to manage the wip levels in front of the first processor. This is why the myopic policy typically has greater throughput than either look-ahead policy, regardless of control policy used at the batch processor. This difference is magnified when the serial processor traffic intensity is high and when the buffer sizes are larger. This is intuitive; the busier the processor, the more likely it is that wip will accumulate in front of that processor, which can lead to rejection of external arrivals. Furthermore, the larger the buffer sizes, the longer it takes for a downstream accumulation of wip to propagate to the front of the system.

The increase in throughput associated with the myopic policy comes with a price: increased cycle time. The myopic policy causes higher cycle times, compared to the greedy look-ahead policy. The greedy look-ahead policy has lower cycle time because it hastens the formation of larger batches for the job type that will be produced next. When the batch processor uses full batch policy, full batches are formed faster. When no idling policy is in effect at the batch processor, more jobs are batched together. The higher the batch processor capacity, the more important it is to form large batches quickly. This is where the greedy look-ahead policy works best.

We have shown that the concept of controlling the serial processor as a function of the wip in front of the batch processor may be able to reduce the mean cycle time of jobs in the system. We have also determined under which conditions a particular concept embodied by a control policy is more effective than the others. This allows us to combine certain aspects of these policies to form a better control policy. A possible control policy is a hybrid look-ahead policy that uses the greedy look-ahead policy when none of the buffers in front of the batch processor has Q jobs in it, and then switches to a balanced look-ahead strategy once one of the buffers has Q jobs in it. Low cycle times are maintained by causing good batches to be quickly formed, while the throughput loss is reduced because premature blocking of a particular job type is prevented.

Although broad trends can generally be identified from the numerical experiments conducted, additional experiments are needed to characterize the curves with greater confidence. It is hoped that with the use of simulation, coupled with the intuition obtained from our small system, will allow us to generate a control policy with the desired performance for complex systems, such as a wafer fab.

REFERENCES

[1] M. Mathirajan, A. I. Sivakumar, V. Chandru, "Scheduling algorithms for heterogeneous batch processors with incompatible job families," *Journal of Intelligent Manufacturing 15, 787-803*, 2004.

[2] R. Bhatnagar, P. Chandra, R. Loulou, J. Qiu, "Order release and product mix coordination in a complex PCB manufacturing line with batch processors," *The International Journal of Flexible Manufacturing Systems 11, 327-351*, 1999.

[3] M. Mathirajan and A. I. Sivakumar, "A Literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor manufacturing," submitted *to Journal of Operations Management*.

[4] R. Uzsoy, "Scheduling batch processing machines with incompatible job families," International Journal of Production Research, vol. 33, no. 10, 2685-2708, 1995.

[5] S. V. Mehta and R. Uzsoy, "Minimizing total tardiness on a batch processing machine with incompatible job families," IIE Transactions, 30, 165-178, 1998.

[6] I. Perez, J. Fowler and W. Carlyle, "Minimizing total weighted tardiness of a single batch process machine with incompatible job families," Computers and Operations Research, 32, 327-341, 2005.

[7] M. Azizoglu and S. Webster, "Scheduling a batch processing machine with incompatible job families," Computers and Industrial Engineering, 39, 325-335, 2001.

[8] F. Jolai, "Minimizing number of tardy jobs on a batch processing machine with incompatible job families," European Journal of Operational Research, 2003.

[9] H. Balasubramanian, L. Monch, J. Fowler and M. Pfund, "Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness," International Journal of Production Research, Vol. 42, No. 8, 1621-1638, April 2004.

[10] R. Deb, and R. Serfozo, "Optimal control of batch service queues," Advances in Applied Probability, 5, 340-361, 1973

[11] R. Deb, "Optimal control of bulk queues with compound Poisson arrivals and batch service," Opsearch, 21, 227-245.

[12] S. Aalto, "Optimal control of batch service queues with Poisson arrivals and finite service capacity," Reports of the Department of Mathematics, University of Helsinki, 1997.

[13] S. Aalto, "Optimal control of batch service queues with compound Poisson arrivals and finite service capacity," Mathematical Methods of Operations Research, 48, 317-335, 1998.

[14] I. Duenyas and J. Neale, "Stochastic scheduling of a batch processing machine with incompatible job families," Annals of Operations Research, 70, 191-220, 1997.

[15] A. N. Avramidis, K. J. Healy, and R. Uzsoy, "Control of a batch processing machine: a computational approach," International Journal of Production Research, Vol. 36, No. 11, 3167-3181, 1998.

[16] J. K. Robinson, J. W. Fowler, and J. F. Bard, "A review of real-time control strategies for furnace batch sizing in semiconductor manufacturing,"

[17] D. J. Van Der Zee, "Look-ahead strategies for controlling batch operations in industry-an overview," In Proceedings of the 2003 Winter Simulation Conference, 1480-1487.

[18] C. R. Glassey, and W. W. Weng, "Dynamic batching heuristic for simultaneous processing," IEEE Transactions on Semiconductor Manufacturing, Vol. 4, No. 2, 77-82, May 1991.

[19] J. W. Fowler, D. T. Phillips, and G. L. Hogg, "Real-time control of multiproduct bulk-service semiconductor manufacturing processes," IEEE Transactions on Semiconductor Manufacturing, Vol. 5, No. 2, 158-163, May 1992.

[20] W. W. Weng and R. C. Leachman, "An improved methodology for real-time production decisions at batch-process work stations," IEEE Transactions on Semiconductor Manufacturing, Vol. 6, No. 3, 219-225, August 1993.

[21] J. K. Robinson, J. W. Fowler, and J. F. Bard, "The use of upstream and downstream information in scheduling semiconductor batch operations," International Journal of Production Research, 33, 1849-1869, 1995.

[22] D. J. Van Der Zee, "Adaptive scheduling of batch servers in flow shops," International Journal of Production Research, Vol. 40, No. 12, 2811-2833, 2002.

[23] L. Solomon, J. W. Fowler, M. Pfund, P. H. Jensen, "The inclusion of future arrivals and downstream setups into wafer fabrication batch processing decisions," Journal of Electronics Manufacturing, Vol. 11, No. 2, 149-159, 2002.

[24] D. J. Van Der Zee, "Real-time adaptive control of multi-product multi-server bulk service processes," In Proceedings of the 2001 Winter Simulation Conference.

[25] J. H. Ahmadi, R. H. Ahmadi, S. Dasu and C. S. Tang, "Batching and scheduling jobs on batch and discrete processors," Operations Research, Vol. 39, No. 4, 750-763, July-August 1992.

[26] B. K. Kim and S. Y. Kim, "Application of genetic algorithms for scheduling batch-discrete production system," Production Planning and Control, vol. 13, no. 2, 155-165, 2002.

[27] C. S. Sung and J. I. Min, "Scheduling in a two-machine flowshop with batch processing machine(s) for earliness/tardiness measure under a common due date," European Journal of Operational Research, 131, 95-106, 2001.

[28] C. S. Sung and Y. H. Kim, "Minimizing due date related performance measures on two batch processing machines,"European Journal of Operational Research, 147, 647-656, 2003.

[29] C. S. Sung, Y. H. Kim and S. H. Yoon, "A problem reduction and decomposition approach for scheduling for a flowshop of batch processing machines," European Journal of Operational Research, 121, 179-192, 2000.

[30] H. Gurnani, R. Anupindi, and R. Akella, "Control of batch processing systems," In Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991.

[31] J. J. Neale and I. Duenyas, "Control of manufacturing networks which contain a batch processing machine," IIE Transactions 32, 1027-1041, 2000.

[32] H. J. A. Rulkens, E. J. J. Van Campen, J. Van Herk, and J. E. Rooda, "Batch size optimization of a furnace and pre-clean area by using dynamic simulations," in 1998 IEEE/SEMI Advanced Semiconductor Manufacturing Conference, 439-444.

[33] E. Akcali, R. Uzsoy. D. G. Hiscock, A. L. Moser, and T. J. Teyner, "Alternative loading and dispatching policies for furnace operations in semiconductor manufacturing: a comparison by simulation," In Proceedings of the 2000 Winter Simulation Conference.

[34] http://www.in-sourced.com/article/articleview/1384/1/1

[35] S. Mason and J. Fowler, "Maximizing delivery performance in semiconductor wafer fabrication facilities," In Proceedings of the 2000 Winter Simulation Conference. 1458-1463.