Task Rescheduling in Multi-Agent Manufacturing

Martyn Fletcher S. Misbah Deen
Computer Science Department, University of Keele, Staffs., England.
{martyn,deen}@cs.keele.ac.uk

Abstract

We present a model for task rescheduling in multi-agent manufacturing that is geared toward maximising system efficiency and reliability in an environment with predictable failure patterns. We describe a distributed manufacturing system architecture [1] to illustrate the merits of rescheduling. We also develop an action redistribution framework based upon resource cooling, ie migrating actions from the most utilised or faulty (hottest) agents' resources to the coldest (least constrained) ones. Furthermore, the paper discusses the costs versus benefits of resource cooling when rescheduling cascades over parallel resources executing a decentralised production task.

1. Introduction

Task rescheduling plays an important role in the performance and robustness of distributed manufacturing systems with dynamic failure patterns. In this environment, it is highly desirable to invoke an on-line rescheduling scheme in which the schedule modifications are executed concurrently with regular production actions [10]. Thus, in contrast to off-line rescheduling, which is performed while locking out regular manufacturing actions, on-line rescheduling can be undertaken incrementally as a higher priority action [8] inside a market economy [2]. In a centralised manufacturing system, task rescheduling is traditionally performed to maximise throughput. In contrast, the fundamental objective of task rescheduling in multi-agent manufacturing is to provide flexible criterion, for instance minimise disruption or balance workload among agents' resources. We assume that: (i) each task has its constituent actions distributed across resources for concurrent execution; and (ii) every agent performs the management function, ie schedules and migrates actions, for one or more parallel resources under its control.

We present a new model for task rescheduling in multi-agent manufacturing systems which enables the agent to independently determine, at a given instance in time, whether a rescheduling action is cost beneficial given that the rescheduling itself imposes an additional load on the system. Rescheduling is executed dynamically by employing an incremental task migration scheme called resource cooling that moves tasks away from 'hot' resources (ie heavily utilised, constrained or recently damaged by some failure) to 'cold' ones (having more available timeslots to complete processing the task's actions).

Section 2 describes a manufacturing system architecture for studying task rescheduling in the presence of unbalanced resources' workloads and system crashes [6]. Section 3 incorporates our framework into the architecture to determine the optimal task redistribution strategy based upon workloads observed at parallel resources and timing the migration scheme's retrieve and perform phases during rescheduling. As part of the model, Section 4 presents a method for handling rescheduling cascades across multiple resources. Section 5 compares the model with current literature, and finally Section 6 summarises the paper.

2. Manufacturing System Architecture

We envisage a multi-agent manufacturing system architecture (illustrated in Figure 1) would integrate the entire spectrum of activities from order booking, through workpiece design, production and marketing to realise the agile enterprise [3].

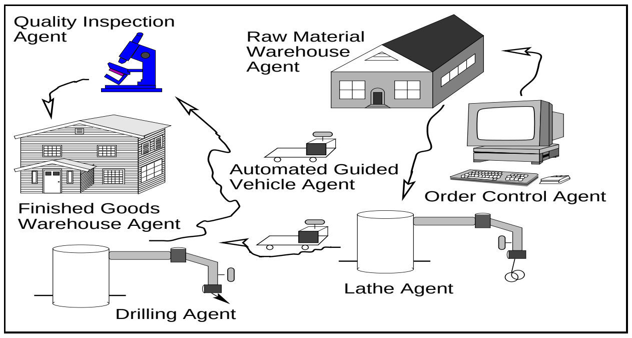
Quality Inspection Agent
Raw Material Warehouse Agent
Automated Guided Vehicle Agent
Order Control Agent
Finished Goods Warehouse Agent
Lathe Agent
Drilling Agent

Figure 1. Manufacturing System Architecture.

Furthermore every agent is defined as an "intelligent" autonomous system that cooperates for transforming, transporting, storing and validating information and physical workpieces. Multi-agent systems provide a flexible but largely unmanned environment for low-volume high-variety manufacturing, where each agent manages its own activities without centralised control. Such agents include stationary machining cells, mobile Automated Guided Vehicle (AGV) robots, expert systems and so on. The internal structure, in terms of functional methods and resources' queues, of an agent is depicted in Figure 2.

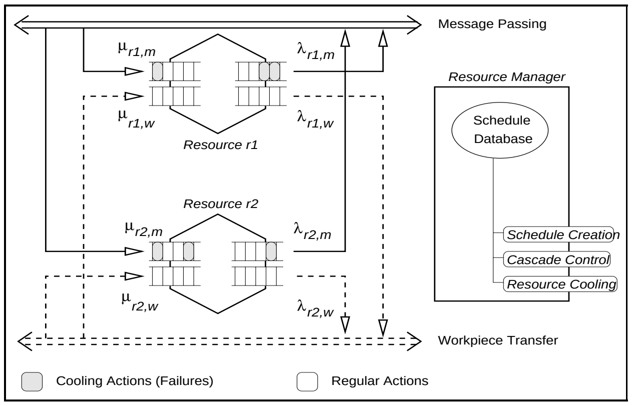

Figure 2. Internal Structure of an Agent.

In our manufacturing system, each agent locally denotes the schedule as a database tuple $\{\ k, a \times j, s \times r\ \}$ indicating that task k's action a is to be executed in timeslot s of interval $j (\leq m)$ upon resource r. The agent's resource manager assigns actions to its resources' timeslots based on traditional scheduling algorithms using criteria, such as priority, arrival time or number of slots required. Rescheduling is demanded due to failures within this distributed system, eg agent breakdowns. This is performed by a redistribution framework, encoded as a cooperation strategy [4] in a suitable agent-based specification language like KQML.

The framework's resource cooling algorithm is invoked either whenever a suitable change is made to the schedule database's state, or at fixed time intervals. The algorithm initially verifies that its trigger conditions are satisfied, including the present workload. If the aggregation of these parameters does not reach some threshold then the system is considered to be balanced and thus no corrective action need be taken.

3. Action Redistribution Framework

Manufacturing a product (ie a task) is decomposed into a hierarchy of related actions $\{a\}$, with p levels and each level having, on average, q sub-actions. Configuring and (re)scheduling such atomic actions into intra- and inter-agent tasks plays an important role in any multi-agent system, especially manufacturing due to its fault-tolerant requirements. For instance machine actions and associated resources are managed as project networks (see Figure 3) during planning and execution so that: completion time is minimised; and reaction to hardware and software faults is quick. As the number and complexity of production tasks increases, so does the probability of failure. Hence agents must decide how they adapt combining actions so skills are continuously available.

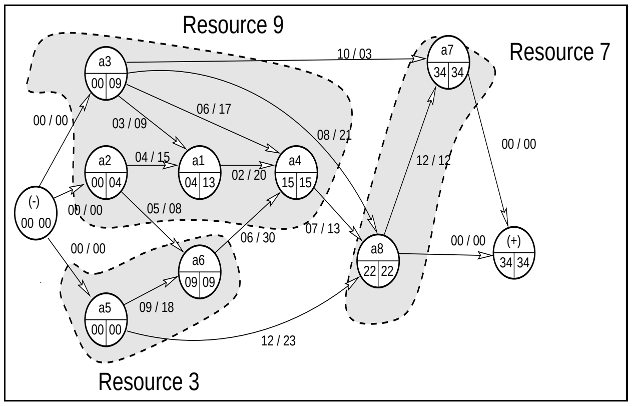

Figure 3. Project Network of Actions in a Task.

We are implementing a resource manager for multi-agent manufacturing systems that can perform schedule control on a task-specific basis while adhering to global constraints and preferences. This control is executed as desired by the product design and the factory configuration to achieve workload equalisation and aid robustness by the discrete allocation and redistribution of actions. To facilitate such load balancing, maintain agent availability and so perpetuate graceful degradation as workload nears the system's saturation, our resource manager tracks statistics [11] on:

- The temperature of actions in a task, ie the ratio between an action's latency and resource r's capacity.
- The latency of actions and resources, determined as a function of timeslots consumed by the task, how many resources the task is distributed over, its priority and the complexity of its inter-action dependencies.

Thus the temperature of action a in task k at time $t_{j,s}$ is:

$$T_r^{k,a}(t_{j,s}) = \left(\frac{Y^a}{N^a}\right) \times \int_{j,s}^{N^a} \frac{e^{-t} \times SRL_r}{c_r^a(t_{j,s}) + m_r^a(t_{j,s}) + q_r^a(t_{j,s})} dt$$

Where $c_r^a(t_{j,s})$ is the cost to resource r of performing action a; $m_r^a(t_{j,s})$ is the number of messages passing along the communication network to execute the action; N^a is the initial number of timeslots to be scheduled; Y^a is the number of timeslots required to complete the action; and $q_r^a(t_{j,s})$ is the reduction in the resource's quality in performing the action during timeslot j, s. Temperature tends to zero because Y^a decreases as the action is executed. P^k is the task's priority. The aforementioned formula ensures that the request patterns of tasks, and hence resources, can be dynamic over time. In practice, we have encountered several cases which display periodical and predictable patterns of regular actions; however the flexibility awarded by the formula can prove critical in unpredictable environments where, at present, the intervention of human expertise is demanded. In our rescheduling model, such patterns (possibly expressed as neural nets) can be incorporated by identifying the number of time intervals such that the resources' latency remains constant over some fixed duration, but is allowed to vary across these intervals. We define the skewed latency of action a in task k to be:

$$STL_{k,a} = P^k \times \sum_{j=1}^{m} L_{k,a,j} \times (t_j - t_{j-1}) \times \delta_{k,a,j,r}$$

$t_j - t_{j-1}$	is the duration of interval j, holding n_j slots.
$L_{k,a,j}$	is the latency of task k's action a in interval j.
$\delta_{k,a,j,r}$	is the weighted factor of performing action k, a on resource r during interval j.

Correspondingly, the skewed latency of resource r is:

$$SRL_r = \sum_{j=1}^{m} L_{r,j} \times (t_j - t_{j-1}) \times \psi_{r,j}(\sum^{k} \sum^{\forall a \in k} L_{k,a,j})$$

Where $L_{r,j}$ is the latency of resource r in interval j and $\psi_{r,j}(\sum^k \sum^{\forall a \in k} L_{k,a,j})$ is the weighted latency applied to executing a number of tasks' actions concurrently during that interval. $L_{r,j}$ is computed as the accumulated latency of all actions that reside upon resource r.

3.1. Resource Cooling Algorithm

To facilitate dynamic action redistribution across resources, a workload equalisation algorithm called *resource cooling* is employed at the commencement of each interval. Fundamentally, resource cooling is a distributed greedy algorithm executed simultaneously, by the resource manager, at each agent. The algorithm determines the optimal candidate action(s) and their destinations to transfer from the most utilised or recently failed resource (with the highest SRL). In our manufacturing domain, there are two types of action: *regular actions* (ie scheduled manufacturing tasks) and *cooling actions* (eg failure alarms causing the changing of execution times for regular actions or the location where they are to be performed). In such a scenario, it maybe beneficial to postpone rescheduling to a later interval as fewer regular actions are affected.

This migration scheme minimises the number of regular actions transferred while maximising system gain, with respect to latency equalisation across resources, in which a substantial proportion of actions exhibit a periodic pattern of request characteristics. Resource cooling comprises two stages to ensure consistency: a retrieve phase, where the hottest resource is determined and its schedule database (representing the action request queue) is queried; and a perform phase, where the candidate actions are transferred into the cooler resources' queues. However, the retrieve phase of migration, and thus the entire cooling algorithm, is not undertaken if the action request queue of the hottest resource is not empty because initiating a rescheduling action then would probably increase the workload imbalance. The migration scheme's retrieve algorithm cycles these steps:

1. Each agent ag_z senses S_z from the current environmental state S (*viz* latencies of agent ag_z's resources and tasks' temperatures) where $S_z \subset S$.

2. The hottest ag_x and coolest ag_y agents are determined based on maximising and minimising $\sum_{r \in S_z} SLR_r$.

3. The hottest agent's *cooling action* set, $\vartheta(S_x) = \cup_{a \in k} \{T_r^{k,a}(t_{j,s})\}$ is created by its resource manager. The number of actions within $\vartheta(S_x)$ is less than or equal to the number of actions within the task k scheduled for execution on r. This is essential in determining the *action contexts* generated for the bidding process.

4. The *action context* set is constructed as the collection of actions $\{k, a\}$ that can be migrated onto agent y's $r2$ resources with compatible skills for greatest return on investment. Formally AC is defined as:

   $$AC \subset (\vartheta(S_x) - L_{r2,j}) : sk_{x,r1}^{k,a} = sk_{y,r2}^{k,a} \wedge RoI(t_j) > 0$$

5. A corresponding tuple for each action within AC is tagged in ag_x's schedule database and a bid is made to ag_y. The bid contains a tuple $\{ag_x \times S_x \times k, a \times RoI(t_j) \times sk_{x,r1}^{k,a} \times SLR_{r1}\}$. The tuples' deletion and insertion are confirmed in the perform stage.

6. Finally, each agent learns by adjusting its environmental state (eg $RoI(t_j)$ estimates) for future task rescheduling.

The aggregate temperature of actions on a given resource is utilised as an indicator of tasks to redistribute because this metric reflects the cost/benefit ratio incurred by reallocation. The candidate action(s) of the task with highest $STL_{k,a}$ upon the hottest resource are thus selected, during the perform phase, to be moved to the resource(s) with least skewed latency that posses the equivalent skill, $sk_r^{k,a}$. There is no global control indicating the latency or temperature of resources, so one agent in the system executes the above retrieve phase of the migration scheme by applying a Contract Net Protocol [9] to determine the hottest and cooler resources. Additional constraints are also observed in selecting the resource onto which the action may be moved, including availability of AGVs to transport sub-assemblies resident in the hottest resource's inbound workpiece buffer.

3.2. Return on Investment

The benefit of the cooling action is realised by examining the workload and resilience measures for candidate resources before and after rescheduling. This return on investment, denoted as $RoI(t_j)$, is calculated as $SLM(t_j) - SLM(t_{j+1})$ where $SLM(t_{j+1})$ is the system latency median that potentially would be associated with the next time interval if the task's actions were migrated to the cooler resource(s). $SLM(t_j)$ is defined as:

$$SLM(t_j) = \sum_{j=1}^{m} \sum_{r \in \mathbf{R}} (\Delta_j - L_{r,j})^2 \times (t_j - t_{j-1})$$

Where $\mathbf{R}$ is the set of parallel resources in the system and $\Delta_j (= \sum_{r \in \mathbf{R}} SRL_r)$ is the median resource latency, over all resources, at time interval j. It should be noted that an action maybe repeatedly moved from one resource to another as a result of the resource attempting to cool itself. To overcome this anomaly, each task is tagged with the resource's unique identity whenever it is transferred so that: i) for fairness, the action cannot be bounced *ad infinitum* between resources without completion; and ii) the action is never passed to a resource which does not posses the correct skill. Observe that if the $RoI(t_j)$ is negative, ie the migration causes greater latency among resources, then the transfer is obviously aborted. Moreover for the redistribution to proceed, $RoI(t_j)$ must exceed the extra cost $C(t_j)$ caused by the reorganisation itself, in terms of message volume and processing, and any cooperation with AGV robots.

In terms of $C(t_j)$, we have implemented several cost model procedures based on different formulae and action/failure patterns. A feasible cost model has formulae:

$$C(t_j) = \sum_{c \in \mathbf{R} \neq h} \sum_{j=1}^{m} \sum_{s=1}^{n_j} g(\Theta_h^{k,a,j,s}, \Theta_c^{k,a,j,s})$$

where:

$$g(\Theta_h^{k,a,j,s}, \Theta_c^{k,a,j,s}) = \begin{cases} 1 & \text{if} T_h^{k,a}(t_{j,s}) > T_h^{k,a}(t_{j,s-1}) \\ \eta & \text{if} T_c^{k,a}(t_{j,s}) \leq T_c^{k,a}(t_{j,s-1}) \\ 0 & \text{otherwise.} \end{cases}$$

In the above formulae, $C(t)$ represents the total temperature difference for executing the action upon the coolest resource c rather than the hottest resource h, over the timeslots in the next m intervals. These equations operate in conjunction to determine the optimal action(s) to transfer to the cooler resource(s) at each interval via calculating the maximum spread between cost and benefit.

A cooling action is only performed if the resource manager estimates that the potential benefit of migrating one or more of its allocated tasks significantly outweighs (namely exceeds coefficient η) the additional cost incurred by moving the actions. Both these risk and return measures take into account the predicted action and failure patterns. To judge the benefit/cost of a cooling action, the hottest resource applies its view of the system latency median as a subjective function. This provides an effective measure of the random action request arrivals when their relative frequencies is asymetric and the distribution is unknown. An objective function would be better, however retaining an accurate and timely latency model for each resource in the decentralised system is practically impossible.

4. Rescheduling Cascades

To tolerate malfunctions at the resource, we distinguish between an action's *scheduled time* $[t_{sch}(k,a) = t_{arr}(k,a) - t_{dep}(k,a)]$ which is action k, a's arrival time in the queue minus its departure time; and its *operational time* $[t_{ope}(k,a) = t_{beg}(k,a) - t_{end}(k,a)]$ that is action k, a's begin time minus its end time. If resource r fails at t_{fail} and recovers by t_{rec}, the breakdown occurs in one of three scenarios:

- Fails while *active*, ie $t_{beg}(k,a) < t_{rec} < t_{end}(k,a)$.
- Fails while *inactive*, ie $t_{beg}(k,a) \geq t_{rec} \geq t_{end}(k,a)$.
- Fails while *neutral*, ie $t_{arr}(k,a) \geq t_{rec} \geq t_{dep}(k,a)$.

When r fails at time t_{fail}, all actions with $t_{end}(k,a) \leq t_{fail}$ must have completed, so the number of complete actions in task k upon r is:

$$u_r(k) = 1 - \sum^{\forall a \in k} \left(\frac{t_{end}(k,a) - t_{fail}}{t_{ope}(k,a)} \times \frac{m^{l-1}}{m-1} \right)$$

There are some actions $\{a_{due}\}$ which, in task k's original schedule, are due to start after t_{rec}. These actions are divided into: (i) those with precedent actions $\{a_{pre}\}$ that are delayed, namely new $\forall a \in a_{pre}$ such that $t_{end}(k,a) > t_{beg}(a_{due})$; or (ii) those not delayed. This is cascade shown in Figure 4. If $(t_{dep}(a_{post}) - t_{rec}) > t_{ope}(a_{post})$ then there is sufficient time to complete $\{a_{post}\}$ within the existing schedule. Lower level actions may need rescheduling, but upper level actions will be unaffected.

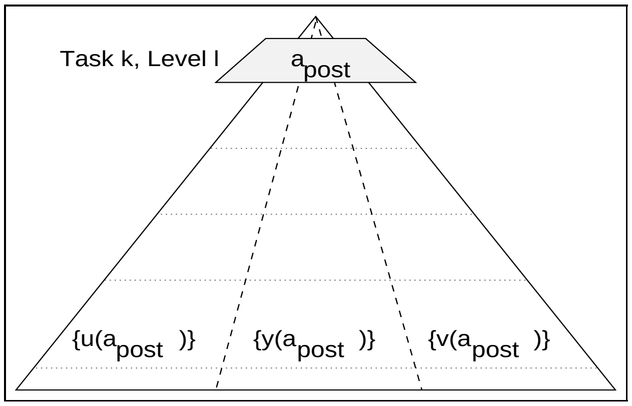

Figure 4. Rescheduling Cascade.

If condition $\Gamma \rightarrow (t_{dep}(a_{due}) - t_{ope}(a_{due})) \geq t_{rec}$ does not hold for lower resources of $\{a_{due}\}$, ie due to having less operational time, then r must reschedule $\{a_{due}\}$. This rescheduling is performed initially upon r and if unsuccesful, due to an inadequate number of timelots, then across the members of **R**. $\{v(a_{post})\}$ is the set of lower level actions *not* requiring rescheduling. In these worst case, the agent must reschedule all lower level actions in $\{a_{post}\}$, namely set $\{y(a_{post})\}$ such that $|\{y(a_{post})\}| = \{u(a_{post})\} \times \frac{m^k-1}{m-1} \times \{v(a_{post})\}$. The cooperation strategy used by an agent to recover an action is:

1. Query a system directory or agent's local cache for suitable resources offering the required skill.
2. Agent requests each resource for available timeslots.
3. Select the resource based on criteria such as geographic proximity or availability.
4. Agent sends the task's details to the selected resource.
5. The resource acknowledges receipt of the action(s) and schedules their execution.

So $|\mathbf{R}| + 3 \times u(k) + 1$ messages are required to recover $u(k)$ actions over l levels. However, an optimised technique could avoid requesting recovery data from an agent more than once, or not querying the availability of resources.

5. Literature Comparison

Our model differs in a number of aspects from other studies proposed in the literature for determining optimal rescheduling in a centrally-controlled manufacturing system [8]. Although the cost and availability of resources were considered, rescheduling was viewed as occurring instantly thus having no discernible affect upon the overall system load. Moreover, these approaches assumed that there are negligible problems associated with the physical transport of workpieces. The model proposed here is an adaption of the earlier holonic system architecture and generic agent model introduced in [5] [4]. The framework used in [5] did not consider irregular task rescheduling patterns as a consequence of resource breakdowns. In contrast, in [4] we considered fault tolerance but all task rescheduling was viewed as the execution of heterogeneous cooperation strategies. Contrary to earlier approaches, we view the retrieve phase of task migration as being triggered when the hot resource's action request queue is empty, while the perform phase is considered as higher priority requests at the corresponding cool resource.

Some traditional multi-agent systems [9] have the potential to modify their interaction and control strategies, however this normally implies suspension of actions during the re-negotiation. This results in a failure causing a task to terminate prematurely even when agents or the network are not directly affected by the fault. Consequently, a manufacturing system that cannot react to reconfiguration demands or tolerate faults may suffer poor availability. Applying the on-line redistribution framework overcomes this anomaly.

6. Results and Conclusions

To illustrate the model's merits and how it could be used by practitioners, we present initial results of a simulation. These experiments use metrics similar to [7], namely mean response time (task view), the effect of global load balancing (system view) and resource utilisation (agent view). We assume: (i) the system has 20 agents, each with 5 resources; and (ii) the simulation is run over 50 time intervals of 10 slots each (1 slot = 60 seconds). The task load is modeled as a Poisson distribution at all resources, while failures are denoted as an exponential distribution. Note that the thresholds, weights and other attributes are set to appropriate values to maximise performance with respect to $SLM(t_j)$.

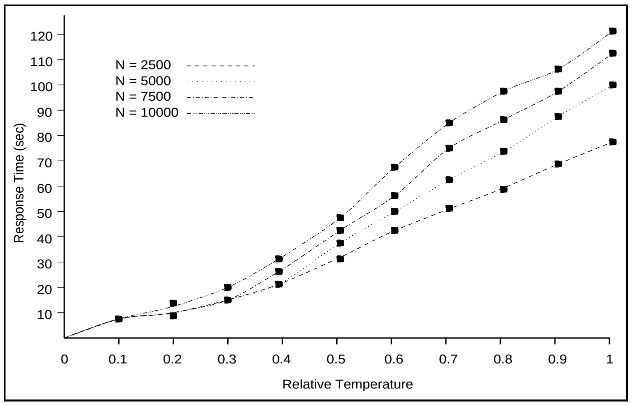

Figure 5. Response Time (Task View).

Figure 5 shows the results of the mean response time for each task when N tasks are executing concurrently within the manufacturing system. The response time is calculated as the difference between actual execution time over multiple resources and the expected duration if the task was executed on a single resource. It shows that the proposed migration algorithm yields substantial benefits in performance when relative system temperatures of 0.6 are exceeded.

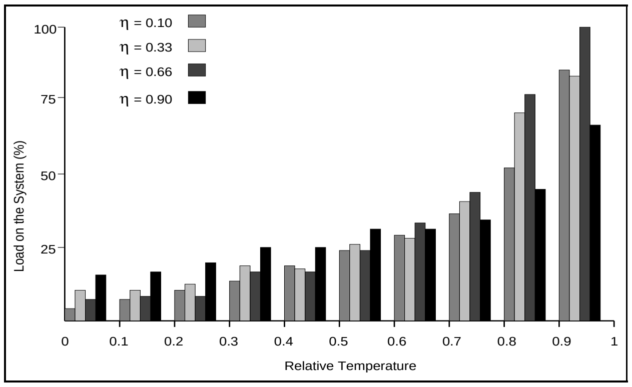

Figure 6. Load Balancing (System View).

In our proposed action migration algorithm, each resource endeavours to transfer its hottest task to the coolest resource while taking priority etc into account. Therefore, the rate at which actions are passed between resources increases in an attempt to balance the utilisation across all agents's resources. This process continues until when the system is so heavily loaded that a saturation point is reached when a transfer does not reduce the overall system's utilisation. Due to the blind migration, ie not taking into account the future schedule of the resource to which an action is passed, a natural equalisation will evolve at the expense of increased system instability. Figure 6 illustrates these issues by showing how the average load across all agents differs as a consequence of the migration algorithm using low and high movement coefficients, namely $\eta \in (0, 1)$ respectively.

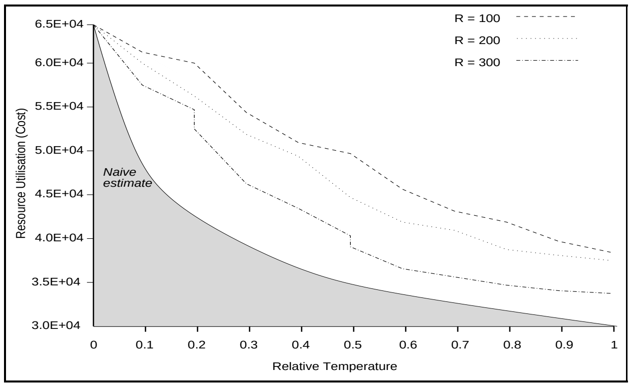

Figure 7. Resource Utilisation (Agent View).

Again the migration algorithm proved to be instrumental in substantially increasing resource utilisation with respect to agents' rescheduling cascades (see Figure 7). Efficient coordination and information exchange between agents results in a 70% increase in the average utilisation of resources. From these experimental results, we can deduce that our model generates desirable resource utilisation and system balancing characteristics in the face of predictable failure patterns. Tasks' response times are also improved.

Summarising, the paper has presented a model of task rescheduling in multi-agent manufacturing that facilitates independent and concurrent reorganisation of actions with respect to their execution timings and resource locations while maintaining inter-action dependencies inside a task. Our model optimises resource availability and so equalises workload under the constriction that resources have different skills and schedules. We are currently extending our algorithms in two directions. Firstly, to handle production environments containing unpredictable action and failure patterns. Secondly, to synchronise schedules across multiple agents at recoverable checkpoints. We are also investigating how the system could reorganise a plan during execution. This material is based on work supported by the European Union programme (EU IMS Project No: 26508) on HMS.

References

[1] Holonic Manufacturing Systems Programme Overview. http://hms.ifw.uni-hannover.de/public/overview.html, 1998.

[2] J. Carlson. Risk Aversion, Foreign Exchange Speculation and Gamblers' Ruin. *Economica*, 65(259), August 1998.

[3] J. Christensen. Holonic Manufacturing Systems: Initial Architecture and Standards Directions. In *First European Conference on Holonic Manufacturing Systems*, 1994.

[4] S. Deen. A Fault-Tolerant Cooperative Distributed System. In *International DEXA Conference*, 1998.

[5] M. Fletcher and S. Deen. Fault Tolerance in Holonic Manufacturing Systems. In *IEEE International Workshop on Embedded Fault-Tolerant Systems*, 1998.

[6] D. Jarvis. Quality Control for Holonic Manufacturing Systems. Technical report, CSIRO, Australia, 1998.

[7] G. Lee. An Adaptive Load Balancing Algorithm Using Simple Prediction Mechanism. In *the Ninth International Workshop on Database and Expert Systems Applications*, 1998.

[8] P. Luh. Scheduling of Manufacturing Systems Using the Lagrangian Relaxation Technique. *IEEE Transactions on Advanced Computing and Automation*, 1993.

[9] R. Smith. The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, C-29(12), 1980.

[10] N. Sugimura. A Study on an Integrated Process Planning and Scheduling System Based on Shape Generation Functions. Technical report, Osaka University, Japan, 1998.

[11] P. Zabback. Database Reorganisation in Parallel Disk Arrays with I/O Service Stealing. *IEEE Transactions on Knowledge and Data Engineering*, 10(5), September 1998.