To appear in Working Notes of the AAAI Workshop on Constraints and Agents
Providence, Rhode Island, July, 1997

Contract-Based Distributed Scheduling for Distributed Processing

David J. Musliner & Mark S. Boddy
Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{musliner,boddy}@src.honeywell.com

Abstract

To an increasing extent, large-scale information processing is a distributed phenomenon. As the trend in computing moves further towards distributed networks of powerful workstations and information servers, we see the growing importance of solutions to *dynamic distributed scheduling* problems. In these domains, resource providers are distributed both geographically and bureaucratically, so that no central authority can dictate a global schedule. Resource consumers are also distributed: tasks arrive at different locations according to arrival functions that are at best stochastically predictable.

In this paper, we describe a distributed constraint-based scheduling system that adjusts task distribution and execution times through the negotiation of *contracts*, using a refinement of the Contract Net protocol. By combining the minimal-commitment capabilities of constraint-based scheduling with the distributed coordination features of contracting, this system responds flexibly to dynamic variations in load balance and unpredictable task arrivals. Large-scale simulations show significant performance benefits to using powerful scheduling methods in the determination of contract negotiation bids. These results can be used to improve the performance of distributed computing systems cooperating to process large-scale shareable tasks.

Introduction

As the trend in computing moves further towards distributed networks of powerful workstations and information servers, we see the growing importance of solutions to *dynamic distributed scheduling* problems. By "scheduling," we mean the allocation of resources over a specific period of time to the performance of a particular task. We are interested in problem domains that are "distributed" in at least two ways. First, they are distributed because tasks may be executed at any of a set of locations, depending on what resources are available at those locations. Second, they are distributed in the sense that there is no central authority that holds knowledge of and control over when and where different tasks are executed. Instead, each location makes its own decisions about what tasks to execute at what times, and coordination is arranged through explicit communication. Finally, the domains of interest are "dynamic" in the sense that they change over time: the desired tasks and available resources may be changing as the system is executing tasks, and the patterns of these changes are not known *a priori*. A static schedule cannot be created for these problems.

The temporal aspect of these problems presents difficulties for conventional approaches to load-balancing in distributed systems; the deadline by which a task must be completed or face a penalty is a significant factor in how tasks should be scheduled at individual sites. In addition, the objective in our domain is *not* to simply balance the load, but rather to maximize the overall system's net earned value. As illustrated by the simulation results discussed in Section , a balanced load may actually be antithetical to maximal system-wide net value.

In this paper, we describe a distributed scheduling system that adjusts task distribution through the negotiation of *contracts*, using a refinement of the Contract Net protocol (SMITH 1980) similar to Sandholm's (Sandholm 1992; 1993). The behavior of this system is investigated through experiments conducted in a simulated domain implementing a simplification of a distributed image processing application for NASA's Earth Observing System. Large-scale simulations have revealed several interesting behavioral aspects of the system. By comparing the behavior of agents negotiating based on complex, constraint-based scheduling systems against agents using simpler queue-based negotiation schemes, we have verified several qualitative and quantitative advantages of the constraint-based scheme. The main result is that there is significant benefit to having the individual negotiating agents use complex, constraint-based scheduling when computing their bids on contracts.

Section describes the domain in more detail, Section provides the technical details of task scheduling and distributed negotiation, and Section sketches the design of the implemented simulator. Finally, Section describe our experimental results. We conclude with a discussion of related work, draw some conclusions from our results, and suggest some areas for further work.

Portions of the work described in this paper were supported by NASA, under contract #94-3050-J1237, administered by Hughes STX

Dynamic Distributed Scheduling for EOSDIS

Our simulated distributed scheduling domain is based on NASA's Earth Observing System (EOS). EOS is a multi-year, multi-billion-dollar project aimed at gathering scientific information about the Earth's environment. With the participation of the European Space Agency, Japan, Canada, and NASA, several satellites supporting dozens of sensors are scheduled to be launched in the late 1990's. These systems, along with additional sensors already in orbit, will be used to collect information on ozone depletion, greenhouse effects, ocean productivity, and other environmental features.

The EOS Data and Information System (EOSDIS) will be responsible for archiving and analyzing the resulting vast amounts of data. EOSDIS functions include managing mission information, archiving and distributing data, and generating and disseminating scientific data products (Dozier & Ramapriyan 1990). Both raw data and the results of analyzing that data will be stored in archives maintained by a set of Distributed Active Archive Centers (DAACs). Much of the data analysis will be performed at these centers, though individual scientists or institutions will also perform individual analyses of raw data (or intermediate results) obtained from the DAACs. Each DAAC will have a collection of specialized computing resources (e.g., Cray XMP and MasPAR computers) to support the analysis done at that site. Individual DAACs may be responsible for different sets of data (from different sensors), generating and archiving different analysis products from one or more sets of data, and supporting different users. Interactions among DAACs may arise because some analyses involve combining multiple sets of data or because one DAAC's "customers" may request data stored elsewhere.

A variety of interesting problems arise in the process of automating the functions of the EOSDIS network. These issues range from managing petabyte[1] data archives, including the automated generation of content-based "meta-data" for indexing, to generating analysis plans to produce a desired analysis product, to scheduling ground equipment for satellite support (Campbell *et al.* 1991; Boddy *et al.* 1995; 1994). In this paper, we are concerned with scheduling analysis tasks at the various DAACs as requests arrive from their respective users. This process is complicated by the fact that satisfying requests may involve getting data from or requesting a component analysis from another DAAC. In addition, some analysis tasks may be offloaded if another DAAC has the appropriate resources and less to do. The negotiation required to find other DAACs to take on such tasks is complicated by the timing restrictions imposed by customer deadlines.

[1] One petabyte = 10^{15} bytes.

Technical Approach

The core of our approach to solving dynamic distributed scheduling problems involves the combination of two fundamental technologies: constraint-based scheduling and contract-based negotiation. The scheduling methods are used to manage task scheduling and execution at individual DAACs in the distributed network; the negotiation methods are used to allow DAACs to exchange tasks and other commitments. This section provides a brief background in each of these independent technologies and then describes the approach we use to combine them.

Contract-Based Negotiation

Negotiation over formal and informal contracts is a longstanding societal mechanism for establishing commitments between agents. Not surprisingly, contract-based negotiation is also a popular paradigm in distributed AI and distributed problem-solving research. Early work by Smith (SMITH 1980) described the Contract Net system of distributed processing agents based on contract negotiation. The Contract Net Protocol (CNP) specifies how contracts are announced by *contract managers* to other agents, how *bids* are returned to the manager by potential *contractors*, and how the contract is then *awarded* by the manager.

Many extensions and variations of the CNP have been used to satisfy various system requirements and avoid certain types of undesirable behavior. The specializations made by Sandholm (Sandholm 1992; 1993) are most relevant to our work. These extensions and modifications to the original CNP include:

- The use of marginal cost calculations to determine what contracts should be announced, and how much agents should bid.
- The restriction that bids are binding commitments, so that an agent must reserve all necessary resources in order to make a bid on a contract. As a result, bidders who are not awarded a particular contract must be sent a *loser* message to release them from the bid commitment.

Constraint-Based Scheduling

Each DAAC agent maintains its own schedule of tasks using *constraint envelope scheduling*, in which schedules are constructed by a process of "iterative refinement" (Boddy, Carciofini, & Hadden 1992). Scheduling decisions correspond to constraining an activity either with respect to another activity or with respect to some timeline. The schedule becomes more detailed as activities and constraints are added. Undoing a scheduling decision means removing a constraint, not removing an activity from a specified place on the timeline. This basic approach is common to a number of scheduling systems (e.g., (Fox & Smith 1984; Smith *et al.* 1990; Sadeh & Fox 1990; Muscettola 1993)). Our implementation of this approach provides unique support for reasoning about partially-specified

schedules and searching through the resulting space for an acceptable fully-ordered execution schedule.

Constraint envelope scheduling is a *least-commitment* approach. We do not assign a set of activities to places on a timeline, assigning each activity a start and end point. Rather, we collect sets of activities and constrain them only as needed. Constraints may express relations between activities (e.g., any analysis task using the results of task A must not start before A is completed) or relative to metric time (this task takes at least 2 hours, and may not start until 10:15). Additional constraints are added as needed to resolve conflicts over resources. So, for example, two tasks that require the same tape drive must be ordered with respect to one another, whereas if they were to use two *different* drives, their ordering would not have to be determined.

The least-commitment nature of our schedules is an important advantage in dynamic domains where task arrivals and changes require rescheduling. If an event arises that makes a resource unavailable, or an ongoing task takes longer than expected, the effect on the schedule is minimized. First, only those activities related by a chain of constraints to the activities explicitly moved will be affected. Second, if the set of constraints in the schedule is consistent with the new event, the projected effect of the schedule can be updated efficiently, without any rescheduling at all.

Combined Technologies

Traditionally, the CNP and its variants are used to exchange contracts that encapsulate complete tasks. This is the primary use in our current implementation as well, where tasks correspond to users' requests for EOSDIS data processing, and contract negotiations allow the DAAC agents to exchange these processing tasks to achieve a balanced system load and maximize throughput. The constraint-based scheduling methods are used within each agent to remain as flexible as possible about the start times for individual tasks, so that the dynamic arrival of new tasks and contract announcements can be accommodated as much as possible.

The links between contract negotiations and task scheduling are forged primarily through functions that compute the value of a task schedule and the value of individual tasks. For example, the CNP requires bidding agents to provide a measure of their cost (a bid) to perform the contract task under consideration. In our system, this bid is the marginal cost of adding the contract task to the bidding agent's schedule. The marginal cost is computed by first storing the value of the agent's existing task schedule, adding the contract task to the schedule, and then finding the new schedule's value. The difference in values, new minus old, is the marginal cost of the addition, and hence the bid. Note that the contract remains on the schedule after this computation; the bid is binding.

Similarly, the decision about which of an agent's existing tasks to announce as a contract open for bidding is made by computing the marginal cost of the task. In our implementation, if the marginal cost of a task is negative (i.e., the agent is better off without trying to execute the task), then the contract is announced for possible award to a different agent[2]. Thus the fundamental decisions required by the CNP are made using value functions evaluated on the constraint-based schedule.

In addition to the CNP modifications made by Sandholm (noted above), we adapted several aspects of the contract negotiation protocol to focus on our thesis investigation and minimize other concerns:

- To limit repetitive and unproductive contract announcements, each contract is announced by a particular manager no more than three times: if the contract is still not awarded to another DAAC (i.e., the manager's bid was the best bid), the contract is executed locally. Also, contracts are not announced more frequently than every 20 time units, to avoid negotiations when no significant changes have occurred that could lead to a different outcome. These limitations were chosen arbitrarily, and seem to be effective at limiting the network communication load while still permitting effective contract movement.
- Rather than using a "focused addressing" scheme in which contract announcements are sent to a select few potential bidders, we broadcast contract announcements to all DAACs, maximizing the potential for useful contract exchange. We are not particularly concerned about the resulting communications load, since the number of contracts "on the market" at any time is restricted through the limitations described above.
- To prevents issues of bid timing from interfering with our investigation, the bidding process is required to operate in a first-come, first-served manner: agents cannot delay their bids to see what other alternatives are available. Combined with synchronized agent clocks, this restriction also allows us to fix an upper bound on the time a contract manager must wait until he is assured of receiving all bids, and can make the contract award with complete information.
- Contract award messages identify which DAAC has won a contract, and they are broadcast. Thus award messages also act as loser messages: each loser DAAC can see that the contract has been awarded to some other DAAC.

The constraint-based scheduling paradigm remains essentially unchanged for this domain: tasks are associated with time intervals whose position in the schedule is restricted by constraints. One significant simplification was made to reduce the complexity of the

[2] Actually, we announce a contract if its net value is less than its price, indicating that some penalty was incurred, and thus that some other DAAC might be able to do a better job.

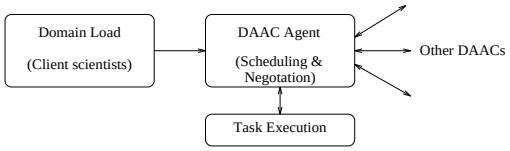


Figure 1: Conceptual role of the implemented DAAC agent.

domain: each DAAC is defined to have only one execution resource it is scheduling (e.g., one data processing engine), and thus the eventual executable task schedule is a fully-ordered list of tasks to run. However, we retain the flexible constraint-based representation of scheduling limitations, so that dynamic changes in a DAAC's load and its contracting environment can be accommodated through incremental changes to the minimal-commitment constraint structures.

System Design and Implementation

We have implemented a distributed, contract-exchanging network of scheduling agents to simulate DAAC behavior in a wide variety of load conditions and operational scenarios. In this section, we describe the system architecture and pertinent implementation details that influence the simulator's flexibility and capabilities.

Conceptual Architecture

Conceptually, the DAAC agents we are simulating exist in an environment similar to that shown in Figure 1. The negotiation and scheduling portions of the DAAC are responsible for receiving new tasks from the external domain (the EOSDIS scientists who submit information processing jobs). Each new job can either be scheduled to execute on local resources, or it can be passed to another DAAC in the network using the negotiation protocol.

Our investigation is focused on the central scheduling and negotiation roles of the DAAC agent, and thus we have implemented only low-detail simulations for the domain and execution components of the system. To avoid the complexities of numerous interconnected processes, we have bundled the peripheral domain and execution components in with the controlling agent itself, yielding an implementation in which a single Lisp process encapsulates all of these components.

Implementation Architecture

As shown in Figure 2, each implemented simulation process has components responsible for the primary DAAC functions:

- Communications with other DAACs, including bootstrap synchronization, contract announcement, bidding, and awarding.
- Scheduling functions, including inserting new tasks, removing tasks, and finding the value of individual tasks and the whole schedule.

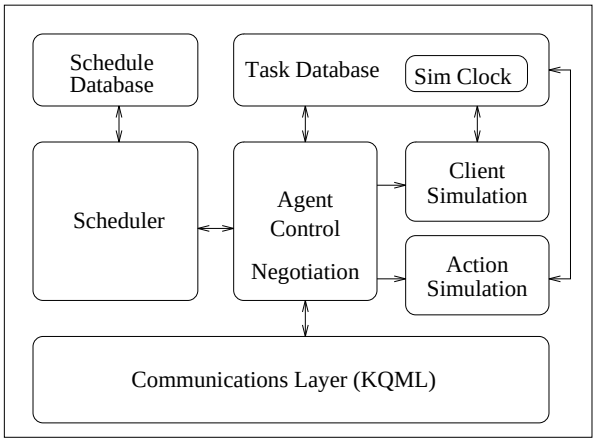


Figure 2: Overview of the DAAC agent architecture.

- Agent control, including coordination of overall communications, simulation, and scheduling activities.

In addition, relatively simple components are included for:

- Client simulation, including generation of new tasks for the local DAAC.
- Action simulation, including maintaining the local simulation clock, starting and finishing scheduled tasks, and notifying the contract managers of status changes.

Scheduling

Each DAAC agent maintains a local constraint-based schedule of tasks it will perform. Based on the existing Honeywell Interval Constraint Engine (ICE) system, the DAAC scheduler provides a function-call based interface to maintain a fully-ordered schedule of tasks by inserting and removing execution time intervals associated with each task.

When a task is defined at a DAAC (either via the client simulation or a contract announcement), a time interval is defined and associated with the task. Constraints are asserted to restrict the possible position of the task interval on the schedule. For example, duration constraints define the minimum length of the interval, earliest-start-time constraints limit the position of the interval's start point, and precedence constraints can be asserted between subtasks in a hierarchical task structure. The interval is then added to the existing schedule, if possible.

To insert a new task interval into the schedule, the current implementation uses a simple greedy algorithm that is neither optimal nor complete; it does not guarantee to find a schedule if one exists. The greedy algorithm takes as arguments a list of the entire set of task intervals to be scheduled (including those that were already on the schedule) and a cost function. The algorithm greedily inserts task intervals from the list onto

```
while (time < end-time)
 {
  foreach message in priority-sort(get-messages())
    case type(message) of
     {
      ANNOUNCE: bid-on-contract(message);
      BID: add-bid-to-contract(message);
      AWARD: if (contractor(message) == self) then
                  accept-contract(message)
             else remove-contract(message);
     }
  award-contracts();
  announce-contracts();
 }
```

Figure 3: Pseudo-code for main agent control loop.

the schedule, checking for consistency, and returns either failure or the value of the resulting schedule. The cost function is used to choose the best possible alternative position for each task interval as it is greedily inserted.

This system has proven quite robust in practice, but optimizing the scheduling algorithm remains one possible direction for future investigation. The cost function is already passed to the scheduler, so the interface would not need to be changed.

Overall Agent Control

The outer loop of the agent program coordinates and controls each of the modular functions shown in Figure 2. Ignoring the domain and execution simulation components, the main loop is shown in pseudo-code in Figure 3.

Action Simulation

Although dynamic environmental conditions such as execution failures are one of the motivations for using constraint-based scheduling, our current implementation does not include that aspect. Tasks are started at the start of their scheduled interval and take exactly their original duration to execute. No uncertainty is currently introduced into the system from the action simulation. The model of action simulation was included, however, as a placeholder for future studies in which execution may fail or be delayed, leading to additional contracting efforts to ameliorate the incipient costs of schedule disruption.

Client Simulation

The client simulation generates the task load that the DAAC agents try to distribute and process in a cost-effective manner. The load is generated stochastically from task-class descriptions that specify the distribution of arrivals, durations, deadlines, prices, and other task features. In the experiments described below, all task distributions are uniform.

We have also implemented "conditional distributions" that can express discrete variations in distributions based on context information such as the time at which a distribution is evaluated. For example, conditional distributions can be used to describe tasks that arrive with a particular probability on Monday morning, but do not arrive at all at other times. Supporting temporal expressions have been built to make it easy to express a wide variety of such conditions based on weekly, daily, hourly, and other time ranges.

Experimental Results

Extensive simulations were run to test the behavior of the DAAC agent network under various loading conditions. In order to provide a basis for comparison and evaluation, two competing version of the contract negotiation system were used. In the first, "simple bidding" system, bids on announced contracts are determined by an approximation to queue duration (specifically, the bid consists of the finish time of the latest queued task; this is not precisely the queue duration because there may be slack time segments in the schedule). In the second, "complex bidding" system, bids are computed as the actual marginal cost of adding the contract to the agent's schedule: the difference in schedule value before and after the addition.

The intention is thus to compare the performance achieved using the full constraint-based scheduling system against a simpler scheme which conceivably could employ a different scheduling method. In fact, only the bidding method is different between the simulated versions; in both systems, bids are made binding by placing the bid-upon contract on the full constraint-based schedule. In the simpler system, the bidding method simply ignores the available schedule-value information and uses queue duration instead.

Common aspects of all the experiments discussed below include:

- The network consisted of five DAAC agents.
- Four of the five DAACs all had the same stochastic task generation functions, giving them, on average, the same level of load from new tasks.
- DAAC-3 was distinguished in that it received medium duration tasks that none of the others did. Its expected load was approximately twice the loads placed on the other DAACs. Hence DAAC-3 must give away tasks or fall further & further behind.
- All task penalties were computed as linear functions of the tasks' lateness [i.e., penalty = penalty-factor * (finish-time – deadline)].
- All experiments were run for 10000 simulated time ticks. Since tasks can arrive at DAACs right up until the end of the simulation time, some tasks are left unexecuted when the simulation halts. These tasks are accounted for in most of the performance metrics (e.g., total schedule value) because they are scheduled as soon as they arrive. Continued negotiation could only lead to earlier execution by a different DAAC. The tasks scheduled after time 10000 are generally a very small portion of the overall data

set, amounting to less than 1% of the total duration of task arrivals.

The main parameter varied throughout the experiment set discussed below is the overall system load level: task durations and deadlines were scaled proportionally to vary the overall system load from 50% to 90%. Those load levels represent the average DAAC load if the actual arriving tasks could be spread evenly amongst the five agents. As noted above, the load arrived from the client simulation with an uneven distribution favoring DAAC-3 with more tasks than the others.

Data Collection

Each DAAC agent writes data to a file continuously as the simulation progresses. In addition, each DAAC periodically writes (to the same file) a complete textual description of its contract schedule and various measures of performance. Thus incremental results are available for evaluation, and much of the DAAC network's overt behavior can be reconstructed from the set of data files produced. Automated data extraction and processing programs have been written to produce graphs and statistics on a wide variety of agent performance measures.

Performance Metrics

We have extracted and evaluated a variety of measures of system behavior and performance. Some track the production of incoming tasks with different distributions and load levels. Other metrics monitor each agent's performance in terms of the number contracts that it completes, the total slack time it accumulates, the total penalties for late contracts, and the net value of the agent's overall schedule including both finished contracts and scheduled future tasks. Still other metrics monitor the agent's network behavior to show how many times a contract was exchanged between DAACs before it was finally executed, the number of contracts bid upon, and the number of contracts won. The experimental results discussed in the following subsections include examples of several of these different performance evaluation metrics.

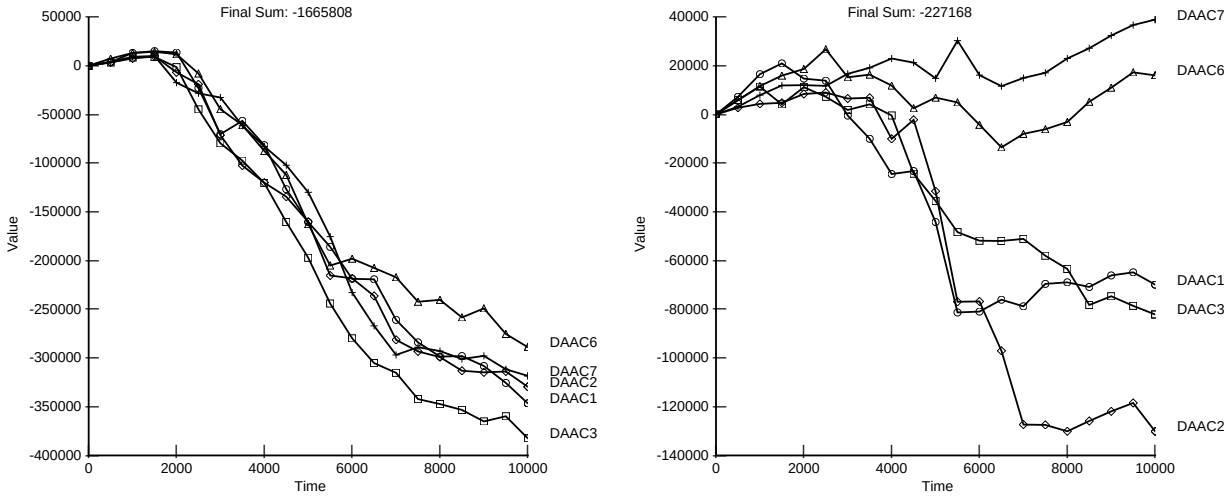
General Observations

The experimental results support our initial hypothesis that the complex, full scheduling bidding method would outperform the simple bidding scheme based on queue duration alone. For example, Figure 4 shows that, at an overall load level of 82%, the simple bidding scheme led to uniformly falling net schedule values, while the complex bidding scheme left room for more variation in DAAC performance. As a whole, the complex network significantly outperformed the simple system: the final summed net value across all the DAACs was seven times lower for the simple system. Furthermore, these superior results were achieved with much lower rates of contracting activity: nearly three times fewer task minutes were exchanged via contract in the complex bidding scheme, and the median number of "hops" by any single contract was cut in half.

It is interesting to note that the simple bidding scheme led to smoother load balancing according to several simple measures, but was not able to outperform the complex bidding method in terms of net value. For example, Figure 4b shows that, in the complex scheme, DAAC-2 actually underperformed the overloaded DAAC-3, which we might normally expect to do the worst. In this case, the bidding scheme has transferred enough penalty-laden jobs to DAAC-2 to overwhelm its normally underloaded state and lead to negative net schedule values. The simple bidding scheme, on the other hand, keeps all the DAAC schedule values very nearly the same (see Figure 4a). Why, then, would the complex scheme do better? Both systems are using the same scheduling algorithm, so what must be happening is that the complex scheme is finding earlier places to schedule tasks in a DAAC's schedule, so that the placement of individual tasks is optimized. The simple scheme, on the other hand, is assuming that tasks will be placed at the end of the queue, which is often inaccurate. Thus the simple bidding scheme will not bid as accurately, and the complex scheme is more likely to place each task in an optimal place.

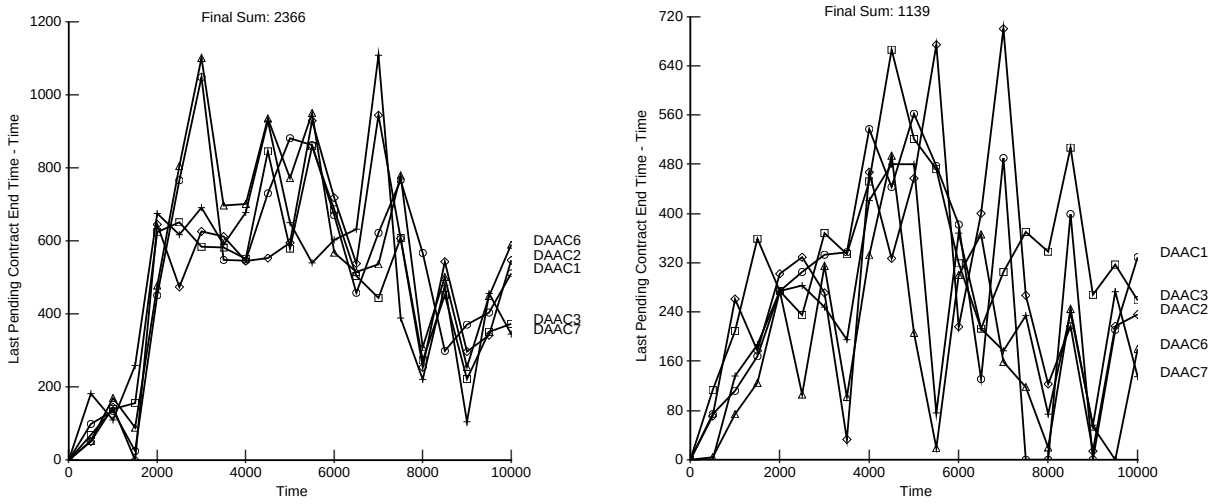
Additional evidence pointing to unbalanced loading providing superior overall performance is provided in Figure 5 and Figure 6. The queue durations are synchronized very closely in Figure 5a, as we'd expect since this value is used to determine bids. In contrast, Figure 5b shows a much wider variance in queue duration. However, the magnitude of the average queue duration is much smaller in the complex bidding case, indicating the effectiveness of the overall system at determining a good place and time to execute each incoming task. At the end of the simulation of task arrivals, the complex scheme had about one-half as much contract time queued up as the simple bidding system.

Similarly, Figure 6a shows closely matched cumulative penalties for all 5 agents in the simple bidding system, but Figure 6b shows wide divergence in the penalties accumulated by the complex bidding agents. This result is particularly interesting because it indicates that DAAC-2, for example, continued to execute high-penalty tasks even after it was already doing worse (in terms of net schedule value) than the other agents. This may initially seem counter-intuitive, since it would seem better to offload tasks that are being scheduled to have a penalty, and this might be expected to even out cumulative penalty. In the short term, the complex bidding scheme does exhibit that behavior: in effect it runs a task on the DAAC that can execute it with the smallest penalty. Note, however, that the cumulative penalty shown in Figure 6 is different from the instantaneous penalty associated with a contract at bid-time: the bid does not take into account how much past penalty has been accumulated by a DAAC, because the primary goal is not to smooth schedule



(a) Simple bidding. (b) Complex bidding.

Figure 4: Net schedule value at 82% load level.



(a) Simple bidding. (b) Complex bidding.

Figure 5: Queue duration at 82% load level.

value, but to maximize the network-wide value. Thus the cumulative performance measures give a picture of system behavior that arises when the agents themselves are not concerned directly with load balancing or average performance measures, but rather with a global, network-wide performance metric.

Changing System Loading

Our experiments were run at three different load levels: 67%, 82%, and 90%. The loading is calculated as the (summed) expected duration of contracts arriving over the entire simulation run, divided by the time available for processing, over the entire system. The results are summarized in Figure 7. Figure 7a graphs the total slack time for each bidding scheme as a function of load level. At 67%, the slack times are effectively equal. Figure 7b is a graph of total penalty incurred by the system as a whole over the entire simulation run. At 67%, despite the fact that both bidding schemes spent the same number of ticks computing, the complex bidding scheme accrues only two-thirds as much total penalty. This difference is consistent across all loading levels. In all cases, the slack times are very similar, but the difference in penalties is quite substantial and increases with increasing load level.

This suggests that what is happening is that the smarter bidding scheme is resulting, not in the execution of a different number of contracts *or* of contracts with different durations, but in the execution of contracts by different DAACs or at different times. In fact, if we compare the numbers of contracts left uncompleted for each scheme at each load level, the difference is in favor of the smarter scheme. The complex bidding scheme, using additional information about the schedule, results in more contracts being completed and a much lower level of penalties.

The fact that the proportional difference in penalties is greatest at an intermediate loading raises some interesting possibilities for further investigation. The additional information available in the smarter bidding scheme should be most useful when two conditions hold: first, when there is substantial room for improvement through smarter scheduling (the load is high enough), and second, when there is sufficient flexibility to make smarter decisions (the load is not too high).[3] If borne out in further analyses, this conjecture would support our arguments in favor of the utility of combining flexible scheduling with informed bidding.

Related Work

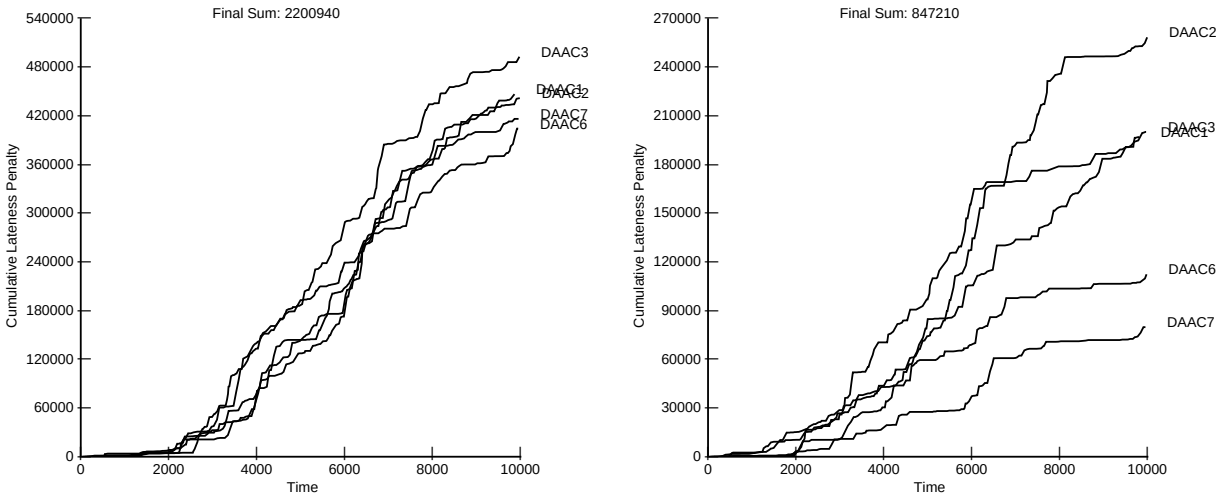
Distributed problem-solving in general, and the allocation of tasks in a distributed system in particular, are both very broad research topics, addressed using a wide variety of approaches in fields ranging from management science, to AI, to operations research, to work on distributed operating systems. Our concerns in this paper are somewhat narrower. We are specifically interested in problems involving the allocation of computing resources to individual tasks, where allocation decisions must be made locally. In additional, these allocations are constrained temporally: there are limits on when the work can be done, where those limits themselves are potentially subject to negotiation.

At least two aspects of our domain distinguish it from those addressed by operating system load balancing techniques (e.g., (Ni, Xu, & Gendreau 1985; Lin & Keller 1986)). First, the temporal character of the client tasks (i.e., earliest start times and deadline) means that the DAACs are not simply operating in a first-come, first-served queue manner, but are maintaining a full schedule of future tasks that must meet stringent timing requirements. This scheduling aspect is difficult to reconcile with the simpler notion of "load" present in operating systems. Second, the objective in our domain is *not* to simply balance the load, but rather to maximize the overall system's net earned value. As illustrated by the simulation results discussed in Section , a balanced load may actually be antithetical to maximal system-wide net value. Thus simple load-balancing methods aim at the wrong target for our problem. Interestingly, however, the approaches used by distributed operating systems are very similar to our contract-exchanging network; they focus more on using various announcement and bidding protocols to manage non-negligible network traffic. Bidding in pure load balancing is trivial; in our domain, high-quality bidding appears to be the key to high performance.

Market-based approaches such as Wellman's WALRAS (Wellman 1993) are difficult to apply to our domain because of the focus on scheduling individual tasks, and the relatively large size and small number of those tasks. Our view of the overall system architecture closely parallels Wellman's description of a coordinating mechanism synchronizing the behavior of a large set of agents with only limited information about what is going on in the system as a whole.

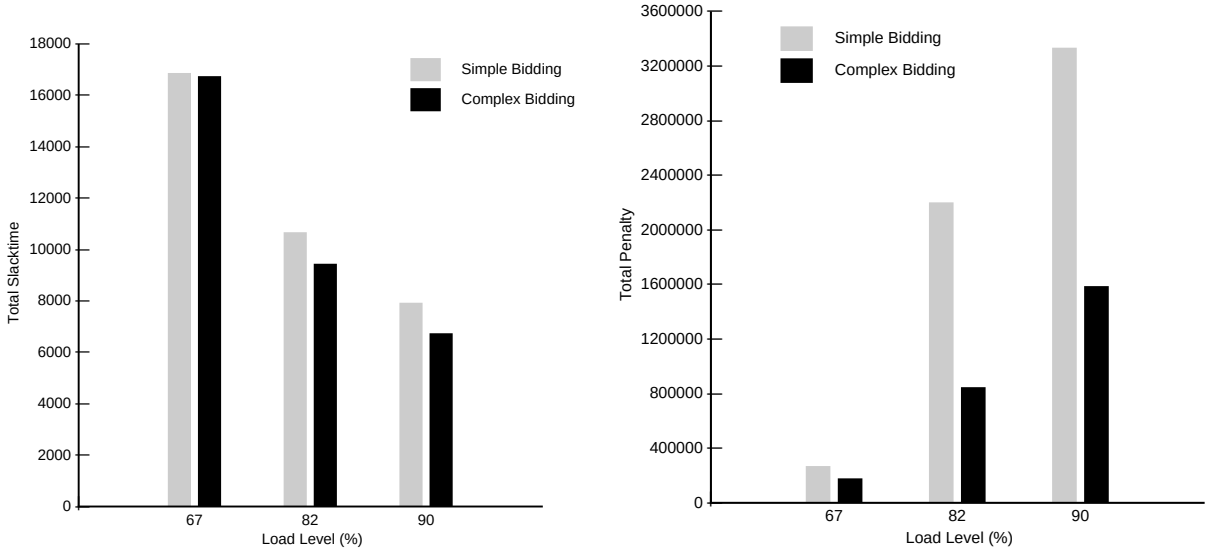
Sandholm's work on task allocation in a transportation domain is closest to ours in flavor, and as discussed previously we have implemented CNP extensions modeled on his. However, our domains are sufficiently different that only limited parallels can be drawn. The main commonality is that negotiating on the basis of marginal cost seems to be a good approach for both domains. While Sandholm has investigated the exchange of "package deals" encompassing multiple tasks, our system is limited to exchanging individual contract tasks. Although DAAC tasks lack the spatial interdependencies inherent in Sandholm's delivery tasks, they may still interact through resource consumption, and thus package deals might be a fruitful area of extension to our existing negotiation scheme.

[3] This parallels results from other problem-solving domains (e.g., Barrett and Weld's results on partial order planning (Barrett & Weld 1994)).



(a) Simple bidding. (b) Complex bidding.

Figure 6: Cumulative penalties at 82% load level.



(a) Total slack time. (b) Cumulative penalty.

Figure 7: Comparative behavior at different load levels.

Conclusions and Future Work

In this paper, we describe the class of *dynamic distributed scheduling problems*, using the EOSDIS DAAC scheduling problem as an exemplar. We present some early results on the use of an architecture integrating constraint-based scheduling with contract-based negotiation to task allocation in this domain. These results clearly demonstrate the utility of incorporating the additional information made available by explicit, site-specific schedules of what tasks will be performed when.

Our comparison of two bidding schemes, one using queue length as a measure of marginal cost, the other using the actual marginal cost calculated by the scheduler, showed clear benefits for the more informed approach. At any loading level, the final value of the schedule executed (task value - penalties for late completion) was substantially better for the smarter bidding scheme. These results were achieved with much lower rates of contracting activity, both in terms of the number of contracts moving and the average number of times any one contract was moved, which means a lower communications overhead. Looking at the behavior of the two bidding schemes as the system loading varies shows that the smarter bidding scheme is using additional information about the schedule to complete more contracts, at a much lower level of penalties, with an increasing advantage as the loading level increases.

The work presented here is best characterized as preliminary results in an area with rich possibilities for further investigation. Dynamic distributed scheduling problems are common, and will only multiply with increasing automation and integration in such areas as manufacturing, distributed communication and control, distributed data management, and air traffic control. Current scheduling practice for applications that are not distributed involves the use of a wide variety of techniques, depending on the detailed requirements for a given system. We expect that the choice of solution methods for distributed scheduling problems will be equally sensitive to these details.

Our current simulation makes a number of simplifying assumptions about how the system operates, both at an individual level and in the interactions between agents. These assumptions can be broadly grouped into those related to the areas of synchronization, expectation, and contract performance.[4] For example, we currently assume that the agent accepting a contract will necessarily complete the associated task by the specified time. Relaxing this assumption will require a number of system modifications, including providing some way to penalize agents for contract violations, and some way for contracting agents to realize that a task is late and may never be completed. The more complicated and dynamic the agents' behavior becomes, the more benefits we expect to realize from the flexibility of the underlying constraint-based representation of the schedule.

We also have plans for future work in the area of inter-agent synchronization. We hope to utilize the constraint-based scheduling capabilities of the individual agents more fully by allowing them to negotiate over the enforcement or relaxation of individual constraints. For example, if DAAC-1 has promised an intermediate data product to DAAC-2 by a certain time, they might negotiate over a modification in that delivery. Providing this capability requires that the negotiation protocol include a language for expressing constraints, including a distinction between modifications which are voluntary ("how about relaxing this deadline by 10 minutes?") and imposed ("I'm going to be late with that data, and there's nothing you can do about it.").

Finally, the agents in this system currently calculate the marginal cost of an added task based on the current schedule, despite the fact that tasks continue to arrive as time passes. We have preliminary results showing that explicit consideration of *expectations* about the arrival of future tasks allows the DAACs to compute a more accurate marginal cost, in the sense that the system makes better decisions about what tasks to schedule when. To date, these results do not include the possibility of task exchanges. In future work we plan to test the additional hypothesis that, by improving the bidding function to take into account these expectations, the full multi-agent, contract-exchanging system will also display improvements in net value earned and other performance metrics.

[4] More generally, most of these assumptions involve the complications involved in accounting for the passage of time.

References

Barrett, A., and Weld, D. 1994. Partial order planning: Evaluating possible efficiency gains. *Artificial Intelligence* 67(1):71–112.

Boddy, M.; White, J.; Goldman, R.; and Short, N. 1994. Planning applications in image analysis. In Hostetter, C. F., ed., *Proceedings of the 1994 Goddard Conference on Space Applications of Artificial Intelligence*, 17–28. Available as NASA Conference Publication 3268, also in Robotics and Computer-Integrated Mfg, V. 11, No. 2, pp 105-110, 1994, Elsevier.

Boddy, M.; White, J.; Goldman, R.; and Short, N. 1995. Integrated planning and scheduling for earth science data processing. In Hostetter, C. F., ed., *Proceedings of the 1995 Goddard Conference on Space Applications of Artificial Intelligence and Emerging Information Technologies*, 91–101. Available as NASA Conference Publication 3296.

Boddy, M.; Carciofini, J.; and Hadden, G. 1992. Scheduling with partial orders and a causal model. In *Proceedings of the Space Applications and Research Workshop.*

Campbell, W.; Short, Jr., N.; Roelofs, L.; and Dorfman, E. 1991. Using semantic data modeling tech-

niques to organize an object-oriented database for extending the mass storage model. In *42nd Congress of the International Astronautical Federation.*

Dozier, J., and Ramapriyan, H. 1990. Planning for the eos data and information system (eosdis). In *The Science of Global Environmental Change.* NATO ASI.

Fox, M., and Smith, S. 1984. Isis: A knowledge-based system for factory scheduling. *Expert Systems* 1(1):25–49.

Lin, F. C. H., and Keller, R. M. 1986. Gradient model: A demand-driven load balancing scheme. In *Proc. Int'l Conf. on Distributed Computer Systems*, 329–336.

Muscettola, N. 1993. Hsts: Integrating planning and scheduling. Technical Report CMU-RI-TR-93-05, The Robotics Institute, Carnegie Mellon University.

Ni, L. M.; Xu, C. W.; and Gendreau, T. B. 1985. A distributed drafting algorithm for load balancing. *IEEE Trans. Software Engineering* SE-11(10):1153–1161.

Sadeh, N., and Fox, M. 1990. Variable and value ordering heuristics for activity-based job-shop scheduling. In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management, Hilton Head Island, S.C.*

Sandholm, T. 1992. Automatic cooperation of area-distributed dispatch centers in vehicle routing. In *Proc. Int'l Conf. on AI Applications in Transportation Engineering*, 449–467.

Sandholm, T. 1993. An implementation of the contract net protocol based on marginal cost calculations. In *Proc. National Conf. on Artificial Intelligence*, 256–262.

Smith, S. F.; Ow, P. S.; Potvin, J.-Y.; Muscettola, N.; and Matthys, D. C. 1990. An integrated framework for generating and revising factory schedules. *Journal of the Operational Research Society* 41(6):539–552.

SMITH, R. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Computers* 29.

Wellman, M. P. 1993. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of AI Research* 1:1–23.