A Coordination Mechanism to Sustain Throughput of Failure-Prone Agent Networks

Graduate School of Systems and Information Engineering

University of Tsukuba

March 2008

Rajesh Gautam

Abstract

We present a multiagent coordination technique to maintain throughput of a large-scale agent network system in the face of failures of agents. The jobs have to be processed at multiple agents in a given sequence and unexpected failure of agents deteriorates system throughput besides creating and changing bottlenecks in the system. Since loss of bottleneck's capacity degrades the overall system performance, the system should identify bottlenecks dynamically and keep their utilization at a high level. In our system, CABS, information about an agent's criticality to fulfill demanded throughput and maintain its designated utilization is passed to upstream agents in the network in the form of messages. The upstream agents utilize this information to identify bottleneck agents and coordinate their actions to provide the bottlenecks with necessary and sufficient jobs for preventing their starvation and congestion. We empirically evaluate CABS using a benchmark problem of the semiconductor fabrication process, which is a good example of a large-scale network system, in comparison with a well-known traditional manufacturing control method, CONWIP. We show CABS outperforms CONWIP in terms of trade-off between throughput and leadtime under various conditions of manufacturing. We also investigate how network flexibility (such as buffers and job releases) makes influences on the performance of CABS, and analyze effectiveness of each component in CABS's message for coordinating agents' behavior.

Acknowledgements

It is a pleasure to acknowledge and thank the people who made this thesis possible.

First of foremost, I would like to express my sincere gratitude to Kazuo Miyashita, Ph.D., who has been my advisor since the beginning of my study. He provided me with many helpful suggestions, important advice and constant encouragement during the course of this work. Without his invaluable help in research and technical writing, this research would never have reached the level it has. Because of him, I thoroughly enjoyed the four years of this research and I feel extremely privileged and grateful for his tutelage, mentoring, and friendship.

I would like to thank the committee Chair Prof. Seiichi Nishihara for his help related to academic matters. I also thank other committee members Prof. Hitoshi Kanoh, Prof. Ushio Sumita and Prof. Yukio Fukui for their valuable advice.

I would like to thank Dr. Haruhisa Kurokawa, the group leader of Distributed System Design Research Group of AIST where I did my research for this thesis. I am extremely grateful to him and other staff members of group for their kind support. I greatly benefited from their valuable inputs and interactions. Their friendship, support and willingness to help me in technical and other matters made my work and stay in Japan very pleasurable.

I would like to thank my wife, Sheelam, for her love, support and cooperation. To help me fulfil my ambition, she has gone through difficult times and besides bearing with my late hours, spoiled weekends and my bad temper, she single-handedly raised our newborn daughter, Sayuri. Without her help and encouragement, this study would not have been completed. My grateful thanks to my parents who always kept me away from other responsibilities and allowed me to concentrate on my studies.

Contents

List of Figures

Chapter 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 desired 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 the 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 the manufacturing cost, 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 affect throughout the system. Because of the size and complexity of the 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 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.

1.1 The Problem

The capital cost to build and equip a semiconductor fabrication facility (*fabs*) 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). The semiconductor fabrication process consists of a complex sequence of process steps, with the number of operations typically in hundreds (Pfund, Mason & Fowler 2006). The various steps of the sequence are to be processed at different workstations in the 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.

Manufacturing systems are a kind of queuing network and are governed by the Little's Law (Little 1961) of queueing theory. Little's Law states that the expected number of work in process (WIP) equals average *leadtime* multiplied by average throughput as shown in Equation 1.1. The *leadtime*, also known as turn-around time, is the duration of time for which a job stays in the system i.e.

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

the sum of its total processing and waiting time in the system. This relation of WIP, throughput and leadtime is shown graphically in Figure 1.1 and Figure 1.2.

$$WIP = Throughput \text{ x } Leadtime \tag{1.1}$$

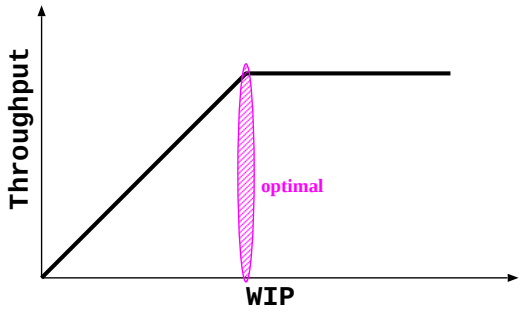

Figure 1.1: WIP-Throughput

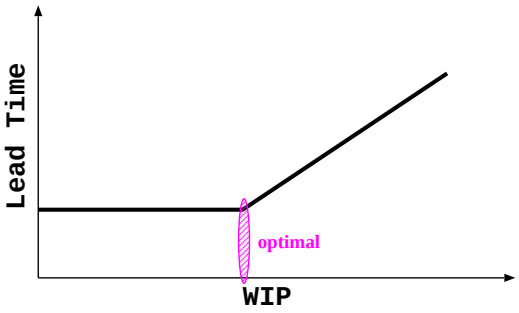

Figure 1.2: WIP-LeadTime

As shown in Figure 1.1, the availability of lots in the system (WIP) has to be increased in order to achieve higher throughput. As the system has finite capacity, after its utilization reaches 100%, increasing WIP further cannot increase the throughput which saturates at its maximum value. Figure 1.2 shows that when there is less WIP, the system has capacity to process its WIP which implies that lots do not wait at workstations before getting processed and the lead time thus remains constant (equal to the processing time). Figure 1.2 also shows that after the system capacity is reached, the extra lots (WIP) have to wait in the queue as the system is already fully occupied and cannot process the lots at increased rate. The amount of surplus WIP increases the queue lengths which results in longer waiting time (leadtime) that the lots experience in the system.

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. In semiconductor fabrication, definitive results are not available until circuits are completely fabricated on the wafer. The number of process steps in semiconductor fabrication is typically in hundreds and average

lead time over a couple of months (Atherton & Atherton 1995) and shorter leadtime enables early diagnosis of problems in products and processes, allowing for faster refinement and avoiding potential losses. As the leadtime is proportional to the amount of WIP (according to Equation 1.1), longer leadtime also causes higher carrying costs for partially finished goods, more space for WIP storage, additional resources for product tracking and control, and many other additional expenses. Consequently, besides achieving high throughput rates due to the capital-intensive nature of the manufacturing infrastructure, simultaneously minimizing the leaditme is critically important in order to be profitable (Kumar & Kumar 2001).

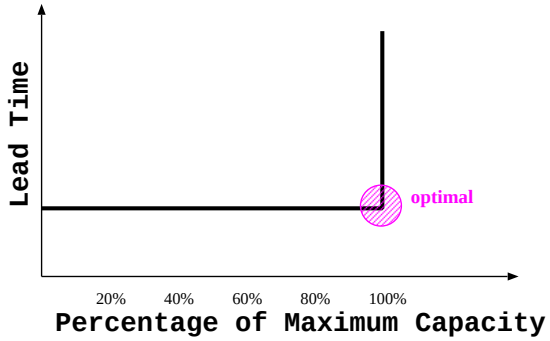

Figure 1.3: Throughput-Lead Time

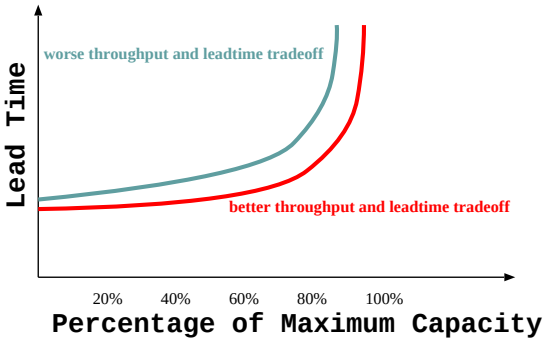

Figure 1.4: Throughput-LeadTime (with failures)

The plot of throughput versus leadtime corresponding to Figure 1.1 and Figure 1.2 is shown in Figure 1.3. Figure 1.3 shows that the leadtime remains constant till the throughput of system increases to its capacity, after which it linearly increases as more WIP is added in the system. The optimal operating condition that maximizes the throughput and has the minimum leadtime is highlighted in Figures 1.1, 1.2 and 1.3. This ideal performance means that the system should have only enough WIP to utilize the full capacity of system. In a network system, due to 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*). The optimal functionality thus means that a new lot invariably arrives at the bottleneck exactly at the time when it finishes processing its current job so that (a) the resource is never idle to keep its 100% utilization

and (b) the lot never waits for processing which is required to keep its leadtime to minimum. As lots have to pass through many workstations depending on the process routes of products, such ideal performance can be achieved only when there is no variability in the system, which is seldom the case in real world.

Variability is anything that causes the manufacturing system to vary from regular predictable behavior. The flow of jobs from a workstation is delayed unexpectedly due to random events such as workstation failure, rework on a job due to some quality problem and operator non-availability. We refer to all such unexpected disruptions as resource failures because all of them have same effect, i.e. the lots to succeeding workstations are delayed for a random duration at random times. The network system usually has 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 resource 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. As such inevitable random events prohibit the system from maintaining ideal behavior, the achieved performance of real systems is shown in Figure 1.4.

As the demand is admitted to realize maximum utilization of system (bottlenecks), the starvation due to failures can cause irrecoverable loss of manufacturing yield. Maximizing throughput of the system therefore means keeping the maximum utilization of the bottleneck resources. High utilization of the such critical resources can be achieved by ensuring availability of jobs for such resources during possible failures of preceding resources. Maintaining extra amount of jobs (WIP) before a resource acts as safety buffer which the resource can process to maintain its utilization when flow of its incoming jobs gets disrupted during random failure on some preceding resource. As increasing the amount of WIP in system increases the leadtime (Equation 1.1), it is desired

that the surplus WIP is kept to minimum. However, with a variable and unpredictable environment, reducing WIP tends to reduce throughput by decreasing the buffering among resources so that downtime periods at any resource have a high probability of forcing idle time spent elsewhere due to lack of work to process. Hence, there is a trade-off between leadtime and throughput in the unstable network systems and the typical performance achieved in real systems is similar to what is shown in 1.4.

Lean manufacturing, that combines high throughput with low WIP has become a minimum requirement for competitiveness in manufacturing industry (Womack, Jones & Roos 1991). The performance of manufacturing system is dependent on (a) the entry of jobs in the system (release control), (b) the movement of jobs within the system by sequencing of jobs at individual workstations, and (c) possible routing. These decisions are taken by a manufacturing control system which is responsible for the execution of manufacturing process. As shown in Figure 1.4, a manufacturing control mechanism that achieves higher throughput with lesser leadtime is considered better. The manufacturing control mechanism can improve system performance by dynamically adapting its behavior to the volatile environment which results from unexpected failures and other disruptions.

The traditional manufacturing control systems are unable to adapt and evolve in the face of disruptions. By aggregating the information that is distributed across the system at a central point, the decision making of centralized and hierarchical control mechanisms can provide good optimization. However, due to their high computational and messaging overheads, effectiveness of such approaches is greatly reduced in large dynamic environments in which frequent random disruptions occur all over the system. The heterarchical control approaches on the other hand are able to provide responsiveness to dynamic situations, but they may not be able to achieve desired global performance due to the myopic vision of its decision making entities.

Because of the limitations of centralized solutions to inherently distributed and dynamic problem, efforts have been made to use distributed intelligent techniques for controlling large manufacturing systems. Recently, multiagent systems have shown great potential in solving complex and dynamic problems. A multiagent system is a distributed system made up of autonomous intelligent

agents that are able to sense and influence only their local environment. Owing to the distribution of decision making, agent based systems can react locally to local changes faster than their centralized counterparts, thus providing scalability and adaptability to rapid changes. Rather than finding a optimal solution by gathering and using all the relevant information at one place, multiagent systems have an emergent behavior which results from the decision making of individual agents. As the overall system performance is dependent on the actions taken by all the agents of system, coordination among agents is of prime importance in order to ensure global coherence of agents' local decisions. In order to make globally coherent decisions, agents coordinate their actions by communicating and interacting with other agents.

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 the 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.

1.2 Objectives and Contributions

The objective of this dissertation is to propose a multiagent coordination mechanism for large scale networks which can improve the overall system performance by promptly and effectively reacting to unexpected failures in the system. In order to mitigate the adverse effect of unexpected disruptions, agents work towards the goal of avoiding starvation of bottleneck agents. Agents monitor

the starvation of resources and avoid it by requesting jobs from other agents by sending messages. By analyzing the information in propagated messages, agents identify the location and requirements of dynamically emerging bottlenecks and coordinate their actions to avoid the bottleneck's starvation. Towards the goal of improving the performance of manufacturing system by using proposed mechanism, the contributions of the thesis are as follows:

- *Agent model of manufacturing system*: We have developed the agent system by modelling the resources (workstations) as the agents. Centralized control mechanisms cannot handle large, complex and dynamic manufacturing systems effectively and trend is to move towards more distributed control. By having a distributed agent model, it is possible to apply distributed control strategies and manage the execution of manufacturing system in a robust manner.

- *New coordination mechanism*: we propose a decentralized coordination mechanism to improve the global performance of dynamic systems. In order to successfully adapt to the changes in the system, the relevant agents should be informed about the changes in the system. Through message passing defined in our coordination mechanism, relevant agents are informed about the changing status of system. Agents decide on their actions by analyzing their local environment and information that they receive from other agents. Besides changing their own actions, agents send additional messages to other agents based on the severity of disruptions.

- *Identification of dynamic bottlenecks*: Due to connectivity and dependency of processing steps, the performance of manufacturing system is critically driven by the bottlenecks in the system. Bottleneck resources limit the throughput of the system and many temporary bottlenecks are created during the course of execution due to capacity losses caused by random failures. Our mechanism identifies bottlenecks in a distributed manner which enables effective coordination of agents' actions resulting in better system performance.

- *Development of Simulation Prototype*: The software prototype is developed to realize the proposed mechanism. The experiments are done using

realistic industrial test data in order to study and evaluate the performance of proposed mechanism.

- *Measurable performance improvement*: Simultaneously increasing the throughput and decreasing the leadtime in large failure-prone manufacturing systems is a challenging task. By using our coordination mechanism, relevant resources of the manufacturing system can autonomously change their behavior appropriately to mitigate the affect of failures. In comparison with popular conventional system, our system achieves a better throughput-leadtime ratio which is an important metric of manufacturing system performance.

- *Scalability of coordination mechanism*: We have improved on the measurable global system performance of relatively large network system (semiconductor fabrication). This is significant as existing sophisticated multiagent coordination mechanisms do not scale well to large agent networks.

- *Applicability to other agent networks*: Networks such as wide-area transport and data networks have similar characteristics and objectives like the manufacturing systems. We have evaluated the performance of proposed coordination mechanism by regulating the system parameters that are more relevant to other networks.

1.3 Organization of Dissertation

The dissertation describes the background of the problem domain, challenges and past research in Chapter 2. In Chapter 3 we define the multiagent coordination mechanisms that is proposed is in dissertation. The proposed coordination mechanism is validated by a series of experiments in Chapters 4, 5 and 6. Chapter 7 and Chapter 8 conclude the dissertation by discussions on conclusions and future works.

Chapter 2 reviews of the state-of-the-art in control of manufacturing systems. After describing the problem of manufacturing control, survey of various techniques are provided. The chapter surveys the various centralized and artificial intelligence based techniques. An overview of agents and multiagent system

is provided and various multiagent based techniques and models from past research are surveyed. The last section of the chapter compares and evaluates the various surveyed techniques, which provides the basis for the definition of our proposed mechanism.

In Chapter 3 the proposed coordination mechanism is defined. The agent model, details of messages and algorithms used by agents for actions and message generations are presented. Later in the chapter, the working of proposed coordination mechanism is explained in details by using an example of a small manufacturing system.

Chapter 4 describes the semiconductor manufacturing process that we have used for experiments in the dissertation. After providing details of experiments, the empirical results of comparison with a conventional system are presented.

Chapter 5 evaluate the applicability of proposed mechanism systems with different characteristics. The effects of buffer capacity and the flexibility of releasing jobs in the system are evaluated in the various experiments presented the chapter. The chapter also describes the integrated deadlock avoidance mechanism that is utililized in the proposed mechanism.

Chapter 6 evaluates the significance of various message parameters that are used for coordination in the proposed mechanism. Results of regulating each individual message parameter are provided in the chapter.

Chapter 7 summarizes the main achievements of the research, presents the general conclusions, and then Chapter 8 proposes directions for future research that might be undertaken.

Additional details about implementation and features which are not the main focus of research are provided in appendixes.

Chapter 2

Past Research

After their entry into the network system, jobs of have to pass (get processed) through a number of nodes before they leave the system after completion. The behavior of network that decides the flow of jobs is dependent on following decisions:

- **Release**: The performance of individual resources and system as a whole is limited by the availability of jobs, which in turn is dependent on the entry of jobs in the system. As the law of conservation should hold, the output from system cannot be more than the number of jobs that enter in it. The entry of jobs, i.e. when and how many jobs enter the system limits the availability of jobs within the system and the flow of completed jobs that comes out of it. In the domain of manufacturing systems this control of feeding jobs to system is also known as *release* control.

- **Dispatching**: In network systems, the nodes generally have buffers in which the incoming jobs are parked before they are processed. Many nodes of network process multiple kind of jobs and at times, different kind of jobs accumulate in node's incoming buffer when it is not able to process the jobs at their incoming rate. When the resource at node becomes ready to process another job, it can choose any one of the buffered jobs to process next and this decision in manufacturing domain is called sequencing or *dispatching* control. As the processing of jobs at a node takes time, the dispatching decisions of a node control the availability of jobs at their subsequent nodes. After its entry into the system, the

dispatching decisions of relevant nodes affects the waiting duration and completion time of individual jobs.

- **Routing**: In many network systems, jobs can be completed by different alternative routes. The possible routes may depend on the capability of nodes and job's requirements. The possibility of alternative routes means that at some nodes, the processed job can be sent to a set possible nodes for further processing. The decision of choosing the node to which the processed job is sent is called *routing* control. **Routing control is not considered in this thesis and the work is done under the assumption that routes of jobs are fixed.**

Because of the capacity constrains of various nodes and dependency relationship between processing steps of jobs, a single decision at a node affects the availability of jobs at other, remote nodes in future. Performance of individual nodes depends on the availability of jobs and final throughput of system is decided by the completed jobs that leave the system. The system behavior and its final throughput results from the cumulative effect of all the decisions taken in system during the course of time.

This thesis focuses on the manufacturing systems which are a good example of network systems. Manufacturing systems have been widely studied in research and a wide range of techniques are utilized to improve the system performance. In particular, this thesis uses semiconductor fabrication process which represents a large, complex and dynamic network. Behavior of manufacturing systems is mainly controlled by the release and dispatching decisions. The routes of jobs in system are relatively fixed, which is in line with our assumptions about routing. This chapter gives an overview of past research in the control of manufacturing systems.

2.1 Conventional approaches

Starting from the raw material, manufacturing process involves processing of jobs at different shared resources in order to output the completed jobs. Before commencing the execution, predictive schedules are utilized to plan other activities such as ordering and preparing the raw materials, tooling, set-up planning,

short-term labour planning, shipping logistics etc. Besides supporting these other planning activities, the schedules made in advance are utilized to direct manufacturing operations. Scheduling can be defined as the optimal allocation of resources to jobs, under the constraints of temporal relationships between jobs and capacity limitations of resources. The goal of scheduling process is to generate a plan which optimizes the various performance objectives. Objective may include the maximization of throughput, the minimization of running cost (i.e. the work in process), the minimization of lead time and minimization of tardiness. Optimal schedules for small scheduling problems can be calculated by simple algorithms. However, scheduling is a non-polynomial (NP) problem and it quickly becomes intractable to find optimal schedule due to exponential number of possible solutions. For a generic m machines and n jobs problem, the possible number of schedules are $(n!)^m$. Due to its high combinatorial aspect scheduling has been widely studied in past research under the implicit assumption that no unexpected events occur during the execution. A review of such deterministic scheduling for can be found in MacCarthy & Liu (1993), Błażewicz, Ecker, Pesch, Schmidt & Wkeglarz (1996), Wiers (1997), Jain & Meeran (1998), Jain & Meeran (1999), Jones & Rabelo (1998) and Pinedo (2002).

The predictive schedules are utilized by the PPC (Production Planning and Control) systems which deal with the higher level control of one or several manufacturing plants, materials and manpower requirement planning and other production activities. After the high-level objectives have been finalized, in order to realize the manufacturing plans and achieve planned production, *Manufacturing Control* system is responsible for managing and controlling the physical activities within a manufacturing plant. The focus of this thesis is *Manufacturing Control*, which makes the job release and dispatching decisions at workstations to produce stipulated finished products by optimizing the performance measures such as minimizing running cost (i.e. the work in process), minimization of lead time, minimization of tardiness etc.

Although predictive scheduling assists in other planning activities, the optimal schedules made in advance quickly become obsolete during their execution because of unpredictable events in system. The classical scheduling models and algorithms are unable to use real-time information and as a result there is a gap between theory and practice of classical scheduling (Cowling & Johansson 2002),

(Shukla & Frank Chen 1996), (MacCarthy & Liu 1993). Most manufacturing systems work in dynamic enviromnements in which various unexpected disruptions affect the system performance. In order to be effective in dynamic systems, the scheduling and planning should take place in tandem with execution by accomodating the unexpected changes.

The real-time events that affect the existing plan and execution of manufacturing can be of two types (Stoop & Wiers 1996), (Suresh & Chaudhuri 1993), (Mehta & Uzsoy 1999), (Cowling & Johansson 2002), (Abumaizar & Svestka 1997), (Schumacher, Verwater-Lukszo & Weijnen 1999), (Shafaei & Brunn 2000), (Sabuncuoglu & Bayiz 2000), (Vieira & Herrmann 2000), (Jensen 2001), (Vieira, Herrmann & Lin 2003).

- **Performance related**: machine failure, operator illness, unavailability of specific tools, unavailability of materials, defective material, etc.
- **Demand related**: rush orders, order cancellation, due date or deadline changes, orders that arrive early/late, change in order priority etc.

In this thesis only the *performance related* events are considered and they are collectively referred to as failures. Similar to the failure event, a resource cannot be utilized for any manufacturing during the event of unavailability of operator and specific tools, which are required to operate the machine. **The *demand related* events are not considered in thesis and it is assumed that the demand remains unchanged during the course of execution and additionally, all the orders are considered to have equal priority.**

As unexpected failures invalidate the existing schedules, dynamic scheduling methods are required that react to dynamic events by using real-time information related to status of the resources and tasks present in system. In a dynamic system, where unexpected failures occur frequently, scheduling is an ongoing process and it is desired that the scheduling method is robust against failures. According to Shafaei & Brunn (1999), *a scheduling method is said to be robust if it provides a schedule, the performance of which remains high in the presence of uncertainties.*

In response to dynamic events, the dynamic scheduling methods change the plans in order to keep them synchronized with updated status of system.

The two policies based on how to change the existing schedule can be classified as *schedule repair* and *complete rescheduling* (Sanmarti, Huercio, Espuna & Puigjaner 1996), (Sanmarti 1997), (Sabuncuoglu & Bayiz 2000), (Cowling & Johansson 2002), (Vieira et al. 2003). *Complete rescheduling* policy is simple in the sense that while rescheduling, it regenerates a new schedule from scratch every time. However, this solution has limited practical applicability because disturbances tend to appear very often, and because of the complexity of scheduling, creating a completely new schedule every time requires prohibitive computation time. Furthermore, frequent changes to the schedule can result in instability of the system which results in additional manufacturing costs. The second, more feasible alternative is to only revise parts of the existing schedule rather than generating the complete schedule from scratch. Compared to complete rescheduling, *schedule repair* is more feasible in terms of computation and provides higher stability (Sabuncuoglu & Bayiz 2000).

Besides controlling the amount of change in the schedule, another decision to make is the frequency of changes in the schedule (Sabuncuoglu & Bayiz 2000), (Vieira et al. 2003). As a regular scheduled activity of production, the schedules can be generated periodically at fixed intervals by gathering the system information. After generation, the schedule is executed and is not renewed till the next scheduling cycle. Although the periodic updating mechanism provides more stability to the schedules, its effectiveness is reduced when significant events occur in the middle of scheduling cycle. The frequency of scheduling in a dynamic environment critically affects the performance of schedules which degrades when the rescheduling period is increased (Muhlemann, Lockett & Farn 1982). As an alternative event-driven rescheduling approach, schedule in revised whenever there is an unexpected event that changes the status of system and affects the applicability of existing schedule. Rescheduling in response to relevant events such as failures improves the adaptability and effectiveness of generated schedules (Yamamoto & Nof 1985). However, in a large dynamic system, where large number of events are constantly generated, rescheduling at all the events might not be feasible due to computation overheads and the high volatility of schedule might effect the system performance adversely (Vieira & Herrmann 2000). In a hybrid rescheduling policy, schedules are updated periodically and additionally when an exceptional event occurs. Besides regular periodic updates, the perfor-

mance improves due to increased responsiveness when additional rescheduling is done for critical events such as machine breakdowns and order related change such as addition, deletion or change in priority of orders (Church & Uzsoy 1992).

The control mechanisms for manufacturing systems can be broadly classified into the following categories (Suresh & Chaudhuri 1993), (Shukla & Frank Chen 1996), (Stoop & Wiers 1996), (Jeong & Kim 1998), (MacCarthy 2001): dispatching rules and simulation-based techniques, artificial intelligence-based techniques, and multiagent-based techniques.

2.1.1 Dispatching and Simulation based Techniques

Since scheduling all the jobs on all the machines is a intractable problem in large and dynamic systems, dispatching heuristics have been used as an alternative. Dispatching at a machine refers to sequencing of buffered jobs for processing according to some criteria. Hundreds of dispatching rules have been proposed by researchers and practitioners (Blackstone, Phillips & Hogg 1982). Generally, dispatching rules are applied at individual workstations and hence they are myopic in nature. Since the best decision to pick the next job for processing depends on future jobs as well as the situation at other machines, no dispatching rule works best in all the conditions. However, due to limited availability of scheduling solutions for large realistic systems, dispatching continues to find extensive use in industry.

In order to analyze different dispatching rules, computer simulations are used to assess their performance under different dynamic and stochastic working conditions. Computer simulations provides a mechanism in which one can capture the essence of a real manufacturing system in the form of a detailed model which can be run, tested, and analyzed in many different ways (O'kane 2000). Due to increased availability of computation power and enhancements in modelling capabilities and visualization of generated results, simulation has been extensively used of analysis and validation for all aspects of manufacturing systems. In practice, the simulation engine is provided with the model of manufacturing system and is fed with the demand information and current status of system in term of location and state of jobs in the system. After feeding the current information, the model is run forward in time by applying selected dispatching

rules at each workstation. The schedules are generated by collecting the information about arrival and departure times of jobs from the simulation results. Different schedules are generated by changing the scheduling rules, and the best one according to the desired performance criteria is selected for application.

Apart from dispatching rules, which sequence the available jobs at a workstation, regulating the number of jobs has also been used as mechanism to control the system behavior. Regulating the workload in the system can smooth the system operations by avoiding congestion and achieve shorter and more reliable lead times. Such WIP regulating control is known as pull mechanism and in such systems, a job moves ahead only when it is authorized by the following workstation. The amount of WIP is regulated at some predetermined level, and the scope of regulation can be at each individual workstation or at the system level. A Kanban (Ohno 1988) pull system uses card sets to tightly control WIP between each pair of workstations. Processing of a job occurs at a workstation only if the job has a card authorizing its processing from the corresponding succeeding workstation. As a result, the Kanban system limits maximum of WIP at each workstation, and the upstream workstation is blocked when the buffer is full. In contrast, a CONWIP (Hopp & Spearman 2000) system maintains a global set of cards to control the total amount of WIP in the system. Unlike Kanban, CONWIP pulls jobs only at the beginning of the manufacturing process and the distribution of WIP within the system is not considered. In CONWIP, when a completed job leaves the system, the card associated with the job is released which authorizes release of a new job in the system. Similar to the selection of dispatching rules, the best WIP levels to be maintained during execution are decided by simulations by generating different schedules with different WIP levels. Past research has shown that besides dispatching rules which take effect after the jobs are available in the system, discretionary input control can provide significant improvements in the system performance. As the new jobs are released in controlled manner by using the input regulation policies, lower and more stable leadtimes can be achieved (Hendry & Wong 1994), (Wein 1988).

In order to simplify the intractable problem of scheduling all the machines, various approaches have been developed which focus only on the critical workstations. In a network system, due to 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 *bottleneck*). As behavior of manufacturing system is dominated by the bottlenecks, it is most critical to schedule them. The problem is thus reduced to independently scheduling the bottleneck machines and non-bottleneck workstations can be scheduled by propagating the schedule of bottleneck. Throughput of system is tied with the utilization of bottleneck resource which has to be ensured by maintaining appropriate amount of jobs before the bottleneck in order to avoid its starvation. The idea is motivated by the concept of "theory of constraints" (TOC) (Goldratt 1990) according to which, any manufacturing system is restricted by one or a few constraint machines. The capacity of the constraint machine determines the overall system throughput. In order to maximize the throughput against limited resources, the constraint machine should be protected with a certain level of work in progress (WIP), which is defined as a constraint buffer. The size of the constraint buffer should be kept big enough to avoid starvation on the constraint machine and, meanwhile, relatively small to cut down the WIP inventories.

In Glassey & Resende (1988), Glassey & Petrakian (1989), Glassey & Weng (1991), Lozinski & Glassey (1988) a concept of bottleneck starvation indicators is presented which is used to develop input regulation policies and dispatching rules. The risk of bottleneck's starvation is accessed by the starvation avoidance indicator. They have experimented with semiconductor manufacturing which has process cycles and each job may visit the bottleneck machine several times during the whole manufacturing process. The cycles are different and or each visit to the bottleneck machine, there is a different sequence of upstream machines from the bottleneck where the jobs need to visit. Each sequence through the bottleneck is denoted as a flow and the risk of starvation is evaluated by computing the desired amount of WIP required in the flows for the bottleneck workstation. Due to its highest utilization, any lost time at the bottleneck due to starvation represents an irretrievable loss of final output. As maintaining large amount of WIP increases the leadtime, the appropriate level of WIP to be maintained in the flows to the bottleneck can be calculated a priori. Dispatching and release rules are then used to maintain the preset level of WIP in the flows. A drawback of the starvation avoidance approach is the assumption

of a single and fixed bottleneck whose location is known a priori. The importance of bottlenecks was also shown by Wein (1988), Wein (1990), Wein (1991), Wein (1992), Wein & Chevalier (1992). They also experimented with semiconductor fabrication with different number of predetermined bottlenecks. They experimented with different dispatching and release rules including a workload regulation rule in which new jobs are released into the system when the total work content for the bottleneck workstation falls below the preset limit. They emphasized the importance of bottlenecks and observed that besides the release control, the performance of dispatching rules is highly dependent on the number of bottlenecks in the system. In dynamic systems, the random capacity losses due to failure can cause the existing bottlenecks to shift and new temporary bottlenecks can emerge. In order to identify shifting bottlenecks more accurately in dynamic scenarios, Roser, Nakano & Tanaka (2001) Roser, Nakano & Tanaka (2002), Roser, Nakano & Tanaka (2003) and Faget, Eriksson & Hermann (2005) utilize the information of prevailing system status to identify primary and secondary bottlenecks along with their magnitudes.

2.1.2 Artificial Intelligence Based Techniques

Artificial Intelligence techniques have been used to develop heuristics that can find satisfactory solutions for complex and realistic manufacturing scheduling problems. Various scheduling applications have been developed based on artificial intelligence techniques which include heuristics, knowledge-based systems, fuzzy logic, neural networks, case based reasoning, meta-heuristics which include tabu search, genetic algorithms, simulated annealing etc. (Suresh & Chaudhuri 1993), (Szelke 1994), (Kerr & Szelke 1995), (Zweben & Fox 1994), (Shukla & Frank Chen 1996), (MacCarthy 2001), (Meziane, Vadera, Kobbacy & Proudlove 2000).

Heuristics Based Techniques

Due to the intractable nature of finding the optimal solution to scheduling problem, heuristics have been used to find reasonably good solutions in a short time. Reeves (1993) defines heuristic as *a technique which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guar-*

antee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is. Even though heuristics are not exact, these algorithms are very useful in efficiently solving hard computational problems such as scheduling.

When an existing schedule becomes infeasible due to disruptions, three common methods are used to update (repair) a schedule: right shift rescheduling, regeneration, and partial rescheduling. Using the right shift rescheduling approach, the remaining operations are postponed (shifted) by an amount which will make the schedule feasible (Abumaizar & Svestka 1997). The right-shift refers to shifting the operations to right in a Gnatt chart. Most of the heuristics update the schedule partially, and only repair the operation which are directly or indirectly affected by disruptions. Partial scheduling is also known as affected operations scheduling. Match-up scheduling is another kind of partial scheduling which repairs the production schedule in an event of disruption (Bean, Birge, Mittenthal & Noon 1991). When a disruption occurs, match-up mechanism re-sequences all jobs scheduled before a matchup point, after which the new modified schedule is completely consistent with the initial schedule. The disruptions can be accommodated during the transient period. The matchup point is increased according to the tardiness of jobs, and if matchup point becomes too large, the jobs are reassigned to different machines using integer program or priority rules.

Meta-Heuristics Based Techniques

Various modern heuristics, also called meta-heuristics have also been used to solve the manufacturing scheduling problem. The techniques include tabu-search, simulated annealing and genetic algorithms. Meta-heuristics are methods that sit on top of local search algorithms and are used to improve the the local search heuristics by enabling them to escape or avoid local optima or premature convergence (Reeves 1993), (Voss, Martello, Osman & Roucairol 1999). Local search heuristics are neighborhood search methods based on the idea of searching neighborhood. Local search algorithms work by starting with a initial solution which is generated randomly or heuristically. The solution is then improved by iteratively changing it. The solution is changed by analyzing the neighboring solutions through iterations. The search process terminates

when the current solution cannot be improved any further by moving it to any neighboring solution. The local search is then stuck in what is known as local optimum. Meta-heuristics are designed to control the local search algorithms in order to improve their performance by enabling them to either avoid or escape from the local optima.

After starting from a feasible solution, the search process in tabu search (Glover 1989), (Glover 1990), (Glover, Taillard & de Werra 1993), (Glover & Laguna 1997) iteratively moves from the current solution to its best neighboring solution even if that move worsens the quality of the result. The search mechanism maintains a history of recently examined solution, which is called a tabu-list. Tabu list is a short-term memory, which contains a pre-determined number of previous moves, which are forbidden or declared tabu. The entries in tabu list are maintained for a certain number of iterations and are used to avoid cycles by avoiding revisits to the them in future iterations. In addition, based on a defined aspiration criterion, a move is made by ignoring its tabu status if it improves the best found solution so far.

Simulated annealing meta-heuristic (Aarts & Korst 1989), (Reeves 1993), (Tan & Narasimhan 1997), (Kolonko 1999), (Satake, Morikawa, Takahashi & Nakamura 1999), (Pham & Karaboga 2000) uses an analogy with the way in which liquids freeze and crystallize. Physical annealing of solids is the process of initially melting a substance, and then lowering the temperature gradually. At high temperature, when the substance is in liquid form, its molecules are not restricted and they can move very freely in relation to each other. As the temperature is lowered, the movement of particles is progressively restricted and the substance begins to solidify. The aim of the physical annealing process is to grow solids with a crystalline structure in which the molecules have minimum energy and the resulting solid is said to be in ground state. However, if the liquid is cooled very rapidly, the resulting solid will not be a perfect crystalline structure but a meta-stable structure with irregularities and defects whose molecules will not be in a minimum energy state. The main idea of simulated annealing mechanism is to solve combinatorial optimization problems by a process analogous to the physical annealing. By associating the moves of iterative improvement algorithm to the re-arrangements of the molecules and the quality of solution to the energy of those molecules, simulated annealing mimics the real annealing

process in order to converge on the global optimum. After starting the search from the initial solution with a high temperature, the successive iterations are made on the basis of temperature, which decreases during the course of execution. During an iteration, if the neighboring solution is worse than the current solution, the move is still made but with an probability that is proportional to the current temperature. As the temperature is reduced with the number of iterations, the probability of accept worsening moves decreases resulting in the convergence of the result.

Genetic algorithms (Holland 1975) mimic the way in which species evolve and are analogous to the Darwinian natural selection and mutations in biological reproductions. The solutions are represented by chromosomes and the search process starts with an initial population of individual chromosomes which are generated randomly or heuristically. Each search iteration is analogous to a generation, and the next generation of individuals is generated by applying crossover and mutation operators to the individuals of the current population. Mutation introduces random modification to the individuals, and crossover combines the genetic materials of two individuals of the current population (parents) in to an individual of the next generation (offspring). The individuals for the new generation are selected according to their fitness which is related to the quality of result that an individual represents. The offsprings with higher fitness level are selected for next generation which mimics the principle of survival of fittest in natural selection.

Meta-heuristics have been used for both static and dynamic scheduling. While repairing a schedule, the local search and simple mechanism can get stuck in a local optimum and meta-heuristics can come over this limitation and enable better repairs (Dorn, Kerr & Thalhammer 1995), (Zweben, Daun & Deale 1994), (Dorn et al. 1995), (Zweben et al. 1994). Various meta-heuristics such as tabu search (Mehta & Uzsoy 1999), (Dorn et al. 1995), genetic algorithms (Jensen 2001), (Chryssolouris & Subramaniam 2001), (Rossi, Dini, Chryssolouris & Subramaniam 2000), (Leon, Wu & Storer 1994), (Wu, Storer & Chang 1993), (Bierwirth & Mattfeld 1999), simulated annealing (Zweben & Fox 1994). These studies have shown that meta-heuristics can find better schedules than local search mechanisms when the existing schedules are repaired in response to the disturbances. They also show that compared to the dispatching

rules approach, the performance of system is better when meta-heuristics are used for scheduling.

Knowledge Based Techniques

Finding optimal schedules for large manufacturing systems is infeasible due to the size of complexity of problem and the computational costs involved. The problem is further compounded by the fact that unexpected disruptions require prompt corrective action in order to ensure that the execution takes place consistently towards achieving the manufacturing goals in a dynamic environment. As a result, besides the use of computation, system-specific knowledge and human judgement is also utilized to take corrective actions during disruptions and finds an important place in practical scheduling. Knowledge-based systems capture the domain specific knowledge and expertise in a knowledge-base which is then used by an inference mechanism to derive conclusions and recommendations from the captured knowledge. In manufacturing scheduling, knowledge-based systems are utilized to make decisions when corrective actions are taken during disruptions. When random disturbances occur, the knowledge-base advises the corrective actions based on knowledge gained from experience (Suresh & Chaudhuri 1993), (Szelke 1994), (Zweben & Fox 1994), (Kerr & Szelke 1995), (Shukla & Frank Chen 1996), (MacCarthy 2001), (Meziane et al. 2000). The information is used to generate and repair the schedules. CABINS framework (Miyashita & Sycara 1995) uses case-based reasoning to repair the schedule in job shops. The knowledge base of cases maintains the repair-context and the corresponding repair actions. Using case-based reasoning, the knowledge about similar situations (contexts) in the past is re-used to make repair accordingly. Using the cases stored in the system, the schedule is repaired incrementally when necessary. Besides repairing the schedules, knowledge based techniques have also be been to select dispatching rules (Jahangirian & Conroy 2000), (Li, Li, Li & Hu 2000).

2.2 Multiagent Based Techniques

The final performance of the manufacturing control system is critically dependent on its control structure. The conventional control architectures can be broadly classified as centralized, which rely on top-down process of command and response. A centralized database is maintained to make the control decisions. The central control entity does the information processing and makes decisions which are distributed to manufacturing resources for execution. In contrast with the control flow, information flows from various manufacturing entities to a central computer, which uses the information for scheduling, dispatching, monitoring any deviation, and dispatching corrective actions.

Although the availability of system's complete information in centralized and hierarchical manufacturing control systems may help in making globally optimal decisions, they are found to be inefficient in dynamic environments of today's dynamic manufacturing processes. Experience has shown that centralized systems are unable to respond to the real-time events in large dynamic systems. The occurrence of events in different parts of the system require the centralized controlling entity to be informed about the unexpected change in status. The centralized controller has to collect, manage and process all the information and respond to it by promptly sending information about updated course of action to various relevant entities of system. Today's large manufacturing systems consist of large number of resources and frequent occurrence of events make the central controller a information and processing bottleneck, thus adversely affecting the responsiveness and performance of system.

Due to the increased dynamism of the markets and environment in which today's manufacturing systems operate, in order to remain competitive, there is need for enhanced responsiveness and robustness from the next-generation manufacturing control systems. The next-generation manufacturing control system should be expandable and reconfigurable, i.e. it should be possible to add, remove or reconfigure the components of manufacturing without stopping or re-starting the process. Due to constant integration of new technologies and the market dynamism, manufacturing resources fail unexpectedly, additional resources and new products are introduced in the system, orders' specification and priorities are changed unexpectedly. It is required that next gener-

ation of manufacturing control systems should be robust in handling the disturbances and should minimize the effect of disturbances by reacting promptly and intelligently. Due to the limitations of centralized hierarchical architectures, the decentralized control architecture has been suggested as an alternative to achieve the desired features. As a result, the current trend has been to design modular and distributed automated systems that can offer robustness, stability and adaptability (Parunak 1993), (Parunak 1995), (Parunak 1998*b*), (Parunak 1998*a*), (Tharumarajah & Bemelman 1997), (Brennan & Norrie 1998), (Liu & Zhang 1998), (Shen & Norrie 1999).

Recent research has shown that multiagent technology is a promising approach to provide robust control for dynamic production systems. The features provided by the agent technology closely match those required for efficient control of manufacturing systems i.e. modularity, decentralization, and dynamic and complex structures characteristics (Parunak 1998*b*). Apart from manufacturing control, agent-based approaches have been applied in variety of other domains such as electronic commerce, e-business, air traffic control, process control, telecommunications etc.

2.2.1 Agents and Multiagent Systems

The concept of computational agents can be said to originate from distributed artificial intelligence (DAI) (Bond & Glasser 1988). Over the time, the focus of AI research has shifted from ideal goal-seeking to resource-bound rational behavior. At the same time, due to the advances in technology, computing is becoming more and more ubiquitous. These developments also coincided with the evolution and popularity of network-based computing technology such as Internet and mobile computing due to which the exchange of information between remote entities is becoming a common practice. As a result of these developments there is an increased interest in the study of multiple cognitive entities acting in communities, which represent the new agent paradigm of computing.

Although researchers broadly agree on the characteristics of agents, many researchers and groups have proposed their definition of agents, and no definition is universally accepted. Various definitions from different disciplines have been proposed for the term *agent*. Wooldridge & Jennings (1995) definition seems

to be the most popular and widely used working definition of agent; according to them: *an agent is a computer system, situated in some environment, and capable of flexible and autonomous action in that environment in order to meet its design objectives. By flexible we mean that the system must be responsive, proactive, and social.* Recently, Ferber (1999) arrived at a minimal common definition of an agent. An agent is a physical or virtual entity that:

- is capable of acting in an environment;
- can communicate directly with other agents;
- is driven by an autonomous set of individual goals or objectives;
- possesses resources of its own;
- is capable of perceiving its environment to some extent;
- has only a partial representation of this environment;
- possesses skills and can offer services;
- may be able to reproduce itself;
- whose behavior tends towards satisfying its objectives, taking account of the resources and skills available to it.

Based on the various definitions of agents, important capabilities of computational agents can be considered as autonomy, reactivity, proactiveness, social ability, flexibility, personalization, adaptation, cooperation, deliberative capability, and mobility (Papazoglou, Laufmann & Sellis 1992).

- `Autonomy`: Agents themselves control their internal state and behavior and operate without the intervention of users.
- `Reactivity`: Agents perceive their environment in which they exist, and respond to various stimuli in a timely fashion in order to meet their design objectives.
- `Proactiveness`: Besides reacting to the changes in their environment, agents constantly exhibit goal-directed behavior in order to satisfy their design objectives.

- `Social ability`: Based on the selected communication language and protocol, agents interact with other agents of the system.
- `Flexibility`: Agents are able to show reactivity, proactiveness, and social ability simultaneously.
- `Personalization`: The agents are able to personalize their behavior according to user specific interests and preferences.
- `Adaptation`: By learning to react and interact with the changing environment, the agents can adapt themselves to dynamic conditions.
- `Cooperation`: The agents can interact act with other agents and to cooperate their actions in order to work towards a common, high-level objective.
- `Deliberative capability`: The agents can plan by reasoning about the world model.
- `Mobility`: The agents can transport itself and migrate to different nodes of a network.

The implemented agents may show only some of the above properties. Nevertheless, what differentiates agent based intelligence from the classical intelligence is the fact that agents generally have incomplete knowledge about the system, and are designed and supposed to have the resources for limited reasoning.

A multiagent system (MAS) is formed by a network of interacting agents. There are various definitions for the term *multiagent system*. According to Durfee, Lesser & Corkill (1989*b*) MAS is *a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver*. More recently, Ferber (1999) defined a MAS as *a system composed of a population of autonomous agents, which interact with each other to reach common objectives, while simultaneously each agent pursues individual objectives*. According to Jennings, Sycara & Wooldridge (1998), the main characteristics of MAS are:

- Each agent has incomplete information, or capabilities for solving the problem, thus each agent has a limited viewpoint;
- There is no global system control;
- Data is decentralized;
- Computation is asynchronous.

In MAS, the agents are autonomous which implies that they are not controlled or managed by any other entity or agent. Knowledge and control of system is distributed across agents and each agent maintains its own knowledge and is responsible for its own actions. In order to achieve the global objective, agents communicate and interact with other agents in the system. As the agents are present in the same environment, the actions and decisions of various agents are affected either by direct communication between them or through the changes that they make to the environment. Due to lack of centralized information and control, a global behavior cannot be imposed on system and emergent behavior results from the interaction of individual agents that have limited knowledge and competence.

As the global behavior depends on the collective behavior of individual agents, the agents of MAS must cooperate, coordinate, and communicate with each other in order to solve the global problem coherently. The interactions are required either because individual agents do not have sufficient capabilities or resources to solve the problem alone, or because there are inter-dependencies between actions of different agents that arise from the sharing of a common environment (Faratin & Jennings 1998). The interactions can be of different types such as cooperation (agents work together to achieve a common objective), coordination (planning interdependent activities to be performed in a coherent manner), and negotiation (a process by which a group of agents reach an agreement on some decision that is mutually acceptable) (Jennings, P. Faratin, Parsons, Sierra & Wooldridge 2001).

Multiagent systems is a approach for cooperative distributed problem solving which can be defined as (Durfee, Lesser & Corkill 1989*a*), (Durfee et al. 1989*b*): *cooperative distributed problem solving studies how a loosely coupled network of problem solvers can work together to solve problems that are beyond their*

individual capabilities. Each problem-solving node in the network is capable of sophisticated problem solving and can work independently, but the problems faced by the nodes cannot be completed without cooperation. Cooperation is necessary because no single node has sufficient expertise, resources, and information to solve a problem, and different nodes might have expertise for solving different parts of the problem.

Achieving a predictable behavior of a large distributed system is a difficult task. Current research such as Xu, Scerri, Yu, Okamoto, Lewis & Sycara (2005), Shen, Zhang & Lesser (2004) focuses breaking the system into smaller manageable local models to generate desirable global properties. By enforcing desired properties on the smaller sub-networks of a large network system, the idea of this approach is to propagate the properties throughput the network. Although this bottom-up approach has potential, it is difficult to achieve a globally coherent behavior from a population of autonomous distributed entities which have only limited knowledge. Due to the individual component's incomplete knowledge about the system, large groups of such entities are liable to experience unexpected emergent behaviour (Scerri, Liao, Lai, Sycara, Xu & Lewis 2004). As a result, systems made up of purely local and small-scale models cannot guarantee a desired behavior at a global level. According to Nwana, Lee & Jennings (1996): *coordination is a process in which agents engage in order to insure their community acts in a coherent manner. Coherent means that the agents' actions gel well, and that they do not conflict with one another.* Additionally, the reasons that might necessitate coordination in multiagent system are summarized as follows:

- **Preventing anarchy or chaos**: co-ordination is necessary or desirable because, with the decentralization in agent-based systems, anarchy can set in easily. No longer does any agent possess a global view of the entire agency to which it belongs. This is simply not feasible in any community of reasonable complexity.

- **Meeting global constraints**: there usually exist global constraints which a group of agents must satisfy if they are to be deemed successful. Agents need to co-ordinate their behavior if they are to meet such global constraints.

- **Distributed expertise, resources, or information**: In multiagent systems, the problem and solution is distributed across multiple agents. Individual agents do not have sufficient competence, resources, or information to solve the problem independently. Resources or expertise distributed across agents may need to be combined to solve a problem.

- **Dependencies between agents actions**: Because of being present in a common environment, the goals and actions of different agents are interdependent. The actions taken by one agent can change some information or part of environment based on which some other agents make their decision.

- **Efficiency**: Even when individuals can function independently, thereby obviating the need for coordination, information discovered by one agent can be of utilized solve the problem in a better way.

As another definition, according to Jennings (1996): *coordination is the process by which an agent reasons about its local actions and the anticipated actions of others to try and ensure the community acts in a coherent manner*. In order enable coherent decision making of distributed agents, variety of coordination mechanisms have been developed.

As it is not feasible/desirable to have centralized information, the partial knowledge of distributed agents can lead to conflicting situations among different agents. Negotiation has been uses extensively used an coordination mechanism to evolve and deduce coherent decisions among interdependent agents. According to Bussmann & Müller (1993): *negotiation is the communication process of a group of agents in order to reach a mutual accepted agreement on some matter*. According to a more generic framework proposed by Jennings et al. (2001), negotiation is considered as a distributed search through a space of potential agreements. During the process of negotiation, the direction of the search is determined by the entities involved in negotiation. Negotiation is viewed as an iterative process in which the negotiating agents repeatedly make and respond to proposals. During successive iterations, the agents need to be able to provide more useful feedback on the proposals it receives. This feedback can either be an evaluation of the proposal, identifying parts of the proposal as undesirable/infeasible or a alternative counter proposal. For the search process

to converge and terminate, the feedback provided by agents should assist the proposing agent to generate more acceptable proposals. Since negotiation involves interaction, the protocol for dialogue between involved need to be defined and various protocols for negotiation have been studied which are described below.

In order to coordinate the activities of various agents, by intelligently deciding which agents perform which task can control the performance of system over time. Dynamic task allocation can take the current system status into consideration and adapt the system's behavior according to the performance requirements and capability of agents. Contract net protocol (Smith 1980) is a very popular technique for distributed task allocation in multiagent systems (Shen, Norrie, D. & Barthes 2001), (Wooldridge 2002). The algorithm is based on the way that companies float the tenders in market to contract some prescribed work to potential bidders. Using the contract net protocol, the tasks are assigned using a contracting mechanism with an objective of assigning the tasks to the agents best suitable for the job. The agent can have two roles in the protocol, manager or contractor. The manager agent advertises the task to be performed by a making an announcement to some other (contractor) agents in the system. On receiving the task announcement, contracting agents evaluate the task with respect to their capabilities and availability and reply the manager with a bid. A bid contains the information about the various parameters of the service that the contractor can provide. The submitted bids are evaluated by the manager agent which selects the most suitable bidder to execute the task and awards the contract to the selected bidder. The contractor then executes the prescribed task and sends a report to manager after the task is completed. Besides contract-net other mechanisms based on markets such as auctions (Hunsberger & Grosz 2000) have also been used to execute negotiation between agents.

Besides market based mechanisms, game-theory based negotiations have also been utilized for coordination (Kraus 1997), (Rosenschein & Zlotkin 1994), (Sandholm 1999). Another kinds of algorithms that are inspired by "swarm intelligence" are also utilized for coordination of multiagent systems. Swarm Intelligence is the property of a system whereby the collective behaviors of (unsophisticated) agents interacting locally with their environment cause coherent

functional global patterns to emerge (Bonabeau, Dorigo & Theraulaz 1999), (Cicirello & Smith 2001), (Tolksdorf & Menezes 2003).

Recent research shows that *token-based* algorithms can address the problems of other coordination mechanisms and provide efficient and scalable coordination for large agent networks in various domains (Wagner, Guralnik & Phelps 2003, Xu et al. 2005, Moyaux, Chaib-draa & D'Amours 2003, Scerri, Farinelli, Okamoto & Tambe 2004). For example, token-based coordination was used to reduce *bullwhip effect*, which is amplification of order variability induced by leadtimes for handling demand fluctuations, in a supply chain network problem (Moyaux et al. 2003). In addition to a token which conveys usual demand information as a request from a market, agents in the supply chain use another type of token for sharing information about their own inventory request to meet the demand fluctuations in a timely manner. Thus, each agent can distinguish between original demands from the market and those of intermediate agents and can avoid reacting blindly to temporal demands. Although a fluctuation of demands has a big influence on performance of the network, it can be reduced outside the network in cooperation with other networks.

2.2.2 Multiagent Control for Manufacturing

Due to the ever increasing global competitiveness, there have been fundamental changes in the way manufacturing systems operate. Today's manufacturing systems have to consistently minimize the product cycle times, maximize productivity, maximize operational flexibility and these goals have to met in a dynamic environment which is characterized by unexpected failures and changes in customer demand. In order to face this challenge, the current trend has been towards deploying automated systems that can offer robustness, stability, adaptability, and efficient use of available resources through a modular and distributed design (Parunak 1993, Parunak 1995, Parunak 1998*b*, Parunak 1998*a*, Tharumarajah & Bemelman 1997, Brennan & Norrie 1998, Liu & Zhang 1998, Shen & Norrie 1999, Maturana & Norrie 1997, Shen et al. 2001). Due to the limitations of centralised hierarchical architectures in handling the challenges posed by current manufacturing systems, the applicability of decentralized control architecture has been explored. By decentralizing the control of manufacturing

system, the idea is to reduce the complexity, increase flexibility, and enhance fault tolerance. Because of their flexibility and robustness against disruptions, it is believed that multiagent systems are a promising approach to build robust and flexible control systems for large, dynamic and complex production systems.

The studies have highlighted that the multiagent based control is better suited for dynamic manufacturing systems. The basic philosophy that differentiates multiagent architectures from hierarchical and centralized manufacturing control systems is that the multiagent systems recognize and correspond to the fact that data and control are distributed through the system. Additionally, this agentification implies natural flexibility as it is possible to add or remove new entities with their attached agents to or from the system with minimal changes in the existing software. The autonomous agents could correspond to different entities in the system such as resources, operators, parts, jobs, orders, etc. This local autonomy of control and data allows the agents to constantly act according to the dynamic state of the relevant (partial) environment. Because of the interaction ability of agents, they can coordinate with other agents to make globally coherent decision. There is no global planning, and the overall system performance emerges through the dynamic interaction of the agents during execution. As the software for each agent in a multiagent control is less complex and simpler than the centralized counterpart, the writing, installation, and maintenance of control software becomes easier. Some surveys on agent-based manufacturing control are present in Ouelhadj (2003), Monostori, Vancza & Kumara (2006), Caridi & Cavalieri (2004), Babiceanu & Chen (2006).

The multiagent architectures used for manufacturing vary in the concentration of control. Centralized/hierarchical mechanisms are complex and due to their rigidity of control structure their applicability is limited in real-time, dynamically changing circumstances. The heterarchical and autonomous architectures on the other hand provide advantage of modularity, reconfigurability, adaptability, fault tolerance and extensibility (Hatvany 1985, Duffie 1990, Vamos 1983, Prabhu & Duffie 1995, Prabhu & Duffie 1996). Nevertheless, due to absence of global information at a single point in such distributed architectures, achieving globally optimal behavior at large scale is difficult and local decisions based on partial information can lead to a undesirable emergent behavior and unstable system (Brennan & Norrie 1998, Shen & Norrie 1999, Bongaerts,

Brussel, Valckenaers & Peeters 1997, Shen et al. 2001, Tharumarajah 2001). Yams (Yet Another Manufacturing System) (Parunak 1987) is one of the earliest application of autonomous architecture in which entities like factory, cell, workstation, machine, parts are modelled as agents. In Yams, for assignment of tasks to resources, task agents use contract net protocol to negotiate with resource agents. Another application of autonomous architecture to cellular manufacturing system treats cells as agent (Shaw 1988). The tasks of orders in their approach are sub-contracted by cell agents to other cells through a bidding mechanism. On receiving the broadcasted bid messages for a given task, the cell agents evaluate the operation and reply with the details of service they can provide in terms of processing time. Based on the bids of different cells, the task is assigned to the cell which optimizes some predefined performance criteria.

In order to achieve robustness against disturbances along with performance optimisation and predictability, several researchers have proposed mediator architectures that combine features of both hierarchical and heterarchical frameworks. In mediator architecture, the basic structure of autonomous local agents is extended with special mediator agents (Brennan & Norrie 1998, Gou, Luh & Kyoya 1998, Shen & Norrie 1999, Bongaerts, Monostori, McFarlane & Kádár 2000, Shen et al. 2001). Compared to local agents, the mediator agents have a larger view of system. Mediator agents work with local agents and coordinate the behavior of multiple agents resulting in a better informed decision making process. Various multiagent control mechanisms are based on mediator architecture, which have different number and roles of mediators (Maturana & Norrie 1997, Maturana, Shen & Norrie 1999, Rabelo, Camarinha-Matos & Afsarmanesh 1998, Bongaerts et al. 1997, Xue, Sun & Norrie 2001, Sun & Xue 2001, Ramos 1994). Mediator agents assist in making globally coherent decisions and Brennan & Norrie (1998), Brennan & Norrie (2001), Bongaerts et al. (2000) and Cavalieri, Garetti, Macchi & Taisch (2000) in their comparative studies have shown that mediator architectures can simultaneously provide stability and satisfactory performance due to enhanced vision of mediators, which enables better planning and coordination in combination with system's ability to react to disturbances.

Market based approaches: Because of various constraints, conflict resolution has been a critical issue for dynamic manufacturing control. Market-

oriented approaches to achieve have been widely used in multiagent control. They have been used for controlling execution by deciding the dispatching at workstations (Maley 1988, Shaw 1988). Market mechanisms have been used for scheduling, where announce-bid-award cycles are applied recursively through different resources in order to plan and schedule the required activities for manufacturing (Iwata, Onosato & Koike 1994, Tseng, Lei & Su 1997, Marcus, Vancza & Monostori 1996). For negotiation, contract-net protocol has been widely used with different variations.

Holonic Manufacturing Systems: A new paradigm of manufacturing control, holonic systems consist of holons or agents that are autonomous, intelligent, flexible, distributed, cooperative (Valckenaers, Bonneville, Van Brussel, Bongaerts & Wyns 1994). Holonic is a flexible approach and can be used to utilize features of both hierarchical and heterarchical systems (Bongaerts et al. 2000). Holons can be organized into flexible hierarchies, which can be both temporal and permanent. Various architectures have been based on the organization and roles of holons in the system. The PROSA reference architecture (Brussel, Wyns, Valckenaers, Bongaerts & Peeters 1998) has three basic kind of holons i.e. resource holons, product holons and order holons, and the architecture defines the information flows in the system. Another system that is based on PROSA architecture, is HoMuCS (agile holonic multi-cell control system) (Langer & Alting 2001). In another similar architecture, ADACOR (Paulo Leit & Restivo 2006), a special supervisor holon acts as a mediator to manage dynamic group formation and coordination of other holons to achieve global optimization. Besides these sophisticated architectures, various simple holonic architectures with fewer kind of holons have also been proposed such as Wiendahl & Ahrens (1997) utilizes order and machine agents, Tseng et al. (1997) uses job and resource agents, and Kadar, Monostori & Szelke (1998) refers to order and machine holons.

2.3 Conclusion

As reviewed in this chapter, various techniques have been developed and utilized in the past to control manufacturing process. The various approaches can be identified as dispatching rules and simulation based techniques, artificial in-

telligence based techniques such as heuristics, meta-heuristics, knowledge-based systems and a relatively new technique of multi-agent systems. The comparative evaluation of various techniques has been done in past by Suresh & Chaudhuri (1993), Shukla & Frank Chen (1996), Stoop & Wiers (1996) and MacCarthy (2001).

Dispatching rules control the system in a distributed manner by sequencing the jobs at each workstation and are easy to implement. The performance of various dispatching rules can vary according to the condition of system. Simulations are used to access the performance of dispatching rule before applying it for execution. As the simulations done in advance utilize the forecasted conditions, the success of dispatching rules depends on accuracy of system information. In dynamic systems, the unexpected events can change the system state to a large extent and all possible scenarios cannot be analyzed due to the computational overheads involved. As a result, such approaches provide an approximated solution and are poor in responding to the real-time events. Dispatching rules represent a local and on-line control mechanism in which no centralized plans are maintained. The other approaches use more global information and search in a large solution space to generate and maintain schedules. As the schedules frequently become unusable due to unexpected disruptions during execution, various approaches have been used to maintain consistency of schedules in dynamic manufacturing environments. Complete rescheduling i.e. regenerating the schedule from scratch in a event of disruption although can provide optimal schedules, it is not feasible due to complexity and computational requirements for large and dynamic system. Schedule repair is a alternative approach which tries to maintain the consistency of schedule by modifying only the part of overall schedule that is affected by disruptions. In order to search for alternative schedules during schedule repair, various local-heuristics have been developed. The searches using local-heuristics may get stuck in a poor local optima and various meta-heuristics have been applied to overcome this limitation. Various meta-heuristics such as tabu search, simulated annealing and genetic algorithms have been utilized for generation of better alternative schedules during the repair. Knowledge-based systems have also been used to make use of acquired expertise and knowledge. By capturing the knowledge about the system and similar scenarios in the past, the knowledge based systems can intelligently sug-

gest actions which can be taken to repair schedules, thus making the schedule repair process more efficient.

Most of manufacturing control systems that are developed previously have centralized and hierarchical control structure. Making use of consistent global view of complete system can enable globally optimal decision making and better control. However, due to centralization, such architectures have high rigidity and they tend to have problems with reactivity to disturbances. In contrast with the centralized control architectures, in multiagent control systems the decision making structure corresponds more closely to the information and decision execution structure. By decentralizing the control of manufacturing control, multiagent system aim to reduce the complexity, increase flexibility, and enhance fault tolerance. Comparative studies have shown that multiagent systems are the most suitable approach for controlling large and dynamic manufacturing systems. The agents in a multiagent system have limited view of system and can make decisions only in a part of system. Due to lack of a centralized information an control structure, the agents rely on interaction with other agents to coordinate their actions. Coordination is vital for decentralized multiagent systems in order to prevent chaos and enable individual agents to make decisions that are globally coherent.

Because of the importance of autonomous coordination, various coordination mechanism based on market based mechanisms, game-theory principles and swarm based interactions have been proposed in the generic multiagent domain. However the sophisticated algorithms which show good performance do not scale to large systems, either due to the extensive messaging that the coordination mechanism requires or due to the computational complexity of locally centralized mediators or auctioneers. Although the simpler, swarm based approaches can provide large-scale coordination, the resulting overall system performance can be very inefficient. In the domain of manufacturing control also, various coordination mechanism ranging from market based to swarm based have been applied. Although the agent-based approach allows for improved, flexible and robust control of dynamic production system, due to lack of efficient and scalable coordination mechanisms, achieving good performance for large and complex manufacturing systems such as semiconductor fabrication remains an issue.

In subsequent chapters the details of multiagent system that we have developed for semiconductor manufacturing is presented.

Chapter 3

Proposed Multiagent Control Mechanism

As reviewed in Chapter 2, centralized control and maintaining global plans although provide optimization but are not feasible in large dynamic manufacturing systems. As an alternative to generating optimal schedules which is an intractable problem, various simplifications and heuristics have been developed and practiced.

The behavior of manufacturing system is dominated by bottleneck resources and their starvation results in irrecoverable loss of throughput (Goldratt & Cox 1992). Due to failures on various resources, the flow of jobs gets disrupted and the resulting drop in utilization of bottlenecks due lack of jobs is hard to recover. Researchers in the past have also highlighted the importance of avoiding starvation of bottlenecks and have proposed mechanisms to ensure the availability of jobs at the bottlenecks. The past studies have assumed that the bottlenecks are fixed and can be determined by preliminary analysis of the system (Glassey & Resende 1988, Glassey & Petrakian 1989, Glassey & Weng 1991, Lozinski & Glassey 1988, Wein 1988, Wein 1990, Wein 1991, Wein 1992, Wein & Chevalier 1992). To avoid starvation of identified bottlenecks, release rules and dispatching heuristics were used in conjunction with simulation to identify adequte safety buffer levels to be maintained during execution. However, due to the capacity loss of resources resulting from random failures, the bottlenecks shift and numerous temporary and secondary bottlenecks can emerge during

subsequent execution. Few researchers have developed methods that utilize the evolving state of system to identify dynamic and shifting bottlenecks (Roser et al. 2001, Roser et al. 2002, Roser et al. 2003, Faget et al. 2005). However, in all the previous studies mathematical or simulation analysis of system is done offline to identify the system bottlenecks. After identification of anticipated bottlenecks, measures to avoid their starvation are calculated (such as level of safety buffers) and applied during subsequent execution. The effectiveness of existing techniques is limited by the frequency of analysis exercise as the conditions in a large and failure-prone system change rapidly and bottlenecks may change during successive planning cycles. In this context, the multiagent coordination mechanism proposed in this thesis has following key features:

- **Autonomous control**: In line with motivations of multiagent technology, the execution of system takes place autonomously without need of any intervention for planning or analysis.
- **Distributed and dynamic bottleneck identification**: The distributed agents monitor performance of resources and dynamically identify the bottlenecks during execution.
- **Integrated planning and execution**: The control in our system is distributed across the agents. Agents autonomously monitor the changes in system and decide on their actions by coordinating with other agents. Through messages within the system, remote agents are informed about location and requirements of emerging bottlenecks and agents change their actions dynamically to avoid the starvation of bottlenecks.

Multiagent systems are characterized by distributed decision making entities which coordinate to achieve desired global behavior. Individual agents do not have knowledge or capabilities to achieve the global system goal. Due to lack of centralized control, the success of a multiagent system depends on effectively calibrating the sub-goals of individual agents such that the distributed agents' autonomous behavior of achieving their local goals results in the desired global behavior. For a multiagent system to be able to respond to unexpected disruptions and changes, the sub-goals of agents should also change according to the modified circumstances. As the multiagent systems are required be

autonomous, the system should autonomously monitor the changes within the system and update the sub-goals of agents accordingly. Due to dependencies between agents' actions, the desired system behavior can be achieved by coordinated actions of multiple agents and the coordination mechanism defines the required protocol of interaction and information flow among different agents in a multiagent system. The existing multiagent coordination mechanisms that show coherent behavior have been shown to work only for small problems and because of their complexity they do not scale to large problems.

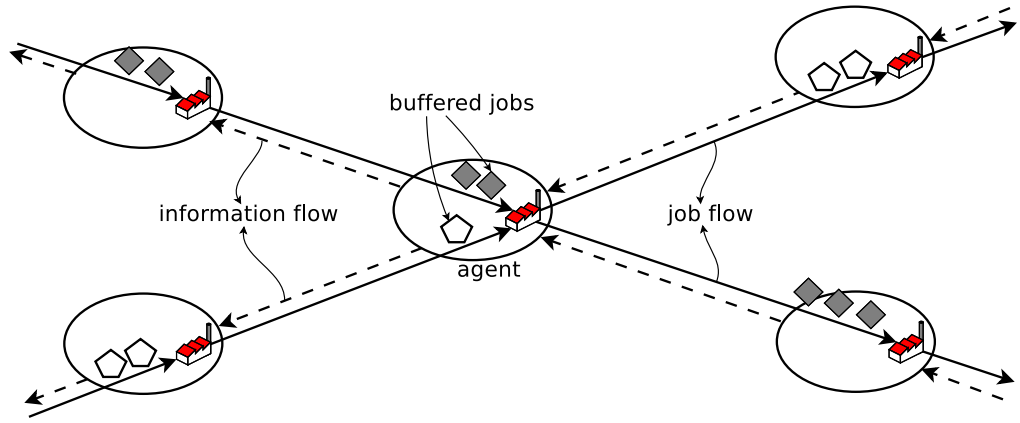

Figure 3.1: Agent interactions in CABS

In this chapter we describe our coordination method: Coordination for Avoiding Bottleneck Starvation (CABS) for improving a trade-off between lead-time and 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 for maintaining the desired flow of jobs to the bottleneck agents. In CABS, corresponding to each kind of job, there is a corresponding flow of information (through messages) in reverse direction as shown in Figure 3.1.

3.1 Agent Actions in CABS

In CABS, the manufacturing problem is modeled by means of a network of agents and the problem assumes a set of jobs $J = \{J_1, ..., J_n\}$ to be processed by a set of agents $A = \{A_1, ..., A_m\}$.

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 consists of a specific number of works to be processed and flows through agents according to its process route. Each agent A_j processes 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 to be completed per unit time. Furthermore, when an agent A_j processes its task T^j_i, it takes a fixed 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_i is a step S^l_i. A preceding agent of the agent A_j in terms of the task T^j_i, $A_{pre(j,i)}$, is in charge of a step S^l_{i-1} and a succeeding agent, $A_{suc(j,i)}$, processes a step S^l_{i+1}.

In addition to the agents that do the actual processing of jobs, *source* agents also exist for each job. A source agent of a job is at the beginning of its process route, and is responsible for releasing new lots of the job in the system by transferring them to the agent processing the first step of that job.

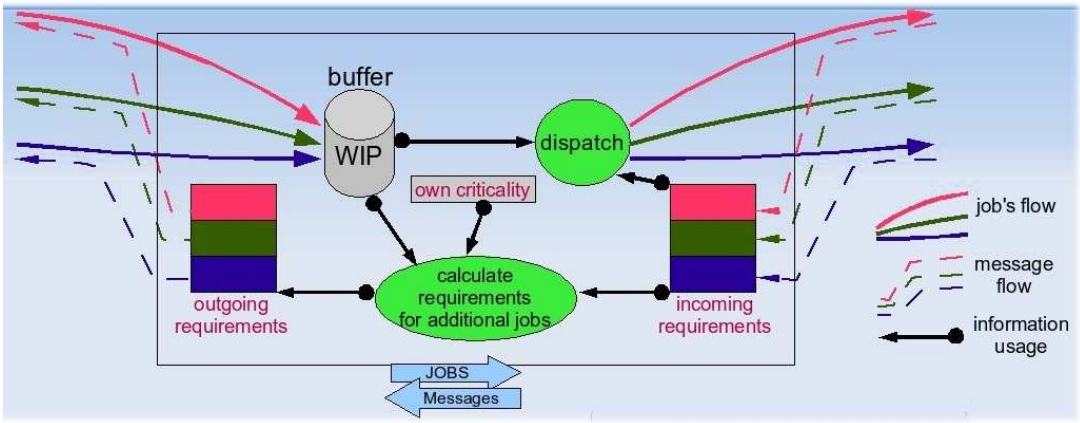

Figure 3.2: Agent's internal details in CABS

The internal details of individual agent in CABS are shown in Figure 3.2.

An agent might be processing multiple type of jobs and it periodically receives requirements (messages) for each kind of job from corresponding succeeding agent in the process flow. The sample agent in Figure 3.2 is processing three type of jobs (shown by three different colors). Agents maintain the latest incoming requirements for each kind of job. The actions of different agents are coordinated by means of requirement messages and each agent (A_j) in CABS performs two actions:

- **Dispatching**: Based on the incoming requirements of its different kind of jobs, T^j, agent A_j selects the next lot to process from its locally buffered jobs (WIP). After processing the selected job of task T_i^j, the processed job is sent to the corresponding succeeding agent $A_{suc(j,i)}$ in the process route.

- **Generating Messages**: Based on the incoming requirements from succeeding agents and the jobs that are already present (locally buffered WIP), agent calculates the requirements for additional jobs. Agent generates requirements for additional jobs for each kind of job that it processes i.e. T^j, and sends these outgoing requirements (messages) to the corresponding preceding agent in the process route i.e. $A_{pre(j,i)}$ for task T_i^j. (The details of "own criticality", which agent uses for calculating outgoing requirements for additional jobs is provided later in the chapter).

In following sections, we describe the basic mechanism of an agent in CABS and then introduce CABS's coordination method developed for mitigating effects of failures and maintaining throughput of the network.

3.1.1 Lot Dispatching

Each agent A_j periodically receives requirements of processing its task T_i^j from the corresponding succeeding agent $A_{suc(j,i)}$. The requirement consists of the following four types of information (detailed definitions will be given later in Section 3.1.2):

`time limit:` time at which agent $A_{suc(j,i)}$ can process another lot for the next step of the task T_i^j.

`request rate`: rate at which agent $A_{suc(j,i)}$ can process the lots for the next step of the task T_i^j.

`amount`: the number of lots that $A_{suc(j,i)}$ requires urgently at `request rate` starting at `time limit`.

`criticality`: criticality of the requirement coming from agent $A_{suc(j,i)}$.

In addition to the requirement information from succeeding agents, for each task $T_i^j \in T^j$, an agent A_j is assumed to have local information such as the demand rate, its current WIP with their due dates, and the total number of lots it has 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, or *due date* of lots if tasks' `time limit` is identical
4: set FET_j as the first task in ET^j
5: **loop**
6: set start time of FET_j at current time
7: $OFT^j \leftarrow \{ET_i^j \mid (ET_i^j \in ET^j) \wedge (ET_i^j \; overlaps \; FET^j) \wedge (criticality(ET_i^j) > criticality(FET_j))\}$ // tasks *overlap* when an intersection exists in their processing periods
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 it 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 thesis, $im[\,].tl$, $im[\,].rr$, $im[\,].am$ and $im[\,].cr$ respectively denote requirement information of

`time limit`, `request rate`, `amount` and `criticality` for the corresponding tasks in the incoming messages of the agent.

3.1.2 Message Passing

As described in Algorithm 1, dispatching of jobs 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_i^j is the time required to process it (i.e., pt_i^j). Each agent has aggregated workloads of all of its tasks based on the demand rates of jobs. The difference between the planned workloads and the total realized workloads due to tasks that have already been processed by an agent denote the current workload deficit of that 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: $t\_ft_j \leftarrow current\_time + \sum_{\forall i \in \{1,\cdots,t_j\}} (t\_w_i^j pt_i^j)$ // 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 workload deficit of A_j
6: $sc_j \leftarrow (1.0 - \sum_{\forall i \in \{1,\cdots,t_j\}} dr_{job(T_i^j)} pt_i^j)$ // surplus capacity of A_j
7: **return** t_wld_j / sc_j

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 the 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 3 **adjRequirements**(task# i, message req) of agent A_j

1: $t\_w_i^j \leftarrow$ amount of current WIP of task T_i^j
2: $t\_tl \leftarrow req.tl + t\_w_i^j / req.rr - pt_i^j$ // time to replenish T_i^j based on request from $A_{suc(j,i)}$
3: $req'.tl \leftarrow max(t\_tl, (current\_time + t\_w_i^j pt_i^j))$ // time at which T_i^j can process new job for $A_{suc(j,i)}$
4: **if** $((req.am == 0)$ `or` $(t\_w_i^j >= req.am))$ **then** // no pending urgent requirement of $A_{suc(j,i)}$
5: $t\_pr_i^j \leftarrow$ total production of task T_i^j until $current\_time$ by this agent (A_j)
6: $req'.tl \leftarrow max(req'.tl, (t\_pr_i^j + t\_w_i^j)/dr_{job(T_i^j)})$ // i.e. further jobs according to demand
7: $req'.am \leftarrow 0$ // i.e. no urgent requirement of job
8: $req'.cr \leftarrow 0.0$
9: **else** // $A_{suc(j,i)}$'s remaining requirements are propagated to A_j's preceding agent after adjustment
10: $req'.am \leftarrow req.am - t\_w_i^j$ // i.e. remaining lots of urgent requirement
11: $req'.cr \leftarrow req.cr$
12: **end if**
13: $req'.rr \leftarrow min(req.rr, 1/pt_i^j)$
14: **return** req'

Algorithm 4 **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: **for all** $i \in \{1, \cdots, t_j\}$ **do**
3: $t\_ft_j \leftarrow current\_time + \sum_{\forall i \in \{1, \cdots, t_j\}} (t\_w_i^j pt_i^j)$ // A_j's earliest time to get starved
4: $t\_cr_j \leftarrow$ `calcCriticality`() // current criticality of A_j
5: $t\_wld_j \leftarrow$ current workload deficit of A_j // as calculated in algorithm *calcCriticality*
6: $im'[i] \leftarrow$ `adjRequirements`$(i, im[i])$ // adjust $A_{suc(j,i)}$'s requirements for propagatation
7: **if** $(t\_cr_j > im'[i].cr)$ `and` $(im'[i].tl > t\_ft_j)$ **then** // A_j is critical `and` processing jobs according to $A_{suc(j,i)}$'s requirements will cause starvation of A_j
8: $om[i].tl \leftarrow t\_ft_j$ // A_j's requirement to avoid its starvation
9: $om[i].rr \leftarrow 1/pt_i^j$ // maximum possible processing (threshold) rate of agent A_j
10: $om[i].am \leftarrow t\_wld_j/pt_i^j$ // amount urgently required by A_j to recover its workload deficit
11: $om[i].cr \leftarrow t\_cr_j$ // A_j's criticality
12: **else**
13: $om[i] \leftarrow im'[i]$ // propagate $A_{suc(j,i)}$'s (adjusted) requirements
14: **end if**
15: **if** (A_j is in failure) **then** // A_j cannot process any jobs
16: $om[i].tl \leftarrow \infty$
17: $om[i].rr \leftarrow 0.0$
18: $om[i].am \leftarrow 0$
19: $om[i].cr \leftarrow 0.0$
20: **end if**
21: **end for**
22: **return** $om[\,]$

To maintain the flow of lots of a task T_i^j to $A_{suc(j,i)}$ at the requested rate $im[i].rr$, A_j requires an incoming lot flow at the same rate from the corresponding preceding agent $A_{pre(j,i)}$. However, the agent itself might be critical and need some jobs earlier and at a higher rate in order to recover its workload deficit. The agent requires a number of jobs immediately and at the maximum rate at which it can process to recover its deficit as soon as possible. Based on the requirement from succeeding agents and its own criticality, the agent generates a consolidated outgoing requirement for its preceding agent. Algorithm 4 describes the calculation of outgoing requirement messages by agent A_j. For each $T_i^j \in T^j$, a requirement tuple om $(om[i].tl, om[i].rr, om[i].am, om[i].cr)$ is generated and sent to the preceding agent $A_{pre(j,i)}$. Algorithm 4 invokes Algorithm 3 to adjust the incoming requirements from $A_{suc(j,i)}$ for propagation to $A_{pre(j,i)}$. $A_{suc(j,i)}$'s adjusted requirements are stored in intermediate im' (Algorithm 4:6), and A_j then uses them and its own current criticality to calculate the final outgoing parameters (om) to be sent to $A_{pre(j,i)}$.

In Algorithm 3, in order to meet the incoming job requirements of $A_{suc(j,i)}$, A_j adjusts the requirements of $A_{suc(j,i)}$ before propagating them to its preceding agent $A_{pre(j,i)}$. The `time limit` and `request rate` from $A_{suc(j,i)}$ indicate the time and rate at which $A_{suc(j,i)}$ can process the new arriving lots of T_i^j. Based on this capacity of $A_{suc(j,i)}$ and its own available WIP, A_j calculates the time at which is should request the next lot from $A_{pre(j,i)}$ (Algorithm 3:2-3). A_j uses the `amount` parameter of $A_{suc(j,i)}$'s requirements and availability of its local WIP to adjust $A_{suc(j,i)}$'s requirements. The `amount` value 0 from $A_{suc(j,i)}$ (Algorithm 3:4) indicates that $A_{suc(j,i)}$ has no special requirement and it needs the lots according to their normal demand. In such a case A_j requests the next lot from $A_{pre(j,i)}$ according to its demand, subjected to its own and $A_{suc(j,i)}$'s availability (Algorithm 3:6). The `amount` and `criticality` parameters are set to zero to inform $A_{pre(j,i)}$ that lots are required according to their demand (Algorithm 3:7-8). The `amount` value other than 0 from $A_{suc(j,i)}$ indicates that $A_{suc(j,i)}$ has urgent requirement of `amount` number of jobs which should be supplied to it at the corresponding `request rate`. If A_j has sufficient WIP to meet the urgent requirement of $A_{suc(j,i)}$ (Algorithm 3:4), A_j resets the propagated requirements to $A_{pre(j,i)}$ in order to ask subsequent lots according to their demand (Algorithm 3:5-8). When A_j does not have sufficient WIP to meet the urgent

requirement of $A_{suc(j,i)}$ by itself, A_j propagates the remaining requirement to $A_{pre(j,i)}$ (Algorithm 3:10-11). In any case A_j cannot process the lots at rate higher than its own capacity and it regulates the `request rate` accordingly before propagation (Algorithm 3:13).

As Algorithm 4 shows, when the agent makes request messages for its preceding agents, the agent A_j uses `criticality` of incoming requirements to identify the location of current bottlenecks in the system. If A_j's own criticality is higher than the incoming `criticality` from $A_{suc(j,i)}$, it means that A_j is more likely to be a bottleneck in the system (Algorithm 4:7). In such a case, if the processing of jobs as per $A_{suc(j,i)}$'s requirements causes starvation of A_j, A_j prioritizes recovering its own workload deficit over satisfying $A_{suc(j,i)}$'s requirement (Algorithm 4:8-11). In order to recover its own deficit at the earliest, A_j sends the time when all its own WIP is used up as `time limit` (i.e., t_ft_j) 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 its workload deficit caused by delayed jobs. The agent passes its own criticality (t_cr_j) as the outgoing `criticality`. This enables the preceding agents to identify the location of current bottlenecks in the system along the process routes.

If `criticality` of $A_{suc(j,i)}$ is higher than that of A_j, it means that $A_{suc(j,i)}$, or some other succeeding agent is a bottleneck. In this case, A_j acts to recover the deficit of $A_{suc(j,i)}$ and generates the outgoing requirements based on the incoming requirements from $A_{suc(j,i)}$ (Algorithm 4:13). As A_j propagates all the requirement parameters of $A_{suc(j,i)}$ to its preceding agent, the preceding agent also gets the information about the location and requirements of critical agent and can hence decide on its actions accordingly.

And, when an agent is in failure, it cannot process any job. Therefore, the agent during the failure period stops requesting jobs from its preceding agents by sending the requirements accordingly i.e., setting `time limit` as $infinity$, `amount` as 0 and `request rate` and `criticality` as 0 (Algorithm 4:16-19).

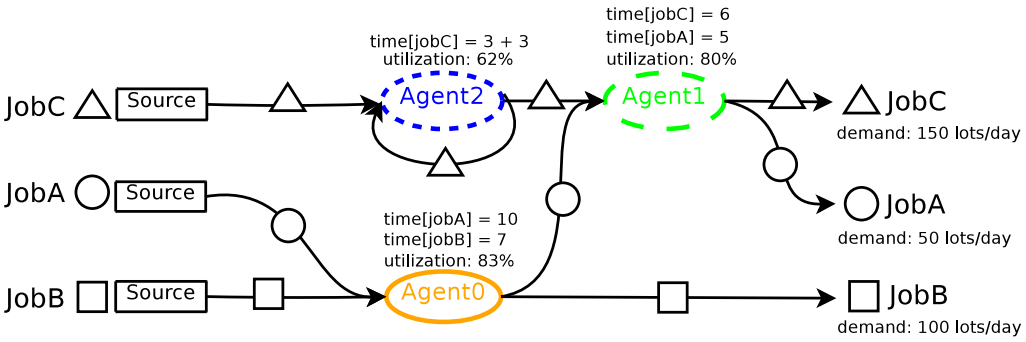

Figure 3.3: Example agent network

3.2 Coordinated Behaviors in CABS

In this section, we show how coordination is achieved in CABS by analyzing its behaviors using a simple example. The example network is shown in Figure 3.3 and it processes 3 types of jobs (*JobA*, *JobB* and *JobC*). The network consists of 6 agents, of which 3 agents i.e. *Agent0*, *Agent1* and *Agent2* are processing agents. Besides these 3 processing agents which do the actual processing on the jobs, there also exist synthetic *Source* agents corresponding to each type of the job. Process routes of the jobs are shown by directed arrows from their corresponding *Source* agents. The demand of completed jobs is shown in terms of lots/day at each process route. In the figure, label *time[job_type]* with each processing agent indicates the time of the labeled agent to process a lot of the*job_type*. *Agent2* is processing 2 steps of *JobC*, one of which is a loop step shown as a cycle in the figure, and each step takes 3 mins. Based on the demand rates and processing times of various jobs, the aggregate utilization of agents is also shown with each agent (in percentage).

For the experiment we simulate a single failure on *Agent2* from time 15000 to 28000 min. First we explain the behavior of a conventional simple manufacturing system during this failure. Then we explain the working of CABS's algorithms during an identical failure and compare its performance with the conventional system.

3.2.1 Behaviors of Conventional System

The conventional manufacturing system that we have chosen for comparison has following properties:

- The jobs are released in the system at the demand rate i.e. the jobs are pushed constantly in system without any consideration to the system status.
- Among the different types of available jobs (WIP), the agents pick the jobs to process in the order of their due dates, i.e. the agents dispatch their jobs according to the earliest due date (EDD) policy.

The above releasing and dispatching policies are simple but widely used in small manufacturing environments because behaviors of the system are predictable to human operators.

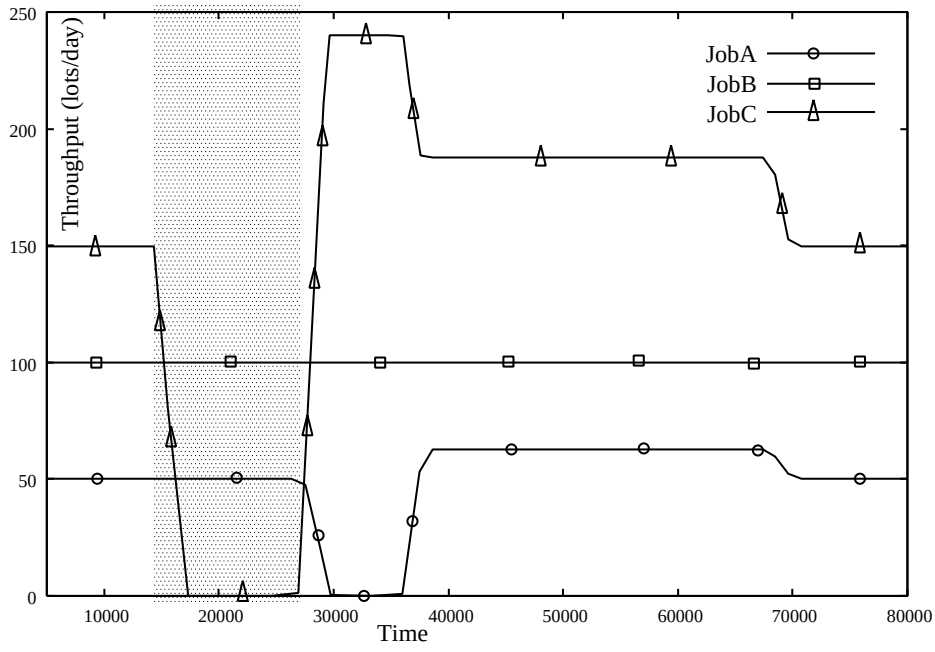

Figure 3.4: Conventional System: Throughput

The behaviors of this system in terms of throughput and finished inventory of different jobs are shown in Figure 3.4 and Figure 3.5 respectively. The duration

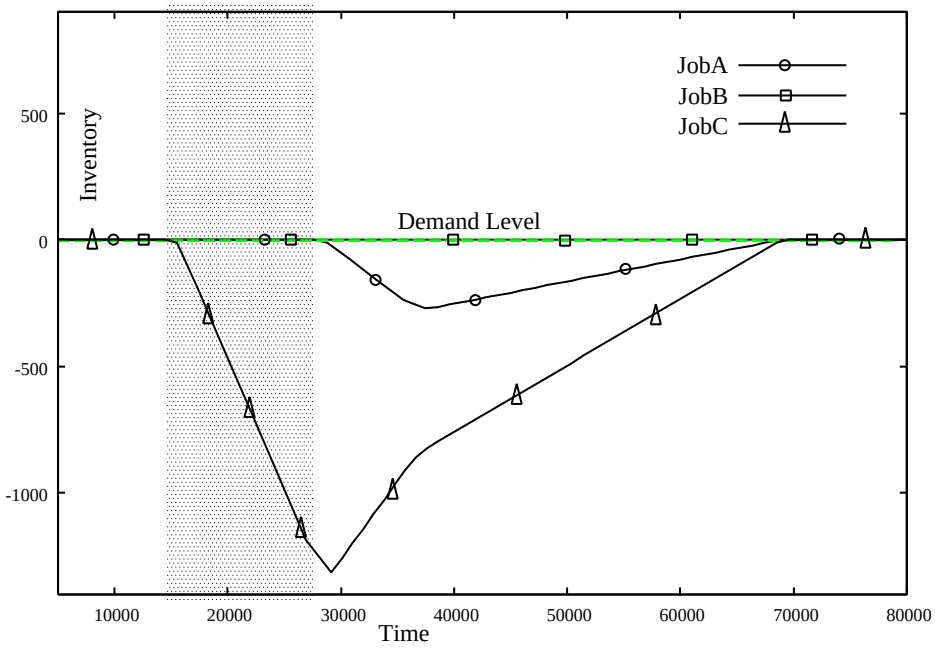

Figure 3.5: Conventional System: Finished Product Inventory

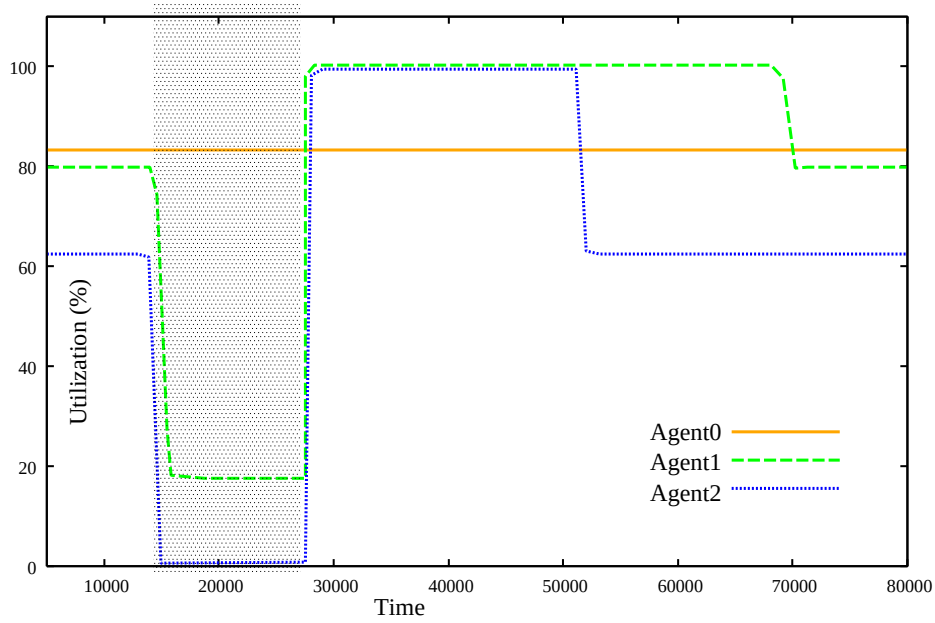

Figure 3.6: Conventional System: Utilization

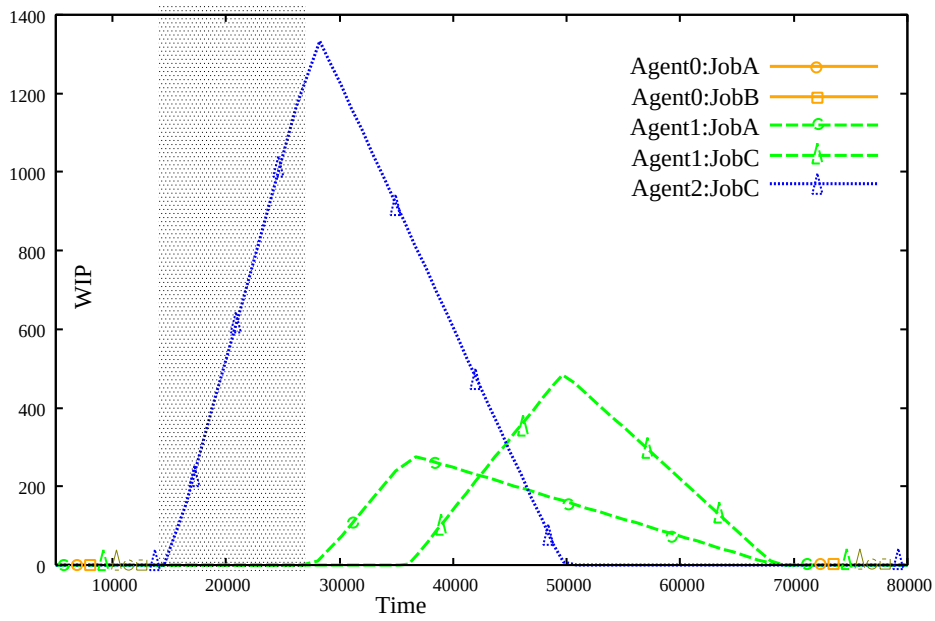

Figure 3.7: Conventional System: WIP

of failure is shown by shaded region. Figure 3.6 shows the capacity utilization profile of agents. Figure 3.7 shows the amount of different WIP that is present at various agents of network.

As the failure on *Agent2* affects *JobC*, its throughput drops to zero with the onset of failure and its finished inventory deficit starts to increase with the progress of failure (dropping lines of *JobC* in Figure 3.4 and 3.5 during the failure). The utilization of *Agent1* drops during failure as it stops receiving *JobC* (Figure 3.6). Please note that system continues to produce the other two jobs with their demanded throughput during failure (the unaltered lines of *JobA* and *JobB* in Figure 3.4 and Figure 3.5 during the failure). Because the jobs are constantly released in this conventional system at demand rate during failure as well, WIP of *JobC* accumulates at *Agent2* (Figure 3.7).

After resolution of failure at 28000 min., *Agent2* starts to clear its accumulated WIP of *JobC* using its full capacity (*Agent2* in Figure 3.6 and 3.4). *Agent1* also starts receiving *JobC* after failure recovery and its utilization also rises to 100% (Figure 3.6). As these backlogged lots of *JobC* have earlier due dates, *Agent1* gives preference to *JobC* over *JobA* in its dispatching. *Agent1* has

limited capacity and initially it utilizes all of its capacity to process *JobC* exclusively. *Agent1*'s exclusive processing of *JobC* results in the stoppage of *JobA*'s production (*JobA* in Figure 3.4). *Agent1* continues to process *JobC* exclusively till time 35000 min., during which the finished inventory deficit of *JobA* increases (Figure 3.5). As *Agent0* continues to process *JobA* at the demand rate, WIP of *JobA* accumulates at *Agent1* (Figure 3.7). By time 35000 min. the deficit of *JobA* and *JobC* becomes equal in terms of delay from their due dates, after which *Agent1* processes both jobs in proportion of their demands. As *Agent1* drops processing of *JobC*, its WIP starts to accumulate after 35000 min (Figure 3.7).

By time 50000 min. *Agent2* has cleared all its accumulated WIP (Figure 3.7) and it restores its processing at demand rate (Figure 3.4). *Agent1* however continues to use it full capacity to clear its accumulated WIP of *JobA* and *JobC*. Because of its less spare capacity *Agent1* takes longer time to clear its accumulated WIP and system simultaneously recovers its finished inventory deficits by time 70000 min. (Figure 3.5). At 70000 min. the recovery is complete and all agents restore their behavior to normal levels.

In this section we explained how the recovery after resolution of failure depends on capacity of *Agent1*. The shortage of *Agent1*'s capacity delays recovery and results in deficits of *JobA*, which is not directly affected by failure. Because of high planned utilization of *Agent1* (80%), the loss of its capacity due to starvation during failure is detrimental to performance of system. Please note that there is no fluctuation in processing of *JobB* during entire scenario.

3.2.2 Behaviors of CABS

The system's recovery can be expedited if *Agent1* has more capacity available during the recovery period. The fraction of its capacity which will be otherwise wasted due to lack of *JobC* during the failure can be utilized to process surplus i.e. more than the stipulated demand of *JobA*. Avoiding *Agent1*'s starvation by processing surplus of *JobA* during the failure will ensure availability of extra capacity and thus faster clearance of *JobC*'s deficit after resolution of the failure.

As *JobA* is being processed by *Agent0* before arriving at *Agent1*, *Agent0* will have to process and supply *JobA* at a higher rate in order to sustain high

utilization of *Agent1*. As *Agent0* is already running near its capacity(83%) to meet the regular demand of *JobA* and *JobB*, it cannot increase the rate of *JobA* high enough to keep *Agent1*'s utilization without reducing its processing of *JobB*. Although *JobB* is not directly affected by the failure, because of indirect dependence between agents in the system, its throughput has to be reduced during the failure in order to maintain utilization of the critical agent *Agent1*. The system in which the agents are running the CABS algorithm achieves this desired behavior.

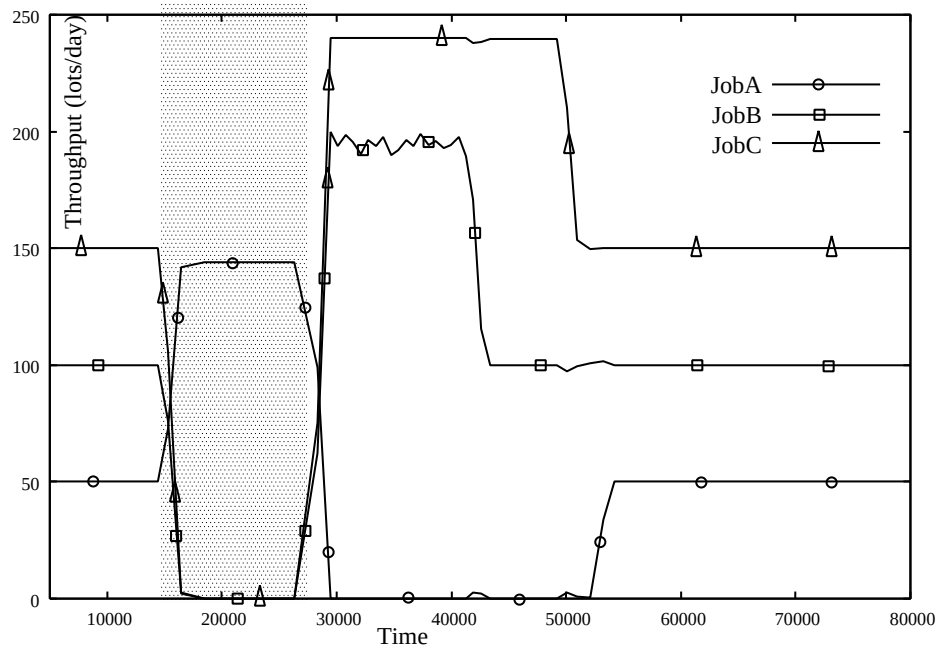

Figure 3.8: CABS: Throughput

Behavior of the system with agents using CABS algorithm is shown in Figures 3.8–3.11. Please note that in comparison to the accumulated WIP of conventional system (Figure 3.7), the system using CABS accumulates very low WIP (Figure 3.11) in this scenario and the scale in Figure 3.11 is different from Figure 3.7 in order to explain the detailed working of CABS. In order to maintain high utilization of *Agent1* during the failure, the CABS system decreases the throughput of *JobB* to increase the throughput of *JobA* (15000 - 28000 min. in Figure 3.8). The surplus of *JobA* produced during the failure allows *Agent1*

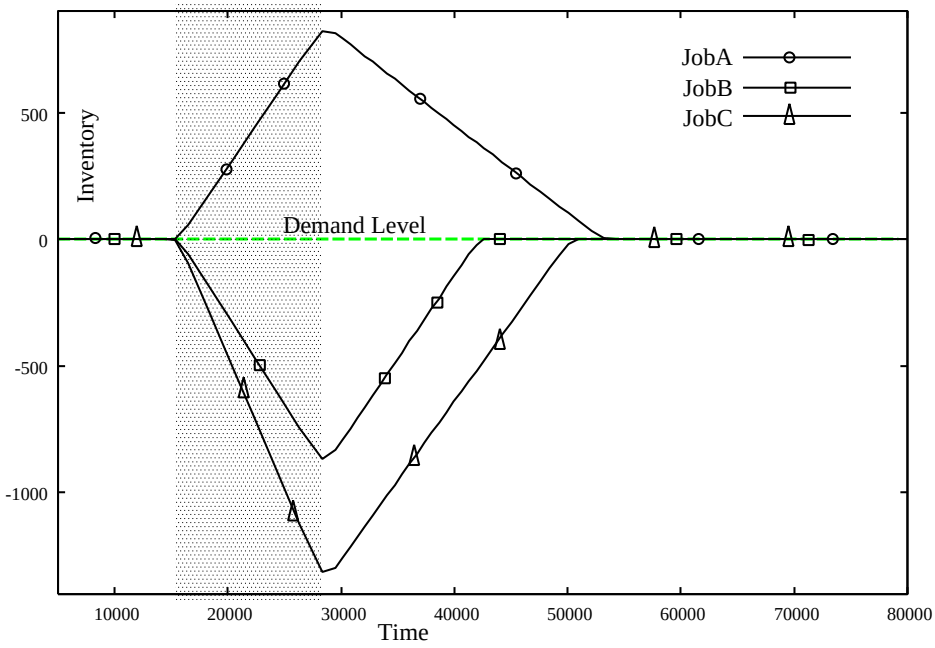

Figure 3.9: CABS: Finished Product Inventory

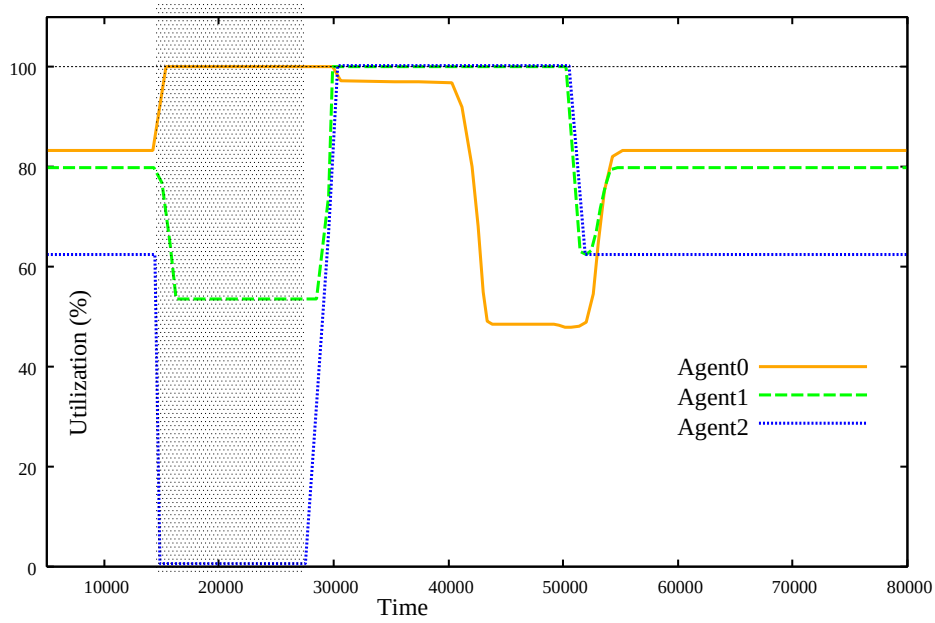

Figure 3.10: CABS: Utilization

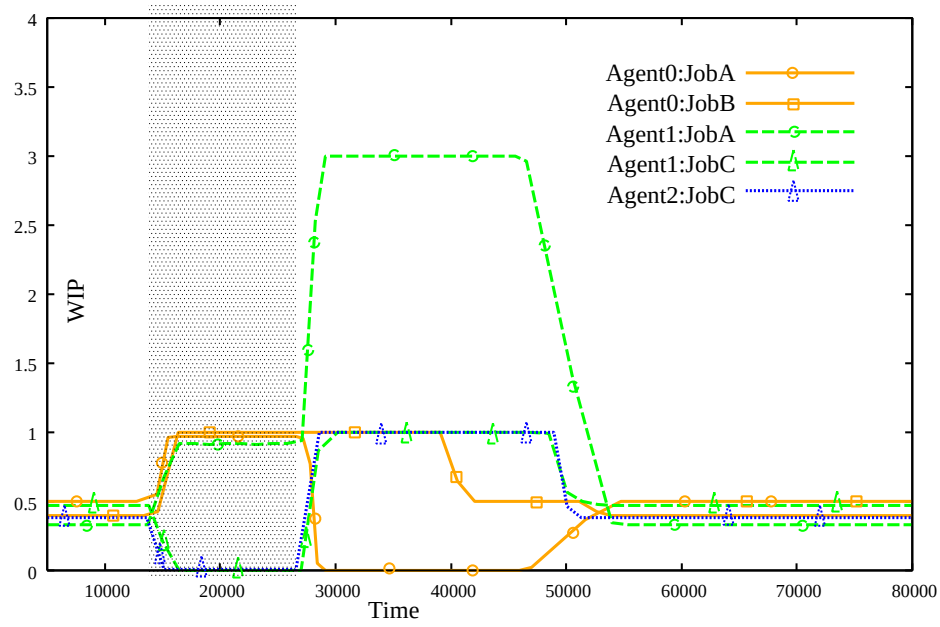

Figure 3.11: CABS: WIP

to have more spare capacity after resolution of the failure, which results in speedy recovery. As a result, compared to the conventional system, the CABS system recovers quickly and clears the deficit of all the jobs by 52000 min. We now explain the detailed working to CABS by analyzing the behavior of agents during different time periods of execution.

Agent0, *Agent1*, *Agent2* are periodically sending the requirement messages to the preceding agents in the process routes of their jobs according to Algorithm 4. The flow of messages is thus in the opposite direction of the flow of jobs shown in the Figure 3.3. After segregating the contents of requirement messages that the agents are sending, we have plotted values of `time limit`, `request rate`, `amount` and `criticality` individually in Figures 3.12 to 3.15 respectively. The X-axis represents the execution time and the Y-axis represents the value of respective requirement parameter that agents are sending at that point of time. In the figures, line *AgentP:JobQ* represents the value of parameter that agent *AgentP* is sending in the requirement message for *JobQ* to its previous agent of *JobQ*. All the parameters except `time limit` are plotted as absolute values as they are sent in the messages. As the time increases monotonously, instead

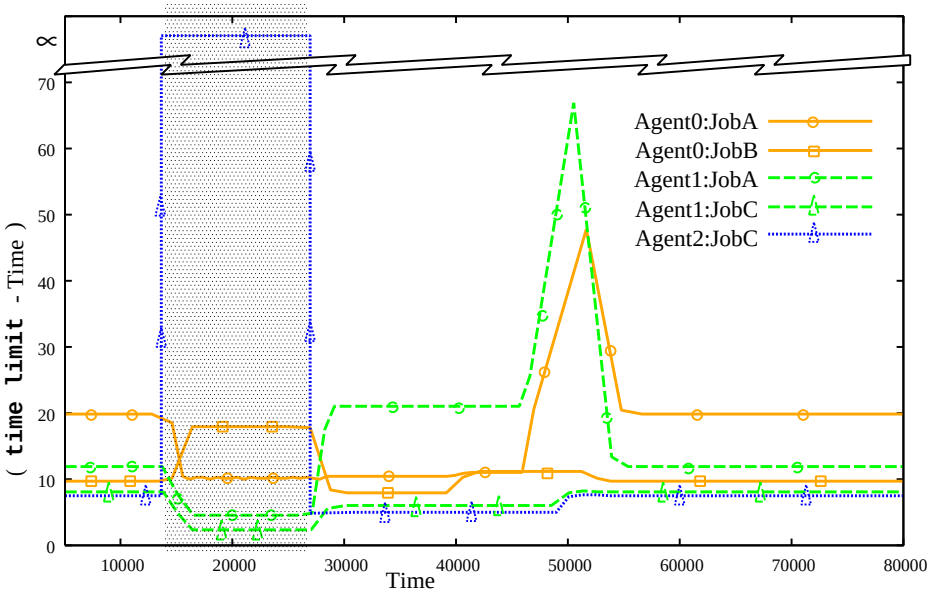

Figure 3.12: CABS Message: `time limit`

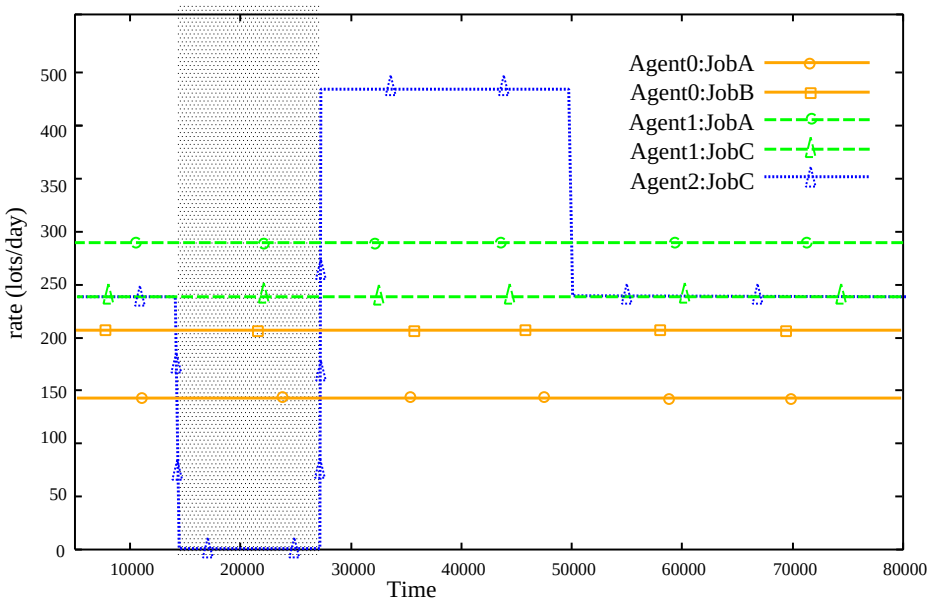

Figure 3.13: CABS Message: `request rate`

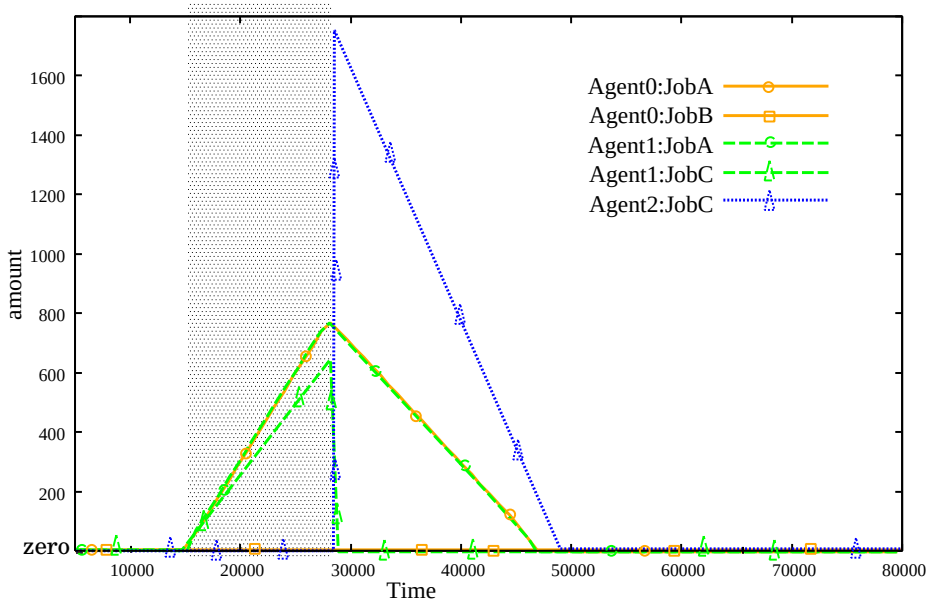

Figure 3.14: CABS Message: `amount`

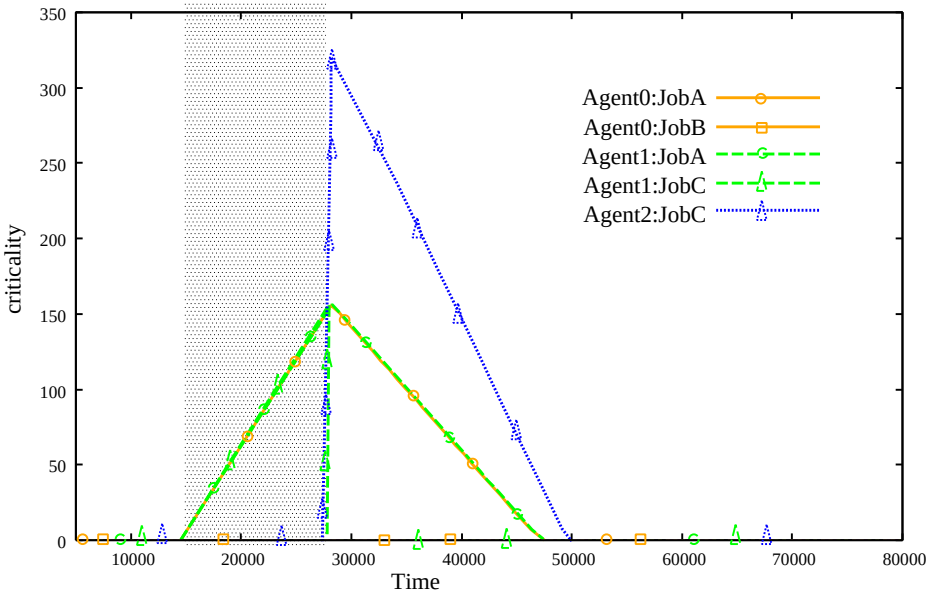

Figure 3.15: CABS Message: `criticality`

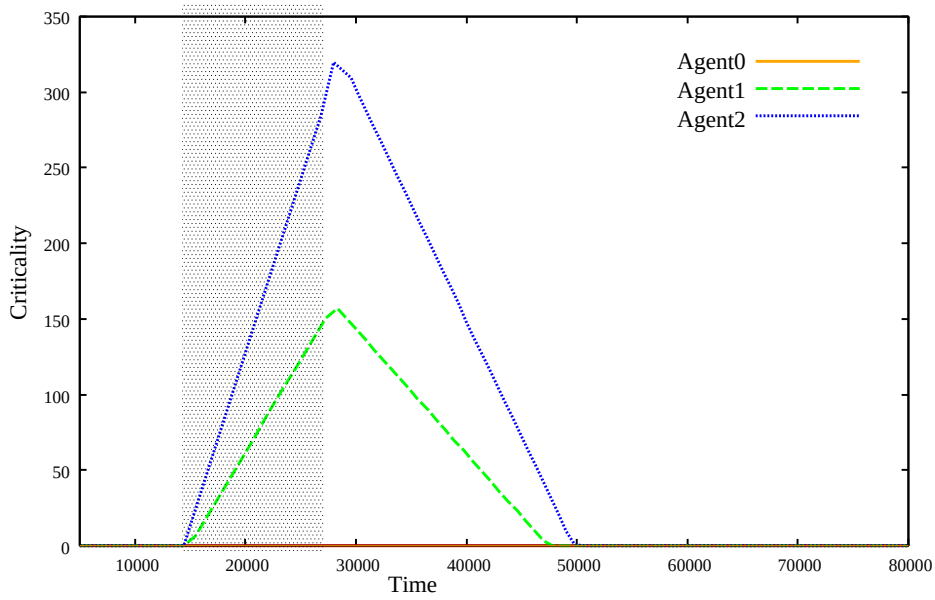

Figure 3.16: CABS: Agents' instantaneous criticalities

of the absolute `time limit` we have plotted its offset from the time when that message was sent (Figure 3.12). At any given time $Time$, (`time limit` - $Time$) indicates how much later in the future the agent wants the next lot of that job. A smaller value of (`time limit` - $Time$) means that the agent needs the next lot immediately, and on the contrary a higher value means that agent needs the next lot at a later time in the future.

Besides the message parameters, Figure 3.16 shows the instantaneous criticality of different agents of the network. Please note that according to Algorithm 4, the `criticality` that an agent is sending in its messages might be different from its own criticality.

Before Failure

In this section we explain the behavior of agents during the smooth running period till 15000 min. During this period the jobs are being steadily produced at the demand rate (Figure 3.8), and hence their finished inventory deficit is maintained at 0 levels (Figure 3.9).

- ***Agent1***: Steady production of jobs at the demand rate during this stable condition results in steady utilization of all agents (Figure 3.10). In this situation, the last agents of jobs assume a static incoming requirement of lots according to their demand (incoming `amount` is 0). Hence, *Agent1* calculates the requirements of *JobC* and *JobA* according to their demand (Algorithm 3:5-8). Since there is no accumulated WIP (Figure 3.11) or surplus production of jobs during this stable condition (Figure 3.9), the `time limit` remains close to current *Time* (Algorithm 3:3,6). *Agent1* sets its own processing capacity as the `request rate` (Algorithm 3:13). Since all the agents are processing their jobs according to the demand and do not have workload deficit, their criticality as defined by Algorithm 2 remains 0 (Figure 3.16). Because of its low criticality *Agent1* passes the demand requirements of *JobC* and *JobA* generated in Algorithm 3 to their respective preceding agents i.e. *Agent2* and *Agent0* (Algorithm 4:13) as shown in Figures 3.12-3.15.

- ***Agent0***: Being the last agent of *JobB*, *Agent0* generates and sends the requirements for *JobB* as *Agent1* does for *JobC* and *JobA* (described above). For *JobA* *Agent0* adjusts the requirements it receives from *Agent1* by Algorithm 3. As there is no accumulation of WIP or surplus production during the steady state, the adjusted `time limit` at *Agent0* also remains close to current *Time*. As the processing time of *JobA* is longer at *Agent0* i.e. 10 compared to 5 at *Agent1* (Figure 3.3), *Agent0* reduces the `request rate` accordingly (Algorithm 3:13). As *Agent0*'s criticality is also low, it propagates the adjusted requirements of *JobA* to its preceding agent i.e. the *Source* Agent for *JobA* as shown in Figures 3.12-3.15.

- ***Agent2***: As *Agent0* adjusts the requirements of *JobA* (described above), *Agent2* adjusts the incoming requirements from *Agent1* for *JobC*. *Agent2* is processing two consecutive steps of *JobC* and its sends the requirements for the first step to its *Source* Agent. Since the requirements for processing the second step of *JobC* is confined to *Agent2* itself, they are not depicted in Figures 3.12-3.15 for the sake of clarity. Although *Agent2* can process its steps of *JobC* at double the rate compared to *Agent1* (see processing times in Figure 3.3), it sends the lower rate of *Agent1* to its preceding

agent (Algorithm 3:13) during smooth operations (Figure 3.13). Since *Agent2* is also not critical during smooth operations, like other agents it also propagates the adjusted requirements as shown in Figures 3.12-3.15.

During Failure

In this section we explain how the agents change their behavior when the flow of *JobC* is blocked at *Agent2* during its failure from 15000 to 28000 min.

- ***Agent1***: With the onset of the failure on *Agent2*, lots of *JobC* stop arriving at *Agent1* and its utilization drops (Figure 3.10). Because of this deficit of *JobC*, *Agent1*'s workload deficit and criticality start to rise due to Algorithm 2 (Figure 3.16). This rise of *Agent1*'s criticality makes it higher than the `criticality` of the incoming static requirements, which is always 0. Due to this raised criticality (Algorithm 4:7), *Agent1* changes its messaging behavior to override the incoming requirements (Algorithm 4:8-11). Now *Agent1* stops propagating static incoming 0 `criticality`, and starts sending its own urgent requirements for both of its jobs (Algorithm 4:2). In order to recover its deficit as soon as possible, *Agent1* continues to request both *JobC* and *JobA* at the earliest timing to avoid its starvation (Algorithm 4:8 and Figure 3.12) and at the maximum rate at which it can process them (Algorithm 4:9 and Figure 3.13). As its workload deficit gradually increases during the failure, the `amount` that *Agent1* needs urgently to recover that deficit also increases with time (Algorithm 4:10 and Figure 3.14). Compared to *JobC*, *Agent1* sends larger `amount` values for *JobA* because the required amount depends on the processing time of the job (Algorithm 4:10), which is shorter for *JobA* than for *JobC* (Figure 3.3). *Agent1* now sends its own criticality in the messages (Algorithm 4:11).

  The key point to note here is the change in *Agent1*'s messaging behavior. When *Agent1* was not critical before the failure, it was propagating the static incoming requirements. As it became critical due to its starvation of *JobC* during the failure, irrespective of the production status of its jobs, it overrides their incoming requirements and starts sending its own requirements with higher criticality, and it does so for all of its jobs.

- ***Agent0***: *Agent0* receives the *JobA*'s higher `criticality` requirement of *Agent1*. As 83% of its capacity is already utilized before the failure, it cannot meet the higher rate requirement of *JobA* without reducing its processing of *JobB*. Because of higher incoming `criticality` from *Agent1*, *Agent0* starts giving preference to processing *JobA* at a higher rate, according to its dispatching policy *selectTask* described in Algorithm 1. Although *Agent0* has some WIP of *JobB* also during this period (Figure 3.11), due to contineous dispatching of *JobA* instead of *JobB*, the `time limit` that *Agent0* sends for *JobB* has a higher offset from *current time*. Because of its limited capacity, processing more *JobA* results in the reduction of its throughput of *JobB* (Figure 3.8). And due to this increased processing of *JobA*, *Agent0*'s utilization rises to 100% (Figure 3.10).

  As the criticality of *Agent0* itself remains low (Figure 3.16), it propagates its incoming requirements to its previous agents i.e. the *Source* agents after adjusting them appropriately.

- ***Agent2***: Although *Agent2* continues to receive requirements from *Agent1*, it stops requesting further jobs during the failure (Algorithm 4:16-19). Meanwhile, its utilization remains 0 (Figure 3.10) and the criticality gradually rises to a high value (Figure 3.16).

After Resolution of Failure

In this section we explain the behavior of agents after the failure get resolved at time 28000 min.

- ***Agent2***: After the resolution of its failure, during which it had accumulated workload deficit and became the most critical agent in the system, *Agent2* starts to request *JobC* immediately and at its threshold rate (Figure 3.12 and 3.13). Although the incoming `request rate` from *Agent1* is lower, because *Agent2*'s criticality is higher it sends its own higher rate at which it can process its process steps (Algorithm 4:9 and Figure 3.13). *Agent2* uses its full capacity to recover its workload deficit as soon as possible (Figure 3.10). By processing the jobs at a higher rate, *Agent2* gradually reduces its workload deficit and criticality (Figure 3.16). The

`amount` that it is requesting also decreases gradually due to the same reason (Figure 3.14). It sends its own criticality with the request (Figure 3.15). It continues this behavior till its criticality drops to 0 by time 50000 min, after which it restores its behavior to normal.

- ***Agent1***: Since the delayed jobs of *JobC* have earlier due dates, *Agent1* gives preference to them over *JobA* in its processing according to Algorithm 1. *Agent1* exclusively processes *JobC*, and it results in increase of *JobC*'s throughput and stoppage of *JobA*'s production (Figure 3.8). This behavior of *Agent1* simultaneously decreases the finished inventory deficit of *JobC* and surplus of *JobA* simultaneously (Figure 3.9). As *Agent1* steadily receives jobs from *Agent2*, it is able to utilize its full capacity to process them (Figure 3.10). During the failure, the increased flow of *JobA* could not compensate for the stopped flow of *JobC* and *Agent1* also accumulated some workload deficit and increased its criticality as a result (Figure 3.16). Because of its high utilization due to availability of *JobC* after the failure resolution, *Agent1*'s workload deficit and criticality decreases gradually (Figure 3.16). Till its criticality drops to 0 at time 47000 min., *Agent1* continues to send small `time limit` and appropriate `amount` for both *JobC* and *JobA* that is required to recover its workload deficit. Although *Agent1* is sending a small `time limit` value for both *JobC* and *JobA* (Algorithm 4:2) and it as available WIP of both the jobs (Figure 3.11), it continuously defers the processing of incoming *JobA* in favor of *JobC* due to its dispatching policy (Algorithm 1). As a result of this dispatching behavior of *Agent1*, the `time limit` values that *Agent1* sends for *JobA* during the recovery period have persistent higher offset from the *current time* (Figure 3.12). When *Agent1*'s criticality drops to 0 at time 47000 min., it starts sending requirements of *JobA* according to its demand (Algorithm 4:12). Since *JobA* still has a surplus production at 47000 min. (Figure 3.9), the `time limit` parameter for *JobA* has a positive offset from the current *Time* till the production of *JobA* is restored to the demand level (Algorithm 3:6 and time 47000 to 52000 min. in Figure 3.12).

- ***Agent0***: At *Agent0*, although the incoming requirement of *JobA* has higher `criticality` compared to the static 0 incoming criticality of *JobB*, *JobA*'s `time limit` has a consistent higher positive offset from the *current time* as described above (Figure 3.12). Because of this time offset, processing *JobB* does not disturb the requirement of *JobA* and *Agent0* is able to dispatch *JobB* (Algorithm 1). Because of its ability to process *JobB*, *Agent0* requests *JobB* more frequently to recover its finished inventory deficit by sending lower `time limit` values of *jobB* to *Source* agent during this period (Figure 3.12). As *Agent1* has altogether stopped the processing of *JobA* during this recovery period which results in a persistent offset in *JobA*'s `time limit`, *Agent0* exclusively processes *JobB* at a higher rate (Figure 3.8). This production of *JobB* at a higher rate results in recovery of its finished inventory deficit, which is complete by 40000 min. Since *Agent0*'s criticality remains 0, it propagates the requirements of *Agent1* at all the time. Hence, during the time 47000 to 52000 min. *Agent0* also sends the higher `time limit` value according to the surplus production of *JobA*.

The system recovers the deficit of all the jobs by time 52000 min., after which the all the agents restore their requirements and behaviors to the normal level, as they were before the occurrence of the failure. In essence, because the capacity of *Agent1* was utilized during the failure by producing *JobA* in excess, it could afford to reduce the processing of *JobA* and utilize more of its capacity to recover the deficit of *JobC* after the failure resolution.

3.3 Conclusion

In this chapter we explained our proposed multiagent coordination mechanism, CABS. In CABS, each workstation is modelled as an agent. The agents coordinate their actions by the requirement message defined in the system. The flow of messages is in reverse direction of the flow of jobs and agents make use of messages' contents to decide their actions. We explained the dispatching algorithm which agents use to sequence their buffered jobs by analyzing the incoming requirement messages from their succeeding agents. Agents make use of

incoming requirement messages and their own criticality to identify bottlenecks in the system. The agents use their own performance w.r.t. to the demand to identify their dynamic workload, which is then used to calculate their criticality. We explained the algorithm that agents use to generate requirement messages for preceding agents according to the location and requirements of dynamic bottlenecks in conjunction with its own WIP. The working of CABS algorithms using a small manufacturing system and single failure was explained in detail, as to how the proposed coordination mechanism avoids the starvation of critical resources in order to achieve better performance than a conventional system. In the following chapter we evaluate CABS with more realistic scenarios of semiconductor fabrication process with repeated random failures.

Chapter 4

Empirical Validation

In this chapter we describe our experimental setup and present the results which empirically shows that CABS outperforms a conventional system in dealing with complex network systems.

4.1 Experimental Setup

In our research, we use a manufacturing problem, especially a semiconductor fabrication process, as a benchmark for controlling large-scale network systems. Semiconductor fabrication is among the most complex manufacturing processes. For example, the manufacturing steps for semiconductor manufacturing usually number a few hundred, with numerous repetitive re-entrant loops. Its leadtime extends over a couple of months (Atherton & Atherton 1995, Pfund et al. 2006). Furthermore, fierce competition of global market place and short technology life cycles require 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 early part of experience curves of manufacturing.

For experiments a simulation system is developed to model the manufacturing process with agents to test proposed algorithms in CABS. The system is built using SPADES (Riley & Riley 2003) middleware[1], which is an agent-based discrete event simulation environment. It provides libraries and APIs to build

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

agents that interact with world by sending and receiving time-based events. The details of event mechanism of CABS are provided in Appendix B.

4.1.1 Experimental Agent Network

For empirical validation, we used a dataset from the Measurement and Improvement of Manufacturing Capacity (MIMAC) testbed datasets of the wafer fabrication processes (Fowler & Robinson 1995) from Arizona State university[2]. The dataset specifies the production steps of semiconductor manufacturing. We have used the same dataset for all our experiments.

Type of product	Non-volatile memory	
Number of Products	2 (Product1 & Product2)	
Number of Workstations	83	
	Product1	Product2
Number of Steps	210	245
Total processing time (hour)	313.4	358.6
Demand rate (wafers/day)	336.0	168.0

Table 4.1: Details of test problem

Table 4.1.1 shows the properties of test problem we chose from MIMAC datasets. It has basic characteristics of a semiconductor manufacturing process such as lengthy process flows with many repetitive re-entrant loops and a couple of critical workstations. There are 2 products in the test problem and Figure 4.1 depicts their process flows. Each node in network represents a workstation and average planned utilization of three workstations (i.e, No.67, 76 and 78) is higher than 80%.

Notwithstanding the high planned utilization of some workstations, many other workstations can also become temporary bottlenecks when they suffer large capacity losses due to unexpected failures in system. Figure 4.1, which depicts the process flows of products through the workstations in the experiment problem, can be viewed as a complex network. It is noteworthy that, although the number of nodes in the network is moderate (less than one hundred), because

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

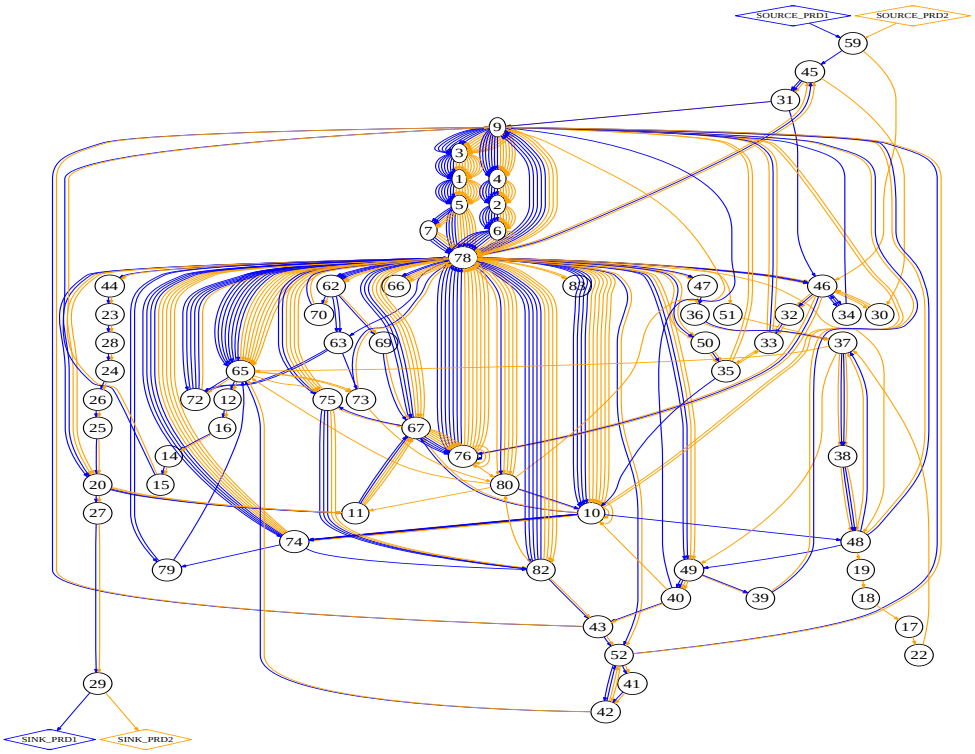

Figure 4.1: Process flows of test problem

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 of "complex networks" (Barabási 2002).

In the experiments, we made the following assumptions to focus our investigative attentions to the basic properties of CABS: (1) there is no variability in processing times of operations, (2) batches are not considered, (3) no setup time is considered, (4) operators are not considered in the model, (5) there is neither product rework nor scrap, and (6) stochastic machine failure is modeled using exponential distribution.

4.1.2 Criteria of Performance Evaluation

According to Kumar & Kumar (2001) Performance Metrics are the indexes that are utilized to measure production performance through simulation case studies. The following are some commonly used metrics in semiconductor manufactur-

ing processes: costs, cycle time, machine utilization, yield, and Work-In-Process (WIP) inventory. From the various proposed performance metrics, it is apparent that the performance of a system is gauged not only by the throughput it achieves, but also the quality of service that it is able to provide. *Lean manufacturing*, that combines high throughput with low WIP has become a minimum requirement for competitiveness in manufacturing industry (Womack et al. 1991), although it is hard to achieve in unreliable and dynamic systems. As a criteria to evaluate performance of CABS, we exploit leadtime that achieves the same level of throughput. Little's Law (Little 1961) in queueing theory states that in order to reduce leadtime by maintaining the same throughput requires WIP to be reduced (Equation 1.1). However, in a variable and unpredictable environment, reducing WIP tends to decrease throughput by cutting back the safety buffer of jobs with the agents. During failures, the reduction of safety buffers increases the probability of forcing idle time of agents 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 ($ProductSet$) for representing overall throughput of the system:

$$Aggregated\ Processing\ Time = \sum_{\forall i \in \{ProductSet\}} Process\_Time_i Throughput_i$$

For representing overall leadtime of the system, we calculate the aggregated leadtime as:

$$Aggregated\ Lead\ Time = \sum_{\forall i \in \{ProductSet\}} Leadtime_i Throughput_i$$

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.

4.1.3 Details of Experiments

All the agents in the network fail repeatedly according to exponential distribution. The mean-time-to-recovery (MTTR) and mean-time-between-failures (MTBF) of the distribution is different for each agent. Each agent is initialized with a different random seed and failure pattern of agents is thus independent of each other.

We have collected the results after executing the simulation of specified semiconductor system for a duration of 2 months. In the high demand experiments, the total number of jobs that got processed through the system were around 760. The total number of failures that occurred in the system during the simulation period was 1400. All agents failed randomly according to their individual MTBF and MTTR parameters of exponential distribution.

For both CABS and CONWIP, we conducted eight independent runs of the system with different random failures. The average of those results is shown as a single datapoint. The results of experiments were taken after the system's initial stabilization at its startup.

4.2 Experimental results of CABS

We conducted a series of simulation experiments with CABS system. For evaluation, different CABS experiments were done by changing the demand levels of products. Figure 4.2 shows the performance of CABS in terms of *Aggregated Processing Time* and *Aggregated Lead Time* for each demand level. Each point in the figure corresponds to a demand level and more than 10 demand levels were used. According to the range of demand levels, the resource utilization for the most heavily loaded agent in the steady state ranges from 50.8% to 86.5%.

The large number of random failures cause starvation of various agents for different durations resulting in a very dynamic system. Due to the resulting dynamic workload deficits of agents, their criticality (as defined in CABS) also changes dynamically. Figure 4.3 shows how the criticality of various agents changes during the course of execution in one of the experiments. As the criticality represents the bottleneck factor of an agent, numerous intersections of

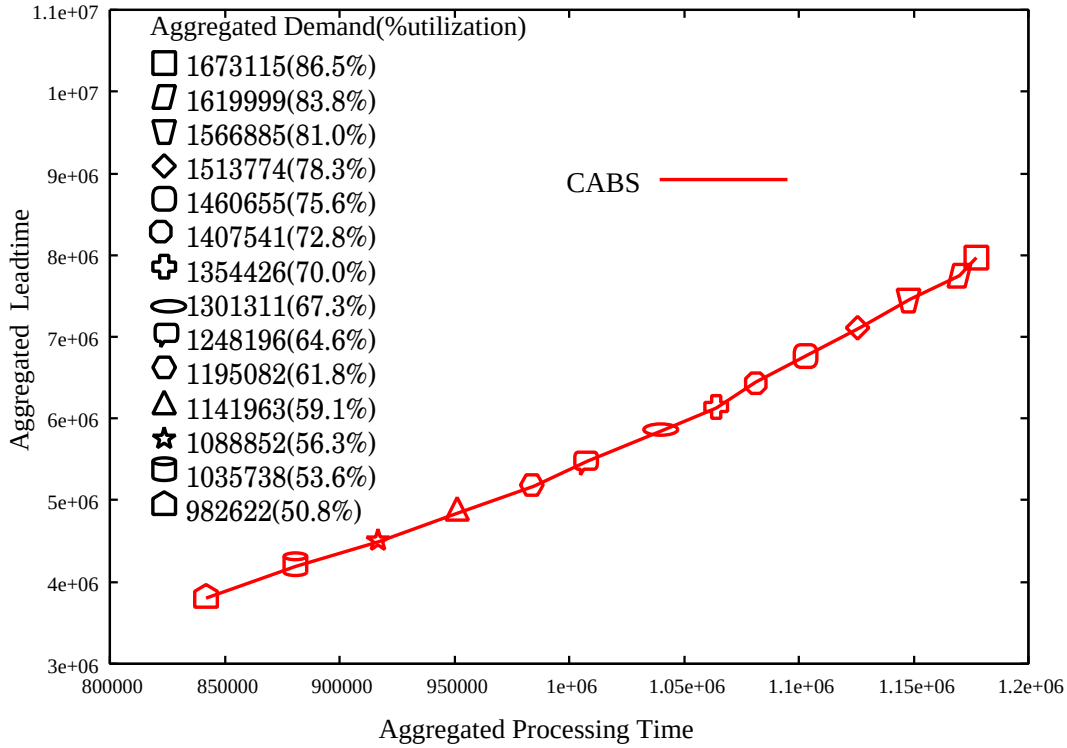

Figure 4.2: Experimental results of CABS

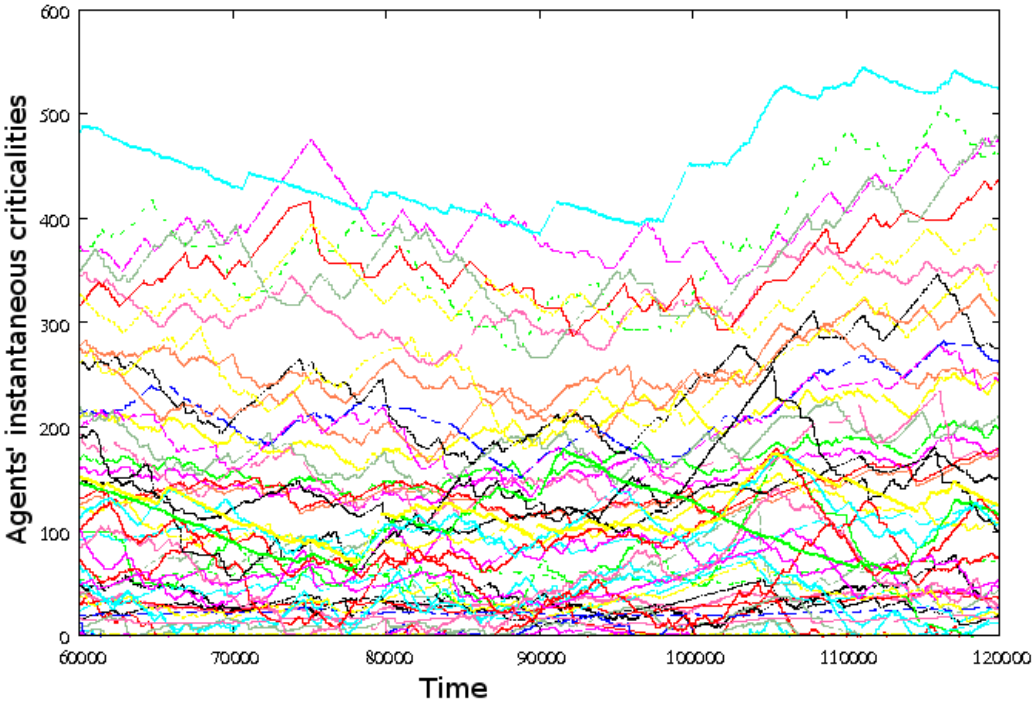

Figure 4.3: CABS: Agents' instantaneous criticalities

agents' criticalities in the figure shows that various dynamic bottlenecks are created and they change over time.

4.3 CONWIP

In order to achieve desired performance from manufacturing system various techniques have been developed. Many systems control the flow of material into and within the system, and a lot of research has been done on them. Broadly these mechanisms can be categorized into *push* and *pull* systems. The Material Requirement Planning (MRP) systems which release jobs into the system according to schedule based on customer orders are an example of *push* system. Push systems do not accept feedback from the system. They release jobs in the manufacturing process according to predetermined schedule. Due to absence of a tight feedback mechanism in push systems, undesirable WIP piles up in system as release of jobs and dispatching at workstations continues according to the schedule even when a following workstation is blocked due to failure or some other disruption. Due to the disadvantages in term of reactivity and robustness to disturbances, *push* systems are considered inferior to the alternative, *pull* systems. In contrast with a push system, in a pull system a job advances only when the following entitiy in the system authorizes the move. Push and Pull represent extremes of WIP management, which is a crucial control for manufacturing systems. In conjunction with MRP or some equivalent demand prediction method, based on demand the push systems develop schedule of amount of jobs to be released (i.e., pushed) into system. In the pull approach, production is authorized (i.e., released or dispatched) only as inventory is actually (or virtually) consumed. For enforcing the designated WIP levels, use of kanban cards (Ohno 1988) has been popular. In this system, a kanban card is used as an authorization signal and the predetermined number of cards limit the amount of WIP in the system.

CONWIP (Spearman, Woodruff & Hopp 1990) stands for Constant Work-In-Process, and designates a pull based control strategy that always maintains a fixed number of jobs in the complete system. Once a job leaves the system after completing its processing at the last agent of its process route, the system releases another job by sending it to the first agent in its process route. The

steady WIP in the system acts as safety buffer and avoids the starvation of resources during failures. In various comparative studies, CONWIP is considered better than the push systems (Ovalle & Marquez 2003). According to (Hopp & Spearman 2000), following are the advantages of CONWIP system over a pure push approach:

- The WIP level (which serves as the trigger for the introduction of products into a Pull production line) is directly observable, while the release date in a Push system must be set with respect to (an unobservable) capacity.
- It requires less WIP on average to attain the same throughput.
- It is more robust to errors in the control parameters.
- It facilitates working ahead of schedule when circumstances permit.

Compared to other pull systems, CONWIP is considered superior and tolerant against instability of systems and is widely used in practice (Bonvik, Couch & Gershwin 1997, Stevenson, Hendry & Kingsman 2005).

Since pull CONWIP system controls WIP and observe throughput, their performance is critically dependent on the choice of WIP level. In systems that use cards (kanbans) to govern WIP, setting WIP is done by choosing card counts. To meet customer requirements, WIP levels (or card counts) must be large enough to achieve the desired throughput. They must also be small enough to prevent excessive WIP. Therefore, a basic problem facing pull systems is determining the minimum WIP level to attain desired throughput rate (Hopp & Roof 1998). In practice the levels of WIP to be maintained during execution is decided through simulations by using the information of forecasted demand and system performance. Since the optimal WIP level for CONWIP can be decided only through trials and errors, we have conducted a series of simulations with different levels of WIP. As the semiconductor fabrication process of our experiments produces two products, we have conducted experiments with different combinations of WIP levels for both products. The *Aggregated Processing Time* (throughput) achieved by CONWIP with different combinations of WIP for one set of experiments is shown in Figure 4.4. The total number of WIP combinations (data points) in for each set of CONWIP experiments is close to 1000.

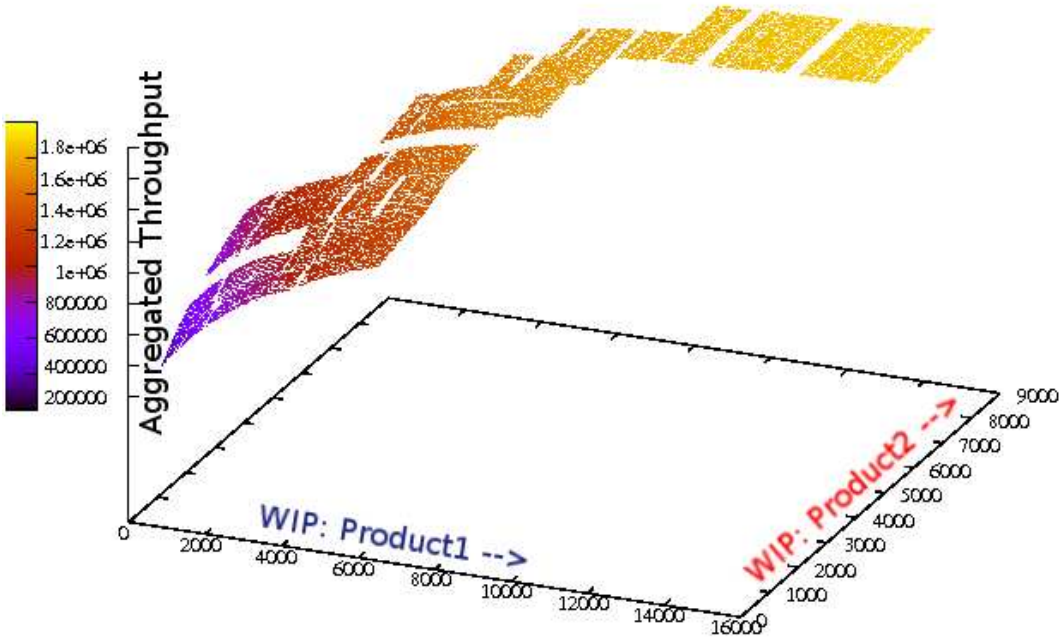

Figure 4.4: CONWIP: Different WIP levels

4.4 Comparison with CONWIP

In this section we compare the performance of CABS with CONWIP. We compared the performances of CABS with two variations of CONWIP, (a) CONWIP using the earliest due date first (EDD) dispatching rule and (b) CONWIP using the Least Slack rule. Details of various dispatching rules are provided in Blackstone et al. (1982). For each experiment of CABS we conducted a series of different CONWIP simulations with different WIP levels. From the series of CONWIP experiments we have selected the best CONWIP levels that achieve the same aggregated throughput with that of CABS. Then we compare the best result of CONWIP with the result of CABS. The comparison of CABS and CONWIP in terms of *Aggregated Processing Time* and *Aggregated Lead Time* is shown in Figure 4.5.

Unlike to a single failure scenario in Section 3.2.2, in these experiments, since failures occur randomly and at all agents based on exponential distribution, CABS may not be able to fully exploit its flexibility of controlling flows of tasks

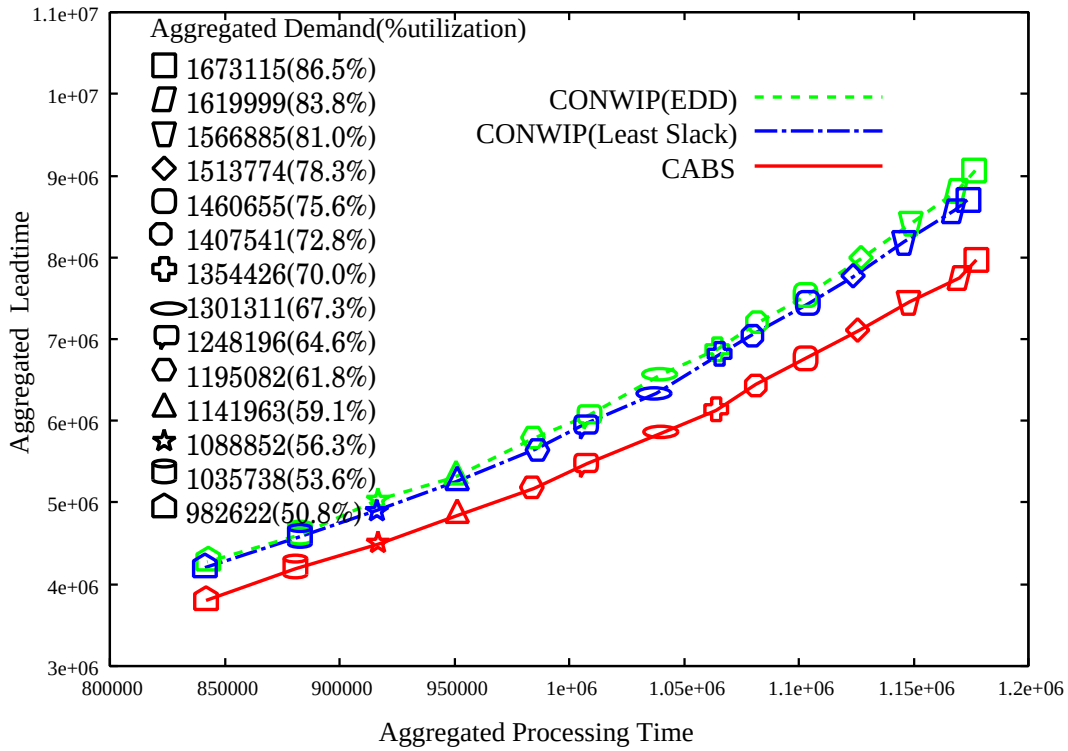

Figure 4.5: Comparison of CABS and CONWIP

to increase production of the appropriate products during failures and after their resolutions. However, the presented results show that the CABS is better than CONWIP from two perspectives:

- **Better Performance**: Figure 4.5 shows that compared to CONWIP, CABS achieves same *Aggregated Processing Time* with a smaller *Aggregated Lead Time*, specially in the high demand region. As *Aggregated Lead Time* corresponds to amount of WIP (Little 1961) and *Aggregated Processing Time* refers to the system throughput, the results also means that CABS achieves a better tradeoff between throughput and leaditme and requires lesser amount of WIP to achieve a given throughput, thus improving the responsiveness and running cost of the system.

- **Better Control**: To be noted is that best WIP levels of CONWIP that are used for comparison had to be determined in advance by using a

series of simulations whereas CABS determines its dynamic WIP level autonomously without any input from users. This signifies that CABS can handle more dynamism in the system and autonomously adapt to a range of conditions. In practice, it is difficult to anticipate all the possible scenarios in dynamic systems in which workstations can fail unexpectedly for random durations. Due to the size and dynamism of system, it is not possible to evaluate the performance of selected WIP level in huge number of possible failure scenarios. The WIP levels selected by simulations done in advance can only provide a average performance over a limited set of scenarios. The failures that occur during real execution can be very different from the simulated scenarios, thus undermining the utility of predetermined WIP levels in CONWIP. CABS on the other hand dynamically reacts to the occurring disturbances, and its coordination mechanism regulates flow of jobs and WIP levels autonomously.

4.5 Conclusion

In this chapter we presented the details of our experimental setup and the results of experiments used to compare the performance of CABS with a popular conventional control mechanism, CONWIP. A simulation platform was developed to model the semiconductor fabrication process as multiagent system in which agents utilize algorithm of CABS to coordinate their actions. Random failures were simulated on all the workstations of system and by using different demand rates, a series of experiments were conducted to record the performance of CABS in terms of achieved throughput and leadtime. Under identical failures, a series of experiments were conducted with different WIP levels of conventional CONWIP system. After selecting the best CONWIP levels, the performance of was CABS was compared with CONWIP. Using the comparative results, we showed that CABS autonomously achieves a better throughput and leadtime tradeoff than the CONWIP. Using its coordination mechanism, CABS dynamically manipulates the flows of various jobs to avoid the starvation of bottlenecks in the system. The flow of jobs in the system also depends on the availability of buffers, and in the following chapter, we evaluate the applicability of CABS in the situations when the flow of jobs cannot be increased either due to re-

strictions on the release of jobs or due to restrictions on buffer capacity in the system.

Chapter 5

Applicability of CABS

The results in the previous chapter highlight the fact that CABS derives its advantage from its flexibility to increase the flows of various jobs at appropriate times. In those experiments we assumed that CABS has the liberty of freely increasing the processing rate of jobs with the only constraint being the capacity of agents. In order for CABS to realize its full potential, the target network system should have desired flexibility. To evaluate the applicability of CABS, in this chapter we show behavior of CABS by putting constraints on favorable properties of network system.

The rate at which an agent can process jobs is limited by its capacity, and it drops to zero during failure of agent. If an agent is not processing its incoming jobs at the rate at which they are arriving, those jobs accumulate as WIP with the agent. Hence for an agent to be able to process new jobs, its succeeding agent should have available buffers to hold those newly processed jobs. As the buffers in real-life are tangible resources, they are finite in number. The buffer capacity of an agent thus puts a constraint on its previous agents' processing of jobs and restricts when they can process more jobs. As the jobs flow through a sequence of agents, the saturation of buffers leads to a cascading effect and more and more preceding agents have to restrict their processing. In the previous experiments of previous chapter we have assumed that system has infinite buffers. In this chapter we will show the performance of CABS by limiting the buffer sizes in the system.

In the previous chapter we have also assumed that the jobs can be released

in the system freely, i.e. any number of jobs can be released in the system at any time, without any consideration to their original demand. This flexibility of *Source* agents to release the jobs at any time adds to the advantage of CABS and enables it to pull and process jobs before their stipulated time. As shown in the example in Section 3.2.2, the early releasing (and processing) of *JobA* during the failure helps in avoiding starvation of the critical agent (*Agent1*). As this advance of job release is a deviation from the planned behavior, it may not be feasible and/or not desirable in many real-life systems. For example, due to the short product life cycles in semiconductor industry, it is undesirable to commit surplus material for products because of their uncertain future demand. Similarly in the road traffic network, the release time is supposedly fixed as the entry of commuters or vehicles cannot be advanced from their stipulated time. Later in this chapter we will also show how performance of CABS changes by limiting advance of job release to the system.

5.1 Effect of Limitation on Buffer Capacity

To highlight the utility of buffers in system, we first show a simplified scenario using the semiconductor fabrication network of Figure 4.1. In this simplified scenario, a single failure occurs at time 50,000 and recovers at time 90,000 on an agent (Workstation No. 19 in Figure 4.1) that is processing only the 105^{th} step of $Product_2$. To emphasize the effect of buffers we compared results of CABS on same network but having different buffer capacities, and also with a conventional system. In experiments we have restricted the maximum number of jobs that can be present in system at any time. The restriction is imposed by system-wide capacity of buffers for each product type. The behavior of CABS on a system having system-wide capacity of 100 buffers for both products is shown in Figure 5.1, 5.3 and 5.5 (in terms of system-wide WIP levels, throughput and finished inventory respectively). The graph of system-wide WIP shows the total number of jobs present in system at a given point in time. During an identical failure, behavior of CABS on a system having a system-wide capacity of 38 buffers for both products is shown in Figure 5.2, 5.4 and 5.6 (in terms of system-wide WIP levels, throughput and finished inventory respectively). The failure duration is shown by the shaded zone in the graphs.

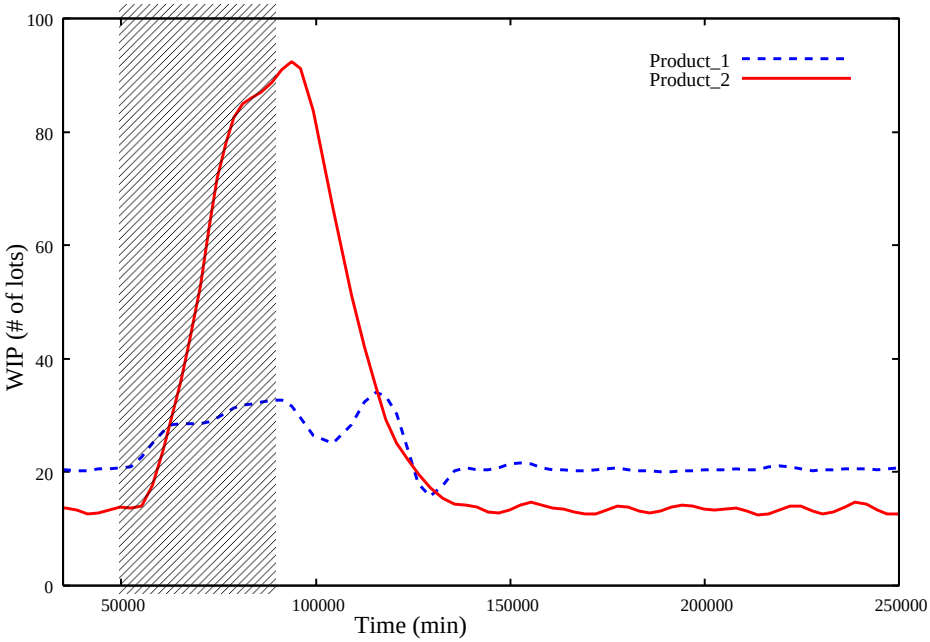

Figure 5.1: CABS(Buffers=100): **WIP**

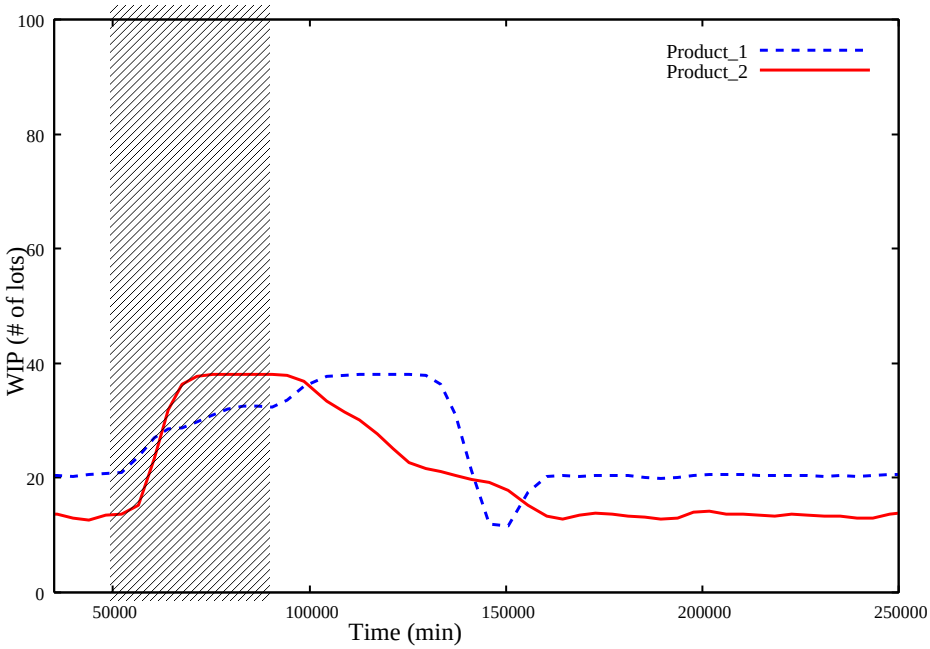

Figure 5.2: CABS(Buffers=38): **WIP**

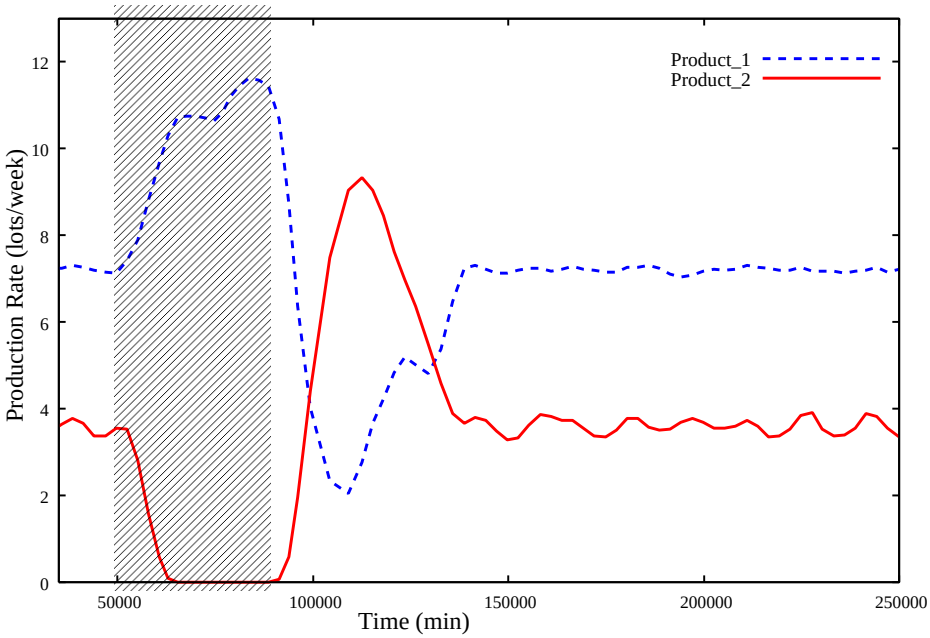

Figure 5.3: CABS(Buffers=100): **Throughput**

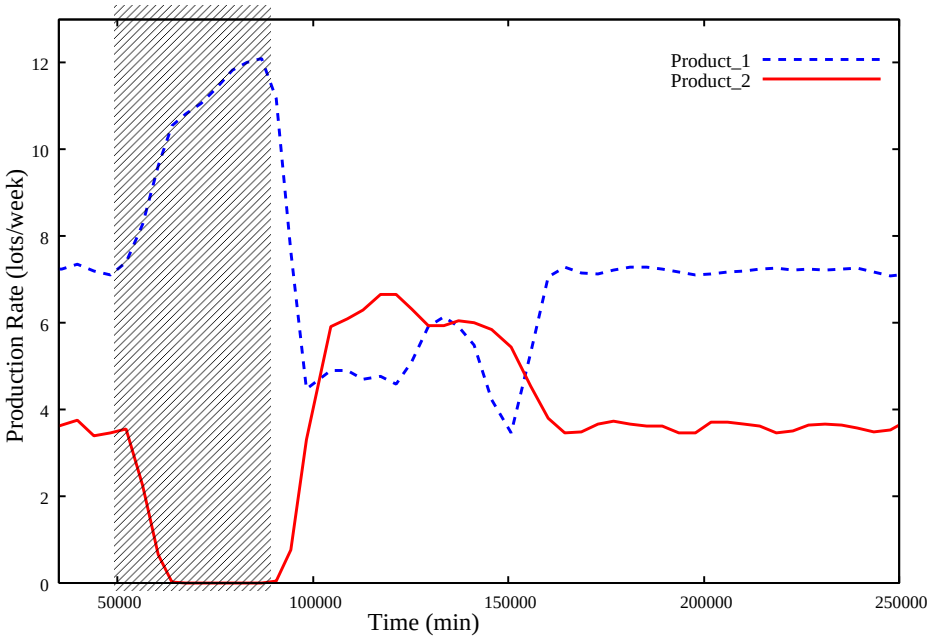

Figure 5.4: CABS(Buffers=38): **Throughput**

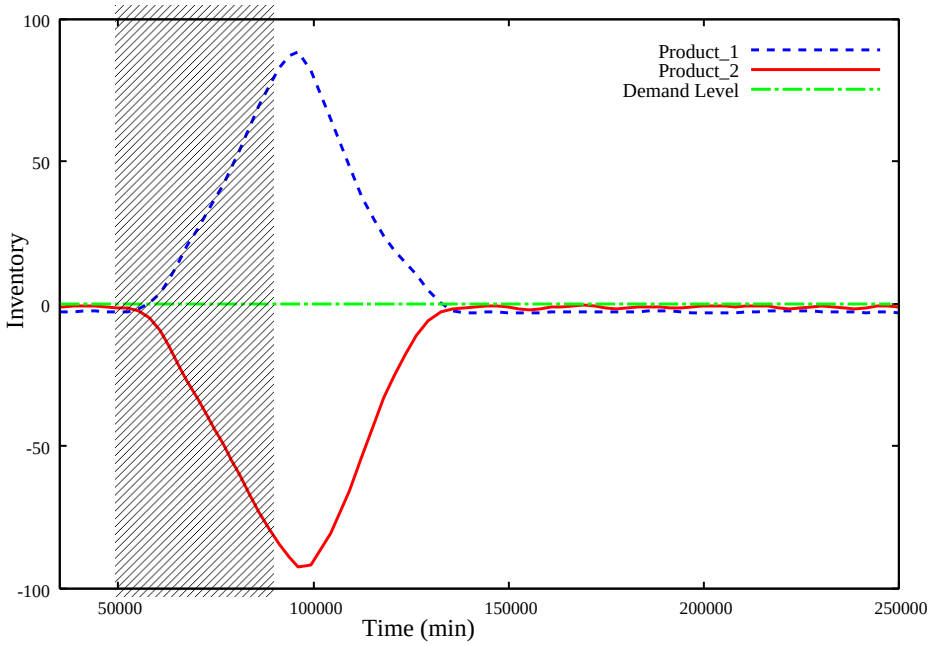

Figure 5.5: CABS(Buffers=100): **Finished Product Inventory**

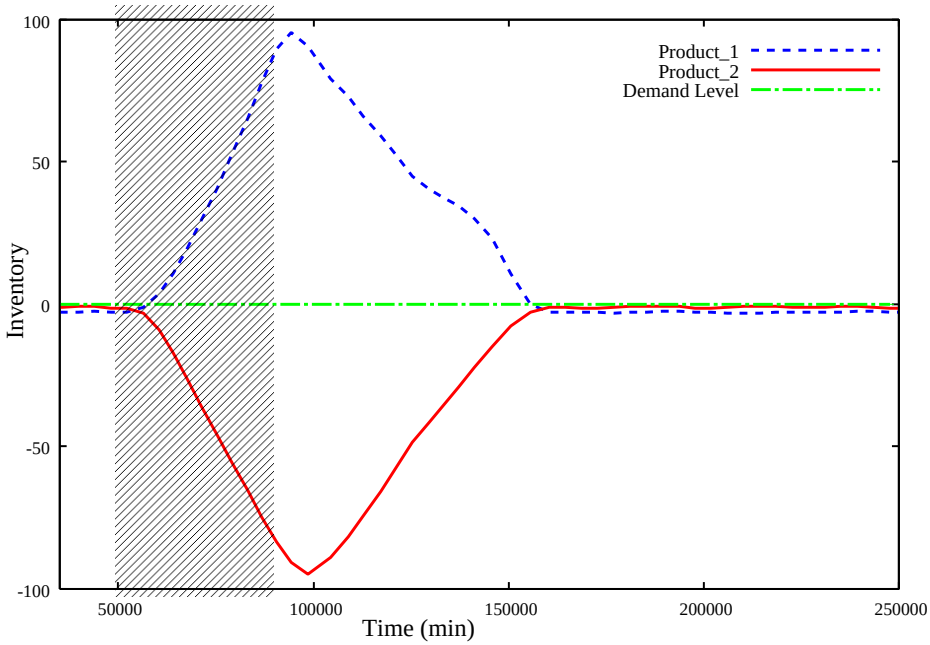

Figure 5.6: CABS(Buffers=38): **Finished Product Inventory**

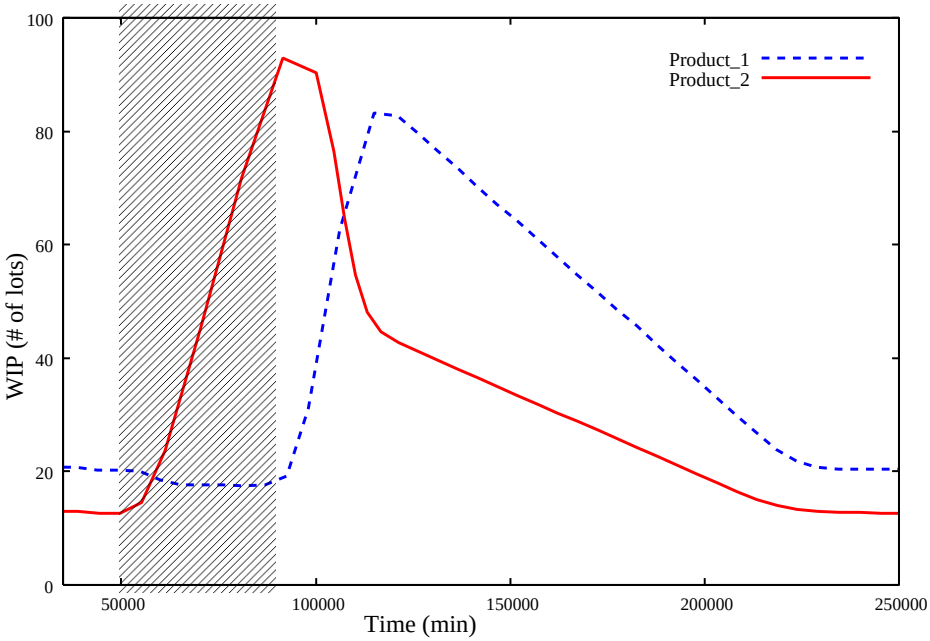

Figure 5.7: Conventional System(Buffers=100): **WIP**

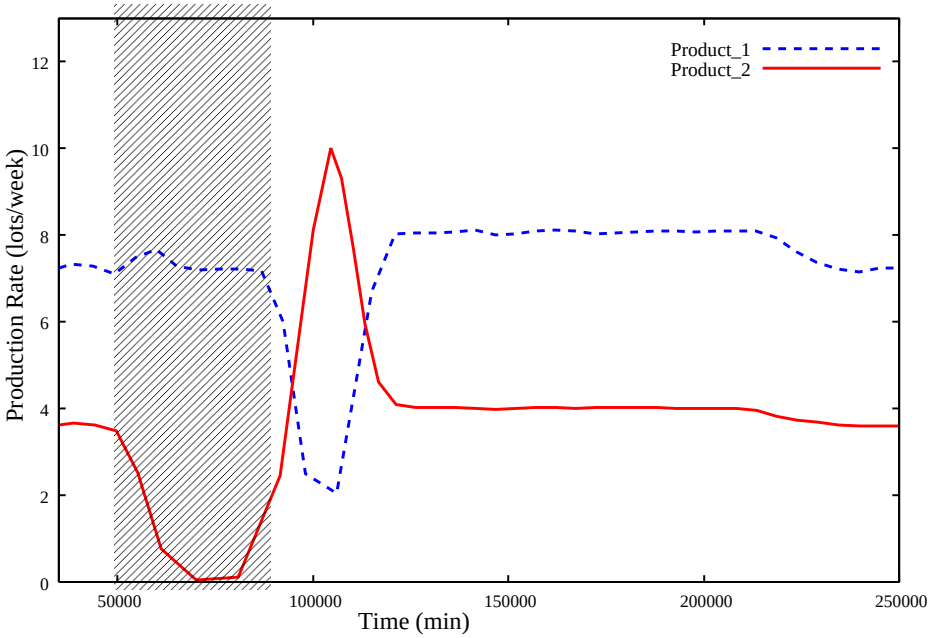

Figure 5.8: Conventional System(Buffers=100): **Throughput**

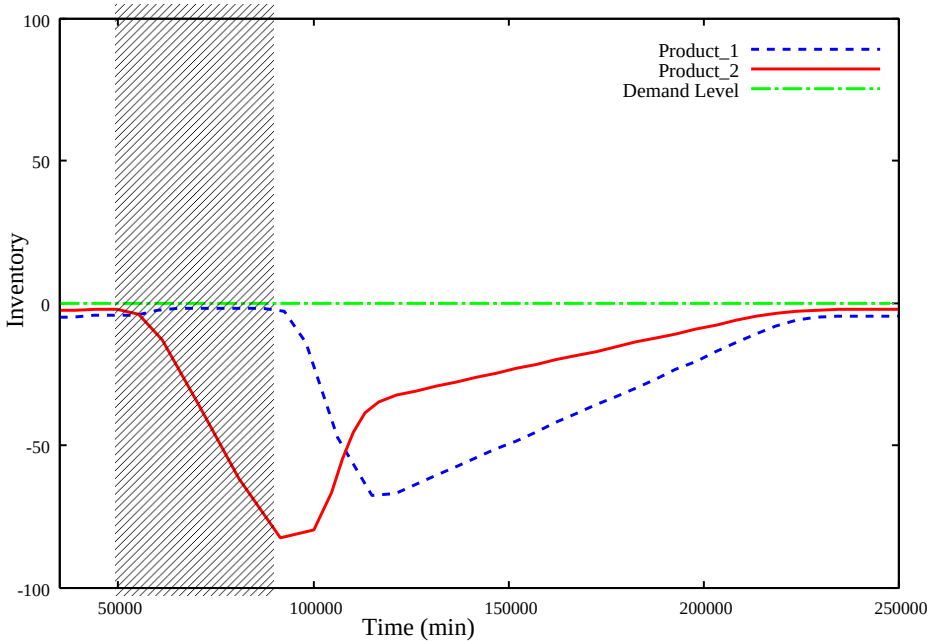

Figure 5.9: Conventional System(Buffers=100): **Finished Product Inventory**

As the failed workstation (Workstation No.19) processes only $Product_2$, $Product_2$'s throughput drops to zero during failure period of 50,000 to 90,000 min. (Figure 5.3 and 5.4). The criticality of agents which are following Workstation No.19 in $Product_2$'s process route starts to rise during failure. As some of those agents are processing both products, they start requesting both $Product_1$ and $Product_2$ at a higher rate by changing their behavior according to CABS algorithm. This behavior increases the throughput of $Product_1$ during failure and these following agents are able to compensate their loss of workload of $Product_2$ by processing $Product_1$ in excess. Among the agents that are processing only $Product_2$, the agents preceding Workstation No.19 can maintain their utilization only by maintaining their processing of $Product_2$. As $Product_2$ cannot advance beyond failed workstation and must be accumulated in network as WIP, processing by these preceding agents is limited by capacity of available buffers.

In the system having 100 WIP buffers, CABS's agents utilize them to pull and process $Product_2$ during failure (Figure 5.1). After resolution of failure,

agents process and clear this accumulated WIP to raise throughput of $Product_2$ to high level (Figure 5.1 and 5.3). By time 140000 min., finished inventory deficit of $Product_2$ and system performance returns to normal (Figure 5.5).

In the system having only 38 buffers, buffer limit of $Product_2$ is reached during failure period itself, by time 70000 min. (Figure 5.2). After saturation of buffers, agents have to stop further processing of $Product_2$, which restricts their ability to maintain their utilization. Because of the lack of accumulated WIP during failure, system can increase $Product_2$'s throughput only to a limited level after its resolution (Figure 5.4). Because of the capacity loss caused by limitation of buffers, system is slow to recover its finished inventory deficit and returns to a normal state by time 160000 min. (Figure 5.6).

Using an identical failure scenario, behavior of a conventional constant-release system (described in Section 3.2.1) using a constant releasing rule and an EDD dispatching rule is shown in Figure 5.7, 5.8 and 5.9 (in terms of system-wide WIP levels, throughput and finished inventory respectively). The buffer capacity of this system is also limited as 100. This benchmark system does not handle failures with a special care and continues production of $Product_1$ at the same demand rate during the failure (Figure 5.8). Thus, due to suspension of flow of $Product_2$, critical agents suffer a capacity loss and system takes long time to recover the production shortage incurred during failure (Figure 5.9). The failure adversely affects production of $Product_1$ as well. Since EDD dispatching rule tries to balance the deficit of both products, finished inventory of $Product_1$ also drops after resolution of failure. Comparison of Figure 5.5 and Figure 5.9 shows that recovery in this conventional system is much slower (about at time 220,000) than CABS. Also to be noted is that CABS with limited WIP also recovers earlier (Figure 5.6) than conventional system and utilizes only half the amount of buffers (Figure 5.2).

This simple example of a failure highlights the significance of buffers in a dynamic system. We also explained how the behavior of CABS is affected by capacity of buffers in system. We now show the performance of CABS in a more realistic scenario when there are continual failures on all agents. Based on the scope of buffer capacity, we show two results. The first result is generated by regulating buffer capacities at the system level. Although, in reality the buffers are associated with each individual agent/workstation, system level buffer ca-

pacity metric does not consider capacity of each individual agent. Nevertheless, the system wide WIP regulation approach is widely used in practice and studied in research e.g. CONWIP (Bonvik et al. 1997, Stevenson et al. 2005). In the second result we show behavior of CABS by regulating buffer capacity at agent level. The same network (Figure 4.1) is used in experiments and failures occur randomly on all agents with exponential distributions (as specified in Section 4.1.3).

5.1.1 Maximum System Wide Buffer Capacity

Figure 5.10 shows the result of CABS with different number of available buffers at system level. The performance metrics are same as in Section 4.1.2. A line in the graph shows the performance of CABS with same buffer capacity and different production demands. The target demands are represented by different points in line. Similar to the preceding example of a single failure, these results also show that availability of buffers assists in maintaining utilization of critical agents during failures. For the higher demand levels, CABS is able to achieve better throughput if more buffers are available in system. For the lower demands, performance is not affected much by the buffer capacity. As agents have lower workload requirements in case of lower demands, even if they stop processing due to shortage of buffers, their resulting workload deficit remains low. In the lower demand region, agents have more spare capacity due to their lower utilization levels and hence they can quickly recover their workload deficits incurred during a failure.

For comparison, we have also generated the results of CONWIP (described in Section 4.4) with identical buffer capacities. The behavior of CONWIP is independent of demand rates, and the WIP level it maintains is equal to system's buffer capacity. For a given buffer capacity, please note that CONWIP achieves marginally less throughput than corresponding CABS (with highest demand). This is despite the fact that CONWIP maintains the maximum level of WIP (equal to the capacity) throughout its execution. CABS on the other hand autonomously manipulates its WIP level during execution, and its lower lead times from corresponding CONWIP means that CABS's management of WIP is better.

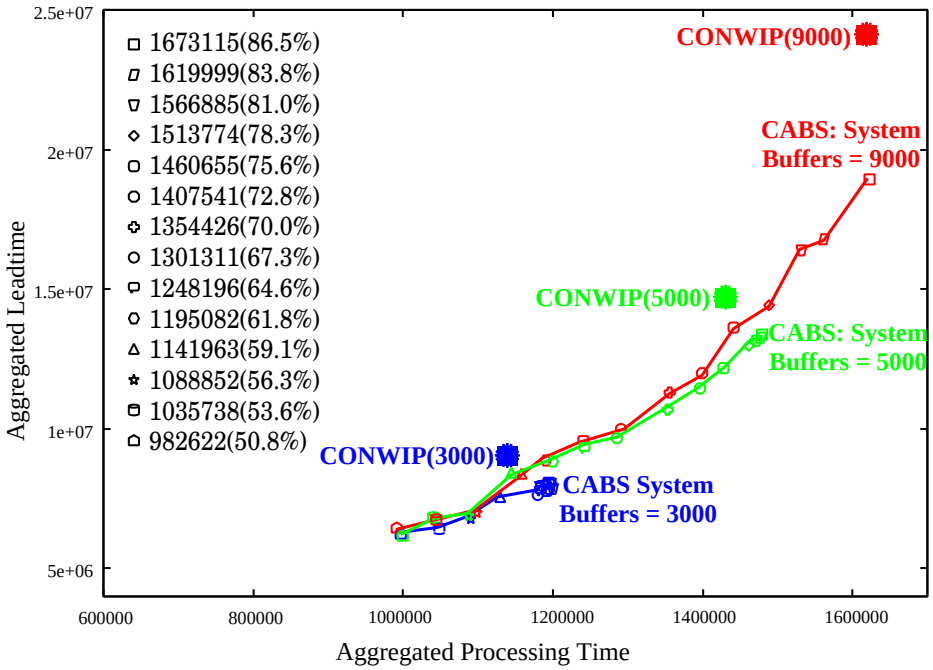

Figure 5.10: CABS with different system-wide buffer capacities

5.1.2 Issue of Deadlocks

Unlike many other network systems, semiconductor fabrication processes are characterized by existence of large number of re-entrant cycles in their process routes (Figure 4.1). 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 low 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 introduced a simple mechanism of reserved buffers to avoid permanent deadlocks in system. We explain our proposed mechanism of reserved buffers by using an example. Figure 5.11 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 *AGENT2*, 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 *Kanban* mechanism (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. The details of implemented kanban mechanism are provided in Appendix A.

In Figure 5.11 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 5.11 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.

Figure 5.12 shows the behavior of CABS (having specific buffers) during the same scenario. As *AGENT1* is processing two product steps (*stepP* and *stepR*), it has a specific buffer for each of them and we assume both of them are also

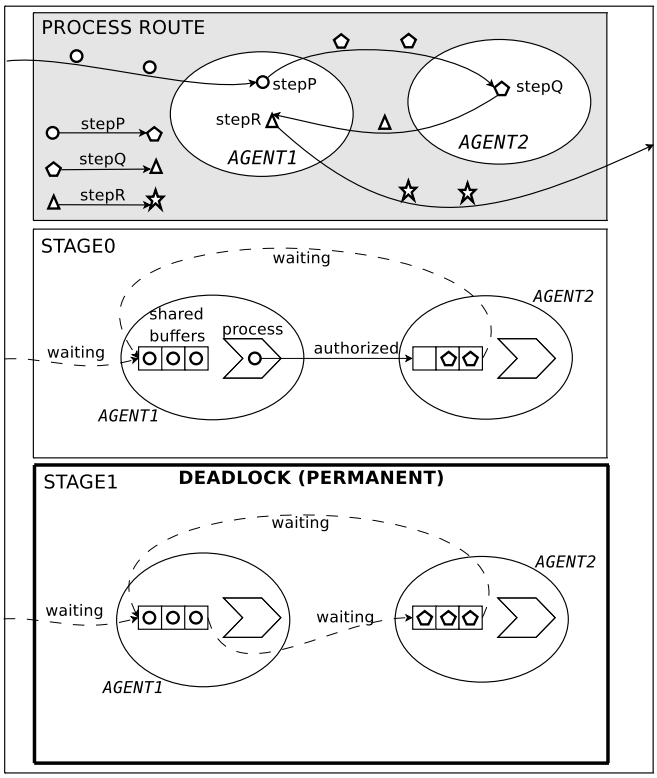

Figure 5.11: Deadlock: without specific buffers

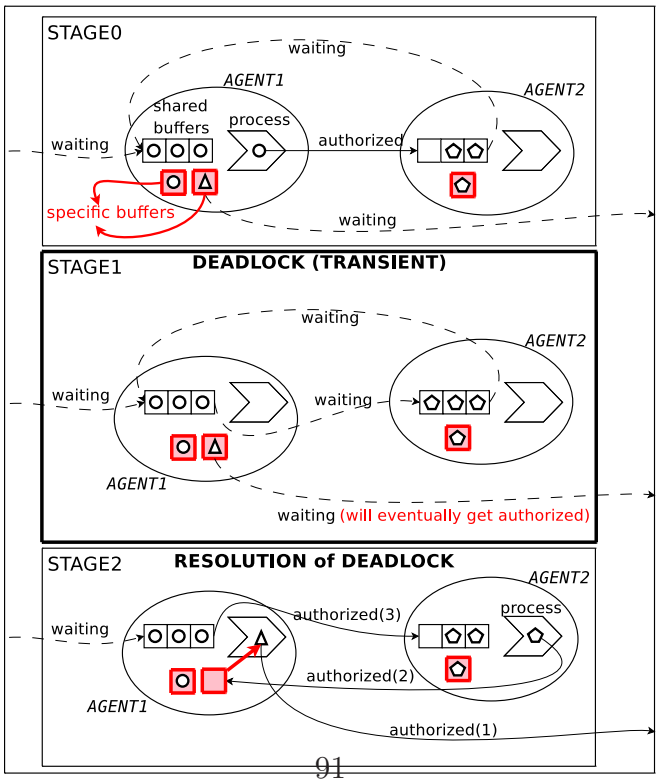

Figure 5.12: Deadlock: with specific buffers

occupied as shown in *STAGE0* of Figure 5.12. A deadlock occurs in this scenario also when the buffers of both agents become full in *STAGE1*. The provision of exclusive buffers guarantees that for each type of job that an agent processes, it will either have a job of that type or will be able to authorize the preceding agent to receive a job of that type. In our example *AGENT1* is already having a job of *stepR* in its specific buffer when the deadlock occurs (*STAGE1* of Figure 5.12). Due to availability of both jobs (*stepP* and *stepR*), *AGENT1* has the choice of processing either of them and hence waits for authorization from two agents, from *AGENT2* for *stepP* and from the corresponding succeeding agent for *stepR*. As all agents of system have such specific buffers, the agent at the end of process route will also have a specific buffer reserved for last process step of the product and hence will eventually free its buffer when it finishes a job. As the intermediate agents also process their jobs similarly and authorize their preceding agents for further processing, *AGENT1* in our example will also eventually get authorized to process its job of *stepR*. The specific buffer for *stepR* becomes free as *AGENT1* it picks *stepR* for processing after receiving authorization (*authorization(1)*) for it (*STAGE2* of Figure 5.12). The newly freed specific buffer authorizes (*authorization(2)*) *AGENT2* to process another job, which will eventually authorize (*authorization(3)*) *AGENT1* for processing its accumulated job of *stepP*, hence resolving the deadlock. As explained, the approach of using specific buffers avoids the occurrence of permanent deadlocks and hence alleviating the need of external intervention which is undesirable in autonomous systems. Agents decide on their dispatching according to the buffer status of succeeding agents and the implementation details of mechanism are provided in Appendix C.

5.1.3 Individual Agents' Buffer Capacity

By maintaining the specific buffers as described above, we investigate how the size of shared buffers has effects on performance of CABS in our semiconductor manufacturing problem. Figure 5.13 shows performances of CABS in semiconductor manufacturing problem of Section 4.1 with different shared buffer sizes. Each line shows the performance of CABS with a certain shared buffer size for various demand rates. In experiment, each agent of CABS has the same size

of shared buffer as depicted in a graph. Similar to system-wide buffer limitations of previous results, from the graph it is clear that performance of CABS degrades with reduction of buffer size. In the hypothetical system-wide buffers used in previous result, there is no restriction on each agent's buffer capacity and any single agent can use all the available system buffers. On the contrary, buffer capacity limitation of each agent is more restrictive as buffers of many agents remain under-utilized due to their immobility.

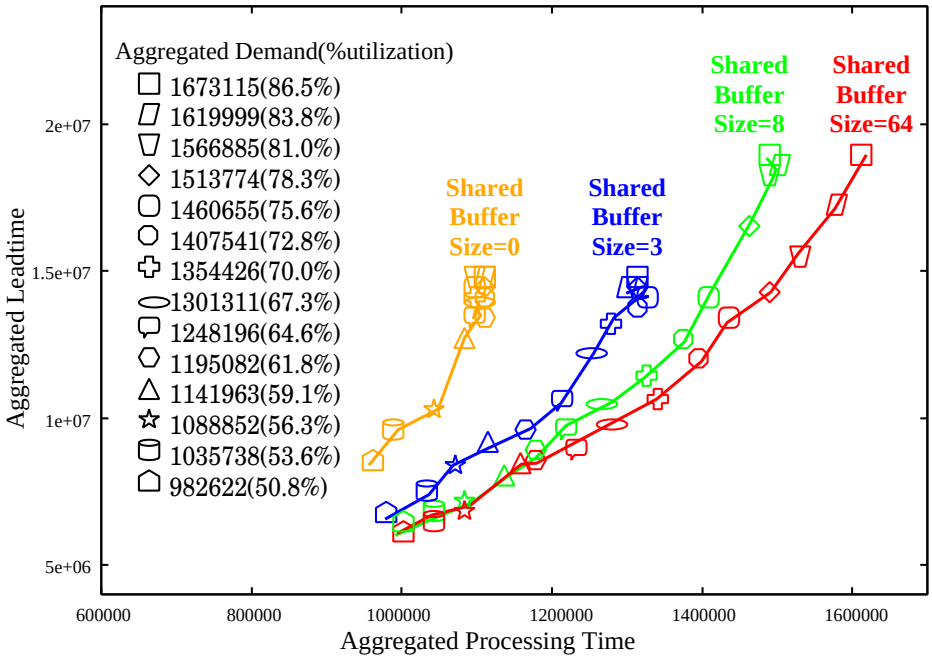

Figure 5.13: CABS with different buffer capacities of agents

5.2 Effect of Limitation on Early Release

In this section we evaluate the performance of CABS by restricting its flexibility of releasing jobs in system. In a leveled, under-loaded and stable system, there is no need for accumulated WIP and hence the jobs do not have to wait for processing at the agents. In order to meet the due-date or deadline (t_{due}) of a job in such condition, the job should be released in system at latest in the timing of ($t_{due} - processtime$), where $processtime$ is the sum of processing time of all its

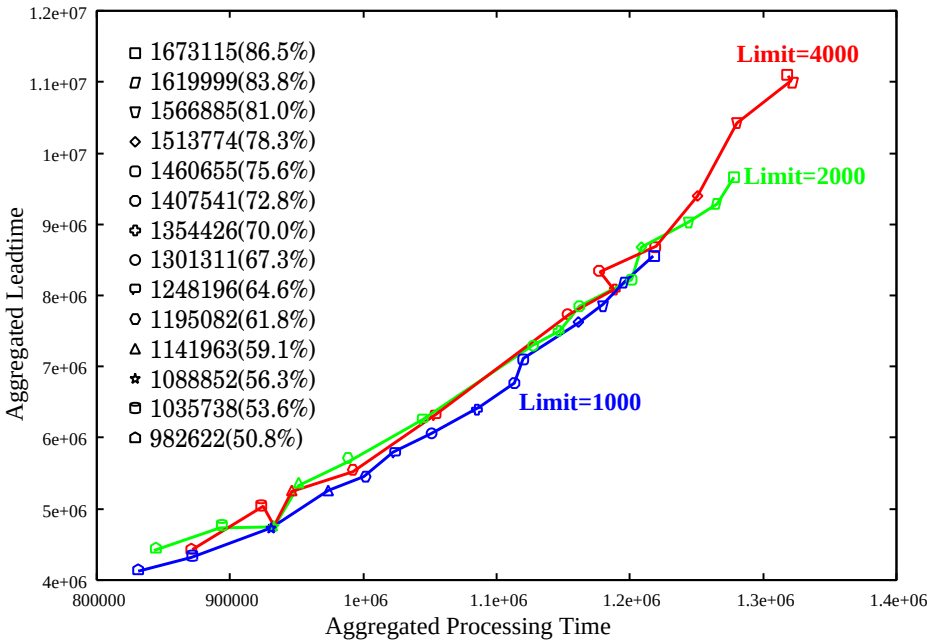

Figure 5.14: CABS with limitations on release flexibility

steps. During failures, when flows of some jobs are reduced below their demand levels, CABS pulls alternative jobs before the timing of their $t_{due} - processtime$ to maintain agents' utilization. When *Source* agent releases more and more jobs before their stipulated release time of $t_{due} - processtime$ to expedite their processing, the offset between stipulated and actual release time $t_{release}$ increases for subsequent jobs. This offset, $((t_{due} - processtime) - t_{release})$ thus indicates the degree of flexibility of job release. We have conducted experiments by putting limits on this flexibility offset. The limit is enforced by each *Source* agent, which releases a new job only when its offset is less than the specified limit.

Figure 5.14 shows the result of CABS with different limits on flexibility offset. The experimental setup is same as in Section 4.1. Each line represents the results of CABS for different demand levels with corresponding *Limit* (in mins.) of flexibility offset. The graph shows that throughput of CABS decreases with reduction in flexibility offset (*Limit*). As the number of jobs that CABS can pull before their stipulated release time is restricted, the possible capacity loss of critical agents increases during failures. This increased capacity loss results in reduced system throughput, and the effect is severer in case of high

demands.

5.3 Conclusion

In this chapter we presented the results of experiments when there are restrictions on buffer capacity and entry of jobs in system. The buffers help in maintaining throughput of resources during disruptions in system. However the number of buffers in any system is limited and results presented in this chapter show how the performance of CABS is affected by buffer capacity. Due to the limit of buffers, such flow based systems are prone to bottlenecks that can stall the operations of system. Resolving permanent deadlock may require external intervention which is undesired in an autonomous system. In this chapter we explained the mechanism of deadlock avoidance which utilizes reserved buffers and in is integrated with CABS. CABS maintains the utilization of bottleneck workstations by increasing flow of alternative jobs when flow of some jobs is delayed due to disruptions. The flow of jobs is limited by the entry of jobs in system and unlike manufacturing systems, jobs cannot be released freely in other systems. We showed how the performance of CABS is affected when release (entry) of jobs in system is restricted. The coordination of CABS is achieved by requirement messages and agents utilize the contents of messages to decide their behavior. In next chapter we analyze the importance of each message parameter in detail.

Chapter 6

Significance of Message Parameters

In the previous chapter we showed influences of several features of network system on performance of CABS. Requirement messages are backbone of CABS coordination mechanism and in this chapter we investigate significance of different message components. In following sections we analyze the significance of each individual message parameter, i.e. `time limit`, `request rate`, `amount` and `criticality` by restricting its strength and scope in system.

6.1 Significance of `time limit` parameter

The `time limit` parameter of a CABS's requirement message specifies the time by which the requesting agent needs another lot of that type of job. When the system is running smoothly, `time limit` parameter remains in accordance with the demand of jobs. When the workload deficit and criticality of an agent rises due to reduced flows of some jobs, agent starts to request its other jobs more urgently. In order to receive more alternative jobs, the agent it changes its behavior and increases the `request rate` parameter and reduces `time limit` in its requirement messages (Algorithm 4:8). As affected agent starts sending lower `time limit` in its messages, offset between stipulated arrival time and `time limit` of the requested job increases with subsequent jobs. As this flexibility of agents to reduce `time limit` in their messages enables them to receive jobs

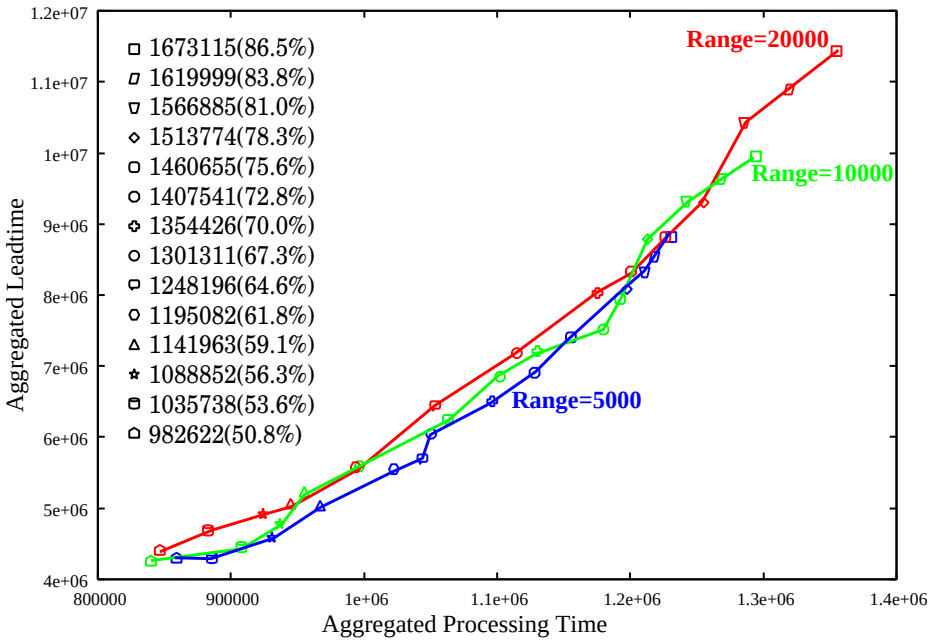

Figure 6.1: CABS with limitations on `time limit`

earlier and recover their deficit, we have conducted experiments by limiting this flexibility.

Figure 6.1 shows performance of CABS with different limits on the time by which an agent can advance `time limit` from the original stipulated arrival time of requested job. The *Range* in the graph shows the amount of time (in mins.) by which the `time limit` for a requested job can be advanced from its original arrival time. The graph shows that in the higher demand region, the throughput achieved by CABS decreases with reduction in flexibility of `time limit` parameter. As `time limit` is the primary parameter that informs preceding agents about urgent requirement of an agent, restriction on *Range* delays the desired flow of jobs at a higher rate. As job of a higher demand is delayed from agent's desired time, agent suffers capacity loss during that delay. Since agents have smaller spare capacities when demands are high, loss of capacity due to restriction on `time limit` thus affects more severely in the region of higher demands.

6.2 Significance of `request rate` parameter

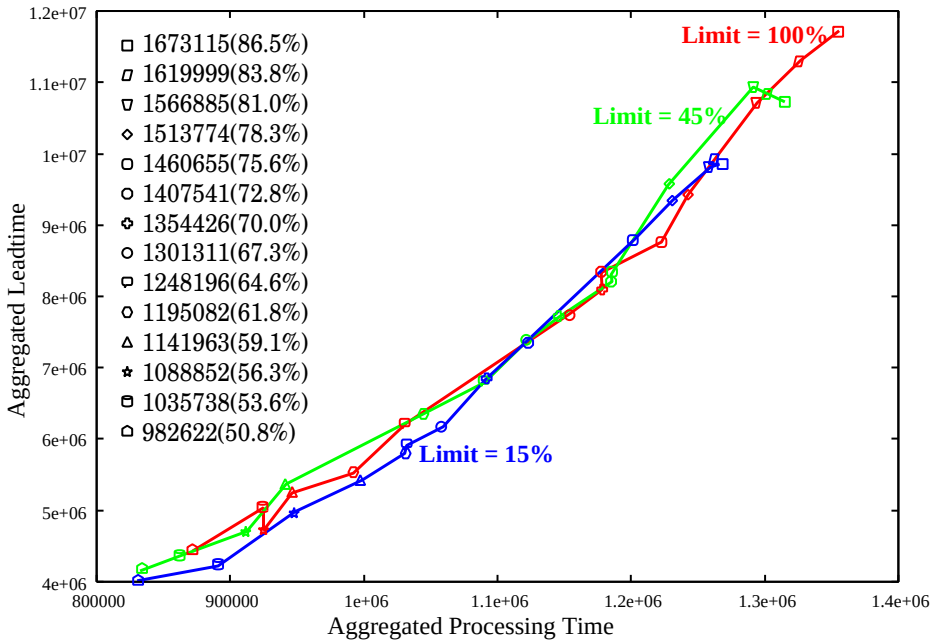

Figure 6.2: CABS with limitations on `request rate`

The `time limit` parameter of a requirement message specifies the time by which requesting agent can process another job of that type. Starting at time `time limit`, the rate at which succeeding agent can process subsequent jobs of that type is specified by `request rate` parameter. During smooth operation of system, `request rate` which is propagated after adjustment by all agents because of their low criticality (Algorithm 4:13). When an agent becomes critical, it changes its behavior and starts requesting jobs at the maximum rate at which it can process them (Algorithm 4:9). This maximum threshold rate at which an agent can process jobs depends on the processing time of job. We have conducted experiments by limiting the flexibility of agents to raise their `request rate`.

Figure 6.2 shows the performance of CABS with different limits on agents' flexibility to increase `request rate` parameter of their messages. The *Limit* in graph specifies the fraction of agent's threshold rate to which an agent can increase `request rate` from corresponding *demand rate* of jobs. The line with

Limit=100% thus shows the behavior of CABS when agents have complete freedom i.e. when they become critical, they can send their threshold rate as `request rate`. The graph shows that in high demand region, CABS achieves lower throughput when `request rate` is restricted. The rate requirement of critical agents is propagated unaltered by their preceding agents (Algorithm 3:13 and 4:13), subjected only to the capacity of preceding agents. The preceding agents use their incoming `request rate` to maintain an appropriate flow to critical agent, till the critical agent's workload deficit is recovered completely. The restriction enforced by *Limit* in experiments reduces strength of flow towards the critical agent, which increases its capacity loss and slows the rate at which its workload deficit is recovered. Since agents have smaller spare capacities when demands are high, restriction on `request rate` affects more severely in region of higher demands.

6.3 Significance of `amount` parameter

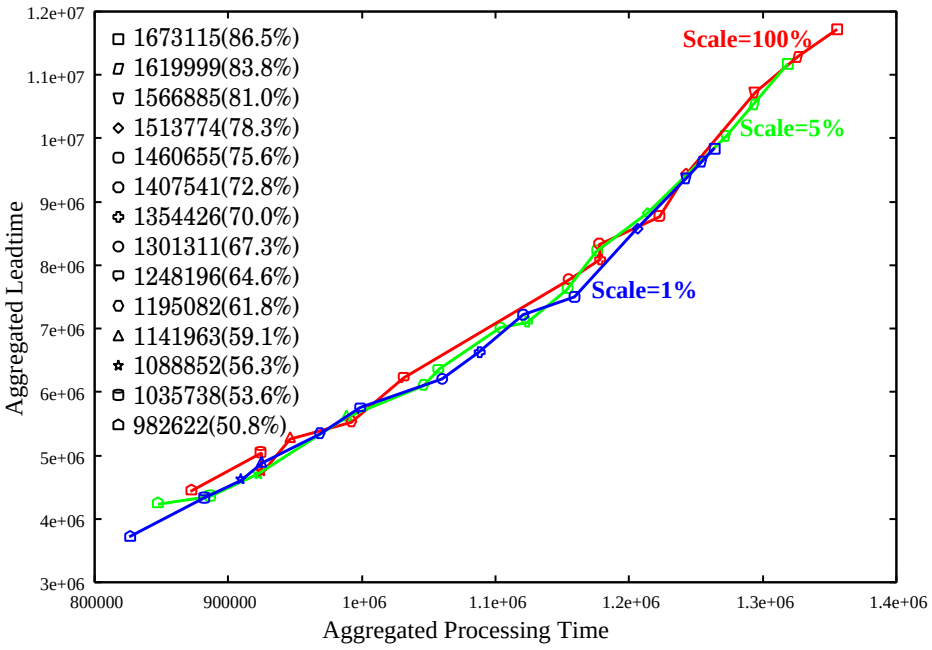

Figure 6.3: CABS with limitations on `amount`

Starting at `time limit`, the number of lots that are required urgently at `request rate` by an agent is specified by `amount` parameter in requirement message. As agents have no urgency of jobs during system's smooth operation, the `amount` that they send in their requirement messages is 0. As preceding agents are also not critical during smooth operations, they propagate this 0 `amount` requirement (Algorithm 3:7). The `amount` parameter is coupled with `request rate` parameter and agent changes both of them simultaneously when it becomes critical (Algorithm 4:9,10). In order to recover their workload deficit during critical phase, agents request some number of jobs at their threshold rate (Algorithm 4:9). The `amount` an agent requests at its maximum rate is limited and depends on agent's workload deficit (Algorithm 4:10). We have conducted experiments by scaling down the `amount` that agents request at higher rate during their critical phase.

Figure 6.3 shows the performance of CABS with different reduction factors on the `amount` parameter of requirement messages. While requesting the jobs at higher rate during criticality, the agents scale down their actual amount requirement by a factor of *Scale* before sending it as the `amount` in requirement messages. The line *Scale*=100% thus shows the behavior of CABS where agents are sending their real amount requirements in the `amount` parameter. In line *Scale*=5% the critical agents send only 5% of their actual amount requirements in `amount` parameter of their requirement messages. The graph shows that throughput achieved by CABS decreases as `amount` parameter is reduced from the original amount requirement. As the number of lots that preceding agents expedite to critical agents is limited by `amount` parameter, its reduction from original amount requirement results in increased capacity loss of critical agents.

The preceding agents propagate `amount` requirement of critical agents after successively reducing it according to their local WIP (Algorithm 3:10). After `amount` number of lots are made available for critical agent, remaining preceding agents revert to normal behavior of requesting jobs according to demand (Algorithm 3:5-8). A large `amount` requirement of critical agent means that its requirements are propagated to more upstream agents, and this transition to normal requirements takes place farther from the critical agent. Due to successive and lengthy failures in our experiments, the workload deficit and amount requirements of critical agents always remains high. The high amount

requirement of critical agents is not met by preceding agents within the system and requirements are thus propagated till *Source* agents. By reducing actual amount requirement of critical agents by a small *Scale* does not cause the reduced `amount` to be met by preceding agents within system, and thus behavior of all agents remains same. The *Scale* has to be reduced to a very low level (1%-5%) so that the reduced `amount` can be met by preceding agents within system and the effect of change in agents' behavior can be visualized by changing *Scale* parameter.

6.4 Significance of `criticality` parameter

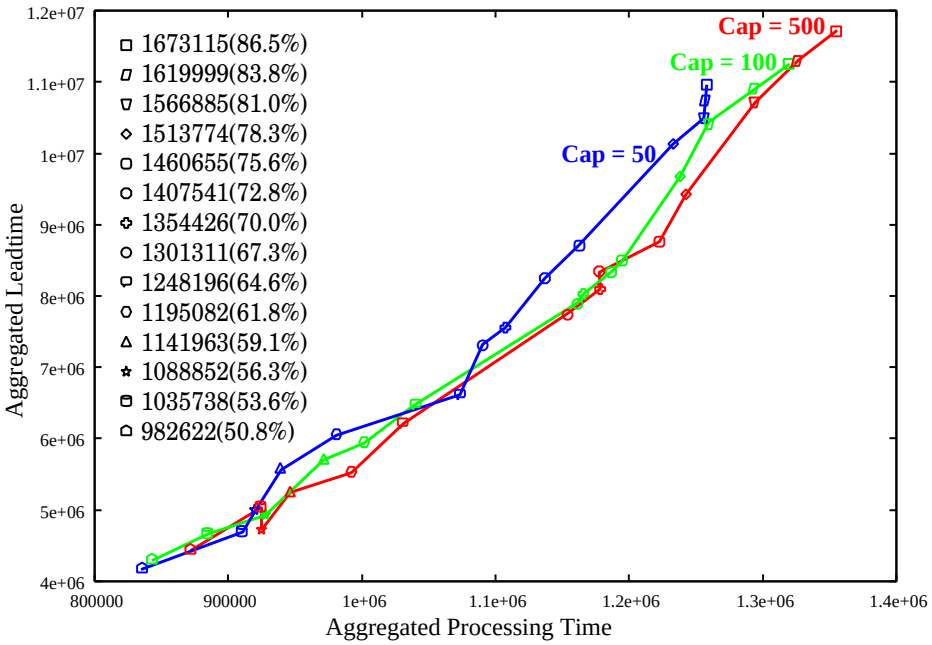

Figure 6.4: CABS with limitations on `criticality`

The `criticality` parameter of requirement message specifies the criticality of agent from which requirement has originated. In a smooth running system, criticality of all agents remains zero and all agents are sending zero `criticality` in their requirement messages. When an agent becomes critical due to reduced flows, it starts sending higher requirements for all its jobs with an appropriate

higher criticality (Algorithm 4:11). We have conducted experiments by truncating agent's criticality, i.e. value of `criticality` that agents send in their requirement messages.

Figure 6.4 shows the performance of CABS with different limits on `criticality` parameter of requirement messages. Irrespective of actual criticality of an agent, *Cap* specifies maximum value that it can send as `criticality` in its messages. *Cap* is same for all the agents in system. The graph shows that throughput achieved by CABS decreases as *Cap* is reduced. Agents use this `criticality` parameter to identify the location of current bottleneck (Algorithm 4:7), and accordingly perform their dispatching and messaging to ensure desired flow of jobs to that most critical agent (Algorithm 1 and Algorithm 4:13). By truncating the criticality of agents, preceding agents fail to identify the correct bottleneck among agents that have criticality higher than *Cap*. As preceding agents fail to identify and ensure desired flow to real bottleneck, the bottleneck suffers capacity loss. By reducing *Cap* further, many other less-critical agents are treated at par with the real bottleneck, causing further degradation of CABS performance.

6.5 Conclusion

Agents in a CABS system coordinate their actions through requirement messages which contain four parameters: `time limit`, `request rate`, `amount` and `criticality`. Agents periodically send and receive requirement messages and use messages' information to adapt to changes in system. Agents use incoming requirement messages parameters to identify the dynamic bottlenecks and their requirements. Based on the incoming requirement messages and their own status, agent calculate parameters of requirement messages to other agents. In this chapter we analyzed the significance of each individual message parameter by restricting its value. By restricting the values of parameters, accuracy and scope of information in system is reduced. The results show that all parameters of messages play significant role in the coordination of CABS and restricting their values adversely affects the performance of CABS which degrades with the level of restriction on parameters.

Chapter 7

Conclusions

Because of the ever increasing competition and changing market scenarios, manufacturing systems are under increased pressure to improve their performance while simultaneously decreasing the operational costs. Semiconductor fabrication is carried on highly capital intensive infrastructure and manufacturing takes place in a highly dynamic environment. Besides catering to the dynamic demand from customers, the semiconductor fabrication is required to consistently perform with high efficiency in the face of frequent disruptions within the system. In order to remain competitive, the manufacturing control system is required to maintain high utilization of costly equipment besides achieving smaller leadtimes at the same time. To achieve high throughput with smaller leadtimes is a challenging task in large failure-prone networks, and it is an open ended goal to improve the tradeoff between throughput and leadtime in not only manufacturing systems but other similar systems as well. It is a challenge to develop flexible and adaptive manufacturing control systems that can achieve the desired system performance in the face of dynamic goals and frequent disruptions.

The multiagent coordination mechanism described in this thesis is proposed to provide an autonomous manufacturing control system that can improve system performance by adapting to unexpected random failures in system.

In the proposed system, CABS, the manufacturing system is modeled as network of agents where each workstation is considered as an agent. In CABS, the actions of agents are coordinated through a message-passing mechanism.

Agents pull jobs from their preceding agents by sending requirements of jobs in messages. Along with the priority of requirement, messages contain time, rate and number of jobs required. In normal conditions, agents send requirements to pull jobs according to the demand. Agents monitor starvation of resources and change requirements to pull alternative jobs at a higher rate to avoid their starvation. Agents process (dispatch) their buffered jobs according to incoming requirements that they receive from their succeeding agents. Depending on resource's capacity and magnitude of starvation, agent's job requirements are assigned a criticality which is utilized to prioritize the possibly conflicting requests. In order to maintain the flow of jobs according to incoming requirements, agents propagate requirements to their preceding agent by relaxing them according to their WIP. The proposed architecture highlights the first contribution of our research:

Decentralized Manufacturing Control Architecture: The agents in CABS are autonomous and there are no hierarchies involved in the structure. As a result, the resulting architecture is modular, scalable and flexible. In CABS, agents send messages to only the neighboring agents from which they receive jobs due to which communication and interaction requirements of proposed coordination mechanism are strictly limited by the underlying manufacturing process. Irrespective of the complexity, topolgy and size of manufacturing system, agents have knowledge of only their neighboring agents. As a result, the architecture is highly flexible as manufacturing system can be easily modified by adding, removing or reconfiguring workstations (agents) just updating information of only the directly connected agents.

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 WIP in 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 trade-off between throughput and leadtime than a conventional manufacturing control method CONWIP. These results highlight another contribution of our research:

Global Optimization in Decentralized Control: Compared to tradi-

tional centralized control architectures, distributed control through multiagent architecture although provides the desired adaptability and robustness to disruptions, achieving coherent global behavior of from a large team of autonomous decision making entities is still an open issue. Success of a multiagent system depends on effectively calibrating the sub-goals of individual agents such that distributed agents' autonomous behavior of achieving their local goals results in desired global behavior. The local goals in CABS are autonomously calculated in a distributed manner and agents coordinate their actions of dispatching and message passing to achieve their local goals. By using coordination mechanism of CABS, the autonomous decisions made by distributed agents according their locally identified goals are coherent and resulting emergent behavior achieves desired global goals. Agents make their decisions based on local goals which are in line with the global goal of (a) achieving desired throughput and simultaneously (b) minimizing the leadtime:

- *Maximizing Throughput*: Achieving the goal of desired throughput is directly related to the goal of avoiding bottleneck's starvation. Each agent in CABS independently monitors its own starvation and calculates criticality that depends on the magnitude of starvation. Starving agent parametrize their locally identified goal of avoiding starvation in terms of job requirements and send them to preceding agents. Other agents identify dynamic bottlenecks of system by comparing the criticalities of different agents which are propagated to them through requirement messages. Agents take actions to meet the goal of avoiding starvation of bottlenecks by (a) dispatching jobs to meet requirements of identified bottlenecks and (b) propagating requirements of identified bottlenecks to other agents so that required flow of jobs towards bottleneck can be sustained. Due to the constant monitoring and efficient message passing, agents can promptly identify dynamically emerging bottlenecks. Relevant agents coordinate their dispatching actions to meet requirements of identified bottlenecks, which results in minimizing bottleneck's starvation and achieving higher system throughput.

- *Minimizing Leadtime*: Leadtime is directly proportional to the WIP in system and goal of minimizing leadtime can be achieved by avoiding

unnecessary WIP in system. In CABS, WIP is regulated at each agent through its mechanism of calculating job requirements. Before propagating the job requirements, agents in CABS adjust them according to available WIP and request subsequent job at a time when their current WIP will be consumed. In CABS, accumulation of WIP happens only when it cannot be avoided in the situations where agents have to pull surplus jobs to meet the requirements of bottleneck for avoiding its starvation. By regulating WIP at each agent and by limiting WIP accumulation only to avoid the starvation of bottlenecks, the amount and duration of surplus WIP is reduced which results in lower leadtimes.

In many traditional manufacturing control mechanisms, the performance of system is analyzed discretely, either periodically or in response to some event. Based on system status, the corrective measures are planned and evaluated offline and applied to system during subsequent execution. Such analysis and evaluation typically requires a human operator to interpret the data and simulation results. This highlights another contribution of our research:

Autonomous Control: In our results we have shown that for semiconductor fabrication process, CABS outperforms CONWIP which is a popular control mechanism and is widely used in practice. In practice, the optimal WIP levels of CONWIP are decided in advance by simulations and are updated continually according to changing status of system. CABS on the other hand works autonomously without external intervention and performs better than CONWIP. Besides performance, another achievement of CABS is an integrated deadlock avoidance mechanism. Networks, such as production systems are prone to deadlocks and permanent deadlocks can bring the system to a halt. Permanent deadlocks cannot be resolved without external intervention which is against the principles of autonomous control. CABS has an integrated mechanism of deadlock avoidance which is achieved in a distributed manner through an additional parameter of requirement messages. The review of past research in multiagent domain shows that effective autonomous control for large systems such as semiconductor fabrication is difficult to achieve. Nevertheless, results of our research emphasize the advantage of using autonomous multiagent architecture in which distributed agents can improve the overall system performance by promptly reacting to locally identified disturbances and changes.

We have experimented and presented results of CABS with variations of different parameters, which highlights another contribution of our research:

Practical Feasibility: We have showed that CABS gracefully handles restrictions on availability of resources and flexibility of target system. Buffers in a real system are limited and generally coupled with resources. The performance of CABS degrades logically as the number of available buffers is reduced. In some situations, it might be required to quickly pull jobs within the system to avoid starvation of critical resources. To maintain adequate flow of jobs to bottlenecks, CABS utilizes pull mechanism to release new jobs into system. However, releasing jobs in system before their stipulated time may not be desirable or feasible due to various practical reasons such as dependency on other system or procurement of raw material. We have evaluated the performance of CABS by restricting flexibility of job release, and performance of CABS logically degrades with available flexibility. These two sets of experiments show that CABS can gracefully handle different practical considerations.

Discussion About Applicability to Other Systems: In this paper we have focused on manufacturing process control and conducted experiments using the semiconductor manufacturing problem, which is the most complex problem in its domain. Although we have empirically evaluated CABS by using only the semiconductor fabrication datasets, we believe that coordination mechanism of CABS is also suitable for other complex and unstable network systems such as road transportation and communication networks. Like semiconductor manufacturing, other network systems also have the primary requirement of achieving high throughput in order to maximize the return on investment (RoI). Like manufacturing problem, they also have a similar constraint of simultaneously maintaining a high quality of service (QoS), which is related to service or waiting time in system. The semiconductor fabrication processes, like the one used in this paper, have fewer nodes (agents) compared to the large scale communication or transportation networks (e.g. Internet or state highways). Even though the size of semiconductor fabrication processes is smaller, the complexity of problem is high because of complex process routes of products which overlap and have numerous cycles in them.

Although agents inside a network system can freely control the processing and movement of jobs after they have entered in network, control over the

time when a job enters in system may depend on type of network/job under consideration. For example, unlike the passive jobs of manufacturing domain, commuters of road network or data packets of live media cannot be pushed in system before their scheduled time. For completeness, in this thesis we have presented additional results of CABS by limiting release flexibility in semiconductor fabrication process.

Chapter 8

Future Work

As coordination in distributed systems is achieved via the exchange of messages, analysis of messaging requirements is significant for evaluation of coordination algorithms. Messaging requirements of underlying coordination protocol significantly affect the scalability of a mechanism. Frequent messaging not only requires high communication bandwidth, it also puts a demand on the processing capacity of agents which is required to process incoming messages. Nevertheless, exchange of information messages between agents is the key for successful coordination. Although we have shown the significance of individual message contents in this paper, we have not formally reported the complexity of algorithm in terms of number of messages. As a future work we want to evaluate the message complexity of CABS. In data networks, coordination messages share the same communication channel used for data transfer and it is desired that communication overheads of coordination algorithm are minimum. Depending to the size of networks, communication bandwidth required for coordination can become a bottleneck in networks of other domains as well. In the current implementation of CABS, communication bandwidth is assumed unlimited and agents send new messages as and when there is a change in their requirements. The bandwidth that CABS's coordination mechanism utilizes can be reduced by regulating the number and frequency of requirement messages. Restrictions on messaging will reduce the accuracy of information and responsiveness of agents, and as another future work we want to analyze how system behavior is affected by availability of communication bandwidth.

Analysis of past research in coordination algorithms shows that algorithms which involve all agents in coordination cannot scale up to handle large networks due to prohibitive processing and/or messaging requirements. The algorithms that coordinate only with neighboring agents can be inefficient due to myopic vision of their agents i.e. lack of remote but significant information. In order to have a scalable coordination mechanism that is efficient and can still handle large agent networks, it is important that local coordination in a large system involves only the necessary and sufficient agents. The agents that are relevant for local coordination many also change over time due to unexpected events and dynamism in system. As the job requirement of a critical agent has to be met by its preceding agents, propagation of criticality in CABS messages involves the necessary agents in coordination to change their behavior. As critical requirements are reverted to demand level after their successive reduction through sufficient agents, the number of agents that receive critical requirements and change their behavior is limited by the current criticality of dynamic bottlenecks. This property of dynamically controlling the scope of coordination is very relevant to develop efficient and scalable coordination algorithms, and in future we want to formalize and study this feature in greater detail.

CABS derives its advantage by expediting the flow of alternative jobs to bottleneck when some jobs are delayed due to disruption at preceding agents. In semiconductor fabrication processes, most agents are processing multiple kind of jobs. The different kind of jobs can be of different products, and/or numerous cycles of same product. Because of complex and dense process routes, agents, especially the critical agents are connected to a large number of other agents from which they receives jobs. As high demand and long leadtimes of products necessitates high WIP, a large number of variety of jobs are always present in the system. Due to high degree of connectivity and regular availability of jobs at neighboring agents, CABS is successfully able to avoid starvation of an agent by expediting jobs to it from alternative neighboring agents when flow from some neighboring agents is delayed due to disruption. The behavior of CABS is purely reactive, and alternative jobs are expedited towards the starving agent only when the starvation starts to occur. However, the purely reactive mechanism may not be effective when starving agent receives its jobs from a single or very few other agents. Due to lack of alternative neighbors from which jobs

can be pulled, CABS's reactive behavior of expediting alternative jobs only after occurance of starvation cannot avoid starvation of poorly connected agents. Starvation of such poorly connected agents can be avoided only by proactively maintaining a safety buffer of surplus WIP before them, notwithstanding the fact that surplus WIP will increase leadtime. In order to maintain its utilization, the surplus WIP will be processed by starving agent when incoming flow of jobs is disrupted and alternative jobs are unavailable either due to (a) lack of neighbors or (b) neighboring agents cannot supply ample jobs in time. We have shown in our research that centralized methods of identifying adequate safety WIP levels, such as CONWIP are inefficient. The agents in CABS monitor starvation in a distributed manner, and as another future work we would like to add proactivity to CABS through which individual agents can assess and minimize the risk of starvation by dynamically maintaining adequate WIP levels.

In current version of CABS, the flow of messages is in a single direction, i.e. opposite to the flow of jobs. As a result, to meet the incoming requirements of bottleneck, agents can utilize information of only their local WIP to decide on their dispatching actions. However, when an agent doesn't have any job that is required by bottleneck, it dispatches whatever other WIP is available to it. This behavior of preceding agent might starve the bottleneck. At times, a job which is required by bottleneck might be arriving soon at a preceding agent but because the preceding agent has committed its resource to some other job, processing of the important job at preceding agent and its subsequent arrival at bottleneck will be delayed. By having an additional flow of information along the flow of jobs, agents can know about forthcoming jobs in advance and hence plan their dispatching actions more effectively. As another future work, we would like to incorporate this additional messaging and evaluate its effectiveness.

During starvation, in order to compensate for delay of some kind of jobs, agent increases the requirement of all its jobs. Consequently, preceding agents of all kind of jobs simultaneously increase their dispatching rate and intake of jobs to meet the new, higher requirements. To avoid its starvation, bottleneck agent increases the requirements of each individual flow to utilize its full capacity. If increased to the required level, starvation of bottleneck can be avoided by increasing flow of a single alternative job. As bottleneck cannot process incoming jobs at the rate which it has requested them from its preceding agents, this

results in unproductive increase of WIP at preceding agents. The unproductive WIP can be avoided if different preceding agents of various fows coordinate their actions to expedite different jobs only enough to ensure that the cumulative flow of jobs utilizes bottlenecks capacity and avoids its starvation. This optimization requires coordination and information exchange between remote agents of different independent flows, and as another future work we will like to explore this feature.

In the research and experiments of semiconductor fabrication that we have presented in this thesis, routing, batches and variation of demand is not considered. In order to enhance the applicability of proposed research, we would like to incorporate these parameters in our experiments in future.

Bibliography

Aarts, E. H. L. & Korst, J. (1989), *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, Wiley-Interscience Series in Discrete Mathematics, John Wiley and Sons, Chichester.

Abumaizar, R. J. & Svestka, J. A. (1997), 'Rescheduling job shops under random disruptions', *International Journal of Production Research* **35**, 2065–2082.
URL: *http://www.ingentaconnect.com/content/tandf/tprs/1997/00000035/00000007/art00017*

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

Atherton, L. & Atherton, R. (1995), *Wafer Fabrication: Factory Performance and Analysis*, Kluwer Academic Publishers.

Babiceanu, R. & Chen, F. (2006), 'Development and applications of holonic manufacturing systems: A survey', *Journal of Intelligent Manufacturing* **17**, 111–131.
URL: *http://www.ingentaconnect.com/content/klu/jims/2006/00000017/00000001/00005516*

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

Bean, J. C., Birge, J. R., Mittenthal, J. & Noon, C. E. (1991), 'Matchup scheduling with multiple resources, release dates and disruptions', *Operations Research* **39**(3), 470–483.

Bierwirth, C. & Mattfeld, D. C. (1999), 'Production scheduling and rescheduling with genetic algorithms', *Evolutionary Computation* **7**(1), 1–17.
URL: *http://citeseer.ist.psu.edu/bierwirth99production.html*

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

Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G. & Wkeglarz, J. (1996), *Scheduling computer and manufacturing processes*, Springer-Verlag New York, Inc., New York, NY, USA.

Bonabeau, E., Dorigo, M. & Theraulaz, G. (1999), *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, Oxford.

Bond, A. H. & Glasser, L. (1988), *Readings in Distributed Artificial Intelligence*, Morgan Kaufman, San Mateo, California.

Bongaerts, L., Brussel, H. V., Valckenaers, P. & Peeters, P. (1997), 'Reactive scheduling in holonic manufacturing systems: Architecture, dynamic model and co-operation strategy'.
URL: *http://citeseer.ist.psu.edu/bongaerts97reactive.html*

Bongaerts, L., Monostori, L., McFarlane, D. & Kádár, B. (2000), 'Hierarchy in distributed shop floor control', *Computers in Industry* **43**(2), 123–137.

Bonvik, A. M., Couch, C. E. & Gershwin, S. B. (1997), 'A comparison of production-line control mechanisms', *International Journal of Production Research* **35**, 789–804.
URL: *http://www.ingentaconnect.com/content/tandf/tprs/1997/00000035/00000003/art00011*

Brennan, R. W. & Norrie, D. H. (1998), Evaluating the relative performance of alternative control architectures for manufacturing, pp. 90–95.
URL: *http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=713641*

Brennan, R. W. & Norrie, D. H. (2001), 'Evaluating the performance of reactive control architectures for manufacturing production control', *Computers in Industry* **46**, 235–245.
URL: *http://www.sciencedirect.com/science/article/B6V2D-43PHG17-2/2/d5a9953cca20608b2b312841aa48b950*

Brussel, H. V., Wyns, J., Valckenaers, P., Bongaerts, L. & Peeters, P. (1998), 'Reference architecture for holonic manufacturing systems: PROSA', *Computers in Industry* **37**(3), 255–274.

Bussmann, S. & Müller, J. (1993), A negotiation framework for cooperating agents, *in* S. M. Deen, ed., '1992 Proceedings of the Special Interest Group on Cooperating Knowledge-Based Systems', Dake Centre, University of Keele, pp. 1–17.

Caridi, M. & Cavalieri, S. (2004), 'Multi-agent systems in production planning and control: an overview', *Production Planning & Control* **15**, 106–118.
URL: *http://www.informaworld.com/10.1080/09537280410001662556*

Cavalieri, S., Garetti, M., Macchi, M. & Taisch, M. (2000), 'An experimental benchmarking of two multi-agent architectures for production scheduling and control', *Comput. Ind.* **43**(2), 139–152.

Chryssolouris, G. & Subramaniam, V. (2001), 'Dynamic scheduling of manufacturing job shops using genetic algorithms', *Journal of Intelligent Manufacturing* **12**, 281–293.
URL: *http://dx.doi.org/10.1023/A:1011253011638*

Church, L. K. & Uzsoy, R. (1992), 'Analysis of periodic and event-driven rescheduling policies in dynamic shops', *International Journal of Computer Integrated Manufacturing* **5**, 153–163.
URL: *http://www.informaworld.com/10.1080/09511929208944524*

Cicirello, V. & Smith, S. (2001), Wasp nests for self-configurable factories, *in* 'Proceedings of the Fifth International Conference on Autonomous Agents'.

Cowling, P. & Johansson, M. (2002), 'Using real time information for effective dynamic scheduling', *European Journal of Operational Research*

127(2), 230–244.
URL: *http://ideas.repec.org/a/eee/ejores/v139y2002i2p230-244.html*

Dorn, J., Kerr, R. & Thalhammer, G. (1995), 'Reactive scheduling: improving the robustness of schedules and restricting the effects of shop floor disturbances by fuzzy reasoning', *International Journal on Human-Computer Studies* **42**(6), 687–704.

Duffie, N. A. (1990), 'Synthesis of heterarchical manufacturing systems', *Computers in Industry* **14**(1-3), 167–174.

Durfee, E. H., Lesser, V. R. & Corkill, D. D. (1989*a*), Cooperative distributed problem solving, *in* A. Barr, P. Cohen & E. Feigenbaum, eds, 'The Handbook of Artificial Intelligence', Vol. 4, Addison Wesley, pp. 83–147.

Durfee, E. H., Lesser, V. R. & Corkill, D. D. (1989*b*), 'Trends in cooperative distributed problem solving', *IEEE Transactions on Knowledge and Data Engineering* **1**(1), 63–83.
URL: *http://citeseer.ist.psu.edu/durfee95trends.html*

Faget, P., Eriksson, U. & Herrmann, F. (2005), Applying discrete event simulation and an automated bottleneck analysis as an aid to detect running production constraints, *in* 'WSC '05: Proceedings of the 37th conference on Winter simulation', Winter Simulation Conference, pp. 1401–1407.

Faratin, P., C. S. & Jennings, N. R. (1998), 'Negotiation decision functions for autonomous agents', *International Journal of Robotics and Autonomous Systems* **24**(3-4), 159–182.

Ferber, J. (1999), *Multi-agent Systems: Introduction to Distributed Artificial Intelligence*, Addison Wesley.

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

Glassey, C. R. & Petrakian, R. G. (1989), The use of bottleneck starvation avoidance with queue predictions in shop floor control, *in* 'WSC '89: Pro-

ceedings of the 21st conference on Winter simulation', ACM, New York, NY, USA, pp. 908–917.

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*

Glassey, C. & Weng, W. (1991), 'Dynamic batching heuristic for simultaneous processing', *IEEE Transactions on Semiconductor Manufacturing* **4**(2), 77–82.

Glover, F. (1989), 'Tabu search– part I', *ORSA Journal on Computing* **1**(3), 190–206.

Glover, F. (1990), 'Tabu search– part II', *ORSA Journal on Computing* **2**(1), 4–32.

Glover, F. & Laguna, M. (1997), *Tabu Search*, Kluwer Academic Publishers.

Glover, F., Taillard, E. & de Werra, D. (1993), 'A user's guide to tabu search', *Annals of Operations Research* **41**, 3–28.

Goldratt, E. (1990), *Theory of Constraints*, North River Press, Croton-on-Hudson.

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

Gou, L., Luh, P. & Kyoya, Y. (1998), 'Holonic manufacturing scheduling: architecture'.
URL: *http://citeseer.ist.psu.edu/gou98holonic.html*

Hatvany, J. (1985), 'Intelligence and cooperation in heterarchic manufacturing systems', *Robotics & Computer Integrated Manufacturing* **2**(2), 101–104.

Hendry, L. C. & Wong, S. K. (1994), 'Alternative order release mechanisms: a comparison by simulation', *International Journal of Production Research* **32**, 2827–2842.
URL: *http://www.informaworld.com/10.1080/00207549408957103*

Holland, J. H. (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press.

Hopp, W. J. & Roof, M. L. (1998), 'Setting WIP levels with statistical throughput control (STC) in CONWIP production lines', *International Journal of Production Research* **36**(4), 867–882.

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

Hunsberger, L. & Grosz, B. (2000), 'A combinatorial auction for collaborative planning'.
URL: *http://citeseer.ist.psu.edu/hunsberger00combinatorial.html*

Iwata, K., Onosato, M. & Koike, M. (1994), 'Random manufacturing system: a new concept of manufacturing systems for production to order', *Annals of the CIRP* **43**(1), 379–384.

Jahangirian, M. & Conroy, G. V. (2000), 'Intelligent dynamic scheduling system: the application of genetic algorithms', *Integrated Manufacturing Systems* **11**, 247–257.
URL: *http://www.ingentaconnect.com/content/mcb/068old/2000/00000011/00000004/art00003*

Jain, A. & Meeran, S. (1998), 'A state-of-the-art review of job-shop scheduling techniques'.
URL: *http://citeseer.ist.psu.edu/87999.html*

Jain, A. & Meeran, S. (1999), 'Deterministic job-shop scheduling: Past, present, and future'.
URL: *http://citeseer.ist.psu.edu/jain98deterministic.html*

Jennings, N. R. (1996), Coordination techniques for distributed artificial intelligence, *in* G. M. P. O'Hare & N. R. Jennings, eds, 'Foundations of Distributed Artificial Intelligence', John Wiley & Sons, pp. 187–210.
URL: *http://citeseer.ist.psu.edu/jennings96coordination.html*

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.

Jennings, N. R., Sycara, K. & Wooldridge, M. (1998), 'A roadmap of agent research and development', *Journal of Autonomous Agents and Multi-Agent Systems* **1**(1), 7–38.
URL: *http://citeseer.ist.psu.edu/jennings98roadmap.html*

Jensen, M. (2001), 'Improving robustness and flexibility of tardiness and total flowtime job shops using robustness measures', *Journal of Applied Soft Computing* **1**(1), 35–52.
URL: *http://citeseer.ist.psu.edu/jensen01improving.html*

Jeong, K. C. & Kim, Y. D. (1998), 'A real-time scheduling mechanism for a flexible manufacturing system: using simulation and dispatching rules', *International Journal of Production Research* **36**, 2609–2626.
URL: *http://www.informaworld.com/10.1080/002075498192733*

Jones, A. & Rabelo, J. (1998), 'Survey of job shop scheduling techniques'.
URL: *http://citeseer.ist.psu.edu/jones98survey.html*

Kadar, B., Monostori, L. & Szelke, E. (1998), 'An object-oriented framework for developing distributed manufacturing architectures', *Journal of Intelligent Manufacturing* **9**, 173–179.
URL: *http://www.ingentaconnect.com/content/klu/jims/1998/00000009/00000002/00173463*

Kerr, R. M. & Szelke, E. (1995), *Artificial Intelligence In Reactive Scheduling*, Kluwer Academic Publishers Group.

Kolonko, M. (1999), 'Some new results on simulated annealing applied to the job shop scheduling problem', *European Journal of Operational Research* **113**(1), 123–136.
URL: *http://ideas.repec.org/a/eee/ejores/v113y1999i1p123-136.html*

Kraus, S. (1997), 'Negotiation and cooperation in multi-agent environments', *Artificial Intelligence* **94**(1-2), 79–97.

Kumar, S. & Kumar, P. (2001), 'Queueing network models in the design and analysis of semiconductor wafer fabs', *Robotics and Automation, IEEE Transactions on* **17**(5), 548–561.
URL: *http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=964657*

Langer, G. & Alting, L. (2001), 'Trends and perspectives - an architecture for agile shop floor control systems', *Journal of Manufacturing Systems* **19**(4), 267–281.

Leon, J. V., Wu, D. S. & Storer, R. H. (1994), 'Robustness measures and robust scheduling for job shops', *IIE Transactions* **26**(5), 32–43.
URL: *http://10.33.174.170/macc/publications/robust.pdf*

Li, H., Li, Z., Li, L. X. & Hu, B. (2000), 'A production rescheduling expert simulation system', *European Journal of Operational Research* **124**(2), 283–293.
URL: *http://ideas.repec.org/a/eee/ejores/v124y2000i2p283-293.html*

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

Liu, X. & Zhang, W. (1998), 'Issues on the architecture of an integrated general-purpose shopfloor control software system', *Journal of Materials Processing Technology* **76**, 261–269.
URL: *http://www.ingentaconnect.com/content/els/09240136/1998/00000076/00000001/art00358*

Lozinski, C. & Glassey, C. (1988), 'Bottleneck starvation indicators for shop floor control', *IEEE Transactions on Semiconductor Manufacturing* **1**(4), 147–153.

MacCarthy, B. (2001), *Modeling manufacturing systems: from aggregate planning to real-time control*, Springer.
URL: *http://dx.doi.org/10.1002/1099-1425(200101/02)4:1¡68::AID-JOS65¿3.0.CO;2-0*

MacCarthy, B. L. & Liu, J. (1993), 'Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling',

International Journal of Production Research **31**, 59–79.
URL: *http://www.informaworld.com/10.1080/00207549308956713*

Maley, J. (1988), 'Managing the flow of intelligent parts', *Robotics and Computer-Integrated Manufacturing* **4**(3-4), 525–530.

Marcus, A., Vancza, T. & Monostori, L. (1996), 'A market approach to holonic manufacturing'.
URL: *http://citeseer.ist.psu.edu/markus96market.html*

Maturana, F. P. & Norrie, D. H. (1997), 'Distributed decision-making using the contract net within a mediator architecture', *Decision Support Systems* **20**(1), 53–64.

Maturana, F., Shen, W. & Norrie, D. (1999), 'Metamorph: An adaptive agent-based architecture for intelligent manufacturing'.
URL: *http://citeseer.ist.psu.edu/maturana99metamorph.html*

Mehta, S. V. & Uzsoy, R. (1999), 'Predictable scheduling of a single machine subject to breakdowns', *International Journal of Computer Integrated Manufacturing* **12**, 15–38.
URL: *http://www.informaworld.com/10.1080/095119299130443*

Meziane, F., Vadera, S., Kobbacy, K. & Proudlove, N. (2000), 'Intelligent systems in manufacturing: current developments and future prospects', *Integrated Manufacturing Systems* **11**, 218–238.
URL: *http://www.ingentaconnect.com/content/mcb/068old/2000/00000011/00000004/art00001*

Miyashita, K. & Sycara, K. P. (1995), 'CABINS: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair', *Artificial Intelligence* **76**(1-2), 377–426.
URL: *http://citeseer.ist.psu.edu/34322.html*

Monostori, L., Vancza, J. & Kumara, S. R. T. (2006), 'Agent-based systems for manufacturing', *CIRP Annals - Manufacturing Technology* **55**, 697–720.
URL: *http://www.sciencedirect.com/science/article/B8CXH-4MX6G4P-4/2/8642b40c87518dc503facdad069cfdfa*

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.

Muhlemann, A. P., Lockett, A. G. & Farn, C. K. (1982), 'Job shop scheduling heuristics and frequency of scheduling', *International Journal of Production Research* **20**, 227–241.
URL: *http://www.informaworld.com/10.1080/00207548208947763*

Nwana, H. S., Lee, L. C. & Jennings, N. R. (1996), 'Coordination in software agent systems', *The British Telecom Technical Journal* **14**(4), 79–88.
URL: *http://citeseer.ist.psu.edu/nwana96coordination.html*

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

O'kane, J. (2000), 'A knowledge-based system for reactive scheduling decision-making in fms', *Journal of Intelligent Manufacturing* **11**, 461–474.
URL: *http://www.ingentaconnect.com/content/klu/jims/2000/00000011/00000005/00272672*

Ouelhadj, D. (2003), A Multi-agent System for the Integrated Dynamic Scheduling of Steel Production., PhD thesis, The School of Computer Science & Information Technology, University of Nottingham, UK.

Ovalle, O. & Marquez, A. (2003), 'Exploring the utilization of a CONWIP system for supply chain management. a comparison with fully integrated supply chains', *International Journal of Production Economics* **83**, 195–215.
URL: *http://www.ingentaconnect.com/content/els/09255273/2003/00000083/00000002/art00328*

Papazoglou, M. P., Laufmann, S. C. & Sellis, T. K. (1992), 'An organizational framework for cooperating intelligent information systems', *Journal of Intelligent and Cooperative Information Systems* **1**(1), 169–202.

Parunak, H. (1993), 'Autonomous agent architectures: A non technical introduction', *Industrial Technology Institute* .
URL: *http://www.agent.ai/download.php?ctag=download&docID=21*

Parunak, H. V. D. (1987), Manufacturing experience with the contract net, *in* M. Huhns, ed., 'Distributed Artificial Intelligence', Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, pp. 285–310.
URL: *http://citeseer.ist.psu.edu/vandykeparunak87manufacturing.html*

Parunak, H. V. D. (1995), Applications of distributed artificial intelligence in industry, *in* G. M. P. O'Hare & N. R. Jennings, eds, 'Foundations of Distributed AI', John Wiley & Sons.
URL: *http://citeseer.ist.psu.edu/parunak94applications.html*

Parunak, H. V. D. (1998*a*), 'Practical and industrial applications of agent-based systems'.
URL: *http://citeseer.ist.psu.edu/parunak98practical.html*

Parunak, H. V. D. (1998*b*), What can agents do in industry, and why? an overview of industrially-oriented r&d at cec, *in* 'CIA '98: Proceedings of the Second International Workshop on Cooperative Information Agents II, Learning, Mobility and Electronic Commerce for Information Discovery on the Internet', Springer-Verlag, London, UK, pp. 1–18.

Paulo Leit a. & Restivo, F. (2006), 'ADACOR: a holonic architecture for agile and adaptive manufacturing control', *Computers in Industry* **57**(2), 121–130.

Pfund, M., Mason, S. & Fowler, J. (2006), Dispatching and scheduling in semiconductor manufacturing, *in* J. W. Herrmann, ed., 'Handbook of Production Scheduling', Springer.

Pham, D. T. & Karaboga, D. (2000), *Intelligent optimisation techniques genetic algorithms, tabu search, simulated annealing and neural networks*, Springer, London; New York.

Pinedo, M. (2002), Scheduling: Theory, algorithms and systems (second edition), Technical report, Department of Information, Operations & Management Sciences, NYU.

Prabhu, V. V. & Duffie, N. A. (1995), 'Modelling and analysis of nonlinear dynamics in autonomous heterarchical manufacturing systems control', *CIRP Annals - Manufacturing Technology* **44**, 425–428.
URL: *http://www.sciencedirect.com/science/article/B8CXH-4P3DTXT-3C/2/132bf30b6658c3b76a8e49c2a03c1d95*

Prabhu, V. V. & Duffie, N. A. (1996), 'Modelling and analysis of heterarchical manufacturing systems using discontinuous differential equations', *CIRP Annals - Manufacturing Technology* **45**, 445–448.
URL: *http://www.sciencedirect.com/science/article/B8CXH-4P3KRDP-3C/2/505e6630ebceed40079ac5ddb9597cd3*

Rabelo, R. J., Camarinha-Matos, L. M. & Afsarmanesh, H. (1998), Multiagent perspectives to agile scheduling, *in* 'BASYS '98: Proceedings of the 3rd IEEE/IFIP international conference on Intelligent systems for manufacturing : multi-agent systems and virtual organizations', Kluwer Academic Publishers, Norwell, MA, USA, pp. 51–66.

Ramos, C. (1994), An architecture and a negotiation protocol for the dynamic scheduling of manufacturing systems, *in* E. Straub & R. S. Sipple, eds, 'Proceedings of the International Conference on Robotics and Automation. Volume 4', IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 3161–3166.

Reeves, C. R. (1993), *Modern heuristic techniques for combinatorial problems*, John Wiley & Sons, Inc., New York, NY, USA.

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.

Rosenschein, J. S. & Zlotkin, G. (1994), *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*, MIT Press, Cambridge, MA.

Roser, C., Nakano, M. & Tanaka, M. (2001), A practical bottleneck detection method, *in* 'Winter Simulation Conference', pp. 949–953.
URL: *http://doi.acm.org/10.1145/564124.564259*

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*

Roser, C., Nakano, M. & Tanaka, M. (2003), Simulation test bed for manufacturing analysis: comparison of bottleneck detection methods for AGV systems, *in* S. E. Chick, P. J. Sanchez, D. M. Ferrin & D. J. Morrice, eds, 'Winter Simulation Conference', ACM, pp. 1192–1198.
URL: *http://doi.acm.org/10.1145/1030818.1030976*

Rossi, A., Dini, G., Chryssolouris, G. & Subramaniam, V. (2000), 'Dynamic scheduling of FMS using a real-time genetic algorithm', *International Journal of Production Research* **38**(1), 1–20. 10.1023/A:1011253011638.
URL: *http://www.ingentaconnect.com/content/tandf/tprs/2000/00000038/00000001/art00001*

Sabuncuoglu, I. & Bayiz, M. (2000), 'Analysis of reactive scheduling problems in a job shop environment', *European Journal of Operational Research* **126**, 567–586.
URL: *http://www.sciencedirect.com/science/article/B6VCT-4135XF5-B/2/6cb46f365829f1018fc6646eebbe56b3*

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.

Sanmarti, E., Huercio, A., Espuna, A. & Puigjaner, L. (1996), 'A combined scheduling/reactive scheduling strategy to minimize the effect of process

operations uncertainty in batch plants', *Computers & Chemical Engineering* **20**, S1263–S1268.
URL: *http://www.sciencedirect.com/science/article/B6TFT-48B0PYH-BC/2/18837f8c96bbd56580df82ebb61b0330*

Sanmarti, Eduardand Espuna, A. P. L. (1997), 'Batch production and preventive maintenance scheduling under equipment failure uncertainty', *Computers & Chemical Engineering* **21**, 1157–1168.
URL: *http://www.sciencedirect.com/science/article/B6TFT-3SHC6FS-9/2/2fc41980f3bd0e90a985252b4429c79e*

Satake, T., Morikawa, K., Takahashi, K. & Nakamura, N. (1999), 'Simulated annealing approach for minimizing the makespan of the general job-shop', *International Journal of Production Economics* **60**(1), 515–522.
URL: *http://ideas.repec.org/a/eee/proeco/v60-61y1999i1p515-522.html*

Scerri, P., Farinelli, A., Okamoto, S. & Tambe, M. (2004), Allocating roles in extreme teams, *in* 'AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems', IEEE Computer Society, Washington, DC, USA, pp. 1502–1503.

Scerri, P., Liao, E., Lai, J., Sycara, K., Xu, Y. & Lewis, M. (2004), 'Coordinating very large groups of wide area search munitions'.
URL: *http://citeseer.ist.psu.edu/686407.html; http://usl.sis.pitt.edu/ulab/cco.pdf*

Schumacher, J., Verwater-Lukszo, Z. & Weijnen, M. P. C. (1999), 'Disturbances and their impact on scheduling', *Comput. Ind. Eng.* **37**(1-2), 75–79.

Shafaei, R. & Brunn, P. (1999), 'Workshop scheduling using practical (inaccurate) data part 2: An investigation of the robustness of scheduling rules in a dynamic and stochastic environment', *International Journal of Production Research* **37**, 4105–4117(13).
URL: *http://www.ingentaconnect.com/content/tandf/tprs/1999/00000037/00000018/art00002*

Shafaei, R. & Brunn, P. (2000), 'Workshop scheduling using practical (inaccurate) data part 3: A framework to integrate job releasing, routing and

scheduling functions to create a robust predictive schedule', *International Journal of Production Research* **38**, 85–99(15).
URL: *http://www.ingentaconnect.com/content/tandf/tprs/2000/00000038/00000001/art00006*

Shaw, M. J. (1988), 'Dynamic scheduling in cellular manufacturing systems: A framework for networked decision making', *Journal of Manufacturing Systems* **7**(2), 83–94.

Shen, J., Zhang, X. & Lesser, V. (2004), Degree of local cooperation and its implication on global utility, *in* 'AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems', IEEE Computer Society, Washington, DC, USA, pp. 546–553.

Shen, W., Norrie, D. & Barthes, J. (2001), *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing*, Taylor & Francis, London.

Shen, W. & Norrie, D. H. (1999), 'Agent-based systems for intelligent manufacturing: A state-of-the-art survey', *Knowledge and Information Systems* **1**(2), 129–156.

Shukla, C. S. & Frank Chen, F. (1996), 'The state of the art in intelligent real-time FMS control: a comprehensive survey', *Journal of Intelligent Manufacturing* **7**, 441–455.
URL: *http://dx.doi.org/10.1007/BF00122834*

Smith, R. G. (1980), 'The contract net protocol: High level communication and control in a distributed problem solver', *IEEE Transactions on Computers* **C-29**(12), 1104–1113.

Spearman, M., Woodruff, D. & Hopp, W. (1990), 'CONWIP: A pull alternative to kanban.', *International Journal of Production Research* **28(5)**, 879–894.

Stevenson, M., Hendry, L. C. & Kingsman, B. G. (2005), 'A review of production planning and control: the applicability of key concepts to the make-to-order industry', *International Journal of Production Research* **43**, 869–898.

URL: *http://www.ingentaconnect.com/content/tandf/tprs/2005/00000043/00000005/art00001*

Stoop, P. P. M. & Wiers, V. C. S. (1996), 'The complexity of scheduling in practice', *International Journal of Operations & Production Management* **16**, 37–53(17).
URL: *http://www.ingentaconnect.com/content/mcb/024/1996/00000016/00000010/art00003*

Sun, J. & Xue, D. (2001), 'A dynamic reactive scheduling mechanism for responding to changes of production orders and manufacturing resources', *Comput. Ind.* **46**(2), 189–207.

Suresh, V. & Chaudhuri, D. (1993), 'Dynamic scheduling–a survey of research', *International Journal of Production Economics* **32**, 53–63.
URL: *http://www.sciencedirect.com/science/article/B6VF8-45JT1J9-1W/2/2d015e0954bacb3b076dca882e736e92*

Szelke, Elizabethand Kerr, R. M. (1994), 'Knowledge-based reactive scheduling', *Production Planning & Control* **5**, 124–145.
URL: *http://www.informaworld.com/10.1080/09537289408919480*

Tan, K. & Narasimhan, R. (1997), 'Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach', *Omega* **25**(6), 619–634(16).
URL: *http://www.ingentaconnect.com/content/els/03050483/1997/00000025/00000006/art00024*

Tharumarajah, A. (2001), 'Survey of resource allocation methods for distributed manufacturing systems', *Production Planning and Control* **12**, 58–68.
URL: *http://www.ingentaconnect.com/content/tandf/tppc/2001/00000012/00000001/art00008*

Tharumarajah, A. & Bemelman, R. (1997), 'Approaches and issues in scheduling a distributed shop-floor environment', *Computers in Industry* **34**, 95–109.
URL: *http://www.ingentaconnect.com/content/els/01663615/1997/00000034/00000001/art00060*

Tolksdorf, R. & Menezes, R. (2003), Using swarm intelligence in Linda systems, *in* A. Omicini, P. Petta & J. Pitt, eds, 'Proceedings of the Fourth International Workshop Engineering Societies in the Agents World ESAW'03'.

Tseng, M., Lei, M. & Su, C. (1997), 'A collaborative control system for mass customization manufacturing', *Annals of the CIRP* **1**(1), 373–376.

Valckenaers, P., Bonneville, F., Van Brussel, H., Bongaerts, L. & Wyns, J. (1994), Results of the holonic control system benchmark at the kuleuven, *in* 'Proceedings of the CIMAT Conference (Computer Integrated Manufacturing and Automation Technology)', pp. 128–133.

Vamos, T. (1983), 'Cooperative systems - an evolutionary perspective', *IEEE Control Systems Magazine* **3**(2), 9–14.

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

Vieira, G. E. & Herrmann, J. W. (2000), 'Analytical models to predict the performance of a single-machine system under periodic and event-driven rescheduling strategies', *International Journal of Production Research* **38**, 1899–1915(17).
URL: *http://www.ingentaconnect.com/content/tandf/tprs/2000/00000038/00000008/art00010*

Vieira, G. E., Herrmann, J. W. & Lin, E. (2003), 'Rescheduling manufacturing systems: A framework of strategies, policies, and methods', *Journal of Scheduling* **6**, 39–62.
URL: *http://dx.doi.org/10.1023/A:1022235519958*

Voss, S., Martello, S., Osman, I. H. & Roucairol, C. (1999), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers.

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 Au-

tonomous agents and multiagent systems', ACM Press, New York, NY, USA, pp. 757–764.

Wein, L. (1988), 'Scheduling semiconductor wafer fabrication', *IEEE Transactions on Semiconductor Manufacturing* **1**(3), 115–130.

Wein, L. M. (1990), 'Scheduling networks of queues: heavy traffic analysis of a two-station network with controllable inputs', *Operations Research* **38**(6), 1065–1078.

Wein, L. M. (1991), 'Brownian networks with discretionary routing', *Operations Research* **39**(2), 322–340.

Wein, L. M. (1992), 'Scheduling networks of queues: heavy traffic analysis of a multistation network with controllable inputs', *Operations Research* **40**(S2), 312–334.

Wein, L. M. & Chevalier, P. B. (1992), 'A broader view of the job-shop scheduling problem', *Management Science* **38**(7), 1018–1033.

Wiendahl, H. & Ahrens, V. (1997), 'Agent-based control of self-organised production systems', *Annals of the CIRP* **1**(1), 365–368.

Wiers, V. C. S. (1997), 'A review of the applicability of OR and AI scheduling techniques in practice', *Omega* **25**(2), 145–153.
URL: *http://dx.doi.org/10.1016/S0305-0483(96)00050-3*

Womack, J. P., Jones, D. T. & Roos, D. (1991), *The Machine That Changed the World : The Story of Lean Production*, Harper Perennial.
URL: *http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0060974176*

Wooldridge, M. (2002), *Introduction to MultiAgent Systems*, John Wiley & Sons.

Wooldridge, M. & Jennings, N. (1995), 'Intelligent agents: Theory and practice', *Knowledge Engineering Review* **10**(2), 115–152.

Wu, S. D., Storer, R. H. & Chang, P.-C. (1993), 'One-machine rescheduling heuristics with efficiency and stability as criteria', *Computers and Operations Research* **20**(1), 1–14.

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.

Xue, D., Sun, J. & Norrie, D. H. (2001), 'An intelligent optimal production scheduling approach using constraint-based search and agent-based collaboration', *Comput. Ind.* **46**(2), 209–231.

Yamamoto, M. & Nof, S. Y. (1985), 'Scheduling/rescheduling in the manufacturing operating system environment', *International Journal of Production Research* **23**, 705–722.
URL: *http://www.informaworld.com/10.1080/00207548508904739*

Zweben, M., Daun, B. & Deale, M. (1994), *Scheduling and rescheduling with iterative repair*, Morgan Kaufman, pp. 241–255.

Zweben, M. & Fox, M. S. (1994), *Intelligent Scheduling*, Morgan Kaufmann.

Appendix A

Mechanism of Kanbans

In CABS, the lots are parked in input buffers at each agent (workstation). When a new lot arrives at agent, it is parked in one of the input buffer where it waits for its processing. Agent has a limited number of input buffers and agent maintains a kanban (card) corresponding to each of its buffer. Each kanban contains following information:

- Owner Agent: The owner agent of kanban, i.e. the agent identification to which the kanban and corresponding buffer belongs.
- Identifier: An index which uniquely identifies the kanban at its owner agent.

During execution, a kanban shuffles between the owner and its preceding agents and flow of kanbans is controlled through passing of messages between the pair of agents. Although there are other messages also in system (described in Appendix A), the messages relevant for management of kanbans are:

- REQUEST_KANBAN_MESG: Agent receives this message from a preceding agent, notifying the preceding agent's request for a kanban.
- GRANTED_KANBAN: In reply to a kanban request from preceding agent, agent sends a kanban to preceding agent by this message.
- LOT_ARRIVAL: This indicates the arrival of a new lot agent along with a kanban, from preceding agent.

- UNUSED_KANBAN: An unused kanban returns to agent from preceding agent, the kanban is free to be used immediately.

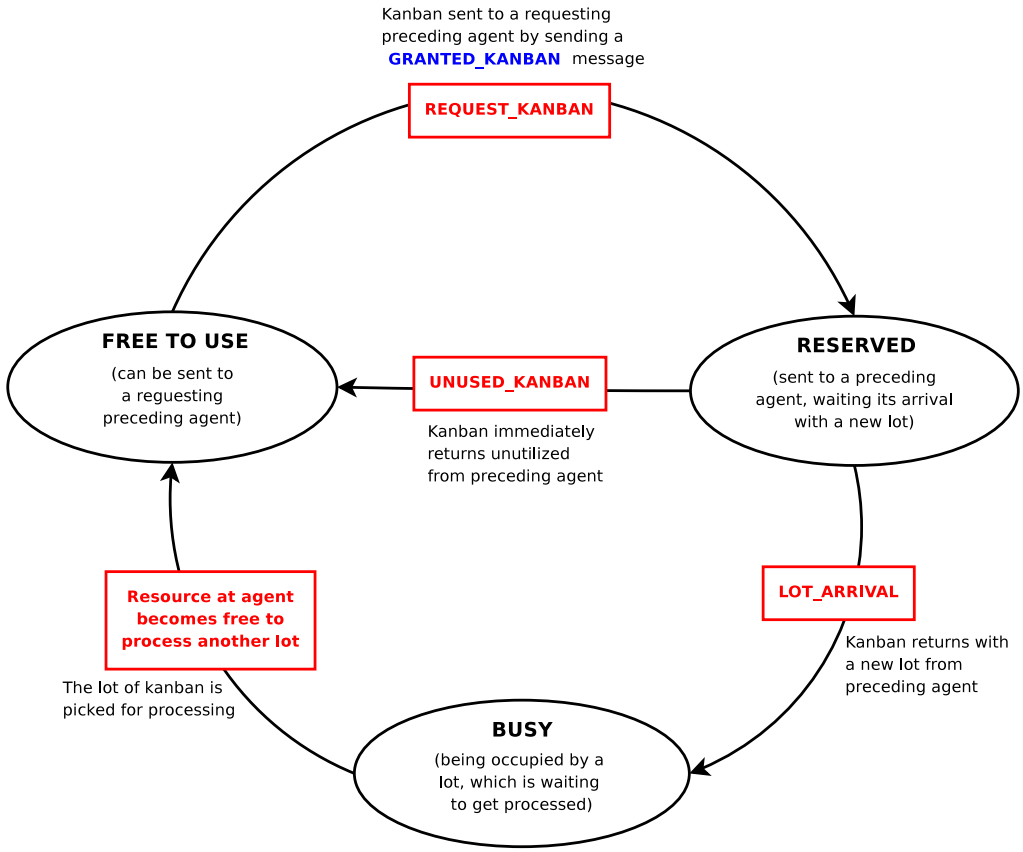

Figure A.1: States of an agent's Kanban

Figure A.1 shows the various states of an agent's kanban. The state of kanban changes according to events shown in boxes on the transition arcs. Before processing of a lot, agent requests a kanban from the succeeding agent to which the processed lot will be sent. In reply to kanban request from a preceding agent, owner agent sends a kanban corresponding to one of its free buffer. By sending kanban, owner agent reserves one of its free input buffer for the forthcoming job. After the requested kanban arrives at requesting agent, it stays with requesting agent till processing of job is finished, after which it is returned to the (succeeding) owner agent along with processed lot. The kanban returns to owner agent with a lot which is parked in a free incoming buffer. The buffer and kanban

remains occupied till the lot is picked for processing by owner agent. The kanban becomes free for reuse and owner agent can again send it to a requesting preceding agent.

The agents in CABS execute autonomously and messaging between agents is asynchronous. The agents may change their dispatching after requesting kanban for a different lot, which results in arrival of unsolicited kanbans from some succeeding agents. Agents return such undesired kanbans to their respective owners with UNUSED_KANBAN message.

Appendix B

Events in CABS

For experimentation, the agent based semiconductor fabrication process is modeled as a parallel agent discrete event based system. The system is built using SPADES (Riley & Riley 2003) middleware[1], 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. In CABS, each workstation is modeled as an agent which executes independently. The semiconductor fabrication system might be processing different kind of products, and each product has a specific process route which specifies the sequence of agents on which a job has to be processed along with processing time required for each operation. The information that an CABS's agent maintains is as follows:

- **Process Steps**: Based on the process routes of various products an agent might be processing multiple process steps. Agents maintains a list of process steps that they process. Each entry of the list has following information:
  - `Product Identifier`: Identification of product to which the processing step belongs.
  - `Step Index`: The step identifier in process route of product. Products may have cycles involving an agent; step index uniquely identifies each processing step.

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

- `Processing Time`: Time required to process the job for corresponding processing step.
- `Succeeding Agent`: The agent which processes next processing step after this processing step. After processing a job of this step, the job has to be sent to the corresponding Succeeding Agent.
- `Preceding Agent`: The agent which processes the preceding processing step before this processing step. The jobs of this processing step come from the corresponding Preceding Agent.
- `Processed`: This is the total number of jobs of this step that have already been processed by agent.
- `Incoming Requirements`: Agent periodically receives requirement messages for the processing step from its corresponding succeeding agent. Agent maintains the latest incoming requirements of processing step i.e. `time limit`, `request rate`, `amount` and `criticality`.
- `Reserved Kanban`: Section 5.1.2 describes the use of reserved buffers that are used for avoiding deadlocks in system. A single buffer is reserved for each process step, and information of kanban corresponding to that reserved buffer is maintained with each process step. The details of general kanban mechanism are provided in Appendix A and details of using reserved kanbans are provided in Appendix C.

- **Demand Information**: Agents maintains information about the demand of each product that it processes.
- **WIP**: The information about each lot, that is buffered with the agent. Each lot in the system contains following information:
  - `Lot Identifier`: an unique number for a particular lot.
  - `Product Identifier`: product to which the lot belongs.
  - `Step Index`: processing step index in the process route of product, this is incremented as lot finishes processing of steps and advances in its process route.

- **Kanbans**: In order to regulate the amount of its individual WIP, agent maintains a card corresponding to each of its free buffer. The details of kanban mechanism are provided in Appendix A.

- **Resource Information**: The information about state of agent's processing resource i.e free, failed or busy with information of job getting processed currently etc.

As the prototype is developed as a discrete event based system, various manufacturing and coordination related events are generated in system during execution. Besides time, events carry additional information regarding the event. Agents of system know about other agents and can send directed events to specific agents. The processing at an agent is triggered by various events that it receives. During its processing, agent may generate further events. According to its destination, the events may be categorized into two types, local and remote. Local events are the events that an agent generates for itself. Remote events are sent to other agents in system. The local events are:

- **PROCESS_FINISH**: When agent (workstation) starts processing a lot, it schedules this event at appropriate time in future (according to the time required for processing step) to indicate completion of processing.

- **RESOURCE_FAILURE**: Based on the random number generated according to random distribution (MTBF), agent generates this event to indicate failure of its resource.

- **RESOURCE_UP**: When an agents fails, based on the random number generated according to random distribution (MTTR), agent generates this event to indicate resolution of failure.

- **CHECK_REMINDER**: This is a heartbeat event, which is utilized to avoid inactivity of agent for long duration. Agent schedules this event regularly at a fixed predetermined interval. In absence of any other events, agent does its health check and necessary actions in response to this event.

- **LOT_RELEASE_REMINDER**: This event is used only by the source agent, the first (hypothetical) agent in the process route of a product. The

source agent schedules this event to trigger the release of a new job which it does by sending it to the first workstation in process route.

The remote events that agents send to other agents in the system are:

- **LOT_ARRIVAL**: After finishing processing of a lot, agent send the lot to corresponding succeeding agent of process step with this event. Along with the lot, the associated kanban is also returned to agent.
- **NEED_BY_MESSAGE**: According to CABS algorithm described in thesis, agent periodically sends requirements for each of its process step to corresponding preceding agent. The messages are sent as this event which contains Product Identifier and Step Index with requirement parameters i.e. `time limit`, `request rate`, `amount` and `criticality`.
- **REQUEST_KANBAN_MESG**: Before starting processing of a lot, agent requests a kanban from the corresponding succeeding agent by sending this event to the succeeding agent.
- **GRANTED_KANBAN**: An agent grants an kanban to requesting preceding agent by sending this event which includes the identifier of kanban.
- **UNUSED_KANBAN**: Agent returns an unwanted kanban to its owner agent by this event.

In case their is a cycle in process route involving a single agent, i.e. the next process step is also processed on same agent, the agent sends remote messages mentioned above to itself. In such special case, remote events can also be considered as local events. The agent is oblivious to the target of events as delivery of events is transparently managed by simulation middleware platform. Figure B.1 shows relationship between various events in terms of triggers. An arc from event A to event B means that on receiving event A, there is possibility that agent may generate event B. Arcs represent possible triggers and actual generation of event will depend on the situation. For example, a local event PROCESS_FINISH indicates that processing of current lot has completed. Due to PROCESS_FINISH event, agent will necessarily generate a remote LOT_ARRIVAL event for its succeeding agent. However, the agent will

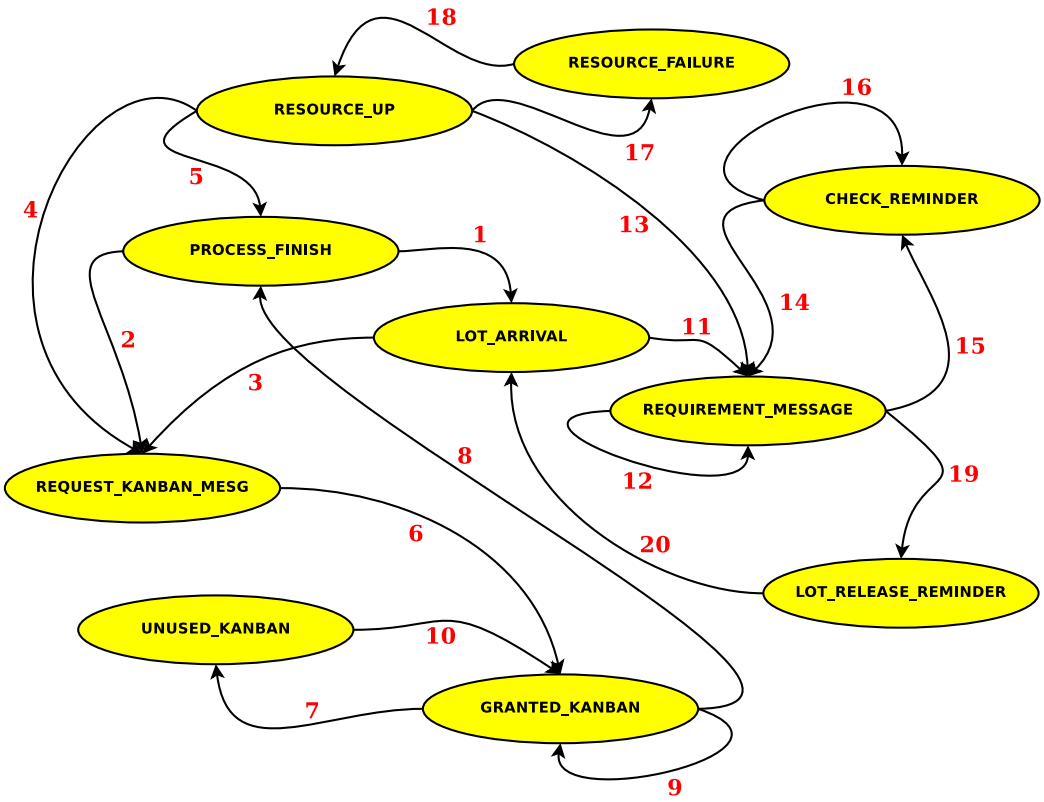

Figure B.1: Triggers for event generation

#	Description of the action(s) performed by agent
1	Send the processed lot to its corresponding succeeding agent
2	If WIP is available, request kanban for the lot to be dispatched next
3	If the resource is free, request kanban for dispatching arrived lot
4	If the resource is free and WIP is available, request kanban for dispatching
5	If some lot was being processed during failure, continue its processing
6	If a kanban is free, send it to requesting agent
7	If the kanban is not required, return it to its owner
8	Start processing a lot with the arrived kanban
9	If there is a pending kanban request, send the kanban to requesting agent
10	If there is a pending kanban request, send the kanban to requesting agent
11	Send new out-requirements after updating them by accounting new lot
12	Send new out-requirements according to received in-requirements
13	Send new out-requirements according to changed capabilities
14	Send new out-requirements according to current status (criticality etc.)
15	Schedule a self status check after fixed predetermined time interval
16	Schedule a self status check after fixed predetermined time interval
17	Schedule next failure at a random time according to distribution parameter (MTBF)
18	Schedule resolution of failure at a random time according to distribution parameter (MTTR)
19	**Only @ Source Agent**: Schedule a lot release according to the `time limit` parameter of incoming requirement
20	**Only @ Source Agent**: Release a new lot by sending it to the first agent in process route

Table B.1: Description of agent's actions during event triggers

generate another remote REQUEST_KANBAN_MESG event to request a kanban only if it has some other WIP to dispatch. Table B describes the conditions and actions that agent performs while generating an event. The number in the first column identify the trigger as labeled in Figure B.1.

Appendix C

Deadlock Avoidance

In Section 5.1.2, the basic mechanism of avoiding deadlocks by utilizing the reserved buffers is described. Here we describe the implementation details of that mechanism. Besides the shared buffers, which can hold any kind of lots, CABS maintains single reserved buffer for each type of jobs it processes. Buffers are managed through kanbans and there is a reserved kanban corresponding to each reserved buffer (details of kanban mechanism are provided in Appendix A. If all shared buffers are full, in response to a kanban request from preceding agent, reserved kanban corresponding to requested process step is granted. Besides reserved kanbans, CABS utilizes an additional message parameter, `ETA`, to avoid deadlocks. Along with `time limit`, `request rate`, `amount` and `criticality`, `ETA` is also send 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. 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 5 is called before CABS dispatching algorithm (Algorithm 1).

Algorithm 5 **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 5 removes tasks with higher `ETA`s, 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, `ETA`s of tasks remain 0 and Algorithm 5 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 `ETA`s. The last agent of process route assumes a static incoming `ETA` of 0. The agents towards the end of process generally will have lower `ETA`s and reserved buffers ensure availability of jobs of all process steps at agents. Hence, by prioritizing lots that will complete earlier, the flow of lots is perpetually maintained which autonomously avoids the occurrence of permanent bottlenecks.