Coordination to Avoid Starvation of Bottleneck Agents in a Large Network System

Rajesh Gautam
University of Tsukuba, JAPAN
r.gautam@aist.go.jp

Kazuo Miyashita
AIST, JAPAN
k.miyashita@aist.go.jp

Abstract

In this paper, we present a multi-agent control method for a large-scale network system. We propose an extension of a token-based coordination technique to improve the tradeoff between two conflicting objectives of the network system: reducing the lead time and increasing throughput. In our system, CABS, information about an agent's urgency of jobs to fulfill demanded throughput and to maintain its utilization is passed from downstream agents in the network so that upstream agents can provide necessary and sufficient jobs to bottleneck agents whose loss of capacity degrades the total system performance. We empirically evaluate CABS performance using a benchmark problem of the semiconductor fabrication process, which is a good example of a large-scale network system.

1 Introduction

Network systems have multiple resources that collectively perform a desired task that is not atomic but rather comprises a set of steps to be accomplished in a specific sequence. Queueing theory [Allen, 1990] has addressed analysis and control of network queueing in its steady state. Nevertheless, to understand and control its dynamic behavior is considered critically important for realizing smooth operations of today's complicated network system. Transportation, communication and manufacturing are typical examples of such large networks, for which uninterrupted and stable operations are required. In this paper, we use a manufacturing problem as a benchmark for controlling a large-scale network system.

In queueing theory, Little's Law [Little, 1961] states that the expected inventory of work in process (WIP) equals the average lead time multiplied by the average throughput. Therefore, with a fixed throughput, reducing the lead time requires WIP to be reduced. However, with a variable and unpredictable manufacturing environment, reducing WIP tends to decrease throughput by cutting back job stocks of machines so that machine downtimes have a high probability of forcing idle time of other machines because of a lack of jobs to process. In this paper, we are concerned with improving the tradeoff between the lead time and the throughput in a manufacturing system with unpredictable machine failure.

In a network system, because of connectivity of the steps to be processed, even if a system might have many overcapacity resources, the final throughput of the system is limited by the resource that has the smallest capacity (called a *bottleneck*). Maximizing throughput of the system therefore means keeping the maximum utilization of the bottleneck resource. High utilization of the bottleneck resource is ensured by maintaining a sufficient amount of jobs before it as a safety buffer against random events that might cause its starvation. Hence, to improve the tradeoff between lead time and throughput of a manufacturing system, several methods have been developed to regulate WIP at the lowest safe level that prevents starvation of bottleneck machines [Fowler *et al.*, 2002]. However, those methods subsume that the bottleneck machines in the system are identifiable by preliminary static analyses of the problem and do not evolve over time. However, in the course of manufacturing, the bottleneck machines might shift temporarily because of unexpected random events such as machine failures that disturb the smooth flow of jobs. This phenomenon is called *wandering* bottlenecks. Most existing solutions to the problem are rather philosophical and managerial (such as Kaizen [Imai, 1997] and Theory of Constraint (TOC) [Goldratt and Cox, 1992]) with a few exceptions of identifying wandering bottlenecks [Roser *et al.*, 2002].

To prevent starvation of bottleneck machines, *lot release control* to regulate workload in front of the bottleneck machines by controlling the entry of jobs in the system [Glassey and Resende, 1988] has been widely used in practice. Nevertheless, it has achieved limited success because its centralized decision-making mechanism at the job entry point cannot respond to dynamics of the manufacturing system (such as wandering bottlenecks). Rather than controlling the job entry, it is desired that jobs are processed and requested dynamically by every machine in the system as to maintain a steady flow of jobs leading to the bottleneck machines. The desired control (*lot flow control*) is possible only through coordinated operations of the machines. Centralized control of all the machines shares the same weak point with the lot release control. A decentralized coordination method is required so that every machine decides its job request and job processing in harmony with other machines as an intelligent agent.

In a time-critical manufacturing environment, no machine (i.e. agent) can afford to search and gather all necessary information of other machines for deciding its actions. Con-

sequently, many coordination techniques proposed in multi-agent systems [Jennings *et al.*, 2001; Sandholm, 1999; Faltings and Nguyen, 2005; Durfee, 1996] are inappropriate for our purpose. Just-In-Time (JIT) [Ohno, 1988] is a method of the distributed manufacturing control by exchanging tokens (Kanban cards) among the machines to control flows and amounts of work (WIP) in the system. In fact, JIT and its extensions are instances of *token-based* coordination [Wagner *et al.*, 2003; Xu *et al.*, 2005; Moyaux *et al.*, 2003] and have been widely used in manufacturing and other related fields. However, because of its simplicity, JIT succeeds only in stable and leveled environments.

In this paper, we propose an extension of the token-based coordination method: Coordination for Avoiding Bottleneck Starvation (CABS) for improving a tradeoff between a lead time and a throughput in a large-scale and uncertain network system. In CABS, agents coordinate with other agents to maintain the adequate flow of jobs to satisfy the various demands by preventing starvation of bottleneck agents. That coordination is achieved by efficient passing of messages in the system. The message includes information that enables agents to identify the bottleneck agents and hence coordinate with other agents by maintaining the desired flow of jobs to the bottleneck agents. In Section 2, we explain a generic manufacturing problem and the semiconductor fabrication process used for experiments. The details of algorithms in CABS are explained in Section 3. Section 4 illustrates basic behaviors of CABS using an example manufacturing scenario. Section 5 shows results of simulation experiments and validates the higher effectiveness of CABS than other conventional manufacturing control methods. Finally, Section 6 concludes the paper.

2 Problem

In this section, we first describe a general model of a manufacturing problem and then introduce a semiconductor fabrication process as an example of the most complicated systems in the current manufacturing industry.

2.1 Definition

The manufacturing problem requires processing a set of jobs $J = \{J_1, ..., J_n\}$ by a set of workstations, which are modeled as agents $A = \{A_1, ..., A_m\}$ in this paper. Each job J_l consists of a set of steps $S^l = \{S^l_1, ..., S^l_{s_l}\}$ to be processed according to its process routing that specifies precedence constraints among these steps. Every lot of the jobs flows through agents according to its process route. Each agent A_j has identical p_j machines to process its t_j tasks $T^j = \{T^j_1, ..., T^j_{t_j}\}$. Each job J_l has a demand rate dr_l, which is the number of lots of J_l to be completed in one hour. Furthermore, when an agent A_j processes its task T^j_i, it takes a process time pt^j_i. A task of the agents corresponds to a step in the jobs. Hence, precedence constraints among steps create a complicated directional network of agents. Presume an agent A_j's task T^j_q is a step S^l_i. A preceding agent of the agent A_j in terms of the task T^j_q, $A_{pre(j,q)}$, is in charge of a step S^l_{i-1} and a succeeding agent, $A_{suc(j,q)}$, processes a step S^l_{i+1}.

In addition to the agents that model the workstations, two types of synthetic agents exist. One is a *sink-agent* for each kind of job, which receives the completed lots from the last agent of the job's process route. Another synthetic agent, a *source-agent*, releases every job in the system by transferring it to the agent processing the first step of the job.

2.2 Semiconductor Fabrication Process

Semiconductor fabrication is among the most complex manufacturing processes. For example, the production steps for semiconductor manufacturing usually number a few hundred, with numerous repetitive reentrant loops. Its lead time extends over a couple of months [Atherton and Atherton, 1995].

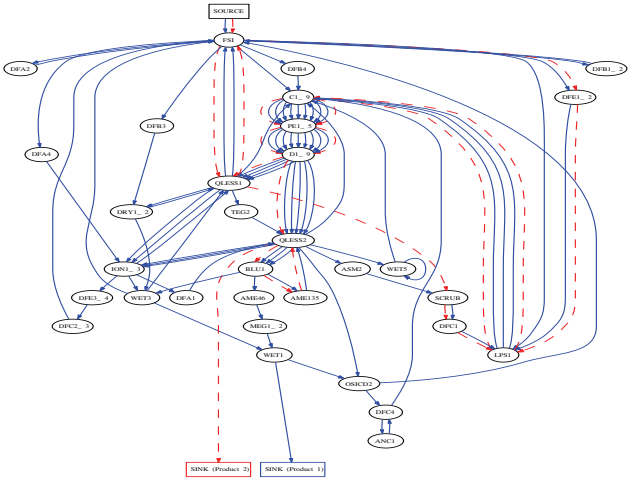

Figure 1: Network of agents in a semiconductor fab

For empirical validation of CABS, we used the Measurement and Improvement of Manufacturing Capacity (MIMAC) testbed datasets of the wafer fabrication processes [Fowler and Robinson, 1995], available from Arizona State university (http://www.was.asu.edu/˜masmlab/home.htm). The data set specifies the production steps of semiconductor manufacture. The factory model that we have chosen for our experiments represents a factory with 38 workstations. Two products, $Product_1$ and $Product_2$, are produced in the system. $Product_1$ has 92 processing steps and $Product_2$ has 19 steps. Many cycles exist in the process routes involving both products. The total process time for $Product_1$ is $4,423$ min (73.7 h); for $Product_2$, it is $1,097$ min (18.3 h). Figure 1, which depicts the process flows of products through the workstations in the experiment problem, can be viewed as a complex network of agents.

3 Coordination through Requirements

In CABS, actions of the agents are coordinated using the messages transmitted among agents. An agent uses requirement information in the incoming messages from succeeding agents for making lot processing decisions and for generating messages to send to its preceding agents.

3.1 Action Selection

The CABS system utilizes token-based coordination so that an agent selects its lot-processing actions based on requirements from its succeeding agents in the process flow. CABS realizes a *pull mechanism* like a JIT system that does not process jobs until they are "pulled" by downstream agents.

Each agent A_j periodically receives a requirement for processing a task T_q^j from a corresponding succeeding agent $A_{suc(j,q)}$. The requirement consists of the following three types of information (detailed definitions will be given later in Section 3.2):

`time limit`**:** time by which agent $A_{suc(j,q)}$ needs another lot for the next step of the task T_q^j.

`request rate`**:** rate at which agent $A_{suc(j,q)}$ needs the lots for the next step of the task T_q^j, starting at `time limit`.

`criticality`**:** criticality of the agent $A_{suc(j,q)}$.

In addition to the requirement information from the succeeding agents, for each task $T_q^j \in T^j$, an agent A_j is assumed to have local information such as the demand rate, current WIP and a total number of lots already produced.

Algorithm 1 selectTask(message $im[\,]$) of agent A_j

1: $\forall i \in \{1, \cdots, t_j\}$ set $t\_w_i^j$ as current WIP of task T_i^j
2: $ET^j \leftarrow \{T_i^j \mid (T_i^j \in T^j) \wedge (t\_w_i^j > 0)\}$
3: sort ET^j according to time limit (i.e., $im[\,].tl$) of tasks
4: set FET_j as the first task in ET^j
5: **loop**
6: set start time of FET_j at current time
7: $OFT^j \leftarrow \{ET_i^j \mid (ET_i^j \in ET^j) \wedge (ET_i^j\ overlaps\ FET^j) \wedge (criticality(ET_i^j) > criticality(FET_j))\}$
// $im[\,].cr$ decides criticality of a task
8: **if** $OFT^j \neq \emptyset$ **then**
9: remove FET_j from ET^j
10: set FET_j as the first task in ET^j
11: **else**
12: **return** FET_j
13: **end if**
14: **end loop**

Agent A_j uses the requirement information from its succeeding agents for choosing the next lot to process (i.e. *dispatching*) when any machine of the agent A_j becomes free. Algorithm 1 describes the dispatching algorithm for the agent A_j. It returns a task with the earliest `time limit` whose dispatching will not delay a task of any higher `criticality` beyond its `time limit`. In algorithms of the paper, $im[\,].tl$, $im[\,].rr$ and $im[\,].cr$ respectively denote requirement information of `time limit`, `request rate` and `criticality` for the corresponding tasks in the incoming messages of the agent. In addition, tasks mutually overlap when an intersection exists in their processing periods (i.e., (`time limit - process time`) of one overlaps with (`current time + process time`) of other).

3.2 Message Passing

Dispatching of agents in CABS is decided solely on requirements from succeeding agents. Hence, information in the requirement is a key to coordination among agents.

An agent tries to meet the requirements of succeeding agents for all of its tasks. Aside from meeting the requirements of succeeding agents, the agent must also minimize its *workload deficit* at all times for satisfying the demand rates of jobs. For example, A_j's workload of a task T_q^j is the time required to process one lot of the task (i.e. pt_q^j). Each agent has aggregated workloads of all of its tasks based on the demand rates of jobs. The difference between the workloads and the total processing time of the tasks that have been processed is a current workload deficit of an agent.

Algorithm 2 calcCriticality( ) of agent A_j

1: $\forall i \in \{1, \cdots, t_j\}$ set $t\_w_i^j$ as current WIP of task T_i^j
2: $t\_ft_j \leftarrow current\_time + \sum_{\forall i \in \{1,\cdots,t_j\}} (t\_w_i^j pt_i^j / p_j)$
// earliest time to finish current WIP
3: $\forall i \in \{1, \cdots, t_j\}$ set $t\_de_i^j$
as total demand of task T_i^j until t_ft_j
4: $\forall i \in \{1, \cdots, t_j\}$ set $t\_pr_i^j$
as total production of task T_i^j until current time
5: $t\_wld_j \leftarrow \sum_{\forall i \in \{1,\cdots,t_j\}} (t\_de_i^j - (t\_pr_i^j + t\_w_i^j)) pt_i^j$
// current estimated workload deficit of A_j
6: $sc_j \leftarrow p_j(1.0 - \sum_{\forall i \in \{1,\cdots,t_j\}} dr_{job(T_i^j)} pt_i^j / p_j)$
// surplus capacity of A_j
7: **return** t_wld_j / sc_j

An agent can recover its workload deficit by processing more lots than demand rates of jobs. The time needed to recover the deficit depends on the amount of deficit and surplus capacity available to the agent. Algorithm 2 calculates an agent's *criticality* as a ratio of its workload deficit and surplus capacity. In CABS, an agent with a large criticality is considered a bottleneck agent. Dynamic change of an agent's criticality represents *wandering* of bottlenecks.

To maintain a continuous lot flow of a task T_i^j to $A_{suc(j,i)}$ at the requested rate $im[i].rr$, the agent requires an incoming lot flow at the same rate from the corresponding preceding agent $A_{pre(j,i)}$. However, the agent itself might need the jobs earlier and at a higher rate in order to recover its workload deficit. The agent requires jobs immediately and at the maximum rate at which it can process materials to recover the deficit rapidly. Based on the requirement from the succeeding agent and its current workload deficit, the agent generates a consolidated outgoing requirement for its preceding agent. Algorithm 3 describes calculation of outgoing requirement messages by the agent A_j. For each $T_i^j \in T^j$, a requirement tuple $(om[i].tl, om[i].rr, om[i].cr)$ is generated and sent to the preceding agent $A_{pre(j,i)}$.

The agent acts to satisfy the requirement of its succeeding agent when agent A_j is not critical (i.e., its workload deficit is less than that of $A_{suc(j,i)}$) or has enough WIP of other tasks to process. In this case, the agent can postpone the `time`

Algorithm 3 makeRequest(message $im[\,]$) of agent A_j

1: $\forall i \in \{1, \cdots, t_j\}$ set $t\_w_i^j$ as current WIP of task T_i^j
2: $t\_ft_j \leftarrow current\_time + \sum_{\forall i \in \{1,\cdots,t_j\}} (t\_w_i^j pt_i^j / p_j)$
// earliest time to get starved
3: $t\_cr_j \leftarrow calcCriticality()$
// current criticality of A_j
4: **for all** $i \in \{1, \cdots, t_j\}$ **do**
5: $t\_ft_i^j \leftarrow current\_time + t\_w_i^j * pt_i^j / p_j$
// earliest time to get starved of T_i^j
6: $t\_tl_i^j \leftarrow im[i].tl - pt_i^j + t\_w_i^j / im[i].rr$
// time to replenish T_i^j based on request from $A_{suc(j,i)}$
7: **if** $(t\_cr_j < im[i].cr) \vee (t\_tl_i^j < t\_ft_j)$ **then**
8: $om[i].tl \leftarrow max(t\_ft_i^j, t\_tl_i^j)$
9: $om[i].cr \leftarrow max(im[i].cr, t\_cr_j)$
10: $om[i].rr \leftarrow min(im[i].rr, p_j / pt_i^j)$
11: **else** // A_j is lagging and starving
12: $om[i].tl \leftarrow t\_ft_j$
13: $om[i].cr \leftarrow t\_cr_j$
14: $om[i].rr \leftarrow p_j / pt_i^j$
15: **end if**
16: **end for**
17: **return** $om[\,]$

`limit` of requesting many tasks T_i^j beyond the earliest possible timing when the current WIP is emptied (i.e., $t\_ft_i^j$) until the last timing when the succeeding agent's request exhausts the current WIP (i.e. $t\_tl_i^j$). This situation realizes *lean manufacturing*, which is intended to reduce the amount of WIP and shorten lead times. As for `criticality`, agent A_j intends to pass the highest `criticality` along the process route by choosing a higher value of itself or its succeeding agent. The `request rate` is truncated only when the requested value is greater than the maximum capacity of agent A_j.

The agent prioritizes recovering its workload deficit over satisfying the succeeding agent's requirement when agent A_j is critical and has no sufficient WIP to process. Hence, as a requirement to its preceding agent, the agent sends the values of its own `time limit`, `criticality` and `request rate` for the purpose.

4 Simplified Example Scenario

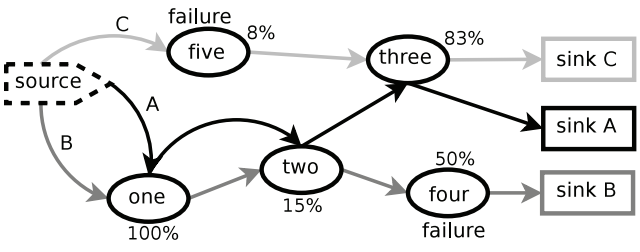

Figure 2: Example production system

The behavior of the algorithms mentioned above is illustrated using a simple scenario of a production system, which produces three job types (A, B, C) according to the process routes shown in Fig. 2. Five workstation agents (labeled one to five) and their utilization according to the demand rate of jobs is shown with the attached percentage. Two failures occur in the system around the same time. Agent4 fails at time 21,000 and recovers at time 60,000. Agent5 fails at time 13,000 and recovers at time 90,000.

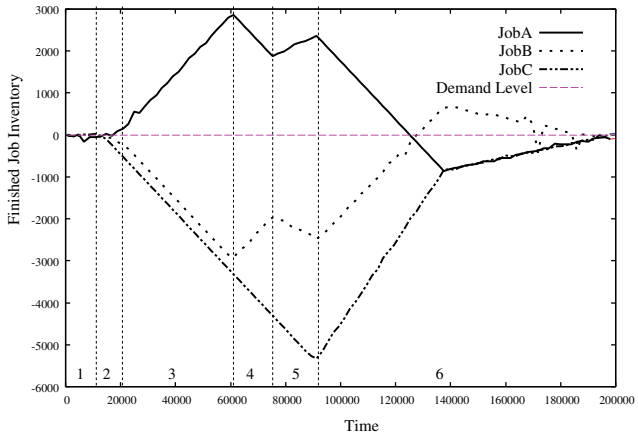

Figure 3: Finished job inventory

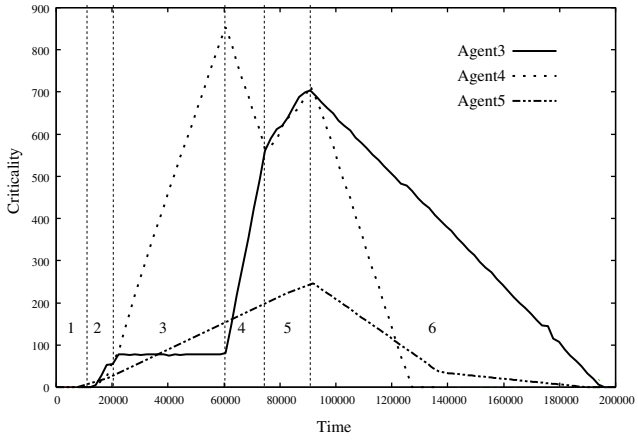

Figure 4: Criticality of agents

The achieved production of the products w.r.t. to the demand is shown in Fig. 3. The criticality of three agents along time is shown in Fig. 4 (the criticality of the other two agents is unimportant this scenario). The time line is divided into six sections, as shown in the figures.

In the second time section, when Agent3 stops receiving JobC because of a failure on Agent5, its criticality rises and it requests the other job, JobA, at a higher rate from Agent2 to meet its workload requirement. Agent2 propagates this request to Agent1.

Then, in the third time section, when Agent4 stops requesting JobB because of its failure, this information is also prop-

agated by Agent2 to Agent1. On receiving these updated requirements, Agent1 stops processing JobB and uses its full capacity to meet the requirement of Agent3 by processing JobA at a higher rate. Because Agent3's requirement of JobA at a high rate is consistently met, its criticality remains low until the recovery of Agent4 (at time 60,000). Although the utilization of Agent3 is higher than Agent4 according to the demand rate, Agent4's criticality rises during the failure and it becomes more critical (i.e. bottleneck) than Agent3.

In the fourth time section, after recovery (at time 60,000) Agent4 requests JobB at a higher rate to recover its workload deficit. At time 60,000, Agent1 stops dispatching JobA, which has a lower criticality (of Agent3) in its requirement, and uses its full capacity to dispatch JobB, which has a higher criticality (of Agent4). As Agent4 starts receiving jobs instead of Agent3, Agent4's workload deficit and criticality decrease and those of Agent3 increase.

In the fifth time section, when criticality of Agent3 and Agent4 become equal at time 75,000, Agent1 uses its capacity to produce both JobA and JobB for balancing the respective criticalities of Agent3 and Agent4. Then, criticalities of Agent3 and Agent4 rise at the same rate until Agent5 restarts processing JobC after its recovery at time 90,000.

In the sixth time section, because Agent3 has a large deficit of JobC, Agent3 dispatches JobC exclusively to recover this deficit and reduce its own criticality at the same time. As Agent1 stops getting requests for additional JobA, it stops processing JobA and starts processing JobB. Consequently, Agent1 recovers the inventory deficit of JobB and also reduces the criticality of Agent4. The system therefore recovers the deficit of all the jobs and returns to normal by time 200,000. Although Agent1 has the highest utilization in this example, it is not relevant because it is unaffected by any failure and its criticality remains low at all times.

5 Experiment

We evaluated performance of CABS using data of the semiconductor manufacturing process described in Section 2.2.

A simulation system is developed to model a manufacturing process with agents to test the proposed algorithms in CABS. The system is built using SPADES [Riley and Riley, 2003] middleware (http://spades-sim.sourceforge.net), which is an agent-based discrete event simulation environment. It provides libraries and APIs to build agents that interact with the world by sending and receiving time-based events.

5.1 Experimental Results

In the experiments, we induced random failures of all the workstations. The failures occur based on the exponential distribution with the MTBF value as 5,000 min and the MTTR value as 400 min. Because of dynamic changes of workstations' capacity, bottleneck workstations shifted temporarily (i.e. criticality of agents changed dynamically).

We compared the performances of CABS to those of a conventional manufacturing control method: constant releasing with earliest due date first (EDD) dispatching. We were unable to make a comparison with more sophisticated methods such as those in [Fowler *et al.*, 2002] because they are inapplicable to problems with wandering bottlenecks.

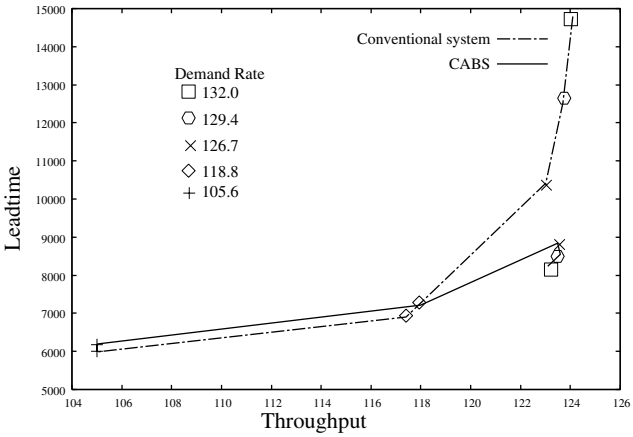

Figure 5: Throughput and lead time of $Product_1$

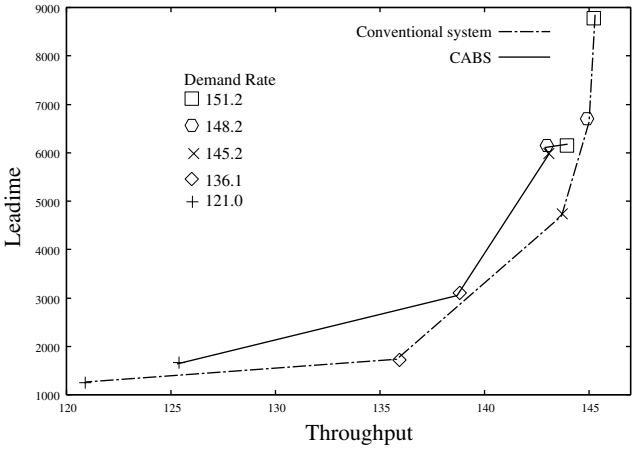

Figure 6: Throughput and lead time of $Product_2$

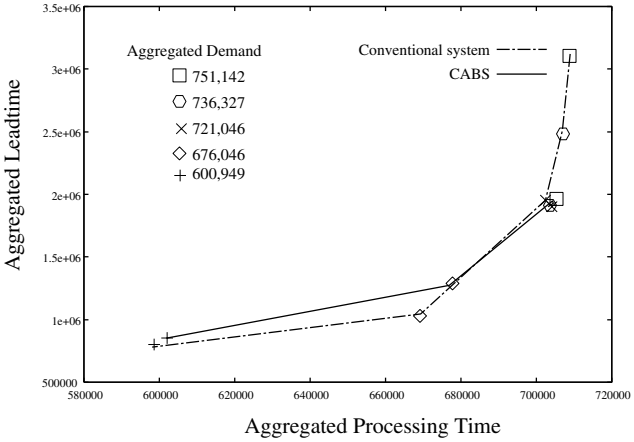

Figure 7: Aggregated result of two products

Figure 5 shows the result of throughputs and lead times for $Product_1$ with different demand rates. When demand rates are high, some agents become bottlenecks and regulate a throughput of the system when they become starved. In such cases, CABS achieved approximately equivalent throughputs with the conventional method, but required much less lead

time (therefore, much less WIP).

Figure 6 depicts the results of throughput and lead times for $Product_2$. This result shows that the performance of CABS is slightly worse than that of the conventional method. But because both products have similar demand rates and $Product_1$ has much longer processing time than $Product_2$, the result of $Product_1$ must have a greater impact on the performance of the entire manufacturing system. Furthermore, the result shows that CABS produced $Product_2$ more than demanded because CABS tried to prevent a loss of agents' capacity caused by failures; the failures did not stop during the experiments. Over-production of $Product_2$ should be decreased thereafter if the failures cease at some time point.

In Fig. 7, the aggregated processing time is calculated as $\sum_i Process\_Time_i Throughput_i$ and the aggregated lead time is calculated as $\sum_i Leadtime_i Throughput_i$. Because $Leadtime = Process\_Time + Wait\_Time$, this result shows that as an aggregate performance of the manufacturing system, CABS required less wait time than the conventional system, which means that CABS has less WIP than the conventional method to produce comparable outputs. In other words, CABS prevented starvation of bottleneck agents (i.e., achieving a high throughput) without increasing WIP in the system (i.e., achieving a short lead time).

6 Conclusion

In this paper, we investigated coordination techniques for a large-scale agent network system. The proposed system, CABS, coordinates the action of agents through a message-passing mechanism that is similar to other token-based coordination methods. By passing and utilizing the information of criticalities and job requirements of downstream agents, CABS can produce high throughput by preventing starvation of wandering bottleneck agents and, simultaneously, achieve short lead times by reducing the amount of inventories in the system. In experiments using data of a semiconductor fabrication process, we have validated that CABS can better improve the tradeoff between throughput and lead time than a conventional manufacturing control method can. We believe that the mechanism of CABS is suitable not only for manufacturing, but also for other network systems.

In the current implementation of CABS, agents might change their requests to the preceding agents drastically based on the small fluctuations of information they possess. Therefore, performance of CABS tends to be unstable during its execution. We plan to make agents decide their requests using a probabilistic threshold so that the behavior of CABS becomes more moderate and controllable.

References

[Allen, 1990] A. O. Allen. *Probability, Statistics, and Queueing Theory*. Academic Press, 1990.

[Atherton and Atherton, 1995] L. Atherton and R. Atherton. *Wafer Fabrication: Factory Performance and Analysis*. Kluwer Academic Publishers, 1995.

[Durfee, 1996] E. H. Durfee. Planning in distributed artificial intelligence. In G. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, chapter 8, pages 231–245. John Wiley & Sons, 1996.

[Faltings and Nguyen, 2005] B. Faltings and Q. Nguyen. Multi-agent coordination using local search. In *Proceedings of IJCAI-05*, pages 953–958, 2005.

[Fowler and Robinson, 1995] J. Fowler and J. Robinson. Measurement and improvement of manufacturing capacities (MIMAC): Final report. Technical Report Technical Report 95062861A-TR, SEMATECH, 1995.

[Fowler *et al.*, 2002] J. Fowler, G. Hogg, and S. Mason. Workload control in the semiconductor industry. *Production Planning & Control*, 13(7):568–578, 2002.

[Glassey and Resende, 1988] C. Glassey and M. Resende. Closed-loop job release control for vlsi circuit manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 1(1):36–46, 1988.

[Goldratt and Cox, 1992] E. Goldratt and J. Cox. *The Goal: A process of Ongoing Improvement (2nd rev edition)*. North River Press, 1992.

[Imai, 1997] M. Imai. *Gemba Kaizen: A Commonsense, Low-cost Approach to Management*. McGraw-Hill, 1997.

[Jennings *et al.*, 2001] N. R. Jennings, A. R. Lomuscio P. Faratin, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: Prospects, methods, and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.

[Little, 1961] J. D. C. Little. A Proof of the Queueing Formula $L = \lambda W$. *Operations Research*, 9:383–387, 1961.

[Moyaux *et al.*, 2003] T. Moyaux, B. Chaib-draa, and S. D'Amours. Multi-agent coordination based on tokens: Reduction of the bullwhip effect in a forest supply chain. In *Proceedings of AAMAS-03*, pages 670–677, 2003.

[Ohno, 1988] T. Ohno. *Toyota Production System: Beyond Large-Scale Production*. Productivity Press, 1988.

[Riley and Riley, 2003] P. Riley and G. Riley. SPADES — a distributed agent simulation environment with software-in-the-loop execution. In *Proceedings of the 2003 Winter Simulation Conference*, pages 817–825, 2003.

[Roser *et al.*, 2002] C. Roser, M. Nakano, and M. Tanaka. Shifting bottleneck detection. In *Proceedings of the 2002 Winter Simulation Conference*, pages 1079–1086, 2002.

[Sandholm, 1999] T. W. Sandholm. Distributed rational decision making. In G. Weiß, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 201–258. MIT Press, Cambridge, MA, 1999.

[Wagner *et al.*, 2003] T. Wagner, V. Guralnik, and J. Phelps. A key-based coordination algorithm for dynamic readiness and repair service coordination. In *Proceedings of AAMAS-03*, pages 757–764, 2003.

[Xu *et al.*, 2005] Y. Xu, P. Scerri, B. Yu, S. Okamoto, M. Lewis, and K. Sycara. An integrated token-based algorithm for scalable coordination. In *Proceedings of AAMAS-05*, pages 407–414, 2005.