Title:

Centralized Task Distribution in the Presence of Uncertainty and Time Deadlines

Authors:

Satoru Fujita
satoru@swl.cl.nec.co.jp
C&C Research Laboratories, NEC Corp.
1-1, Miyazaki 4-Chome, Miyamae-ku
Kawasaki, Kanagawa 216, Japan
Victor R. Lesser [1]
lesser@cs.umass.edu
Box 34610, Lederle Graduate Research Center
University of Massachusetts
Amherst, MA 01003-4610, U.S.A.

Abstract:

This paper focuses on a deadline-based multiagent task decomposition/scheduling of hierarchical task structures in an environment in which one agent is responsible for assigning subtasks and their execution order to a set of agents. The overall task to be decomposed is specified in an extended version of TAEMS task modeling framework in which task quality and execution time are uncertain before execution and are represented in the form of a probability distribution.

The master agent generates and assigns an initial subtask schedule to agents, and if necessary will revise agent schedules whenever agents make rescheduling requests. These requests are based on the degree of deviation that has occurred in expected quality and duration as a result of actual method execution. Evaluations of scheduler performance indicate a phase transition of rescheduling efficiency with respect to problem difficulty. Specifically, while frequent rescheduling was ineffective for both very easy and very difficult problems, it was extremely effective in the transition region (mid-range) of difficulty.

Topic Areas: Cooperation & Coordination; Distributed Resource Allocation

---

[1]The material in this paper is based upon work partially supported by the National Science Foundation under Grant Nos. IRI-9321324 and IRI-9523419. The content of the information does not necessarily reflect the position or the policy of the U.S. Government, and no official endorsement should be inferred.

Centralized Task Distribution in the Presence of Uncertainty and Time Deadlines

Abstract

This paper focuses on a deadline-based multiagent task decomposition/scheduling of hierarchical task structures in an environment in which one agent is responsible for assigning subtasks and their execution order to a set of agents. The overall task to be decomposed is specified in an extended version of TAEMS task modeling framework in which task quality and execution time are uncertain before execution and are represented in the form of a probability distribution.

The master agent generates and assigns an initial subtask schedule to agents, and if necessary will revise agent schedules whenever agents make rescheduling requests. These requests are based on the degree of deviation that has occurred in expected quality and duration as a result of actual method execution. Evaluations of scheduler performance indicate a phase transition of rescheduling efficiency with respect to problem difficulty. Specifically, while frequent rescheduling was ineffective for both very easy and very difficult problems, it was extremely effective in the transition region (mid-range) of difficulty.

Topic Area: Cooperation & Coordination; Distributed Resource Allocation

1 Introduction

This work focuses on the problem of a deadline-based multiagent task decomposition and scheduling (MADS) of medium grain problem-solving activities in an environment in which one (master) agent is responsible for assigning subtasks and their execution order to a set of (slave) agents. The master agent decides not only how to partition subtasks among the slave agents but also how many agents are required in order to meet the performance goals associated with the high-level task. The master agent is also responsible for rescheduling the slave agents when the master agent is notified that agent execution of activities does not conform to a priori expectations, and integrating the completed partial result of agents into a overall solution to the high-level task. Slave agents, where necessary, transmit intermediate results to other slave agents so as to enable or facilitate execution of their activities. Figure 1 shows the interaction among the master and slave agents.

This approach to task decomposition is obviously different from a negotiation model of task decomposition such as the contract net approach [12] where the task executing agents have more autonomy in deciding what activities they will execute. It also differs from that approach because part of an agent's task assignment may require the agent to

transmit intermediate results not to the contracting agent but rather to another agent before a specific time. This possibility was foreseen in a later paper by Davis and Smith [1] in which, once a task decomposition is established there can be a "result sharing" phase, but to our knowledge this result sharing and its timing was never a consideration in any of the work on actual implementations of the contract net algorithm. Another approach to task decomposition is the one developed by Durfee and Montgomery [8]. They represent the overall task using a hierarchical behavior model and are concerned with finding appropriate abstractions in the behavior space that motivate the decomposition process. Our model of task decomposition is probably closest to the work by Durfee and Lesser on the implementation of hierarchical organizational structures in PGP [7]. However, in that work, the decomposition of activities to agents is given a priori and the focus is on how to appropriately schedule the activities and to whom intermediate results should be sent. In our work, we not only do that but also decide on which activities to assign to specific agents and determine how many agents are required.[2]

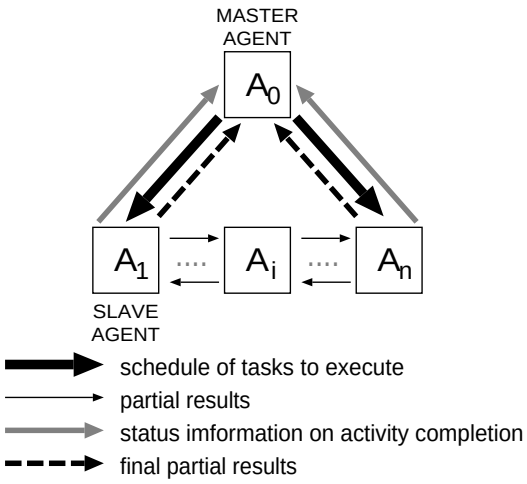


Figure 1: Centralized Task Distribution and Monitoring

We also base these decisions on a more complex and domain-independent model of a task structure than the one used in their work. A key aspect of this model is the explicit declaration of the expected distribution of the time for subtask completion and of the quality achieved. This information together with time deadlines and criterion

[2]Another way of viewing this work is that the MADS algorithm is another coordination module that can be incorporated into the GPGP architecture [4]. From that perspective, it extends the range of agent organizational structures that can be constructed using GPGP.

on acceptable satisfaction for the overall task is used to generate agent task schedules. As part of this process, the possibility of subtask failure is taken into account so that multiple ways of achieving a task may be incorporated into the agent decomposition decision to increase the likelihood of meeting the overall task objectives. In this way, our task structure framework can represent problem-solving activities involving coarse grain search processes in which the search involves a small number of alternative paths.

In addition to our discussion of the MADS algorithm that we have developed for task decomposition and scheduling, we will also present simulation results concerning under what conditions the master agent needs to re-decompose and reschedule agent activities. This issue of rescheduling also relates to issues that Durfee and Lesser raised in their work on the balance between predictability and responsiveness in making coordination decisions [6]. As will be seen, our work represents a much more detailed and extensive investigation of this issue based on the expected distribution of task duration and quality.

The remainder of the paper is structured as follows. Section 2 presents our representation for the task structure that the master agent is responsible for partitioning and scheduling on slave agents. Section 3 details the algorithm used for this process. Section 4 discusses experimental results and a phase transition based on an implemented version of the algorithm. The paper concludes with a discussion of future work and summarizes the contribution of this work.

2 Model

2.1 Task Representation

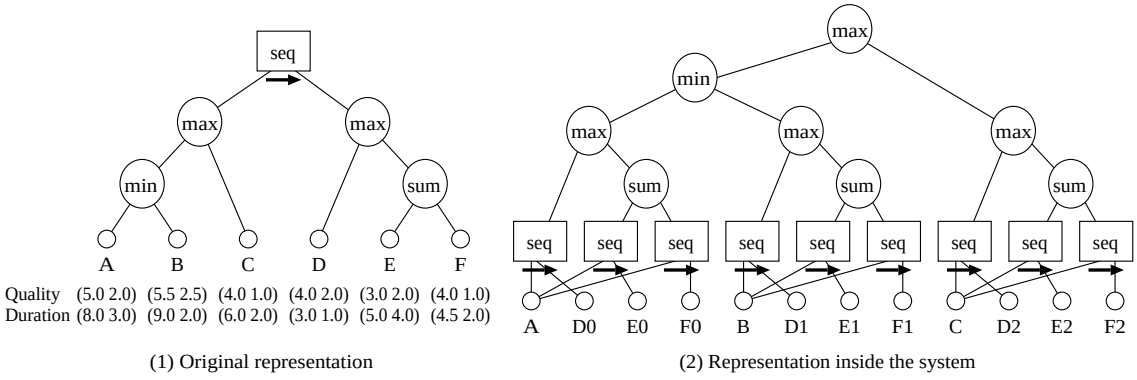


Figure 2: An example task structure

We used an extended version of the TAEMS task representation framework, developed

in [2], to represent the problem-solving task that the master agent has to partition. Figure 2 shows an example task structure that will be used in the simulations discussed in section 4. The task structure consists of basic method nodes and high-level task nodes that specify how to combine the quality achieved by subtasks. In addition to the *sum, max,* and *min* quality accumulation functions specified in the TAEMS formulation, we add a new function *seq*, which indicates that the quality is only accumulated by the final child subtask, and a task just after a *seq* node uses the output of the task just before the *seq* node as an input. Figure 2(2), which is automatically obtained from Fig.2(1), is a representation inside the system; D0 stands for the D that follows A, D1 stands for the D that follows B and D2 stands for the D that follows C; E0, F0 and the rest follow the same notations. Additionally, we explicitly indicate the distribution of the expected values. In Fig.2(1), numbers in parentheses represent the parameters for normal distribution of quality or duration, such as average and standard deviation. For instance, quality column of method A is (5.0 2.0), which stands for the average of 5.0 and the standard deviation of 2.0.

2.2 Goal Criteria for High-Level Tasks

In this model which involves tasks with performance profiles and where the number of slave agents is variable, the specification of the goal criteria for the task allocation and scheduling process needs to be more complex. For example, optimizing expected quality within available time and resource requirements is only one possible choice. In this section, we specify a range of goal criteria that the MADS algorithm can target when trying to find the "best" schedule.

In addition to specifying a deadline when an answer to the high-level task is required, our goal representation allows the specification of the required overall task quality in two ways. One is based on a percentage of the likely quality that would be achieved by the best possible solution. The other way is by specifying overall quality is in terms of an absolute quality. In addition to specifying the quality value, the desired likelihood of producing this value by the deadline is also specified. There is no guarantee that the exact quality value specified in these criteria will be achieved, but the MADS algorithm will use these criteria in deciding how many slave agents will be used, up to the available limit. For example, given the same task structure with the same deadline, a goal specification with a higher value for expected quality and more stringent requirements on how often these criteria need to be met will often require the use of additional slave agents. It is important to re-emphasize that the task representation permits and encourages the specification of alternative ways of achieving the desired solution (alternative search paths) and the specification of different paths which trade off quality of solution for decreased execution time [10]. Thus, by adding more agents to the task decomposition we can

increase the likelihood of achieving higher quality by the deadline, either by examining more alternatives or permitting the speedup of particular high-quality solution paths through concurrent executions of parts of the path, whose completion prior to the deadline without parallelism would be highly unlikely.

3 MADS Algorithm

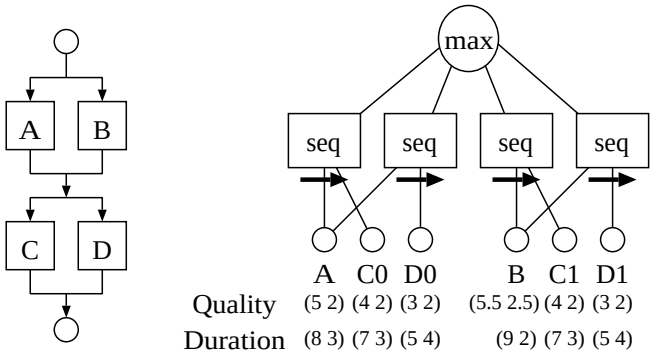
This section describes the MADS algorithm that generates a schedule for multiagents to achieve the goal criteria specified for the high-level task. The algorithm consists of the following sub-components.

1. alternative generation
2. agent number determination
3. macro order generation
4. method rating
5. partitioning and schedule generation

We will discuss this algorithm in the context of a simple coarse grain search problem in Fig.3(a). There are two branching points, each with two alternatives, so that there are four search paths in total; (A, C), (A, D), (B, C), (B, D). Each path can reach a goal state, but each has a different quality value and execution time. Figure 3(b) shows the task structure with expected quality and duration represented in the same notations of Fig.2. The deadline of the problem is 20, and the goal criteria is 80% satisfied quality.

3.1 Alternative Generation

The alternative generator creates alternative ways of achieving each subtask, i.e., alternative plans, and selects the useful combinations from these alternatives. A useful combination is one which is not dominated by another choice. For example, if one plan has an expected quality of 10 and duration of 5, it would not be a good combination if there existed another plan which generated an expected quality of 12 with a duration of 5. However, another plan that has quality of 16 with duration of 7 would be a good candidate for further consideration in later stages of the algorithm. This seemly straightforward comparison can be quite complex since the alternatives' performance is represented in a probability distribution of quality and duration. We chose upper and lower bound comparison for quality and duration ranges, since representative values of distribution, such as average and median, cannot represent ranges of values while exact

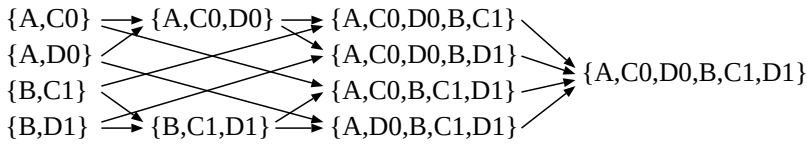


(a) Task Flow (b) Task Structure and sample parameters

Figure 3: A simple example for explanation

comparison of distribution function is hard to calculate. The upper and lower bounds are determined in a confidence interval, such as the 90% confidence interval, and the results of comparison are partial orders of alternatives. [3]

For example, the following partial orders were generated for the sample problem in Fig.3(b). Methods in a brace stand for a combination of the method executions.



In these partial orders, a combination of {A, D0, B, D1}, for instance, does not appear, since the combination is dominated by a combination of {B, C1, D1}. [4] In such a way, several inefficient combinations are removed from the candidate list.

3.2 Agent Number Determination

The number of agents necessary to achieve the required quality within a given deadline is determined in the following two steps. The first step determines the necessary amount

[3] In [9], we discuss in detail how these probability distribution performance profiles are calculated from the task structure and present the algorithm for this comparison process.

[4] The lower and upper bounds of expected quality achieved by {A, D0, B, D1} are 5.9 and 14.3, and those by {B, C1, D1} are 6.6 and 15.4. The lower and upper bounds of expected quality by {A, D0, B, D1} are 15.9 and 38.1, and those by {B, C1, D1} are 12.1 and 29.9. Therefore, {A, D0, B, D1} are expected to require more time to complete it, but produce less quality than {B, C1, D1}.

of computation power to achieve the required quality on a single agent. It selects the alternative that has the minimum duration of all those that achieve the specified criteria for quality, and determines the necessary duration.

The second step heuristically calculates the number of agents needed to meet the deadline. We use a simple model for this calculation that is extremely crude but has worked for the cases that have been tested.[5] The current version of implementation uses a magic number which estimates the effect of twice the number of agents. We temporarily set this number to 1.8, which means that twice the number of agents could provide 1.8 times the computing power. The validity of the number will be discussed later. With this magic number, the process calculates the minimum agents needed to provide the required computing power, and returns either the calculated number or the number of available agents, whichever is lower.

For example, the first step calculates the required computation power of 43.5 units for the sample problem in Fig.3(b), and the second step assigns three agents, which is expected to provide 50.8 ($\simeq 1.8^{\log_2 3} \times 20$) units.

3.3 Macro Order Generation

A macro order is defined as a sequence of partial orders, which corresponds to a rough schedule of problem-solving activities. The macro order generator generates a macro order that is most likely to maximize the performance profiles of the chosen alternatives.

For example, the following macro order was generated for the sample problem in Fig.3(b).

$$\{\text{B, C1, D1}\} \rightarrow \{\text{A, C0, B, C1, D1}\} \rightarrow \{\text{A, C0, D0, B, C1, D1}\}$$

The algorithm does not break up a combination of methods and delay ordering in detail to the next method rating stage, if the likelihood of achieving it with the assigned computing power is more than 0.99. In the above case, {A, C0, B, C1, D1} is decomposed, since the likelihood of achieving it with 50.8 unit power calculated in the previous subsection is 0.98. On the other hand, {B, C1, D1} is not further decomposed, since the likelihood is almost 1.0.

The macro order is an incremental sequence with respect to the members in a list, so that it is convenient to use a differential list as follows.

$$\{\text{B, C1, D1}\} \rightarrow \{\text{A, C0}\} \rightarrow \{\text{D0}\}$$

[5] This is one aspect of our parallel scheduling algorithm that we feel could be significantly improved.

3.4 Method Rating

After the macro ordering is established, priority is given to each basic method in the same level of the macro order with rating heuristics. The heuristics are almost the same as those used in the Design-To-Time [10] scheduling algorithm, shown as follows.

Avoid-not-enabled reduces the score of a method that has unfinished enabling (pre-condition) methods.

Enforce-enables raises the score of a method that enables other methods in the future. This rule contributes to increased parallelism.

Prefer-facilitators raises the score of a method that facilitates other methods in the future.

Longer-method-earlier raises the score of a method corresponding to the expected duration of the method. This rule promotes the execution of longer methods earlier.

The macro order obtained at the end of the previous subsection involves two cases that need the method rating, namely {B, C1, D1} and {A, C0}. B gets a higher score than C1 and D1 because B enables C1 and D1, and C1 gets a higher score than D1 because C1 has longer expected duration than D1, and A gets a higher score than C0 because A enables C0, so that the following order is finally obtained.

$$B \rightarrow C1 \rightarrow D1 \rightarrow A \rightarrow C0 \rightarrow D0$$

3.5 Agent Partitioning and Scheduling

Methods are assigned to specific agents and given a start time for their execution based on the following heuristics.

1. **no-enabling-method-append-early**
   If the method to be assigned waits for no method, then assign the method to the agent with the earliest available time slot.

2. **single-enabling-method-append-tail**
   If the method to be assigned waits for another method that is currently the last process to be executed by an agent, then assign the method to the end of that agent's queue. In this case, two successive methods are scheduled sequentially for the same agent, so that no time is wasted between the two methods.

3. **single-enabling-method-not-append-tail**
   If the method to be assigned waits for another method but that method is not at the tail of the agent's process queue, assign the method to an agent with the earliest available time slot. If the agent is different from that of the enabling method, it may be necessary to insert a waiting period or delay before the assigned method.

4. **multiple-enabling-methods-with-one-extreme**
   If the method to be assigned has multiple enabling methods and one method, whose next position is free, is expected to be far behind the other methods, the assigned method is scheduled immediately after the most delayed method.

5. **multiple-enabling-methods**
   This is the most general case, where the method to be assigned has multiple enabling methods. First, the maximal end time of these methods is estimated, and the method is assigned to the agent whose earliest free time is closest to the maximal end time.

Figure 4 depicts an example of parallel scheduling for the task in Fig.3(b). Note that this figure simply displays the average durations of tasks, which actually have the probability distributions of the durations. At first, B, which has no enabling (precondition) method, is assigned to agent 3 based on the first heuristic, and then C1, which is enabled by B, is assigned to the next position on agent 3 based on the second heuristic. D1 is also enabled by B, but C1 has already been assigned to next to B, so that D1 is assigned to agent 1 based on the third heuristic. In this case, a waiting period indicated by a black rectangle is inserted before D1. Steps 4-5 are almost the same as the steps 1-2. At Step 6, D0 is assigned to agent 2 based on the third heuristic, but no waiting period is inserted, because its precondition method A has also been assigned to the same agent 2. The final schedule is obtained in six steps. Although D0 is estimated to go beyond the deadline, it is still possible it will meet the deadline.

3.6 Rescheduling

After assigning activities to agents, the agents execute the assigned methods and sometimes make rescheduling requests to the master agent. An open function is provided at the end of every method for agents to determine whether to make rescheduling requests. Some typical templates are shown as follows.

every-time makes a request when a method is completed.

far-from-expected-quality makes a request when the error between the actual and expected quality is greater than the standard deviation multiplied by a specified value.

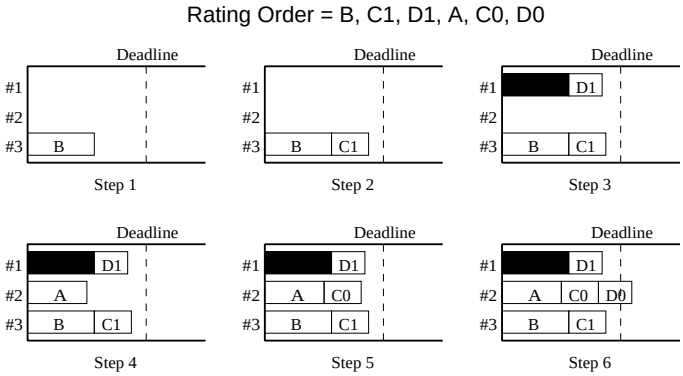


Figure 4: Example of parallel scheduling

far-from-expected-duration makes a request when the error between the actual and expected duration is greater than the standard deviation multiplied by a specified value.

far-from-expectation makes a request when either the far-from-expected-quality or far-from-expected-duration case occurs.

Rescheduling is accomplished in almost the same way an initial schedule is created except that the agent number determination step is skipped and rescheduling starts from the basis of assuming that on-going methods will be completed as expected. The task structure is updated to reflected finished methods by assigning the actual quality obtained to the quality slot of the finished method with zero deviation, and zero is assigned to the duration slot. The on-going methods are marked immovable and the expected durations are subtracted by the amount of the passed period. The unexecuted methods remain as the targets of rescheduling.

4 Experiments and Discussion

4.1 Initial Schedule Performance

The task structure presented in Fig.2 which was previously discussed will be used in the simulations presented in this section. Figure 5 shows schedules created by the MADS algorithm for the example problem using different goal criteria and deadlines. In accordance with the variety of goal criteria and deadlines, different schedules for different

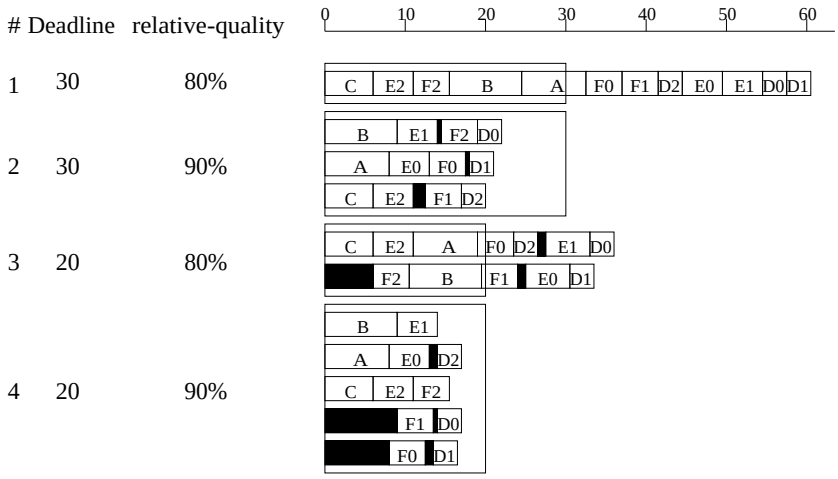


Figure 5: Schedule examples

numbers of agents are created. [6]

For instance, a comparison between schedules #1 and #3 indicates that tight deadlines affect the number of agents assigned. A comparison between schedules #1 and #2 indicates that a harder goal criterion requires more agents and, consequently, more methods to be scheduled, given the same deadline.

In this problem, the execution of methods C and E2 can achieve a solution with the minimum execution time, while it takes much longer to obtain quality with the executions beginning with A or B because of the min-relation between A and B. Additionally, the execution of E and F contributes to a high-quality solution because of the sum-relation between them. As a result, the algorithm gives a high priority to C, E2 and F2.

At the beginning of the schedule #3, there is an idle period that intuitively seems not to be a good idea, but it is appropriate since this schedule has a high possibility of finishing methods C, E2 and F2, while it would lower the possibility of finishing F2 if method B had been scheduled at the beginning and F2 had followed B.

4.2 Improvement of Task Quality

Rescheduling/repartitioning strategy, as a whole, is better than one-time static scheduling/partitioning created at the beginning of task execution, since the durations of task execution are uncertain. However, it is necessary for agents to consider the trade-off be-

[6] In these experiments, we have assumed a zero delay in communication and a zero duration for rescheduling. Due to space limitations, we have not included experiments considering these factors.

tween rescheduling cost and the amount of quality improvement. This subsection discusses possible strategies for this trade-off. As stated in the introduction, Durfee and Lesser have previously addressed this issue in a less formal way and with less experimentation [6].

In order to study these problems, a simulation system was created. This simulation system invoked the MADS algorithm with the fixed number of agents on the example problem to get initial schedules for each agent. The execution of methods was then simulated based on random values being assigned to method quality and duration in conformity with their distribution parameters. If the result of method execution triggered rescheduling, the simulator would, as detailed previously, reinvoke the MADS algorithm. Results shown in this section were obtained from two hundred simulations for each parameter set with different random value assignment.

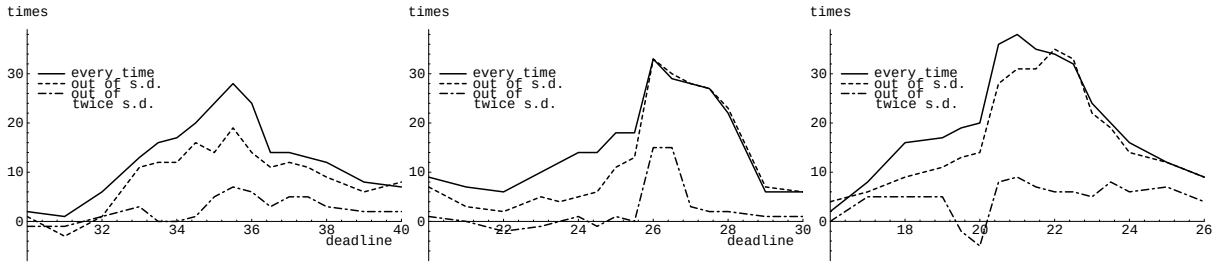


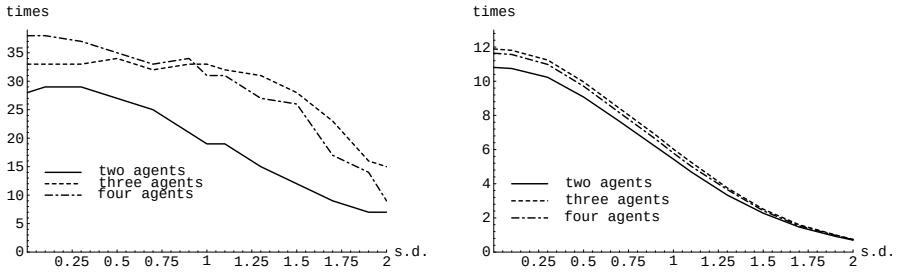
Figure 6: Relation between deadline and improvement

First, the deadline of the task was varied for two, three and four agents. The average duration to finish all methods in the problem with a single agent is 60.5 units, so that two, three and four agents respectively need more than 30, 20 and 15 units of time to finish all methods, although the first solutions could be found within these periods. Two conditions are compared; one is a sample with rescheduling, and another is a sample without rescheduling. We plotted lines on the graphs in Fig.6, connecting the results with specified deadlines and the number of times rescheduling samples produce better solutions than the no-rescheduling samples, subtracted by the numbers in the opposite relations. There are three lines drawn in each graph. Solid lines correspond to the cases where rescheduling was done whenever a method finished. Dashed lines depict the cases where rescheduling was restricted by the threshold that the error between the expected value, such as quality or duration, and the obtained value is not within the standard deviation. Dot-dashed lines depict the more restricted cases where the error is not within twice the standard deviation. For example, when a schedule was created for three agents with a deadline of 26 and using an every-time rescheduling strategy, there were 34 cases

where rescheduling samples were better and one case where the no-rescheduling sample was better, so that the point of the deadline of 26 and the number of improvement times 33—which corresponds to 16.5% of the 200 simulations—was plotted on the second graph in Fig. 6.

These graphs indicate that there is a phase transition in the effectiveness of rescheduling. On one hand, it is hard to achieve solutions with good quality for problems with short deadlines, and thus rescheduling cannot improve the quality significantly. On the other hand, it is easy to achieve solutions with good quality for problems with long deadlines, and thus rescheduling in this case also cannot improve the quality significantly. However, rescheduling is effective for the problems in the transition region between easy and hard problems, since the master agent can exploit the intermediate results so far obtained to either take advantage of more time to schedule more or better alternatives to increase the likelihood of producing higher quality results or to react to the availability of less time or lower quality than expected to find alternative methods to increase the likelihood within the available time left an acceptable level of quality will be achieved. Decker and Lesser in [5] have shown similar results on a slight different problem where they were studying the level of sophistication that should be used to coordinate agents in a lateral/peer type of organization.

Note that these experiments do not consider the cost of rescheduling. If it takes a certain amount of time to reschedule, the time to execute tasks would decrease by the amount, so that the problem would become harder, and then the effect of rescheduling would become smaller.



(1) Deviation and Improvement (2) Deviation and number of rescheduing

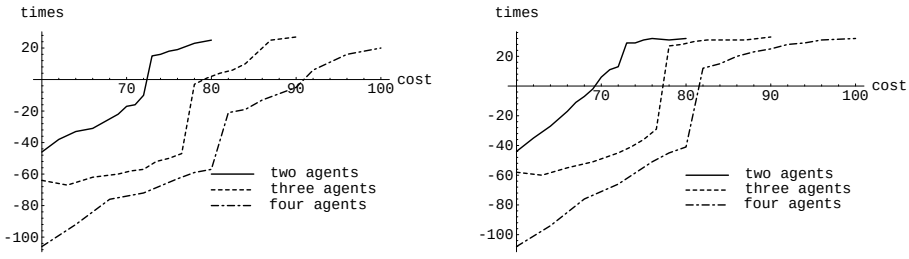
Figure 7: Relation between error threshold and improvement

Compared with the restricted rescheduling, the shapes in the dashed lines are similar to the shapes in the solid lines, while it is not evident that rescheduling is effective in the cases in the dot-dashed lines. Figure 7 show additional graphs that indicate the detail relation between the error threshold and the times of improvement. The deviations in X-axis are shown in the numbers relative to the standard deviation. Three lines in the

first graph correspond to the cases at the peaks of improvement in the graphs in Fig.6. The times of improvement are gradually degraded in accordance with the error thresholds. The threshold of 1.0 standard deviation seems to be a good measure to decide whether to reschedule, while 2.0 standard deviation is too restricted. The number of rescheduling opportunities for the threshold of 1.0 standard deviation is almost a half of that for the every-time rescheduling strategy, so that it can reduce the rescheduling cost to the half.

4.3 Parallel Execution Loss by Multiple Agents

Parallel execution by multiagents can potentially be inefficient due to agents having nothing to do while they wait for results from methods executing on other agents. There are also two more subtle issues in which inefficiency can be introduced. One involves the situation where, at the deadline, there may be methods that are still executing; the results of these methods will be lost. Obviously, the more agents, the more this loss will occur, whereas with a single agent whose processor speed is as fast as the entire agent set, the type of loss will be much less and not affected by processor speed. The second issue involves the situation where there may be multiagents that are searching different alternative paths concurrently. Clearly, the results of one path could make it unnecessary to work on the other path. In a single agent, if rescheduling occurred after the first path's successful execution, the second search path would in most cases not appear in the updated schedule unless it had a reasonable likelihood of getting a better answer and/or there was not other useful work to be done. Thus, the concurrent execution of these alternative search paths by agents could lead to wasted effort [3]. The scheduler uses the magic number described in Section 3.2, such as 1.8, to represent the magnitude of these effects with twice the number of agents.



(1) Without rescheduling (2) With rescheduling

Figure 8: Loss by multiagents

In the graphs in Fig. 8, the solution quality obtained by two, three and four agents is compared with the quality obtained by a single agent with deadline of 60. Cost axis is the

length of deadline multiplied by the number of agents, and times axis is the number of cases where the solutions by multiagents are better than those by a single agent, subtracted by the number of the opposite cases. The left graph is the case without rescheduling, and the right graph is that with rescheduling. Quality for the cost of 60 by a single agent is comparable to that for the cost of 72 by two agents and 91 by four agents without rescheduling, and comparable to that for 69 by two and 82 by four. The loss of parallel execution by twice the number of agents is from 10 to 20%, so that it is reasonable that the magnitude of these effects with twice the number of agents, such as the magic number, is about 1.8, at least for this problem.

5 Conclusion

This paper has examined an algorithm for deadline-based multiagent task decomposition and rescheduling of coarse grain problem-solving activities where the characteristics of operators (tasks, methods) with respect to expected quality and duration are specified in terms of probability distributions. A five-step algorithm, called MADS, was described which decides how many agents were needed and how to partition and schedule activities among these agents. Additionally, different criteria were developed for reinvoking the MADS algorithm when activity execution did not conform to expectations.

Experimental results were then discussed; they showed how different schedules and numbers of agents would be generated as a result of varying goal criteria for a successful answer and the deadline within which an answer was needed. As part of these experimental results, we explored different criteria for rescheduling based on the amount of standard deviation from the expected value. These were experimentally found to be good measures to reduce rescheduling costs. We also found that there was a phase transition in effectiveness of rescheduling in relationship to problem difficulty.

Our MADS scheduling algorithm is closest to that done by Decker et al. [3], and Long and Lesser [11], who clarified the categories of task relationships and created heuristics for parallel scheduling. Our work extends their approaches by exploiting information about the uncertainty of task duration and quality in the parallel scheduling of tasks.

This paper discussed centralized rescheduling, but partial rescheduling among a subset of agents seems to be a good idea for a multiagent system. Another focus of further work is the application an actual agent network. Real tasks will reveal more problems, such as scheduling cost problems, difficulties in unifying distributions, and so on.

References

[1] Davis, R. and Smith, R.G., Negotiation as a Metaphor for Distributed Problem

Solving. *Artificial Intelligence*, vol.20, pp. 63-109, 1983.

[2] Decker, K. and Lesser, V., Generalized Partial Global Planning, *International Journal on Intelligent Cooperative Information Systems*, vol.1(2), pp. 319-346, June 1992.

[3] Decker, K., Garvey, A., Humphrey, M. and Lesser, V., Control heuristics for scheduling in a parallel blackboard system. *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7(2), pp. 243-264, 1993.

[4] Decker, K. and Lesser, V., Designing a Family of Coordination Algorithms. *Proc. of the First International Conference on Multiagent Systems*, San Francisco: AAAI Press, June 1995.

[5] Decker, K. and Lesser, V. A Formal Environment- Centered Framework for Agent Coordination. Submitted to Artificial Intelligence Journal (under review).

[6] Durfee, E.H. and Lesser, V.R., Predictability Versus Responsiveness: Coordinating Problem Solvers in Dynamic Domains. In *Proc. of AAAI-88*, pp. 66-71, 1988.

[7] Durfee, E.H. and Lesser, V.R., Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transaction on Systems, Man, and Cybernetics*, 21(5):1167-1183, 1991.

[8] Durfee, E.H. and Montgomery, T., Coordination as Distributed Search in a Hierarchical Space. *IEEE Transaction on Systems, Man, and Cybernetics*, 21(6):1347-1362, 1991.

[9] Fujita, S. and Lesser, V. Cooperative Tasks in Coarse Grain Search Problems. Computer Science Technical Report 96-28, University of Massachusetts, Amherst, 1996.

[10] Garvey, A., Humphrey, M. and Lesser, V., Task Interdependencies in Design-to-time Real-time Scheduling. *Proc. of AAAI-93*, pp. 580-585, 1993.

[11] Long, Q. and Lesser, V., A Heuristic Real-Time Parallel Scheduler Based on Task Structures. Computer Science Technical Report 95-92, University of Massachusetts, Amherst, 1995.

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