Polite Rescheduling: Responding to Local Schedule Disruptions in Distributed Manufacturing Systems

Thomas Tsukada Kang G. Shin

Real–Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

Abstract

This paper considers the problem of handling schedule disruptions in a distributed manufacturing system. When a cell controller reschedules a manufacturing cell in response to some disruption, it may disrupt the schedule at some other cell, because schedules at different cells may interact. In the approach we propose, a controller at a disrupted cell tries to reschedule in a way which is likely to be least disruptive to other cells' schedules, through negotiation with controllers at other cells. This approach, which we call "polite rescheduling", has the advantage of retaining much of the original schedule, while avoiding wide propagation of the disruption through the rest of the system. Simulation results show that a polite rescheduling algorithm helps isolate disruptions to a small subset of cells.

1 Introduction

1.1 Motivation

Manufacturing systems routinely experience unexpected events, such as machine failures and resource unavailability. Flexible manufacturing systems must be able to recover from such disruptions efficiently. In a distributed manufacturing system, intelligent run-time coordination is necessary for such flexibility, because actions taken at one part of the system can adversely affect other parts of the system. If a disruption occurs at one cell of the system, that cell's controller must take some action. However, this action may result in a disruption in another cell. In such a way, a disruption at one cell may propagate through the whole system.

A good example of this type of propagation of disruptions can be seen in the rescheduling of a cellular manufacturing system. If one cell suffers a disruption, the jobs at that cell may have to be rescheduled. However, because of precedence constraints or resource sharing, this rescheduling may disrupt the schedules of other cells, by the late arrival of parts or resource unavailability.

The work described in this paper was supported in part by the NSF Under Grant IRI-9209031.

1.2 "Polite Replanning"

We address the problem of recovering from a disruption in a distributed plan, specifically a problem of how an individual agent handles the recovery from such a disruption. In our approach, which we call "polite replanning", the affected agent attempts to solve locally the problem of finding a response to the disruption, in such a way that it will be least disruptive to other agents. This approach avoids the costs of making the local problem a global problem, while it remains in a cooperative framework by attempting to isolate the effects of the disruption. More importantly, by avoiding complete replanning of the system, and by attempting to isolate disruptions, it attempts to retain as much of the distributed plan as possible.

In order to find a response which is least disruptive to other agents, the affected agent must have some information about how its actions will affect those other agents. Because an individual agent does not have global knowledge about the system, some form of negotiation is needed as a means of gathering information about other agents. Negotiation is a well-studied concept in distributed artificial intelligence (DAI) [4, 6]. The disrupted agent searches for the least disruptive response by negotiating with other agents which could possibly be disrupted by its actions.

We explore these issues in the domain of distributed job shop rescheduling. While finding a good schedule given some measure is a very hard problem, handling the disruption of an already existing schedule also presents an important problem. We consider "polite rescheduling", the application of polite replanning to the problem of recovering from such a disruption in a distributed group of manufacturing cells.

This paper is organized as follows. which is related to ours. Section 2 presents a formal model and an outline of our polite replanning ideas. Section 3 describes the application of our ideas to the job shop rescheduling domain. Section 4 presents some preliminary results of our work. Section 5 presents a summary of this work.

2 Formal Model

Let C be the set of n cells. For each cell i, there is a set $S_i = \{s_{i1}, \ldots, s_{im}\}$ of states. In the job shop scheduling domain, for example, this set would be the set of all schedules. For each cell i, there is a set $D_i \subset S_i$, which is the set of *disrupted* states. A state in S_i which is not in D_i is a *safe* state. A disruption is an event which puts a cell into a disrupted state. A disrupted state in job shop scheduling would be an infeasible schedule. Let set $A_i = \{a_{i1}, \ldots, a_{ip}\}$ be the set of actions which can be taken at cell i.

For each pair of cells i and k, there is a transition function $\mathcal{F}_{ik} : A_i \times S_k \longrightarrow S_k$ which describes how an action taken by cell i affects cell k. Thus, if cell k is in state s and cell i takes action a, then cell k will be put into state $\mathcal{F}_{ik}(a, s)$. If cell i takes an action which puts cell j into a disrupted state, we say cell j has been *disrupted* by cell i. If cell i is in state s, and then takes action a, it will itself be put into state $\mathcal{F}_{ii}(a, s)$.

For cell i and state $s \in D_i$, let $\mathrm{RA}_i(s) = \{a : a \in A_i, \mathcal{F}_{ii}(a, s) \in S_i - D_i\}$ be the set of recovery actions for cell i while it is in disrupted state s. For cell i and state $s \in D_i$, let $\mathrm{GA}_i(s) = \{a : a \in \mathrm{RA}_i(s), \mathcal{F}_{ij}(a, s') = s' \text{ for all } s' \in S_j, j \in C, j \neq i\}$ be the set of *guaranteed-local* recovery actions. A guaranteed-local recovery action will not affect any other cell, regardless of what state it is in. For cell i, let $T_i = \{s : s \in D_i, \mathrm{GA}_i(s) \neq \emptyset\}$ be the set of *semi-safe* states for cell i. A semi-safe state is a disrupted state from which a cell can recover with a guaranteed-local recovery action. For cell i and action $a \in A_i$, let $M_i(a) = \{j : j \in C, j \neq i, \exists s \in S_j, \mathcal{F}_{ij}(a, s) \in D_j \}$ be the set of remote cells which could possibly be disrupted by cell i taking action a. Clearly, if $a \in \mathrm{GA}_i(s)$ for some cell i and action a, then $M_i(a) = \emptyset$.

Disruptions can be classified by how much they result in propagation of disruptions. Consider a system state x in which cell i is in disrupted state $x_i = s_d \in D_i$, and in which every other cell is in a safe or semi-safe state. We call the disruption which caused cell i to be disrupted a disruption of *type 0* if $\mathrm{GA}_i(s_d) \neq \emptyset$. By taking an action $a \in \mathrm{GA}_i$, cell i can recovery from a type 0 disruption without disrupting other cells. Likewise, we call the disruption a disruption of *type l*, where $l > 0$, if there is an action $a \in \mathrm{RA}_i(s_d)$ such that $M_i(a) \neq \emptyset$, and a cell $k \in M_i(a)$, such that $\mathcal{F}_{ij}(a, x_j) \in T_j$ for all $j \in M_i(a)$, $j \neq k$, and such that, if cell k is disrupted by action a (that is, if state $\mathcal{F}_{ik}(a, x_k) \in D_k$), it is a disruption of type $l - 1$. Cell i can recover from a disruption of type n without the disruption being propagated more that l levels. If a disruption is not of any of these types, then this model cannot describe how the system can recover from this disruption.

Outline of Approach

In our 'polite replanning' approach, we assume that, in searching for the lowest cost response to an outside disruption, it is best to try to limit the propagation of the disruptions. Even though the best solution might entail the disruption of every cell in the system, we limit the search space by trying to find a solution which involves the least disruption propagation.

When a cell experiences a disruption, it tries to determine whether this disruption is of type 0. If it determines this, then it takes a recovery action which will not result in a disruption of another cell. If not, then it tries to determine through negotiation with other cells whether the disruption is of type 1. If it determines this, it takes the action which results in a propagation of the disruption of at most one level. Here we do not go beyond disruptions of type 1, but this approach can be extended, at the cost of much more communication in the negotiation process.

Consider a disruption which puts cell i into disrupted state s_d. The controller at cell i uses some heuristic G to try to find a guaranteed-local action $a \in \mathrm{GA}(s_d)$. If it can find such an action, it will take that action. If not, some communication is necessary for the selection of a good recovery action. Thus, the controller at cell i uses some heuristic H to select a recovery action $a' \in \mathrm{RA}(s_d)$ which seems likely, given local information, not to be very disruptive to other cells. Cell i then sends a proposal message to all members of $M_i(a')$, proposing action a'.

When a cell j receives a proposal message proposing action a', it first determines whether action a' will cause a disruption at cell j. If not, then it returns an ok-0 message. Otherwise, it tries to determine whether the disruption caused by a' will be one of type 0, which can be handled locally. If so, it returns an ok-1 message. Otherwise, it will return a **not-ok** message, perhaps along with some information J which can be used by the disrupting cell's heuristic H to propose a better solution.

When the controller at the disrupted cell receives replies to its proposal, if all replies are **ok-0** messages, then it takes the proposed action. If all replies are either **ok-0** or **ok-1** messages, then the controller can take the proposed action. If there is a **not-ok** reply, then the controller knows that action a' will not isolate the disruption to the cells in $M_i(a')$, so, with whatever information has been gathered, it uses heuristic H again to propose a new recovery action, unless it determines that further negotiation will not be useful.

Here we have considered only the simplest case in which only one cell suffers an initial disruption. This approach is of course only a simple outline of an algorithm for handling this problem. The real issues are what kinds of heuristics G and H are, what kinds of information J is to be exchanged, and what to do when no proposal is acceptable to the other cells. At least some of these answers are domain dependent, and cannot be more fully described in this very general model.

3 Polite rescheduling

3.1 Background

In this section, we consider the use of polite replanning in the domain of scheduling in a cellular manufacturing system. The scheduling domain is chosen to investigate this problem because it contains easily definable interactions among cells, in the form of precedence constraints among jobs. One important aspect of scheduling is the actual execution of an already-constructed schedule (a *preschedule*). One approach to handling unexpected events is dynamic scheduling, in which no preschedule is constructed. All scheduling decisions are made at run-time, by dispatch rules [3], or by least-commitment opportunistic planning [5]. Another approach is to construct a new schedule when events render the old one infeasible. One very fast way of doing this is to "push back" the existing schedule until it becomes feasible. This method is widely used in practice, but very often produces an inefficient schedule.

These approaches, however, do not make use of a good preschedule. We choose instead to follow the matchup scheduling approach of Bean *et al.* [2]. In this approach, when unexpected events disrupt the preschedule, the scheduler attempts to schedule production so that the system can return to ("match up with") the original preschedule. Thus, the good

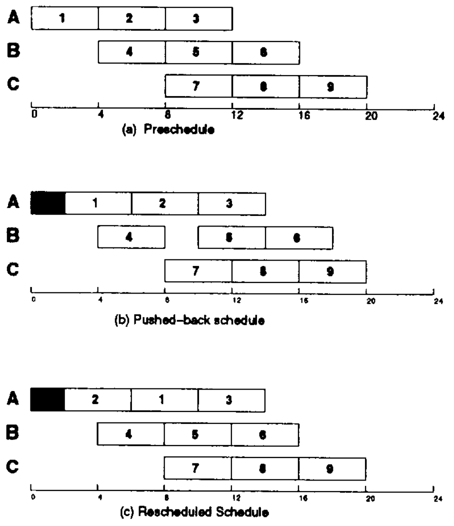

Figure 1: A simple example.

preschedule need not be discarded when disruptions occur.

3.2 Polite rescheduling

Our approach, as previously discussed, is to have local cell schedule controllers reschedule in response to schedule disruption in such a way as to limit the disruption, either to the cell itself, or to a small subset of cells. In order to evaluate various rescheduling approaches, we consider the following class of job shop problems. Each job is to be processed on any machine of one specific cell. Jobs may have successors at other cells; a successor job may start processing only after its predecessor has been completed. We assume that a preschedule has already been constructed for this set of jobs, and that this preschedule tries to minimize the sum of tardiness over all the jobs. Tardiness is a common measure, but minimizing tardiness for even simple problems is NP hard.

We assume that each cell has knowledge from the preschedule about which of its jobs have successor jobs, and these times those successor jobs are scheduled to begin processing at other cells. We call the times the *precedence deadlines* of the predecessor jobs; precedence deadlines are not to be confused with due times. Likewise, each cell has information about which of its jobs have predecessor jobs. However, cells do not have any other information about the schedules at other cells.

In this type of problem, cells interact solely through

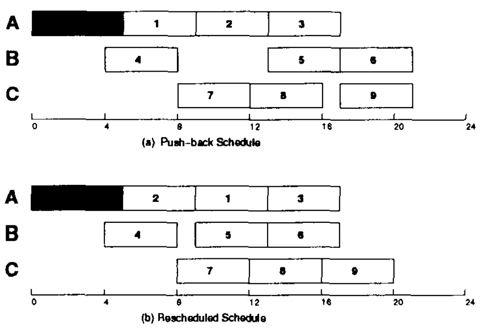

Figure 2: A simple example (cont'd).

precedence constraints among jobs. Consider the very simple example in Figure 1. Here there are three cells with one machine per cell. Here job 2 has job 5 as a successor, which in turn has job 9 as a successor. The preschedule is shown in (a). In (b), the machine at cell A is unable to process any job from time 0 to time 2. Cell A's schedule has been pushed back, disrupting the schedule at cell B because of the late processing of job 2.

The algorithm we propose is based upon the outline described in Section 2.2. When a disruption is identified at a cell, that cell will try to reschedule itself without disrupting schedules at other cells; such rescheduling would be a guaranteed-local recovery action. It will thus try to find a new schedule in which jobs with successors complete processing before their successors are scheduled to begin processing (in the preschedule). If such a non-disrupting schedule can be found, then the cell will attempt to implement a good non-disrupting schedule.

If such a schedule cannot be found, then the cell will try to find a schedule that is likely to be least disruptive to other cells. It then will propose that schedule to the cells which may be affected by it. Each of these other cells will either accept this schedule, if it determines that it can reschedule in response to any disruptions caused by the proposed schedule without disrupting other cells, or reject this schedule, if it cannot determine this. If all of these cells accept the proposed schedule, then the originally disrupted cell will implement it, and the cells disrupted will find and implement new non-disrupting schedules which deal with the disruptions caused by the proposed schedule.

In the simple example described before in Figure 1, while the pushed-back schedule in (b) resulted in a schedule disruption at cell B, the schedule in (c) reschedules cell A without disrupting cell B. In our algorithm, cell A would try to find such a schedule before beginning any negotiations with any other cells. Had the machine of cell A been down from time 0 to time 5 instead, as in Figure 2, then cell A first would try to find a non-disruptive schedule, and would fail because none exists. It then would try to find a schedule least likely to be disruptive to cell B. It would then propose this schedule. Were it to propose the pushed-back schedule in (a) of Figure 2, cell B would not accept the proposal, as it would be unable to avoid disrupting the schedule at cell C. The schedule in (b) of figure 2, if proposed by cell A, would be accepted by cell B, as it can find a non-disruptive schedule to address the late completion of job 2.

3.3 Implementation

The implementation of this algorithm requires at each cell a *negotiator* module and a *rescheduler* module. The negotiator module determines what kinds of schedules to propose to other cells, and determines the priorities through the use of information resident at the node or gathered through communication. The rescheduler produces new schedules according to priorities determined by the negotiator. These priorities determine what kind of schedule will be produced. For example, if a disruption has just been identified, then the initial action will be to try to find a non-disrupting schedule. The priority for such a schedule is for all jobs with successors to complete processing before their precedence deadlines. Except for one machine scheduling, even this simple problem of scheduling to meet deadlines is NP-hard, so the rescheduler usually cannot search for optimal solutions. The following algorithm is a heuristic for finding non-disruptive schedule for a cell with one machine: It uses the modified due date (MDD) heuristic described in [1] which combines aspects of the well-known shortest processing time and earliest due date heuristics, and is good when minimizing tardiness is a goal.

```
1. Schedule predecessor jobs in ascending
   order of precedence deadline;
2. For each predecessor job in order:
   2.1. Order the unscheduled jobs
        by smallest MDD;
   2.2. If there is an unscheduled job
        which can be inserted into
        the schedule without making a
        precedence job late, insert the
        lowest such job and go to 2.1;
3. Schedule the remaining jobs by MDD.
```

The goal of this algorithm is to find a non-disruptive

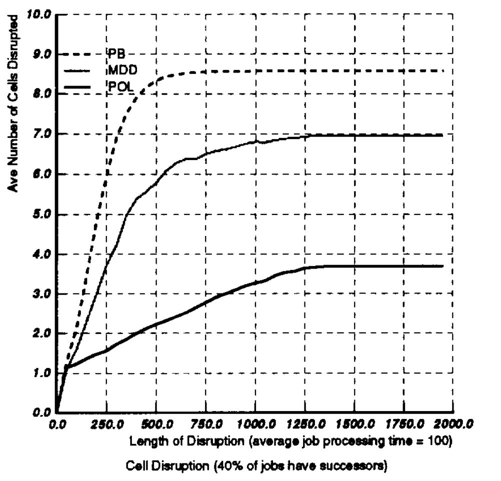

Figure 3: Disruption Propagation

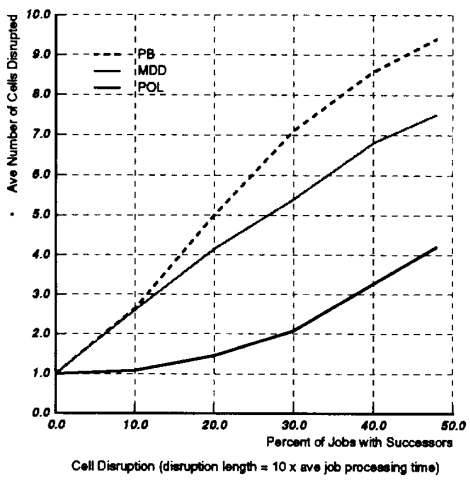

Figure 4: Disruption Propagation

schedule. However, because it uses MDD in scheduling the jobs without successors, it also attempts to find a low tardiness schedule.

4 Evaluation

We evaluate the priority scheduling algorithm through simulation of disruptions in a generic manufacturing system. In these simulations we compare the results from our "polite" priority rescheduling algorithm (POL) with the results from two similarly fast algorithms which do not consider how the rescheduling of one cell may affect another: the pushback algorithm (PB), in which schedules at disrupted cells are simply pushed back, and the MDD algorithm, in which disrupted cells use the modified due date heuristic to reschedule from the point of disruption.

In these simulations, a preschedule is constructed for a generic manufacturing system of ten cells of two machines each. The preschedule is generated from a randomly generated set of 250 jobs. Each job is either a member of a predecessor-successor pair, or else does not any precedence relations with any other job. Only jobs with no successors have due times. In each simulation, one of the machines at one cell is disrupted, and the system is rescheduled using each of the three rescheduling methods described above.

The measures we use to compare these rescheduling methods are the average tardiness for each job in the schedule, and the number of cells disrupted in the rescheduling process. Average tardiness is the measure used in constructing the original preschedule, and it is a measure of the quality of the schedule. The number of cells disrupted is a measure of how disruptive the rescheduling process is to the manufacturing system. When a schedule of a cell is disrupted, the work in that cell is disrupted not only because of the need to reschedule, but also because the other schedules such as those governing transportation of parts and availability of tools may also be disrupted.

Figure 3 shows the number of cells eventually disrupted from the propagation of one original disruption, versus the length of the original disruption. These results show that the priority scheduling algorithm isolates disruptions much more than the other two rescheduling methods. Figure 4 shows the number of cells eventually disrupted from the propagation of one original disruption versus the proportion of jobs with successors. These results demonstrate that isolating disruptions becomes harder as scheduling constraints become tighter.

Figure 5 shows the average tardiness over the whole schedule versus the length of the original disruption. These results show the cost of attempting to isolate disruptions. The priority scheduling algorithm has a higher tardiness cost because it emphasizes meeting precedence constraints over meeting due dates. As shown in figure 6, for the originally disrupted cell, the priority scheduling algorithm has a much higher tardiness cost than the MDD method. However, as shown in figure 7, for the *other* cells, it has a smaller tardiness cost, because of the effects of the original disruption on remote cells has been reduced.

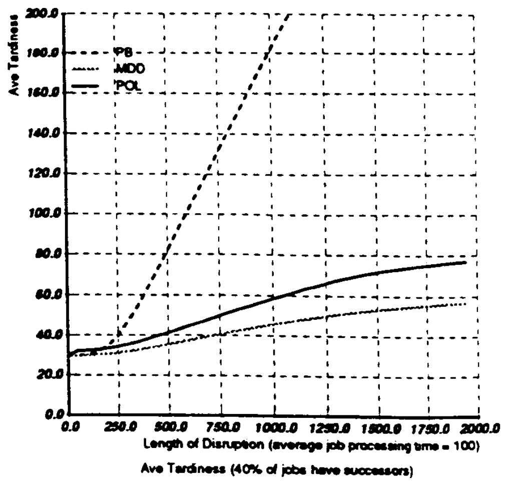

Figure 5: Tardiness

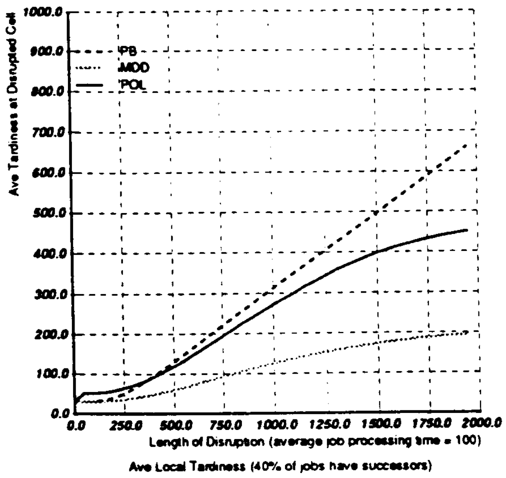

Figure 6: Tardiness at Remote Cells

5 Summary

We have presented a new approach to handling schedule disruptions in a distributed manufacturing system. Our approach takes into consideration the possibility that responding to a disruption in one part of the system may cause disruptions in other parts of the system. "Polite rescheduling" thus attempts to respond to disruptions local to one manufacturing cell so that other cells are disrupted as little as possible. Our preliminary results show the advantages of using a scheduling algorithm that emphasizes precedence constraints over due dates. Our future work will further explore and evaluate these and other applications of DAI techniques to this area.

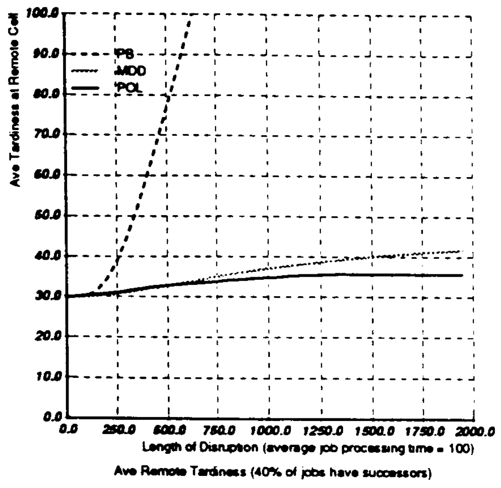

Figure 7: Tardiness at Remote Cells

References

[1] K. R. Baker and J. W. M. Bertrand. A dynamic priority rule for sequencing against due dates. *J. Opns. Mgmt.*, 3:37–42, 1982.

[2] J. C. Bean et al. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483, May-June 1991.

[3] J. H. Blackstone, D. T. Phillips, and G. L. Hogg. A state-of-the-art survey of dispatch rules for manufacturing job shop operations. *International Journal of Production Research*, 20(1):27–45, 1982.

[4] S. E. Conry et al. Multistage negotiation for distributed constraint satisfaction. *IEEE Trans. on Systems, Man, and Cybernetics*, 21(6):1462–1477, November 1991.

[5] P. S. Ow and S. F. Smith. Viewing scheduling as an opportunistic problem-solving process. *Annals of Operation Research*, 12:85–108, 1988.

[6] K. Sycara et al. Distributed constrained heuristic search. *IEEE Trans. on Systems, Man, and Cybernetics*, 21(6):1446–1461, November 1991.