Centralized, Distributed or Something Else? Making Timely Decisions in Multi-Agent Systems

Tim Harbers, Rajiv T. Maheswaran and Pedro Szekely
University of Southern California - Information Sciences Institute
4676 Admiralty Way, Suite 1001, Marina Del Rey, CA 90292
{tharbers, maheswar, pszekely}@isi.edu

Abstract

In multi-agent systems, agents need to share information in order to make good decisions. Who does what in order to achieve this matters a lot. The assignment of responsibility influences delay and consequently affects agents' abilities to make timely decisions. It is often unclear which approaches are best. We develop a model where one can easily test the impact of different assignments and information sharing protocols by focusing only on the delays caused by computation and communication. Using the model, we obtain interesting results that provide insight about the types of assignments that perform well in various domains and how slight variations in protocols can make great differences in feasibility.

Introduction

A multi-agent team is a collection of independent entities that make observations and decisions to achieve or optimize some common goal. Agents must efficiently share knowledge about the environment, their decisions and the team goal in order to perform well. Software agents that facilitate coordination in domains like disaster rescue, joint military operations, and project management must additionally do so in a timely manner. Appropriate information should get to agents before the moments at which decisions need to be made. The best approach depends on the structure of the problem and delay due to communication and computation.

Agents may attempt to pull information ("how good is it if I do this?") or push information ("I'm going to start doing..."). This can involve gathering data from multiple sources and processing it to produce desired or relevant output. Should a single agent be responsible for this? If the team goal and interactions can be decomposed, should the processing be distributed to as many agents as possible? Perhaps a partially-centralized approach is best, but how does one determine the optimal partition of responsibility?

In order to answer these questions, we need a way to handle diverse assignments and a variety of protocols for processing and propagating information for any given task decomposition and event sequence. In this paper, we propose a model of an agent and agent interaction based on message processing to capture the impact of computation and communication delays. We abstract the detailed logic that agents use for decision making into rules that govern the generation and transmission of messages. We implemented a simulator based on this model where we could elicit the timing information from messages propagated under these rules.

Motivated by a project to build a team of coordinator agents, we needed to investigate if our protocols for sharing information were feasible or could be improved with intelligent assignment of responsibility. Our model and simulator allowed us to test various protocols and assignments in different circumstances which yielded insights into protocol feasibility. In addition, we discovered that certain types of partial centralization lead to improved performance with respect to minimizing delay in getting information to agents.

The question of centralization vs. decentralization is fundamental and has been addressed for problems such as multi-processor scheduling and distributed constraint optimization. Queueing theory offers insights for many models for evaluating delay in networks. However, this is the first attempt to combine agent assignment, information sharing protocols and delay to analyze performance of multi-agent systems in time-critical domains. This is an important problem because in real-time dynamic environments, delayed information can make the quality of information useless.

Agent Interaction Model

Our model of a multi-agent system (MAS) focuses on aspects that effect the timeliness of decision-making. The reasoning that contributes to the quality of the decision-making is abstracted away. Thus, we are concerned with how information about *events* (e.g., observations of the environment, agents' actions or queries) are propagated through the system where delay is the primary metric.

Single Agent Model

We consider an agent (see Figure 1) to be a processing resource. An agent is either busy or free. Processing is triggered by *messages* that are taken from one of three *boxes*: the *inbox*, *todobox*, and *outbox*. The inbox stores messages received from other agents or events generated at the current agent. Processing an inbox message is equivalent to deserialization. The result will be the creation of messages that go to the todobox. Messages in the todobox represent local computations. Processing a todobox message can lead to the creation of messages in the todobox or outbox of the same

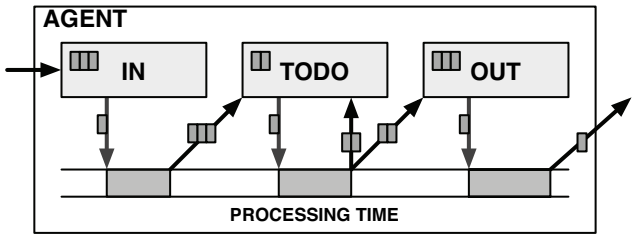

Figure 1: Agent Model

agent. Messages placed in the todobox model the need for further processing. Messages placed in the outbox model information to be passed to other agents. Processing an outbox message encapsulates the costs of serialization or lookup, and creates a message in the inbox of the appropriate agent.

Team Goal and Structure

The team goal of a multi-agent system is either to achieve some state or optimize a common reward function. The structure that describes how agents actions and observations affect the team goal can be represented as a graph. Nodes represent whatever makes sense in the domain (agents' actions, reward function components, or environmental information). An example of this for a hierarchical task structure can be seen in Figure 2. Each node has an *owner* which is an agent in the system. The nodes are classified as either *external* (the owners are fixed by the domain) or *internal* (the owner can be determined by the system designer). A key investigation in this paper is the best strategy to assign owners to internal nodes. Any two nodes may have multiple *relationships* represented as "typed" directed links in the graph. In Figure 2, relationships include "parent" and "enabler". This can model most common MAS formalisms (CTAEMS/HTNs, DCOPS/DCSPs, MMDP/Dec-MDPs, Market/Graphical-Games).

Message Processing Model of a MAS

For a given problem, we have a graph G as a set of external nodes $\mathcal{N}^E$, internal nodes $\mathcal{N}^I$ and relationships $\mathcal{R}$. The message processing model of a multi-agent system is constructed as follows. The message abstraction does not contain "data" as we care only about the delay in information propagation. Each message has a *class* that categorizes its processing time requirements and also, how many and what type of messages are created after processing. Formally, a message m is a tuple, $(c,n_s,n_d;a,b,t)$ where m has class $c \in \mathcal{C}$, with a single source and destination node, $n_s,n_d \in \mathcal{N} = \mathcal{N}^I \cup \mathcal{N}^E$ that appeared in box $b \in \{b^{in},b^{todo},b^{out}\}$ of agent $a \in \mathcal{A}$ at time t.

Let $p : \mathcal{C} \rightarrow \mathbb{R}$ represent the computational cycles a message requires by class. Let $d(a_1,a_2)$ be the inter-agent delay for communicating between agents a_1 and a_2 [1]. A message selection function, $s(\cdot)$ determines how an agent chooses a

[1] While p and d can be distributions, they are deterministic functions in this paper

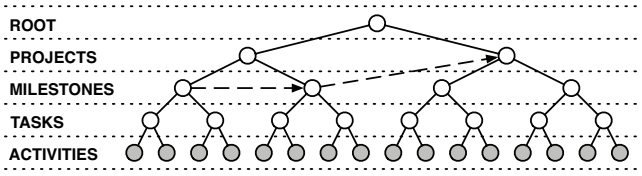

Figure 2: Task Hierarchy Graph

message from its three boxes when it is not busy. A message creation function, $h(\cdot)$, contains the rules by which new messages are generated when a message is processed. An ownership assignment function $w : \mathcal{N} \rightarrow \mathcal{A}$ denotes which agent is responsible for managing a given node. Once G,d, p,s,h,w are given, we can simulate the multi-agent system for any event stream $\mathcal{E}$. An event stream is a collection $\{(c,a,t)\}$ whose elements denote the class of a message that arrives in the inbox of a given agent at a given time. While the designer has control to varying degrees over many of the parameters discussed above, the focus of our investigation is the impact of the ownership assignment w over the internal nodes $\mathcal{N}^I$ on delay in processing events. Given an ownership assignment, message creation and processing are the causes of delay in propagating information. Because the message creation function, $h(\cdot)$, is responsible for the way this information is propagated, it plays a key role.

Message Creation

Here, we consider a message creation function based only on classes and relationships [2]. Suppose that a node n_s is related to node n_d via a relationship r. The $h(\cdot)$ function specifies what new messages to create on node n_d when the agent owning node n_s processes a message of class c. This depends on the relationship between n_s and n_d, namely r. Thus, the $h(\cdot)$ function takes a class c and a relationship r as input and can be expresses as $h(c|r)$. The output of $h(c|r)$ is a collection of classes which is a subset of all possible classes. The $h(c|r)$ function will create a set of classes for any node that shares the r relationship with node n_s [3].

When an agent chooses a message from its inbox, it contains information from a different agent that affects a node that it owns. Let us call this the *relevant* node. It then looks at the class of the message and puts a message in its own todobox with same class and relevant node. Let us call this the *relevant* class. When an agent chooses a message from its todobox, the agent looks at the relevant class c, and finds all relationships r for which $h(c|r)$ is non-empty. For each of these relationships, we then find all nodes that share that relationship with the relevant node. Let us call these nodes the *affected* nodes. For each affected node, and for each element $\tilde{c} \in h(c|r)$, a message of class $\tilde{c}$ is created. This message is put in the agent's own todobox, if the agent owns the affected node. Otherwise, it is put in the agent's outbox. When an agent chooses a message from its outbox, it sends a message with the same class to the inbox of the agent who

[2] Rules based on agents can represent heterogeneous reasoning

[3] Note $h(c|r) = \emptyset, \forall r \in \mathcal{R}$ denotes a terminal message class

owns the affected node.

Once a message is chosen to be processed, it is no longer considered for selection in the future. Events instigate the cascade of messages. To make timely decisions, appropriate agents must have the required information as quickly as possible after an event occurs. Messages are tagged with an identification of the event that triggered it and the time they finished processing. Once can then use any desired delay metric based on these parameters to evaluate the performance of the system. Our metric is the time from event occurrence until the time when the last message tagged with that event identification is processed.

We do not consider issues such as multithreaded processing, heterogeneous agents, memory limitations, multiple ownership of nodes or noisy communication, but the model can be extended easily to capture those situations. Nevertheless, the model described above can capture many multi-agent system protocols and in particular, a diverse set of approaches that we considered for a coordination problem discussed forthwith.

The Real-Time Scheduling Problem

We consider a real-time multi-agent system where software agents re-plan and re-schedule activities in response to dynamic events in an uncertain environment such as disaster rescue or joint military operations. The team goal is represented by a graph. We begin with a tree where the goal or objective function (root node) is decomposed recursively into subtasks (intermediate nodes) until we reach potential activities (leaf nodes) [4] In addition, there can be other dependencies, such as enabling, between nodes, represented by a directed link between them. An enabling link means that, in order to succeed, any descendant of the destination node must begin after the source node has been accomplished. An example of this can be seen in Figure 2.

The agents have an initial schedule of activities to perform. However, the durations and resulting qualities of these activities are uncertain. Thus, both the set of activities being performed and the timing need to be changed dynamically in order to increase performance or achieve a goal.

The leaf nodes (potential activities) can only be performed by certain agents. Thus, they are external nodes in our model, and the owners are fixed. The root and intermediate nodes (internal nodes in our model) can be assigned to any agent. These nodes represent some computation that evaluates the status and projects the future of the task associated with the node based on information from directly related tasks. Assigning all internal nodes to a single agent is equivalent to having a single manager for the whole project (centralization).

In a dynamic real-time system, agents will send queries about potential activities, such as "what if I do this?", "what if I don't do this", or "what if I delay this?" This information must be processed through all the relevant nodes to know the impact on the team reward before an agent can make a decision. The node assignment question is important because we want all agent queries to be answered as quickly as possible. When agents get their queries answered, they will have more information when they make decisions, which generally results in better decisions.

[4]For this paper, the names of the intermediate layers are significant only to identify the depth of the nodes

Message Processing Model of Problem

Relationships ($\mathscr{R}$): Given a source and destination node from a graph, they can have a *parent* or *child* relationship (through the tree), or a *source* or *target* relationship (through directed enabling links). Also, all nodes have a relationship with themselves, labeled *self*.

Classes ($\mathscr{C}$): Classes are important because they determine how messages propagate throughout the system, which reflect the underlying reasoning being used by the agents. We have investigated and represented many schemes used in the real system within this model. Here, we discuss the propagation protocols that result from two of such schemes.

The first, *behavior protocol*, has one message class, c^{beh}. These messages contain information about new behavior that an agent has or is considering at a certain node. An agent receiving a behavior-update message will calculate the new behavior of any affected node it owns. Once new behavior is determined, a message is sent to the parent and target nodes of the affected node as they are affected by this change.

The second type, *cost protocol*, adds a second message class, c^{cost}. These messages capture the resource costs of all descendant leaf nodes (activities) for any given behavior. If a new cost message is received (from a child node), an agent must update its own cost estimates and investigate a potential change in its behavior. The node then informs its parent about its new costs and its parent and targets about its new behavior.

Message Creation Function (h) and Message Selection Function (s): Both protocols above are captured by the following message creation function: $\{h(c^{beh}|parent) = c^{beh}, \quad h(c^{beh}|target) = c^{beh}, \quad h(c^{cost}|parent) = c^{cost}, \quad h(c^{cost}|self) = c^{beh}\}$. For all other conditions, $h(c|r) = \emptyset$. The interplay between cost and behavior messages is that cost messages create behavior messages, which means that additional behavior messages are propagated through the graph. The creation of classes for these protocols are visualized in Figure 3. The root has no parents or targets by definition, and thus does not propagate any messages. Because all other nodes have parents, all message flows terminate at the root for these propagation protocols. The selection function prioritizes the boxes as $(1)out, (2)todo, (3)in$ and then chooses the earliest arriving message within the highest priority box.

Experimental Setup

Delay (d) and Processing Cost ($p(c)$): The goal of receiving information as quickly as possible will be hindered by message processing and message communication. We defined $d(a_1, a_2)$ as the communication delay: the time it takes a message to get from agent a_1 to agent a_2. For all agents $a_1, a_2 \in \mathscr{A}$, $d(a_1, a_2) = d$ was a constant throughout a given experiment. Note that these delays are only incurred when

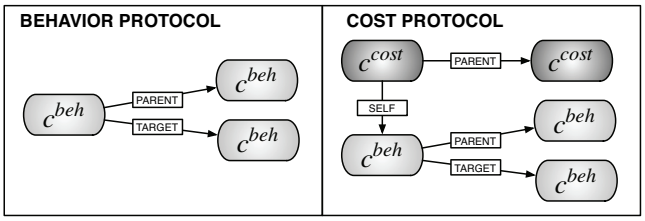

Figure 3: Propagation Protocols

a message is transmitted between nodes owned by different agents. Otherwise, new messages are put directly into the todobox of the same agent.

The computational cycles taken to process a message from the inbox and the outbox was fixed to 1 *time unit* as a reference. The cycles taken to process a message from the todobox (denoted p) was held constant over all classes for a single experimental trial. This means that $p(c^{beh}) = p(c^{cost}) = p$ whenever processing a todo message. We investigated many combinations of d and p, as we will discuss later.

Graph (G), Nodes($\mathcal{N}$) and Agents ($\mathcal{A}$) : Graphs were generated using the structure shown in Figure 2 with 4 children per node at all levels except that the number of milestones per project could vary between 4 and 6. Thus, graphs had between 85 and 125 internal nodes and between 256 and 386 external nodes. Graphs either had no enabling relationships or "many" enabling relationships. [5] The number of agents in each trial were chosen such that the ratio of nodes to agents was approximately 5. Thus, we had between 68 and 102 agents.

Ownership Assignments (w): : This is the main property we wished to investigate. It is also the property over which we as designers have the greatest control. While the graph is constrained by the scenario and the protocol must facilitate the transfer of useful information, we can employ almost any arbitrary assignment. At the top of Figure 4, we display the following 7 assignment classes.

- (**C**) a single agent owns all nodes.
- (**T**) a single agent owns the root, all projects and milestones; tasks are distributed among remaining agents.
- (**M**) a single agent owns the root and all projects; milestone owners also own the tasks below them.
- (**MT**) a single agent owns the root and all projects; milestones and the tasks are distributed among remaining agents.
- (**P**) a single agent owns the root; project owners own the milestones and tasks below them.
- (**PM**) a single agent owns the root; the projects and milestones are distributed with milestone owners also owning the tasks below them.
- (**D**) all nodes are distributed among all agents.

[5] In the "many" case, for every node, sequentially, a target was chosen uniformly from the remaining nodes. An enabling link was added if it would not create a cycle.

Here, C is centralization, D is full distribution, and the others are various forms of partial centralization.

Event Stream ($\mathcal{E}$) and Performance Evaluation: Events represent changes in the environment that agents must address. We model events using messages arriving at the external nodes (activities). Events arrive at discrete times that we call *pulses*. Pulses are generated every 500,000 time units. We chose this number so that if processing an inbox message takes 1 microsecond, then pulses arrive every half a second.

We generated an event stream by choosing the number of events per pulse and the number of empty pulses between event-arrival pulses from normal distributions rounded to the nearest non-negative integer. The main experiments used $N(6,4)$ for both distributions. In our experiments, the class of all events are c^{cost} when using the cost protocol and c^{beh}, under the behavior protocol. Events are deposited uniformly over the external nodes (activities) in the graph. All messages that result from this event are tagged with a unique event identification. The difference between the time when the final message from a particular event has been processed and the time this event entered the system is stored as the *event-propagation time*. Because the goal is to minimize the time taken to propagate information, we used the average event-propagation time as a metric to evaluate performance.

Experimental Results

We focus on three scenarios. The first has no enablers and uses only the behavior protocol. The second has many "enabling" nodes and uses only the behavior protocol. The third has many "enabling" nodes and uses the cost protocol.

A single experiment consisted of a graph and an event traffic stream as described earlier with a given assignment, communication delay and todobox message processing cost. For each trial of that experiment, we measured the average event-propagation time in pulses (instead of time units). An experiment was created for all 7 assignments, all communication delay values $d \in \{0, 5000, 10000, \cdots, 50000\}$ and all todobox message processing costs of $p \in \{0, 1000, 2000, \cdots, 10000\}$ (847 experiments). The step sizes for delay and processing were doubled for the many-enablers cost-protocol scenario, because the average event-propagation times were very large (252 experiments).

We ran 61 unique trials for each experiment to ensure statistical significance. Figures 4 shows the combined results for all three scenarios. For each scenario, the upper-left "landscape" graph has a symbol for each delay-processing cost pair. This symbol refers to the set of assignments that were *not dominated* by any other assignment. If an assignment is *not* in the set marked by the symbol for a particular pair, some other assignment was better, as verified by a *t-test* with 99% significance probability. The assignments in the set can be considered the "winners" for that situation. In addition, three other bar charts show the average event-propagation time in pulses for all assignment for three delay-processing cost pairs. These three instances are shaded with a grey background in the landscape graph. Our basic expectations are that a centralized scheme will win when process-

ing cost is very low, and a distributed scheme will win when delay is very low.

In the no-enablers behavior-protocol scenario, our basic expectations are met. For the majority of the landscape, partially-centralized schemes tend to be the winners. Looking deeper, we see that assignments that centralize at the bottom (P, PM) are the ones that dominate the landscape graph, unless processing costs are very low. We also note that the delay had to be 15 times the processing cost for centralization to be dominant. However, in all cases, the average event-propagation time was less than half a pulse. Thus, it seems that without "enabling" relationships, the assignments are essentially equivalent.

In the many-enablers behavior-protocol scenario, where almost every node is a source of an "enabling" relationship, we expect that all agents and especially the root will be much busier. Thus, we suspect that assignments which do not overload the root agent's responsibility should do best. Our basic expectations regarding centralization and distribution hold again. However, centralization only wins when processing costs are zero and distribution is bettered by one of the partially-centralized schemes (PM). Assignments that overload the root, such as centralization and T (which gives a single agent all nodes in the top three levels), tend to perform poorly. Assignments that distribute the top two levels (P, PM, D) tend to the best.

In the many-enablers cost-protocol scenario, we expect that the root will be loaded even more because the cost message create additional behavior messages that travel to the root. The main result is that for most assignments, the average event-propagation time is over 100 pulses. This indicates that the protocol is infeasible in most dynamic domains, regardless of the quality of the information it shares. Agents will not get the information before decisions need to be made. The only region where it may be feasible is when the processing cost is extremely low and one uses an assignment that distributes the top two levels (P, PM, D). Even in these cases, the average event-propagation time is about 10 pulses, which is quite high. Interestingly, in this scenario, distribution is dominated in many situations where there is zero inter-agent delay. This goes against our basic expectations of performance.

Finally, we note that processing cost seems to have a far greater impact on average event-propagation time than delay, especially in many-enablers scenarios. When comparing the (10000,10000) and (50000,2000) delay-processing-cost pairs, the former is an order of magnitude worse.

Related Work

This agent assignment problem is similar to the the Multiprocessor Scheduling Problem (Garey & Johnson 1979). In the latter, given an acyclic graph of nodes, or jobs, each node has to be assigned to a processor, or agent, and takes a certain amount of time to be processed. A node can have predecessors (all the nodes pointing to it) that have to be processed first. A communication delay exists for every pair of nodes. The goal is to minimize the overall processing time, i.e,. to minimize the time for the final job to complete. The problem

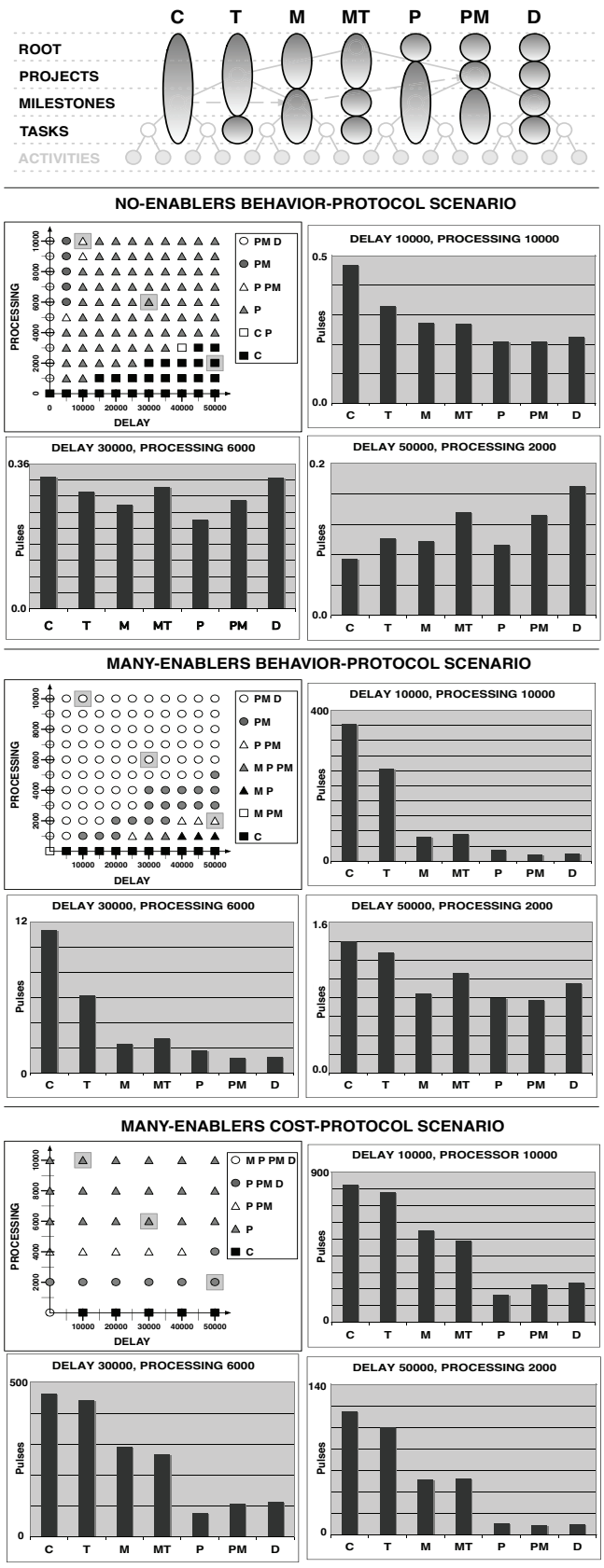

Figure 4: Ownership Assignments and Results

is NP-hard (Garey & Johnson 1979) and many algorithms to solve it have been proposed.

However, there are some important differences. In our model, a node can trigger processing more than once. Furthermore, an agent's work is determined by message queues and the processing is dependent on a complex protocol and dynamic events, whereas in the MSP, the processing order is fixed. Finally, we assume that some nodes have fixed owners (external nodes), so the choice of assignment is not completely unrestricted.

Our problem is also similar to the Distributed Constraint Satisfaction Problem (DCSP) (Yokoo *et al.* 1998) and Distributed Constraint Optimization Problem (DCOP) (Petcu & Faltings 2005). In DCSPs and DCOPs, every agent has its own nodes (or variables) and has to reach a solution by communicating with other agents over inter-agent constraints. A centralized assignment reduces the problem to a CSP (COP). In addition, there are distributed (Modi, Shen, & M. Tambe 2005) and partially-centralized (Mailler & Lesser 2004) approaches and the impact of centralization has been investigated (Davin & Modi 2005). The impact of cluster sizes for partial-centralization on the number of terminal solutions for approximate DCOP algorithms was also investigated (Pearce, Maheswaran, & Tambe 2006). A significant difference is that DCOPs are attempting to maximize the quality of a static problem while we are trying to maximize the timeliness in a dynamic environment.

Because our model makes use of message processing and tries to analyze the effect of traffic patterns, our work is also related to queueing theory.(Gross & Harris 1985) Events are generated using a particular distribution and propagated through the network, with the overall goal of minimizing delay. The main difference is the use of agent assignment to nodes, which is not a critical issue in queueing theory.

Conclusion

In this paper, we propose a model of a multi-agent system based on message processing to capture the effects of computation and communication in terms of delay. Our primary interests were to (1) study the impact of agent assignment on the timeliness of processing events in a dynamic environment and (2) evaluate the feasibility of information sharing protocols. We implemented a simulator based on this model and tested scenarios motivated by a real-time scheduling problem. In the question of centralized vs. distributed, we found that the best was usually something else. In domains with no enablers, centralization at the lower levels of a task hierarchy led to better performance. In domains with many enablers, distributing the top levels of the task hierarchy was essential to get good performance. While these may seem intuitive in hindsight, we were unable to predict exactly which partially-centralized schemes would perform best and what the landscape graph would look like *a priori*. In addition, we found that basic expectations, such as where distribution should dominate, didn't always hold true. A key result was that a small change in protocol can make a huge difference in feasibility. Thus, even if the protocol led to very good information, it would be useless in a dynamic environment.

As a multi-agent system designer, it is useful to have a method to evaluate protocols in terms of information delay and optimize with respect to agent responsibility. Our model and implementation allows one to test a large landscape of conditions and schemes in a quick and easy manner. Developing this area of research is valuable because in real-time dynamic environments, information delay is as important as information quality. [6]

References

Davin, J., and Modi, P. J. 2005. Impact of problem centralization in distributed constraint optimization algorithms. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 1057–1066.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Gross, D., and Harris, C. M. 1985. *Fundamentals of queueing theory (2nd ed.)*. New York, NY, USA: John Wiley & Sons, Inc.

Mailler, R., and Lesser, V. 2004. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, 438–445.

Modi, P. J.; Shen, W.; and M. Tambe, M. Y. 2005. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal (AIJ)* 161:149–180,.

Pearce, J. P.; Maheswaran, R. T.; and Tambe, M. 2006. Solution sets for dcops and graphical games. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, 577–584. New York, NY, USA: ACM Press.

Petcu, A., and Faltings, B. 2005. Dpop: A scalable method for multiagent constraint optimization. In *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 05)*, 266–271.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering* 10(5):673–685.

[6]The work presented here is funded by the DARPA COORDINATORS Program under contract FA8750-05-C-0032. The U.S. Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them. We also thank NASA and ONR for support provided under the awards NNA05CS29A (CMMD) and N000014-03-C-0222 (CARTE), respectively.