6

Scalable Coordination Mechanism to Maintain Throughput of Dynamic Multiagent Networks

Rajesh Gautam[1] and Kazuo Miyashita[2]
[1]*DInSys Technologies,*
[2]*AIST,*
[1]*India,*
[2]*Japan*

1. Introduction

Many real-world systems are a manifestation of queuing networks. The queueing theory (Allen 1990) has addressed analysis and control of such networks in a steady state. Nevertheless, to understand and control their dynamic behavior in unstable situations is considered critically important for realizing smooth operations of today's complicated network systems. Transportation, communication and manufacturing are typical examples of such large networks, for which uninterrupted and stable operations are highly required. Influences of failures propagate unexpectedly in a complex network system. Network systems have multiple resources (i.e. nodes) that collectively perform tasks that are not atomic but rather comprise a set of steps to be accomplished in a specific sequence by different resources. As each resource of network is involved in intricate interactions with other resources, even a small failure at a single resource can make ripple effects and damage operations of the entire network. Heavy traffic jams in a transportation network and large-scale blackouts in a power-transmission network are typical outcomes of such cascading phenomena. Therefore, a robust method for controlling behaviors of the network to avert catastrophe caused by failures and maintain smooth operations is of keen interest among many researchers (Barabási 2002).

Manufacturing processes are examples of such networks which have become increasingly complex over time. Due to globalization of economy, manufacturing industry has also become very competitive and has to face new challenges. In addition to the persistent challenge of reducing manufacturing costs, same manufacturing infrastructure is utilized to simultaneously produce numerous customized products which have aggressive time to market and short life cycles. Simultaneously, in order to avoid technological obsolescence and remain competitive, parts of manufacturing infrastructure constantly get modified which adds to the volatility of manufacturing process. In such large, complex and dynamic systems, unexpected failures can have unanticipated effect throughout the system. Because of the size and complexity of problem, analysis and provisioning of preventive measures for the huge number of possible conditions is not possible during the planning phase. To maintain desired performance of such time-critical systems in the face of unexpected failures, developing robust control mechanisms is an active area of research. As a

Source: Multiagent Systems, Book edited by: Salman Ahmed and Mohd Noh Karsiti,
ISBN 978-3-902613-51-6, pp. 426, February 2009, I-Tech, Vienna, Austria

benchmark for controlling large-scale network systems, we have used the semiconductor manufacturing process which is among the most complicated and capital-intensive manufacturing processes in the world.

Semiconductor fabrication processes (*fabs*) consist of complex sequence of process steps, with the number of operations typically in hundreds and lead times extending over a couple of months (Pfund et al., 2006). The various steps of sequence are to be processed at different workstations in a given order. The process routes contain numerous cycles and *fab* produces a diversity of products (having different process routes) simultaneously which result in complex flow of jobs through the system. The capital cost to build and equip a semiconductor fabrication facility runs into billions of (US) dollars[1]. This requires the manufacturer to utilize every opportunity to increase the utilization and throughput of *fab* in order to maximize the return on investment (RoI). Besides increasing the throughput of manufacturing system, another objective of manufacturers is to simultaneously minimize the leadtime of jobs. With shorter leadtimes, a manufacturer can meet the dynamic customer orders more quickly and be more responsive to the market by reducing the time to market for new products. Furthermore, the fierce competition in the global market place and short technology life cycles require manufactures in the semiconductor industry to always deploy state-of-the-art manufacturing technologies. It causes their manufacturing processes to be unstable and unpredictable because they most of the time operate in the early part of the experience curves of manufacturing.

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, it is difficult to achieve the desired performance. The network systems usually have multiple and overlapping flows of tasks. When a failure occurs at a resource in the system, the flows using that resource are blocked in the middle and their tasks are delayed. As a result, workloads from the failed resource and downstream resources of its tasks are reduced during the failure and throughput of the affected tasks decreases. After recovery of the failure, for restoring throughput of the affected tasks, downstream resources of the failed resource must process excess flows of these tasks. If those resources should also process other tasks that are not affected by the failure as usual, the resources get congested and deteriorate throughput of those tasks as well. Besides degrading the throughput, the failure causes the lots to be held up for longer duration in the queues which adds up to their leadtimes of completed lots.

1.1 Conventional control approaches

In a manufacturing system, because of connectivity of the steps to be processed, even if a system might have many overcapacity resources, final throughput of system is limited by the resource that has the smallest capacity (called a *bottleneck*). Maximizing throughput of system therefore means keeping 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 leadtime and throughput of a manufacturing system, several

[1] http://www.icknowledge.com/economics/fab costs.html

methods have been developed to regulate WIP at the lowest safe level that prevents starvation of bottleneck machines (Fowleret al., 2002). However, those methods subsume that the bottleneck machines in system are identifiable by preliminary static analyses of problem and do not evolve over time. However, in the course of manufacturing, 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 & 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 bottleneck machines by controlling the entry of jobs in system (Glassey & 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 the dynamics of manufacturing systems (such as wandering bottlenecks). Rather than controlling the job entry, it is desired that jobs are processed and requested dynamically by every machine in 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 machines. Centralized control of all machines shares the same weak point with the lot release control (Miyashita et al., 2004). 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.

1.2 Multiagent based coordination approaches

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. Consequently, many coordination techniques proposed in multiagent systems (Jennings et al., 2001, Sandholm 1999, Faltings & Nguyen 2005, Durfee 1996) are inappropriate for our purpose. In a stable and leveled manufacturing system, a *pull control* method (Liberopoulos & Dallery 2000), in which an upstream machine starts processing a new task only when it receives a request from its downstream machine, has been investigated and shown to be efficient. Just-In-Time (JIT) (Ohno 1988) and CONWIP (Hopp & Spearman 2000) are the best-known examples of such pull control methods. In JIT, a machine exchanges tokens (*Kanban* cards) between its adjacent machines to control flows and amounts of WIP in the system. In fact, JIT and its extensions such as CONWIP are instances of *token-based* coordination (Wagner et al., 2003, Xu et al., 2005, Moyaux et al., 2003) and widely used in manufacturing and other related fields. However, because of their simplicity, they cannot correspond smoothly to changes of the environment such as demand fluctuations and machine failures. Hence, as a key of their successful application, emphasis was put on eliminating such deviations, which are inherent and inevitable in the semiconductor manufacturing process.

Although multiagent technology is an active area of research, its success in large complicated systems such as semiconductor fabrication has been limited. Coordination among agents is the cornerstone of distributed multiagent systems and new coordination algorithms are constantly being developed. The sophisticated coordination algorithms that require extensive interaction among large number of agents for making globally optimal decisions cannot work for large complex networks due to high messaging and computations requirements. On the other hand, the coordination algorithms which use simple interactions between small number of agents are although scalable, their efficiency is poor and the

resulting emergent system behavior can deviate greatly from desired behavior. Although multiagent framework suits well to the distributed nature of manufacturing systems, it is still a challenge to develop autonomous and distributed manufacturing control which is robust against unpredictable failures and achieves desired global optimization from today's dynamic manufacturing systems.

We view a manufacturing system as a network of agents that are in charge of processing specific steps of products. Thus each agent represents a machine and its buffers in the manufacturing system. In the manufacturing system, routing of tasks is partially fixed at a product design phase, but dispatching of tasks can be fully and dynamically controlled during manufacturing process. We have proposed an extension of the token-basedcoordination method: Coordination for Avoiding Bottleneck Starvation (CABS) for improving a tradeoff between leadtime and throughput in a large-scale and uncertain network system (Gautam & Miyashita 2007*a*, Gautam & Miyashita 2007*b*). 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 to maintain desired flow of jobs to the bottleneck agents.

In this paper, we show that CABS can be effectively applied to the production control of the semiconductor fabrication process. In Section 2, we explain a generic manufacturing problem and the details of coordination algorithms in CABS. Section 3 illustrates how CABS compensates for production loss caused by machine failures using a simulation result of a single failure scenario. Section 4 explains the distributed deadlock avoidance mechanism of CABS. Section 5 empirically validates that CABS succeeds to achieve desired throughput with shorter leadtime than a wellknown conventional manufacturing control method, CONWIP. Finally, Section 6 concludes the paper.

2. Coordination mechanisms in CABS

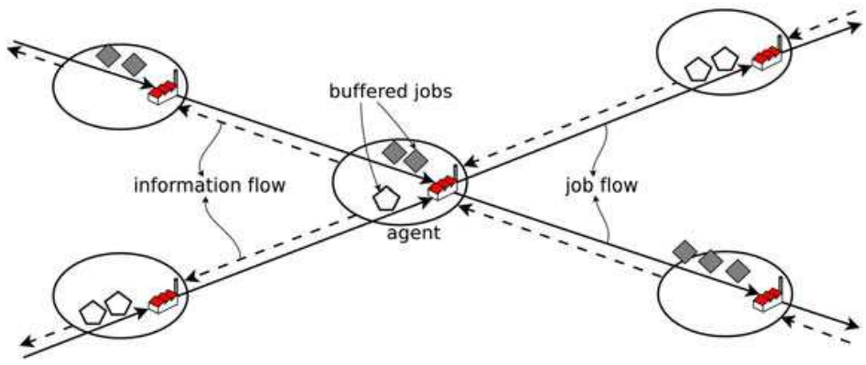

Fig. 1. Agent interactions in CABS

In this section, we first describe a general model of manufacturing problem and then introduce the coordination method developed for mitigating the affect of failures and maintaining throughput of network. In CABS actions of agents are coordinated using the messages transmitted among agents. As shown in Figure 1, an agent uses requirement information in the incoming messages from succeeding agents for making task processing decisions and for generating messages to send to its preceding agents.

2.1 Problem 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. Lots of a job flow through agents according to the job's 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 agent corresponds to a step in the jobs. Hence, precedence constraints among steps in jobs 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 of A_j, $A_{suc(j,q)}$, 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 Action selection

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

```
1: set ET as ∅
2: for all i ∈ {1, ··· , t_j} do
3:    set W_i as current WIP of task T^j_i
4:    if (W_i > 0) then
5:       add T^j_i to ET
6:    end if
7: end for
8: sort ET according to time limit (i.e., im[ ].tl) of tasks
9: set FET as the first task in ET
10: delete FET from ET
11: loop
12:    set start time of FET at current time
13:    set OFT as ∅
14:    for all ET_i ∈ ET do
15:       // im[ ].cr decides criticality of a task
16:       if (criticality(ET_i) > criticality(FET))
            ∧ (FET delays ET_i) then
17:          add ET_i to OFT
18:       end if
19:    end for
20:    if OFT ≠ ∅ then
21:       set FET as the first task in ET
          // i.e. next task with earliest im[ ].tl
22:       delete FET from ET
23:    else
24:       return FET
25:    end if
26: end loop
```

CABS utilizes token-based coordination so that an agent selects its lot-processing actions based on requirements from its succeeding agents in the process flow. Thus, 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 2.3):

1. `time limit`: time by which agent $A_{suc(j,q)}$ needs another lot for the next step of the task T_q^j.
2. `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`.
3. `criticality`: criticality of the agent $A_{suc(j,q)}$.

In addition to the requirement information from 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, its current WIP and the total number of lots it has already produced.

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 any other task with 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, a task t_1 **delays** task t_2 if processing t_1 before t_2 at current time (t_{curr}) delays the completion of t_2 beyond its `time limit`, *im*[t_2].*tl*.

$$t_1 \textbf{ delays } t_2 = \begin{cases} true & \text{if } (t_{curr} + pt_{t_1}^j) > \\ & \quad (im[t_2].tl - pt_{t_2}^j) \\ false & \text{otherwise} \end{cases}$$

2.3 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 those requirements, the critical agents must also minimize their *workload deficit* at all times for satisfying the demand rates of jobs. For example, A_j 's workload of a single lot of task T_q^j is the time required to process it (i.e., pt_q^j). Each agent has aggregated workloads of all of its tasks based on the demand rates of jobs (i.e., dr_l). The difference between the workloads and total processing time of tasks that have already been processed is the current workload deficit of an agent.

An agent can recover its workload deficit by processing more lots of any task than the corresponding demand rate. The time needed to recover the deficit depends on the amount of deficit and surplus capacity available to agent. Algorithm 2 calculates an agent's *criticality* as a ratio of its workload deficit and available 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.

Algorithm 2 calcCriticality( ) of agent A_j

$\forall i \in \{1, \cdots, t_j\}$ set W_i as current WIP of task T_i^j
$FT \leftarrow current\_time+$
$\sum_{\forall i \in \{1, \cdots, t_j\}} (W_i pt_i^j / p_j)$
// earliest time to finish current WIP
$\forall i \in \{1, \cdots, t_j\}$ set TD_i
as total demand of task T_i^j until FT
$\forall i \in \{1, \cdots, t_j\}$ set TP_i
as total production of task T_i^j until now
$WLD \leftarrow \sum_{\forall i \in \{1, \cdots, t_j\}} (TD_i - (TP_i + W_i)) pt_i^j$
// current estimated workload deficit of A_j
$SC \leftarrow p_j (1.0 - \sum_{\forall i \in \{1, \cdots, t_j\}} dr_{job(T_i^j)} pt_i^j / p_j)$
// surplus capacity of A_j
return WLD/SC

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

$\forall i \in \{1, \cdots, t_j\}$ set W_i as current WIP of task T_i^j
$ST \leftarrow current\_time+$
$\sum_{\forall i \in \{1, \cdots, t_j\}} (W_i^j pt_i^j / p_j)$
// earliest time when A_j gets starved
$CR \leftarrow calcCriticality()$
// current criticality of A_j
for all $i \in \{1, \cdots, t_j\}$ **do**
  $RT_i \leftarrow im[i].tl - pt_i^j + W_i / im[i].rr$
  // time to replenish T_i^j based on request from $A_{suc(j,i)}$
  if $(im[i].cr >= CR)$ **then**
    $om[i].tl \leftarrow max(ST, RT_i)$
    $om[i].rr \leftarrow min(im[i].rr, p_j / pt_i^j)$
  else
    $om[i].tl \leftarrow ST$
    $om[i].rr \leftarrow p_j / pt_i^j$
  end if
  $om[i].cr \leftarrow max(im[i].cr, CR)$
  if (A_j is in failure) **then**
    $om[i].tl \leftarrow \infty$
    $om[i].rr \leftarrow 0.0$
    $om[i].cr \leftarrow 0.0$
  end if
end for
return $om[\,]$

To maintain a continuous lot flow of task T_i^j to $A_{suc(j,i)}$ at the requested rate $im[i].rr$, the agent requires an incoming lot flow at same rate from the corresponding preceding agent $A_{pre(j,i)}$. However, the agent itself might be critical and 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 to recover the deficit rapidly. Based on the requirement from succeeding agent and its current workload deficit, the agent generates a

consolidated outgoing requirement for its preceding agent. Algorithm 3 describes the calculation of outgoing requirement messages by 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 agents use `criticality` of incoming requirements to identify the location of current bottlenecks in the system. If `criticality` of $A_{suc(j,i)}$ is higher than that of A_j, it means that $A_{suc(j,i)}$ is more likely to be a bottleneck in the system. In such a case, A_j acts to recover the deficit of $A_{suc(j,i)}$ and generates the outgoing requirements based on the incoming requirements from $A_{suc(j,i)}$. The agent postpones `time limit` in the outgoing requirements to the time when the current WIP is emptied (i.e., ST in Algorithm 3). This realizes *lean manufacturing*, which is intended to reduce the amount of WIP and shorten leadtime. `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 more critical than $A_{suc(j,i)}$. In order to recover its own deficit at the earliest, A_j sends the time when its own WIP is used up as `time limit` and its maximum production rate as `request rate` in requirements to its preceding agent. By sending high `request rate` and short `time limit` to all the preceding agents, the agent tries to expedite the production of all the available jobs for recovering the workload deficit caused by delayed jobs.

As for `criticality`, agent A_j intends to pass the highest `criticality` along the process route by choosing a higher value of itself and its succeeding agent. This enables the preceding agents to identify a location of a current bottleneck in the system along the process routes.

When an agent is in failure, it cannot process any job. Therefore, the agent during the failure period stops requesting jobs to its preceding agents by sending the requirements accordingly (i.e., setting `time limit` as ∞ and `request rate` as zero). `Criticality` of the failed agent is set to zero so that preceding agents can avoid responding to the requests from the failed agent.

3. Covering capacity loss caused by machine failures

In this section, we explain how CABS can cover the capacity loss of failed agents using a simplified scenario of a single failure. 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 & Riley 2003) middleware[2], 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.

3.1 Test problem

For empirical validation, we used the Measurement and Improvement of Manufacturing Capacity (MIMAC) testbed datasets of the wafer fabrication processes (Fowler & Robinson 1995) from Arizona State university3[3]. The dataset specifies the production steps of semiconductor manufacturing.

[2] Available online at: http://spades-sim.sourceforge.net.

[3] Available online at: http://www.was.asu.edu/~masmlab/home.htm.

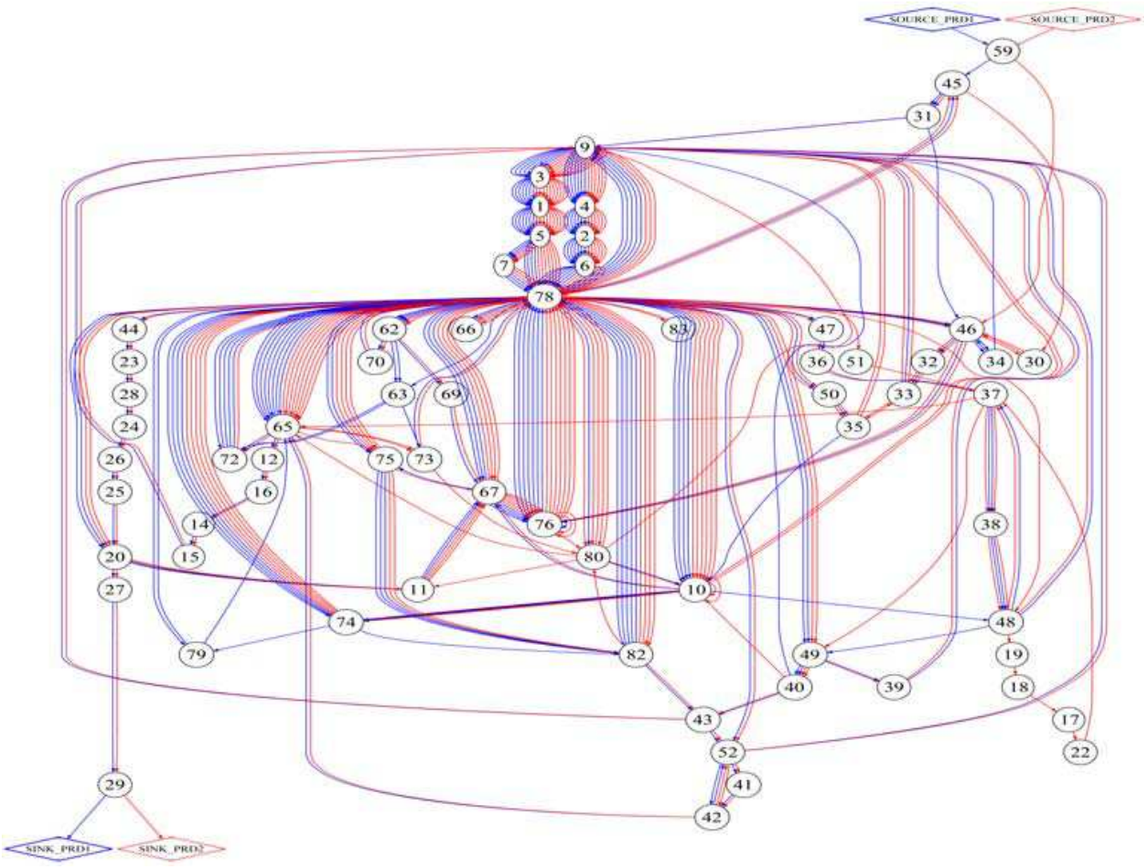

Fig. 2. Process flows of test problem; 83 agents represent workstations and process 455 steps of two types of products.

Table 1 shows the properties of the test problem we chose from the MIMAC datasets. It has basic characteristics of a semiconductor manufacturing process such as lengthy process flow with many repetitive reentrant loops and a couple of bottleneck workstations.

Type of product	Non-volatile memory
Process flows	2
Products	2 (i.e., 1 per process)
Workstation groups	83
Workstation	265
Steps	210 (Product 1) 245 (Product 2)
Raw processing time (hour)	313.4 (Product 1) 358.6 (Product 2)
Demand rate (wafer/day)	336.0 (Product 1) 168.0 (Product 2)
Lot size (wafers)	48(Product 1,2)

Table 1. Specification of test problem

In the experiments, we made the following assumptions to focus our investigative attentions to the basic properties of CABS: (1) there is no variabilities in processing times of operations,

(2) no setup time is considered, (3) operators are not considered in the model, (4) there is neither product rework nor scrap, (5) stochastic machine failure is modeled using exponential distribution, and (6) the demand rates are tuned to realize 87% resource utilization for the most heavily loaded workstation in the steady state.
Figure 2, which depicts the process flows of products through the workstations in the test problem, can be viewed as a "complex network" (Barab´asi 2002). Each node in the network represents a workstation group, which may consist of multiple workstations. Three workstation groups (i.e, No.67, 76 and 78) have average utilization which is higher than 80% and can easily become bottlenecks when unexpected events occur in the manufacturing process. It is noteworthy that, although the number of nodes in the network is moderate (less than one hundred nodes), because they are connected with directional, weighed and multiple links, analysis of the network's behavior is far more intractable than that of networks, which is a current research subject in the area of complex networks.

3.2 Single failure scenario

In this simplified scenario, a single failure occurs at time 50,000 and recovers at time 90,000 on an agent (Workstation group No.19 in Figure 2) that is processing only the 105th step of $Product_2$. To emphasize characteristic behaviors of CABS, we compared the results of CABS with those of a benchmark system using a constant releasing rule and an EDD (i.e., the earliest due date first) dispatching rule. The behaviors of CABS and the benchmark system are shown in Figures 3 - 5 and Figures 6 - 8 in terms of finished product inventory, production rate and WIP levels respectively. The failure duration is shown by the shaded zone in the graphs.
We first explain the behavior of CABS in detail. During the failure, the flow of $Product_2$ is stopped after the failed agent and its production starts to drop (shown as concave lines of $Product_2$ in Figure 3 and Figure 4). Due to unavailability of $Product_2$, the succeeding agents to the failed agent begin to starve, and their workload deficit increases. Consequently, as explained in Algorithm 2, `criticality` of those agents increases during the failure. Among the succeeding agents, some agents are processing both $Product_1$ and $Product_2$. In order to compensate for the shortage of $Product_2$, these agents start to request $Product_1$ early at their maximum rate (see Algorithm 3: lines 10 - 11). This behavior of agents increases the production rate and finished inventory of $Product_1$ during the failure (see rising $Product_1$ lines in Figure 3 and Figure 4).

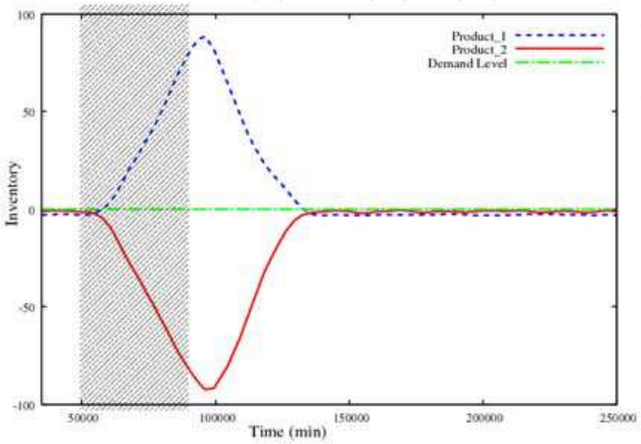

Fig. 3. CABS: Finished Product Inventory

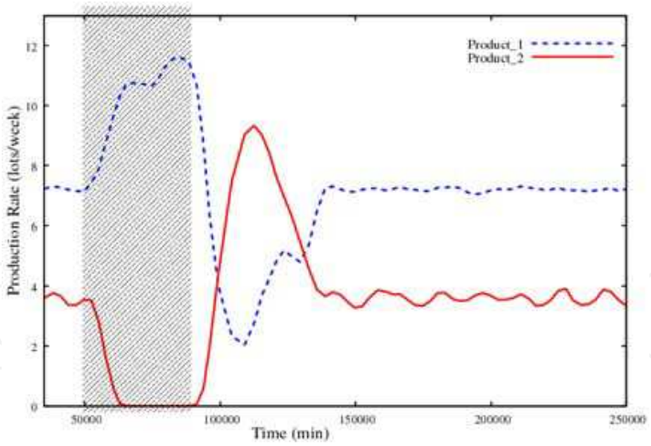

Fig. 4. CABS: Production Rate

In order to recover workload deficits, the agents pull both $Product_1$ and $Product_2$ at high rates during the failure. But due to the failure, $Product_2$ cannot be processed and its WIP is accumulated as shown in Figure 5. When the failure is recovered, the agents increase production rate of $Product_2$ by utilizing the extraWIP accumulated during the failure (see rising of $Product_2$ lines after the failure recovery in Figure 4 and Figure 3). The production rate of $Product_1$ is reduced after the recovery to bring the finished inventory of both the products to the desired demand level by time 130,000 (see Figure 4 and Figure 3). This is achieved by the dispatching rule shown in Algorithm 1, which exploits `time limit` information of different kinds of tasks to pick the next task for processing. Since $Product_1$ is produced in excess during the failure, `time limit` in the requirement from *sink-agent* of $Product_1$ rises during the failure. `Time limit` of $Product_2$ remains low because of its deficit from the demanded production.

Thus, by using the coordination mechanism of CABS, the agents are able to maintain their utilization during failures by processing alternative tasks. This enables them to recover throughput of failed tasks quickly by producing more of them after the resolution of failures.

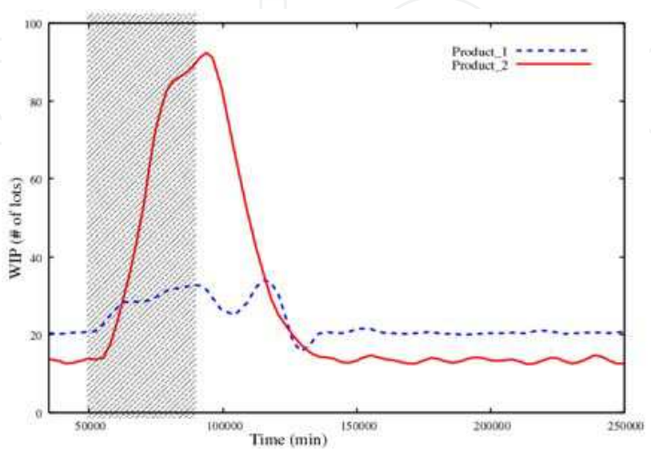

Fig. 5. CABS: WIP

The benchmark system, on the other hand, does not handle failures with a special care. It continues production of $Product_1$ at the same demand rate during the failure (see Figure 7). Thus, due to the suspension of the flow of $Product_2$, the bottleneck agents suffer a capacity loss and the system takes long time to recover the production shortage incurred during the failure. The failure adversely affects production of $Product_1$ as well. Since the EDD dispatching rule tries to balance the deficit of both products, the finished inventory of $Product_1$ also drops after the resolution of failure (see Figure 6). Comparison of Figures 3 and 6 shows that the recovery to the desired product inventory level of both the products is much slower (about at time 220,000) than CABS. More importantly, if the demand rates of the products are higher (or the failure sustains longer), it is more likely that the benchmark system cannot make up for the production loss caused by the failure permanently.

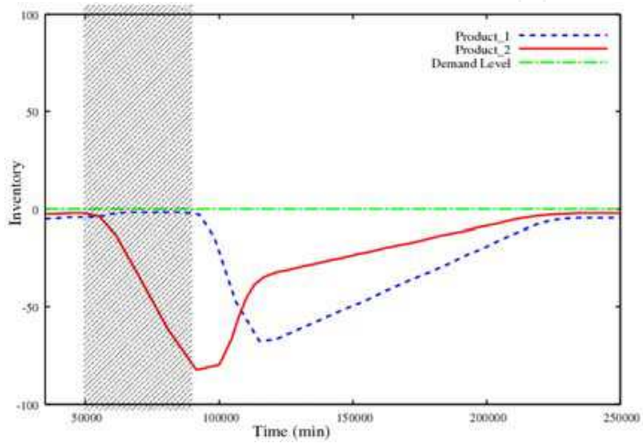

Fig. 6. Benchmark: Finished Product Inventory

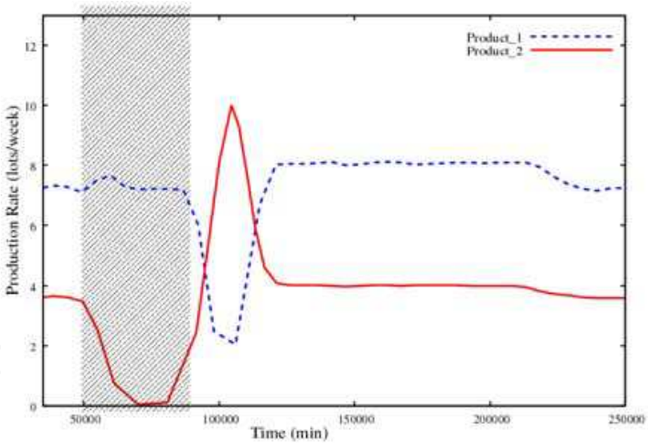

Fig. 7. Benchmark: Production Rate

In the above experiments, we assumed no upper limitation of WIP size in the system. As a result, in both CABS and the benchmark system, WIP was increased up to more than 90 lots during the failure. In realistic manufacturing situations, WIP size should be suppressed under certain level due to physical and economical reasons. Figures 9 - 11 show the behaviors of CABS with a limited WIP size. In the experiment, we limit the total WIP size of the system as 38 lots at its maximum. Figures 9 - 11 show that by limiting the WIP size CABS still performs similarly to CABS with unlimited WIP size but takes more time to compensate

for the production loss caused by the failure. However, to be noted is that CABS with limited WIP is still much faster to recover production loss (about at time 150,000) than the benchmark system and the quick recovery of CABS requires less than half of WIP used in the benchmark system (see Figure 11 and Figure 8).

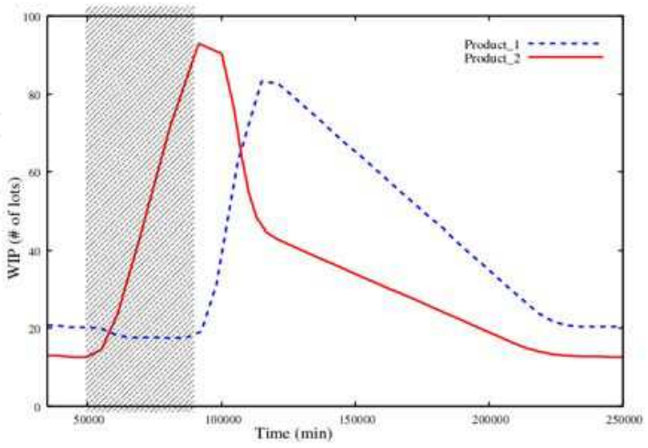

Fig. 8. Benchmark: WIP

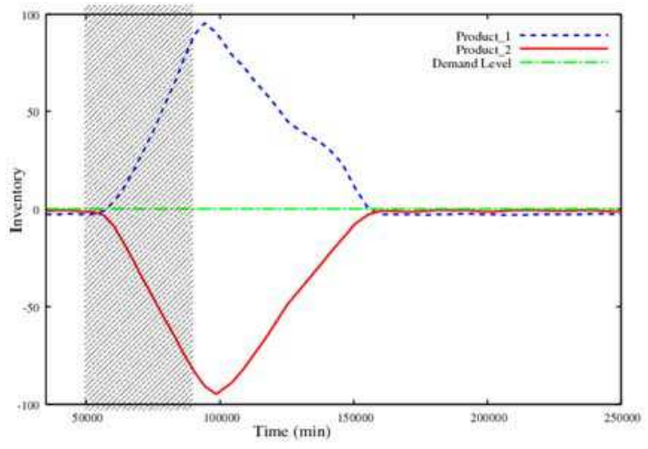

Fig. 9. CABS w/ limited WIP size: Finished Product Inventory

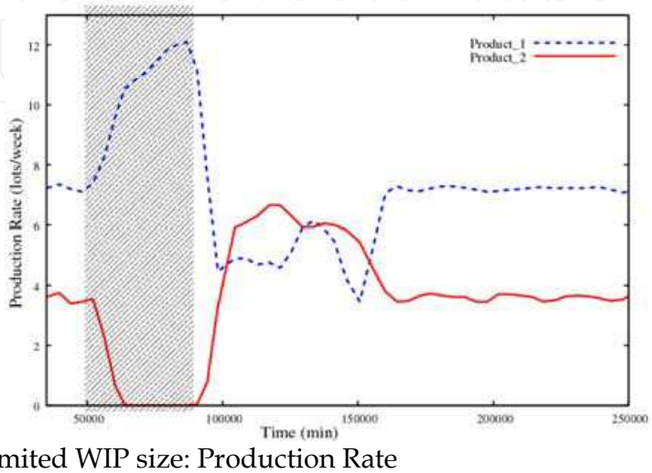

Fig. 10. CABS w/ limited WIP size: Production Rate

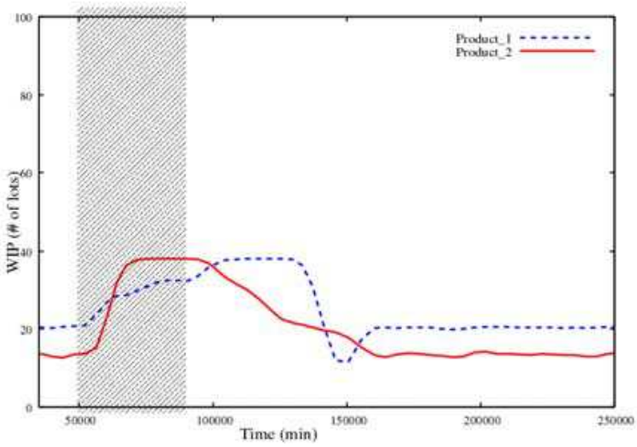

Fig. 11. CABS w/ limited WIP size: WIP

4. Distributed deadlock avoidance

Unlike many other network systems, semiconductor fabrication processes are characterized by existence of large number of re-entrant cycles in their process routes (Figure 2). Although deadlocks can occur in other systems also due to limitation of buffers, semiconductor fabrication processes are more prone to them due to large number of cycles. Deadlocks can be of two types, *permanent* and *transient* deadlock. A permanent deadlock cannot be resolved without external intervention, whereas a transient deadlock resolves itself over time (Venkatesh & Smith 2005). The probability of having deadlocks increases when capacity of buffers in system is reduced. As permanent bottlenecks bring the system to standstill, issue of bottlenecks has to be addressed in order to have an autonomous system that can work with limited buffer capacities. Because of the complexity of system, avoidance, identification and resolution of deadlocks in semiconductor manufacturing processes is a difficult problem and various sophisticated techniques are being investigated under current research (Venkatesh & Smith 2005).

As the techniques for managing deadlocks in semiconductor fabrication processes are still under investigation, in order to focus our attention on behavior of CABS, we have developed an distributed algorithm that avoids permanent deadlocks in system. Our deadlock avoidance algorithm avoids permanent deadlocks by:

- introducing a mechanism of reserved buffers
- utilizing an additional parameter in CABS message

We first explain the concept of our reserved buffers by using an example. Figure 12 describes a permanent deadlock that occurs in the part of system that has a small cycle involving two agents. *PROCESS ROUTE* describes a cycle in process flow of job through *AGENT1* and *AGENT*2, where *AGENT1* is processing two steps of process. As succeeding agent should have a free buffer to park incoming job, agents in system wait for authorization from their succeeding agent before they can start processing a new job. We have used the token based mechanism (similar to *Kanban* (Ohno 1988)) for realizing such authorization. Agents in this example have a shared buffer of size three, which can hold any type of incoming job.

In Figure 12 we first show the occurrence of permanent deadlock in a system that is not using specific buffers, i.e. only has shared buffers. *STAGE0* in Figure 12 shows that *AGENT1* is processing *stepP* as it is authorized by a free buffer of *AGENT2* (shown by directed solid line). Because all buffers of *AGENT1* are full, *AGENT2* cannot process its jobs and is awaiting its authorization from a free buffer of *AGENT1* (shown by directed dashed line). *STAGE1* shows the permanent deadlock that occurs when buffers of *AGENT2* also get full after receiving the additional job from *AGENT1*. As both agents now wait for authorization from each other indefinitely, this deadlock is *permanent* and cannot be resolved without external intervention.

We now explain our mechanism of specific buffers and how it avoids the occurrence of permanent deadlock during the same scenario. In CABS, each agent has two types of input (WIP) buffers: one is a single-sized buffer specific to the WIP of each product step and the other is a buffer shared by any WIP incoming to the agent. Hence, each agent in CABS has (1) multiple single-sized specific buffers whose number is equal to that of the product steps that are processed by the agent, and (2) a shared buffer whose size is not fixed.

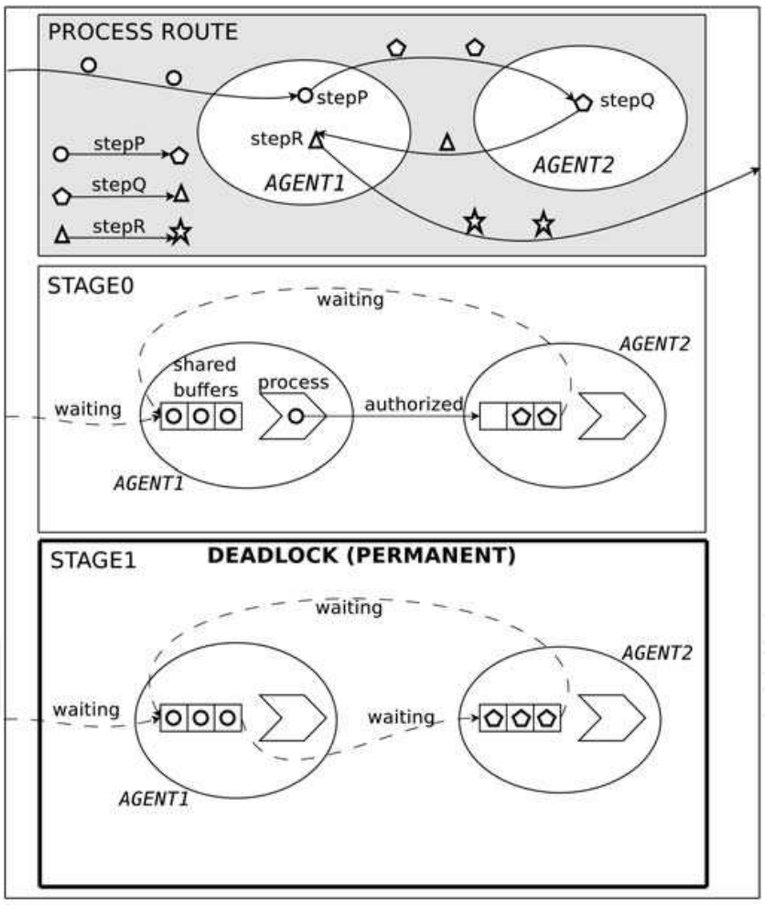

Fig. 12. Deadlock: without specific buffers

We now explain our mechanism of specific buffers and how it avoids the occurrence of permanent deadlock during the same scenario. In CABS, each agent has two types of input (WIP) buffers: one is a single-sized buffer specific to the WIP of each product step and the

other is a buffer shared by any WIP incoming to the agent. Hence, each agent in CABS has (1) multiple single-sized specific buffers whose number is equal to that of the product steps that are processed by the agent, and (2) a shared buffer whose size is not fixed.

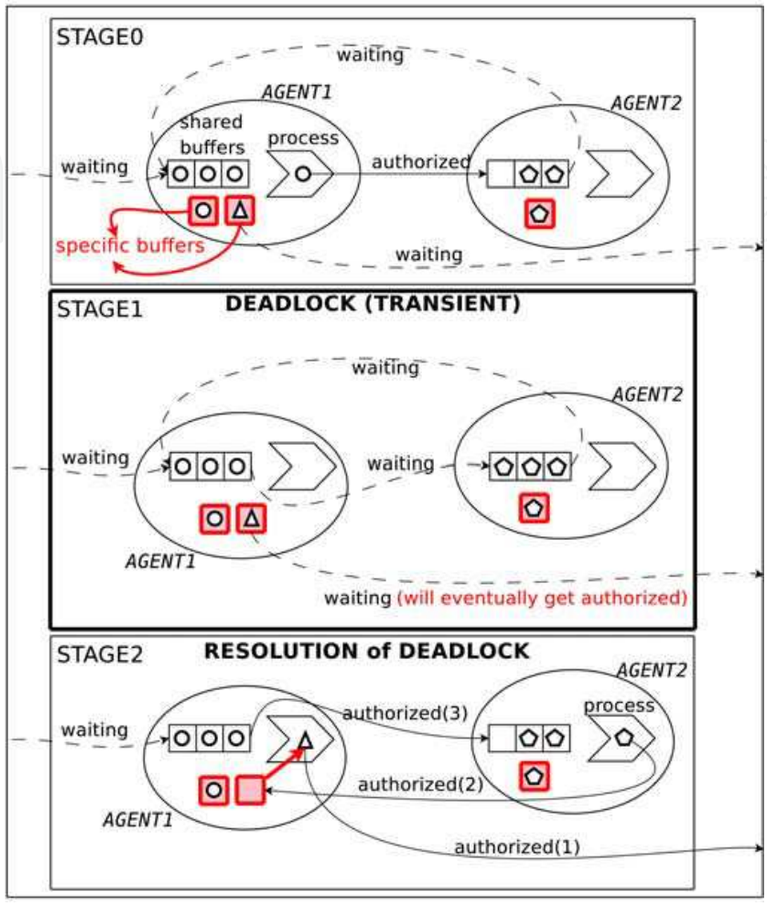

Fig. 13. Deadlock: with specific buffers

We now explain the details of additional message parameter that is used to avoid deadlocks. The additional message parameter, ETA (Estimated Time to Approval), is included in CABS message and is utilized to control the flow of buffers when buffers get occupied. Along with `time limit`, `request rate` and `criticality`, `ETA` is also sent in all the requirement messages of a process step. `ETA` defines the time (in future) at which agent will be able to accumulate a new coming lot of corresponding process step. When an agent has a (shared of specific) free buffer to accommodate a lot, it can immediately authorize the preceding agent to process new lot. However, when the agent does not have any free buffers to accommodate new lot, it sends the ETA (a time in future) when it can accommodate a new lot from its preceding agent. According to the received ETA, the agents plan their dispatching and defer processing of lots accordingly.

If a shared buffer or the reserved buffer of a process step is free, agent sends a value 0 for `ETA` in requirement message to preceding agent. If all shared buffers of agent including the reserved buffer of a process step are occupied, agent calculates the time (`ETA`) at which a buffer will become free according to its dispatching plan.

$$\texttt{ETA} = \begin{cases} \mathbf{0}, & \text{if a shared buffer or reserved buffer of process is free} \\ \textbf{when a buffer will become free according to plan based on dispatching} & \text{if all shared buffers including reserved buffer of process step are occupied} \end{cases}$$

In order to calculate ETA, agent uses the CABS dispatching algorithm (Algorithm 1) to make dispatching plan of its buffered WIP. A new lot can be accommodated when any shared buffer or the reserved buffer of process step becomes free. When all the buffers are occupied, ETA will be a positive time in future and preceding agents should not send lot before that time. In order to honour the ETA of their succeeding agents, agents defer dispatching of their lots accordingly. The dispatching mechanism of CABS is modified to incorporate ETA by addition of the following rule:

RULE: The lot picked for dispatching should have minimum ETA

This additional rules implies that irrespective of other requirements (time limit and criticality), the lots are dispatched in order of their ETA. Alternatively, rather than sitting idle to honour the (late) ETA of a high criticality lot, agent will dispatch a lower criticality job which has earlier ETA. To realize the desired working of ETA mechanism in CABS, Algorithm 4 is called before CABS dispatching algorithm (Algorithm 1).

Algorithm 4 **pruneTasksETA**(message $im[\,]$) of agent A_j

1: $t_{min} \leftarrow$ among $\forall i \in \{1, \cdots, t_j\}$ find i with minimum $im[i]$.ETA // i.e. among all tasks, task t_{min} has the earliest ETA
2: **for all** $i \in \{1, \cdots, t_j\}$ **do**
3: **if** ($im[i]$.ETA $> im[t_{min}]$.ETA) **then**
4: delete $im[i]$ from $im[\,]$
5: **end if**
6: **end for**

Algorithm 4 removes tasks with higher ETAs, and dispatching algorithm (Algorithm 1) then chooses a lot to dispatch from the remaining tasks. In case there are multiple tasks with minimum ETA, dispatching works as usual by considering their other requirement parameters. Most of the times, when free buffers are available at agents, ETAs of tasks remain 0 and Algorithm 4 has no effect. In such normal cases, all tasks are considered for dispatching and execution of CABS takes place according to their requirement parameters. However, when buffers become full, agents in CABS prioritize processing of lots with lower ETAs. The last agent of process route assumes a static incoming ETA of 0. The agents towards the end of process generally will have lower ETAs and reserved buffers ensure availability of jobs of all process steps at agents. Hence, by prioritizing lots that will complete earlier and propagating the ETA to remote agents, the flow of lots is perpetually maintained which autonomously avoids the occurrence of permanent bottlenecks.

5. Empirical validation of CABS

In the experiments, using the same semiconductor fabrication problem as in Section 3.1, we evaluate the performance of CABS when there are repeated failures at all agents in the

system. Failures occur randomly based on the exponential distribution. MTBF of agents ranges from 951.6 min to 47, 899.0 min with average of about 6, 900 min. And MTTR of agents is between 0.0 min and 4, 009.0 min with average of about 500 min. Exact data of failure patterns (MTBF and MTTR of all workstations) can be found in the MIMAC datasets.

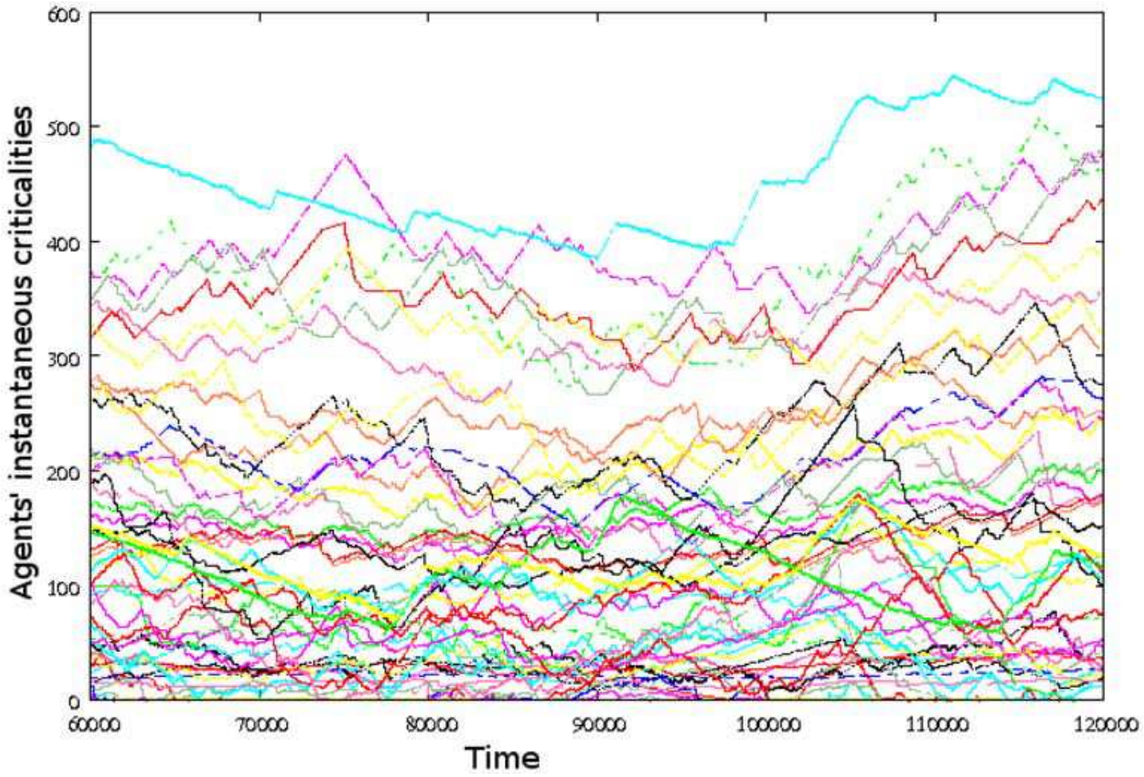

Fig. 14. CABS: Agents' instantaneous `criticality`

Because of dynamic changes of workstations' capacity and disruptions in product flows due to failures, bottleneck workstations shift with time. In CABS we consider the dynamic `criticality` of individual agents as their bottleneck factor. Figure 14 shows how `criticality` values of different agents change in one of the experiments. Figure 14 gives an idea of system's dynamism and shows how different agents can become bottlenecks during the course of execution.

As a criteria to evaluate performance of CABS, we exploit leadtime that achieves the same level of throughput. In queueing theory, Little's Law (Little 1961) states that the expected WIP equals the average leadtime multiplied by the average throughput. Therefore, with fixed throughput, reducing leadtime 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 an idle time of machines due to lack of jobs to process. The system should strike a suitable balance between leadtime (or WIP) and throughput in the face of failures. Hence, the system that requires less leadtime to achieve the same throughput is considered more efficient and robust against failures.

To integrate system's performance on multiple types of products with different manufacturing processes, we calculate the aggregated processing time of all the products as

$$\sum_{i \in ProductSet} Process\_Time_i Throughput_i$$

for representing overall throughput of the system and calculate the aggregated leadtime as

$$\sum_{i \in ProductSet} Leadtime_i Throughput_i$$

for representing overall leadtime of the system. Because

$$Leadtime = Process\_Time + Wait\_Time$$

a system with a smaller aggregated leadtime corresponding to the same aggregated processing time is more efficient and robust against failures.

5.1 Comparison with CONWIP

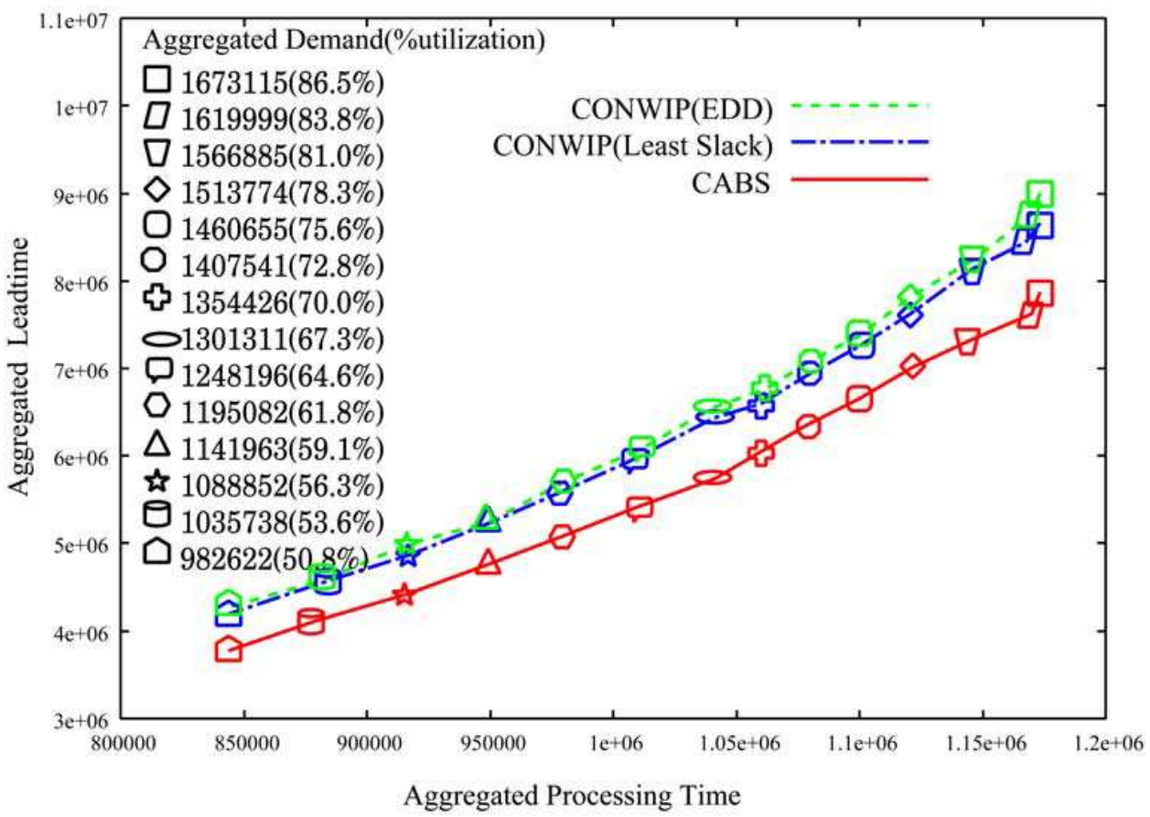

Fig. 15. Comparison of CABS and CONWIP in terms of tradeoff between leadtime and processing time

We compared the performances of CABS with those of conventional manufacturing control methods: CONWIP using the earliest due date first (EDD) dispatching rule and CONWIP

using the Least Slack rule[4]. CONWIP processes jobs at the maximum speed as long as there are jobs waiting to be processed and attempts to maintain a constant level of WIP throughout a manufacturing system by introducing a new task into the system only after a processed task has left the system. Since CONWIP is more flexible than JIT, it is supposed to be more tolerant against instability of systems.

In the experiment, we compare performance of CABS and CONWIP based on more than ten demand rates in which utilization of a bottleneck agent ranges from 50.8% to 86.5%. Since the optimal WIP level for CONWIP can be decided only through trials and errors, we have conducted extensive search (i.e., more than 1,000 runs of CONWIP simulation) of the optimal WIP level for CONWIP at each demand rate of the experiments. Then we compare the best result of CONWIP with the result of CABS. To be noted is that CABS determines its WIP level autonomously in realtime without any input from users.

For both CABS and CONWIP, we conducted five independent runs of the system with different random failures. The average of those results is shown as a single datapoint in Figure 15. The experiments simulated over two months of operation after the system's initial stabilization at its startup.

Unlike to a single failure scenario in Section 3.2, in the experiments, since failures occur randomly at all the agents, CABS may not be able to fully exploit its flexibility of controlling flows of tasks to increase production of the appropriate products during failures and after their resolutions. However, Figure 15 shows that CABS performs better than CONWIP with EDD rule and Least Slack rule, succeeding to achieve higher throughput with shorter leadtime, especially in the region of high demand rates. Figure 15 also shows that both CABS and CONWIP fail to achieve the desired high demands due to failures.

5.2 Effects of buffer size limitations

In this section, we evaluate effects of each agent's buffer size to the overall behaviors of CABS. By using its messaging in conjunction with the specified buffer mechanism, CABS can autonomously avoid deadlocks in a distributed manner. In the experiments, we investigate how the size of shared buffers has effects on the performance of CABS.

In order to see the effectiveness of deadlock avoidance mechanism, we have conducted experiments with different buffer capacity at agents. Figure 16 shows the performances of CABS with the different shared buffer sizes. Each line shows the performance of CABS with a certain shared buffer size for various demand rates. In the experiment, each agent of CABS has the same size of shared buffer as depicted in a graph. From the graph it is clear that as we decrease the buffers the performance of CABS degrades. The drop in performance is logical and we have verified that even with minimal shared buffers (size 0), the system continues to function and does not get stalled due to permanent deadlocks. In absence of our deadlock avoidance mechanism, the system comes to a standstill after executing for some time. The probability of deadlocks increases as we reduce the number of buffers.

The performance drop that is caused by reduction of buffer capacity is logical and is also explained in Section 3.2 for hypothetical system-level buffer capacity (Figures 9 -11). In

[4] See (Blackstone, Phillips & Hogg 1982) for detailed presentation of the rules.

order to maintain utilization, agents need to process alternative jobs when some jobs are unavailable due to failures. In the experiment there are many agents who are not processing both kinds of jobs. Utilization of those agents cannot be maintained by increasing the production of alternative products during failures. In order to maintain utilization, those agents need to keep on processing the products that may not be finished because of failures and utilize the shared buffers to keep WIP of those products. Therefore, CABS with smaller buffer sizes has more chances of losing its agents' capacity, and is more likely to degrade the tradeoff between leadtime and throughput.

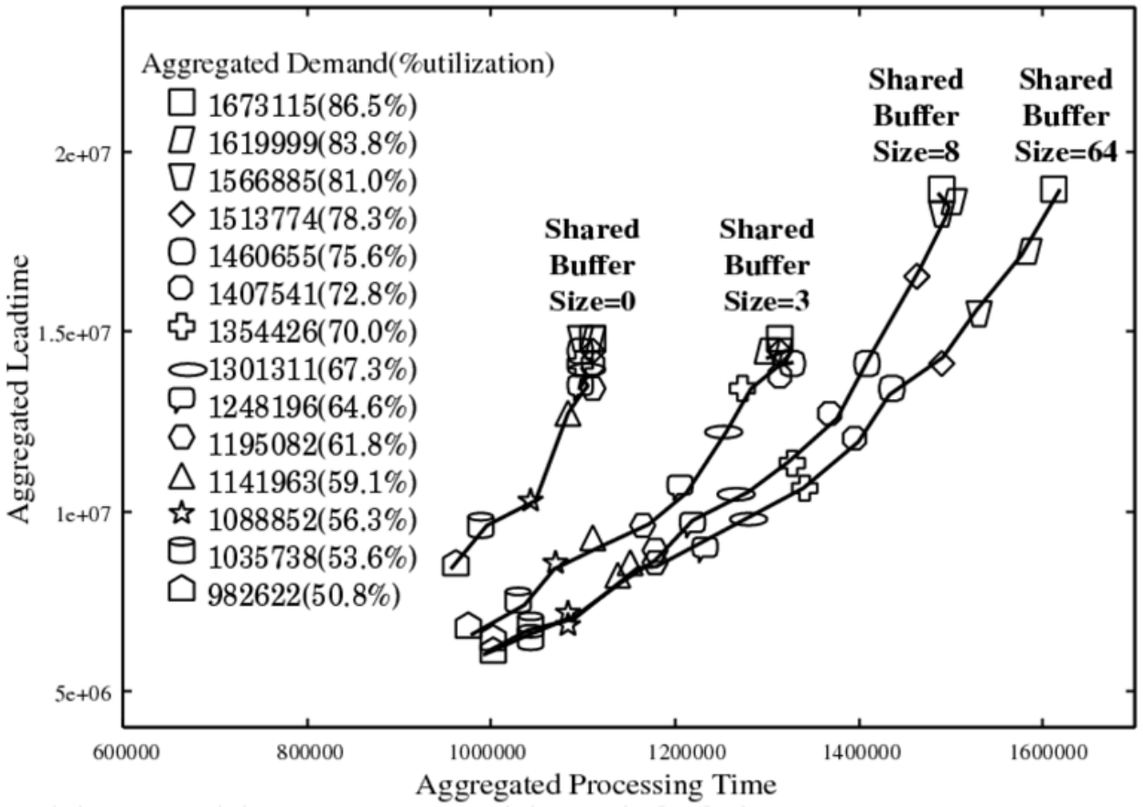

Fig. 16. Effect of buffer size limitations in CABS in terms of tradeoff between leadtime and processing time

6. Conclusion and future works

In this paper, we investigated coordination techniques for maintaining desired throughput of a semiconductor manufacturing system in the face of machine failures. The proposed system, CABS, coordinates the action of agents (i.e., workstations) through a message-passing mechanism that is similar to other token-based coordination methods. Among other methods, what is unique in CABS is the contents of the message and the ways to use them. By passing and utilizing the information of criticalities and job requirements of downstream agents, CABS can sustain high throughput by preventing starvation of wandering

bottleneck agents and, simultaneously, achieve short leadtime by reducing the amount of inventories in the system.

In experiments using data of a semiconductor fabrication process, we have shown that CABS can compensate for capacity loss caused by a machine failure efficiently and validated that CABS achieves a better tradeoff between throughput and leadtime than a conventional manufacturing control method CONWIP. We believe that the coordination mechanism of CABS is suitable not only for semiconductor manufacturing, but also for other complex and unstable network systems such as transportation and communication.

We also proposed our new distributed deadlock avoidance mechanism. The deadlock avoidance algorithm works independently and can be therefore used with other dispatching and control mechanisms, other that CABS. We have shown that the proposed deadlock avoidance mechanism is distributed and it autonomously avoids the occurrence of permanent deadlocks. However, the present mechanism requires marking and reserving of some buffers (specific buffers). This additional constraint may lead to underutilization of reserved buffers which may not get utilized optimally. As another future work we want to investigate if and how this requirement of permanent reserved buffered can be avoided. But dynamically reserving the buffers on need basis, more buffers can be used as shared buffers and overall buffer occupancy and system's efficiency can be improved.

7. References

Allen, A. O. (1990), Probability, Statistics, and Queueing Theory, Academic Press.

Barabási, A.-L. (2002), *LINKED: The New Science of Networks*, Perseus Books Group.

Blackstone, J. H., Phillips, D. T. & Hogg, G. L. (1982), 'A state-of-the-art survey of dispatchin rules for manufacturing job shop operations', *International Journal of Production Research* 20, 27–45.
URL: http://www.informaworld.com/10.1080/00207548208947745

Durfee, E. H. (1996), Planning in distributed artificial intelligence, *in* G. O'Hare & N. R. Jennings, eds, 'Foundations of Distributed Artificial Intelligence', John Wiley & Sons, New York, NY, chapter 8, pp. 231–245.

Faltings, B. & Nguyen, Q. (2005), Multi-agent coordination using local search, *in* 'Proceedings of IJCAI-05', pp. 953–958.

Fowler, J., Hogg, G. & Mason, S. (2002), 'Workload control in the semiconductor industry', *Production Planning & Control* 13(7), 568–578.

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

Gautam, R. & Miyashita, K. (2007*a*), Coordination to avoid starvation of bottleneck agents in a large network system, *in* M. M. Veloso, ed., 'IJCAI', pp. 1281–1286. URL: *http://www.ijcai.org/papers07/Papers/IJCAI07-207.pdf*

Gautam, R. & Miyashita, K. (2007*b*), Robust coordination to sustain throughput of an unstable agent network, *in* 'AAMAS '07: Sixth international joint conference on autonomous agents and multiagent systems'.

Glassey, C. & Resende, M. (1988), 'Closed-loop job release control for vlsi circuit manufacturing', *IEEE Transactions on Semiconductor Manufacturing* 1(1), 36–46. URL: *http://ieeexplore.ieee.org/xpls/abs all.jsp?arnumber=4371*

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

Hopp, W. J. & Spearman, M. L. (2000), *FACTORY PHYSICS*, second edn, McGraw-Hill.

Imai, M. (1997), Gemba Kaizen: A Commonsense, Low-cost Approach to Management, McGraw-Hill.

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

Liberopoulos, G. & Dallery, Y. (2000), 'A unified framework for pull control mechanisms in multi-stage manufacturing systems', *Annals of Operations Research* 93, 325–355.

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

Miyashita, K., Okazaki, T. & Matsuo, H. (2004), Simulation-based advanced wip management and control in semiconductor manufacturing, *in* 'WSC '04: Proceedings of the 36th conference on Winter simulation', Winter Simulation Conference, pp. 1943–1950.
URL: http://www.ingentaconnect.com/content/tandf/tprs/2000/*00000038/00000008/art00010*

Moyaux, T., Chaib-draa, B. & D'Amours, S. (2003), Multi-agent coordination based on tokens: Reduction of the bullwhip effect in a forest supply chain, *in* 'Proceedings of AAMAS-03', pp. 670–677.

Ohno, T. (1988), *Toyota Production System: Beyond Large-Scale Production*, Productivity Press. Pfund, M., Mason, S. & Fowler, J. (2006), Dispatching and scheduling in semiconductor manufacturing, *in* J. W. Herrmann, ed., 'Handbook of Production Scheduling', Springer.

Riley, P. & Riley, G. (2003), SPADES — a distributed agent simulation environment with software-in-the-loop execution, *in* P. J. S. Chick, S., D. Ferrin & D. J. Morrice, eds, 'Winter Simulation Conference Proceedings', Vol. 1, pp. 817–825.

Roser, C., Nakano, M. & Tanaka, M. (2002), Productivity improvement: shifting bottleneck detection, *in* J. L. Snowdon & J. M. Charnes, eds, 'Winter Simulation Conference', ACM, pp. 1079–1086. URL: *http://doi.acm.org/10.1145/1030453.1030609*

Sandholm, T. W. (1999), Distributed rational decision making, *in* G. Weiss, ed., 'Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence', The MIT Press, Cambridge, MA, USA, chapter 5, pp. 201–258.

Venkatesh, S. & Smith, J. (2005), 'An evaluation of deadlock handling strategies in semiconductor cluster tools', *IEEE Transactions on Semiconductor Manufacturing* 18(1), 197–201.

Wagner, T., Guralnik, V. & Phelps, J. (2003), A key-based coordination algorithm for dynamic readiness and repair service coordination, *in* 'AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems', ACM Press, New York, NY, USA, pp. 757–764.

Xu, Y., Scerri, P., Yu, B., Okamoto, S., Lewis, M. & Sycara, K. (2005), An integrated token-based algorithm for scalable coordination, *in* 'Proceedings of AAMAS-05', pp. 407–414.

Multiagent Systems

Edited by Salman Ahmed and Mohd Noh Karsiti

ISBN 978-3-902613-51-6

Hard cover, 426 pages

Publisher I-Tech Education and Publishing

Published online 01, January, 2009

Published in print edition January, 2009

Multi agent systems involve a team of agents working together socially to accomplish a task. An agent can be social in many ways. One is when an agent helps others in solving complex problems. The field of multi agent systems investigates the process underlying distributed problem solving and designs some protocols and mechanisms involved in this process. This book presents an overview of some of the research issues in the field of multi agents. It is a presentation of a combination of different research issues which are pursued by researchers in the domain of multi agent systems as they are one of the best ways to understand and model human societies and behaviours. In fact, such systems are the systems of the future.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen