Distributed Coordination of Massively Multi-Agent Systems

Nadeem Jamali and Xinghui Zhao

176 Thorvaldson Building, 110 Science Place
Department of Computer Science, University of Saskatchewan
Saskatoon, SK, S7N 5C9, Canada

Abstract. Coordination is a key problem in massively multi-agent systems. As applications execute on distributed computer systems, coordination mechanisms must scalably bridge the network distance between where decisions are made and where they are to be enforced.

Our work on the CyberOrgs model[1] addresses this challenge by encapsulating distributed multi-agent computations along with computational and communication resources they require (for carrying out the application's functions as well as for coordinating actions of the agents) plus purchasing power represented by an amount of eCash for acquiring additional resources. Resources are defined in time and space, and are owned by cyberorgs. Resource ownership changes as a result of trade between cyberorgs.

Ownership of resources coupled with an effective and scalable control structure creates a predictable resource environment for multi-agent systems and their coordination mechanisms to execute in. Particularly, the coordination mechanism can reason about the possibility of successful coordinated action based on predictable communication and processing delays.

This paper presents our experience with hierarchical coordination of distributed processor resource for a system of cyberorgs internally distributed across a number of physical nodes. We demonstrate that encapsulation of network resources creates a scalable opportunity for reasoning about distributed coordinated action to support decision making.

Experimental results show that the CyberOrgs based resource-aware approach scalably increases opportunities for successful coordinated distributed actions involving up to 1500 agents (in much larger systems) by reducing the delay in determining their feasibility, as well as helps avoid attempts of infeasible actions.

1 Introduction

A multi-agent computation distributed over a network of computers faces a number of sources of uncertainty. When an agent's decision about the action to take next depends on actions taken by other agents, agents must contend with the uncertainty of other agents' actions. When agents are distributed across a number of physical nodes, both computational as well as resource uncertainties emerge.

Coordination between agents emerges as a key concern for achieving optimal results [5], especially when the computations are distributed [4]. It turns out that requirements

[1] The model is referred to as *CyberOrgs*, and the entities are referred to as *cyberorgs*.

of computation and coordination can be treated as separate and orthogonal dimensions of computing [6], leading to an opportunity for separating concerns and addressing coordination explicitly.

Coordination presents significant challenges when agents execute in a distributed environment with a number of processors connected by communication networks. Specifically, coordination mechanisms must bridge the network distance between the agents whose actions need coordination. This is a difficult problem because network delays are generally unpredictable, in large part because network performance goals are typically systemic, and applications are free to engage in virtually unrestricted competition for network resources, leading to resource dependencies. This is part of a more general problem. In *open systems* [7], when there are both *logical* and *resource* dependencies [5] between agents, resource dependencies sometimes lead to logical dependencies. Unrestricted competition for resources between agents collaborating to achieve a shared goal may hamper progress toward the goal. Coordinating resource access by agents is hence critical to reducing uncertainty and enabling agents to make control decisions for the best global performance [12].

In a bounded resource environment, if a computation can launch other computations as in a multi-agent system, it is difficult to control resource consumption reactively. If an erroneous or malicious agent begins creating other agents with similar characteristics, and if the only mechanism employed for identifying such agents is observation of their own threatening behavior, the rate of growth in the number of agents can be shown to be exponential. Intuitively, this means that irrespective of how conservatively the system purges misbehaving agents, so long as the mechanism relies solely on the observation of individuals' suspicious activity, by the time the system reacts, it may be too late: other agents have potentially been created about whose behavior the system will know nothing until it has observed them individually.

Our approach to controlling such behavior is by bounding resource consumption at the outset, and limiting resources available to a multi-agent computation. In this approach, each agent would receive a resource consumption allowance, which it could utilize or give a part of to other agents. Our work on the CyberOrgs model [9] uses this approach by encapsulating distributed multi-agent computations inside hierarchically organized resource encapsulations. [10] described scheduling strategies for efficiently controlling processor resource for a hierarchy of cyberorgs.

The difficulty of resource coordination is compounded by distribution of the resources. A number of approaches have been used to address the problem. [13] introduces a hierarchical scheduling scheme to apply a set of algorithms that enforce various processor usage constraints. Although the scheduling scheme is used for mobile programs, the interaction paradigm is client/server. Furthermore, network resource is not considered. [14] addresses network delay in the context of distributed scheduling, and provides approximation algorithms for distributed task scheduling problems. This approach focuses on specific global objectives of scheduling, such as minimizing the makespan, minimizing the average completion time, etc. Therefore, for a given network, they have specific ways to schedule the tasks, and no complex coordination is involved. Furthermore, the tasks are assumed to be unrelated, and there are no interactions or constraints between tasks. Coordination between distributed schedulers is

considered in [3], for providing multimedia to multiple clients without conflict. Each scheduler is located on one computer and only has a partial view of the global schedule, requiring the schedulers to coordinate. Because the schedulers have the specific purpose of preventing access conflict, and the distributed clients do not interact with each other, coordination is simple.

Our approach is to carry out resource coordination within distributed resource encapsulations provided by the CyberOrgs model. When a computation can rely on the availability of computational and communication resources, its coordination mechanisms can be assisted by the knowledge in making decisions about coordinated distributed action. This improves overall efficiency by avoiding infeasible coordination attempts.

2 CyberOrgs

CyberOrgs [9] is a model for resource sharing in a network of self-interested peers, where application agents may migrate in order to avail themselves of remotely located peer-owned resources. CyberOrgs organize computational and communication resources as a market, and their control as a hierarchy. Specifically, each cyberorg encapsulates one or more multi-agent distributed computations (to be referred to as computations contained in the cyberorg), and an amount of *eCash* in a shared currency. Cyberorgs act as principals in a market of distributed resources, where they may use their *eCash* to buy or sell resources among themselves. A cyberorg may use the resources so acquired for carrying out its computations, or it may sell them to other cyberorgs.

CyberOrgs treat computational and communication resources as being defined in time and space. In other words, a resource is not available for use before or after the instant of time at which it exists. Sale of a resource is represented by a *contract* stipulating availability of resources to the buyer for a cost. Delivery of resources to cyberorgs is determined by a hierarchy of control decisions. In other words, cyberorg a makes control decisions required for delivery of resources purchased from it by cyberorg b; cyberorg b in turn makes control decisions determining how the resources purchased from it by cyberorg c are to be delivered. Cyberorgs may pre-pay to buy resources which will exist in the future. Cyberorg b may use the resources it owns only if the resources exist at a time when the cyberorg is being hosted by a. In other words, after signing a contract, a cyberorg must migrate to the prospective host cyberorg in order to avail itself of newly acquired resources. Additionally, if b migrates from a while it owns future resources through a contract with a, it cannot use those resources except if it eventually returns to a and if it possesses resources which have not yet expired.

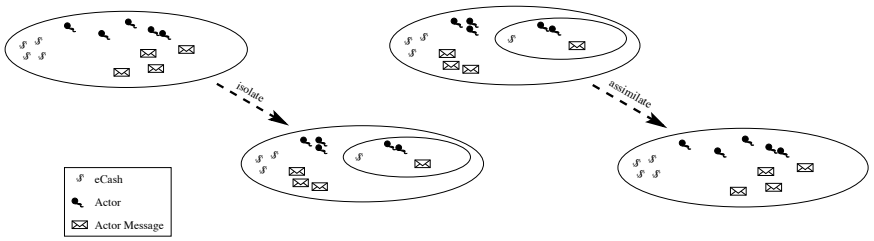
The CyberOrgs model separates concerns of computations from those of the resources required to complete them.

We assume that computations are carried out by primitive agents called actors [1], and we represent the resource requirements of each computation by the sequence of resources required to complete it. *Ticks* serve as the unit of a consumable resource such as processor time. Every computation requires a certain number of ticks to complete.

Progress is represented by transitions occurring with introduction of ticks into the system. When a tick is inserted into a cyberorg, it may pass the tick on to a client

cyberorg, use it for progressing on its system operations (such as for carrying out primitives) or on its actor computations. Whether a tick is passed on to a client or used locally depends on the contracts that the cyberorg has with its clients.

As illustrated in Figure 1, a new cyberorg is created by using the `isolate` primitive, which collects a set of actors, messages, and electronic cash, and creates a new cyberorg hosted locally.[2]



a) CyberOrg Isolation

b) CyberOrg Assimilation

Fig. 1. Creation and Absorption

A cyberorg disappears by assimilating into its host cyberorg using the `assimilate` primitive, relinquishing control of its contents to its host.

A cyberorg may realize that its resource requirements have exceeded what is available by its contract with the host cyberorg. This triggers its attempt to migrate. The tasks required for a cyberorg to migrate are: search (for a potential host), negotiate (a contract with potential hosts), and migrate (to a selected host).

A more formal treatment of the operational semantics may be found in [9].

3 Distributed Coordination

The way cyberorgs encapsulate computational and communication resources creates unique opportunities for scalable distributed coordination. Because delivery of network and processor resources to computations is controlled at a fine grain, idle resources are known precisely. As a result, communication and processing delays in carrying out fixed length system communications required for coordination become predictable. This - in turn - allows the distributed coordination components to reason about the feasibility of coordinated action based on good estimates of delays, and attempt only promising coordinated actions.

We use coordination of distributed processor resource delivery as an example of a distributed coordination problem. A single cyberorg may be internally distributed in that it may own computational resources at a number of physical nodes, on which its agents reside. Network resources would be additionally required by the cyberorg to enable

[2] These primitives bear some similarity to those of the Interaction Abstract Machines (IAM) [2].

communication between its distributed agents. Coordination decisions for the cyberorg may be local to a processor, or they may involve multiple processors. In cyberorg implementations, meta-agents called *facilitators* are responsible for making resource decisions for cyberorgs and interacting with the the processor scheduler to secure those resources. Facilitators, being agents, also require resources for executing. For a distributed cyberorg, there are as many facilitators as the number of processors on which parts of the cyberorg are located. One way of organizing these facilitators is to designate one of them as the *master* facilitator. The master facilitator maintains information required for global scheduling decisions; other *(slave)* facilitators maintain sufficient information for making local scheduling decisions autonomously. Coordinated global scheduling actions implementing cyberorg primitives offer a significant challenge in the context of unpredictable communication delays. However, cyberorgs allow communication as well as computation delays to be locally known based on knowledge of idle resources. If a master facilitator knows the global state of network and remote processor resource availability, it can use that information to predict delays in communicating with other facilitators, based on which it can determine whether a certain primitive operation can in fact be carried out. Particularly, assessment of feasibility of a coordinated action to implement a distributed cyberorg primitive can be made before actual communication with other facilitators regarding the specific operation.

3.1 Coordination among distributed schedulers

Coordinated action involving a number of nodes hosting parts of a cyberorg requires prior agreement between the nodes. Therefore, there is a minimum delay between when the action is conceived and when it can actually be carried out at each of the nodes. If this delay can somehow be estimated, actions requested for sooner than this delay can be summarily dismissed. Given the benefit of knowledge of resource availability in a system of cyberorgs, here we attempt to calculate this delay.

Consider a 2-node (N_1, N_2) request for a coordinated primitive operation (Figure 2) to be carried out across the nodes hosting a cyberorg. The delay Δ in reaching agreement consists of several parts: D_1, the time delay from when the request is generated by the master facilitator on N_1 (or received from a slave) to when the request is scheduled to be processed on N_1; $P(a)$, the computational cost of analyzing the request and creating distributed tasks (where a is the total number of agents involved in the request); $C(N_1, N_2)$, the network delay in sending a message from N_1 to N_2[3]; D_2, the delay from when the request is received on N_2 to when the request is scheduled to be processed on N_2; $P(a_2)$, the computational cost of interpreting the request, and evaluating its feasibility on N_2 (where a_2 is the number of agents on node N_2 involved in the primitive); $C(N_2, N_1)$, the network delay of sending an acknowledgment back from N_2 to N_1; D_3, the delay from receipt of the acknowledgment to when the message is processed on N_1. Therefore, the coordination cost should be:

$$\Delta = D_1 + P(a) + (C(N_1, N_2) + D_2 + P(a_2) + C(N_2, N_1)) + D_3 \qquad (1)$$

[3] We assume that clocks are synchronized within some epsilon.

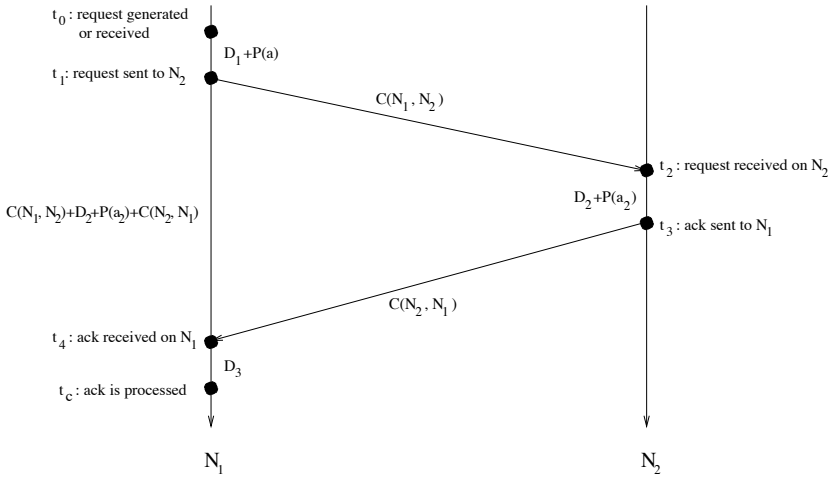


Fig. 2. Cost of coordination between distributed schedulers

The time delays D_1, D_2, and D_3 can be estimated from details of CPU scheduling. Figure 3 shows a scheduler cycle of length of l, where s is the time slice allocated to the facilitator responsible for processing the primitive request. If the primitive request arrives during the time interval $[T_0, T_s]$, probability of which is $p = \frac{s}{l}$, the delay would be 0; if the request arrives during $[T_s, T_l]$, probability of which is $1 - p$, the delay would be non-zero. In the latter case, we can take an average delay as an approximation: $\frac{l-s}{2}$. Therefore, the approximation of time delay between when a request is received on a node and when the request is scheduled is $(p \times 0) + (1 - p) \times \frac{l-s}{2}$, which is:

$$D = \frac{(l-s)^2}{2 \times l} \tag{2}$$

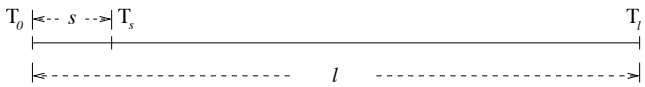


Fig. 3. A scheduler cycle

The communication costs $C(N_1, N_2)$ and $C(N_2, N_1)$ can be estimated from details of network resource control. In the current implementation, network control is message based, and the unit of control is a cyberorg. If r is the network flow rate (messages per second) that a cyberorg receives, the time delay of sending a message from an agent in this cyberorg would be $\frac{1}{r}$. Because a cyberorg processes its messages based on a first come first serve rule, the actual time delay of sending the specific primitive request would be $\frac{m+1}{r}$, where m is the number of messages to be processed before the message carrying the primitive request. After the message in question is processed, it

goes through the network link between N_1 and N_2, and the delay is determined by the bandwidth (b) of the network route and the size of the message (z).

$$C = \frac{m+1}{r} + \frac{z}{b} \tag{3}$$

For convenience, we use a function $f_{N_1,N_2}(r_{12}, m_{12}, z_{12}, b_{12})$ to refer to this network communication cost, where N_1, N_2 are names of nodes.

Using equations 1, 2, and 3, we obtain the approximation cost of achieving group agreement for a coordinated distributed action, which is:

$$\begin{aligned} \Delta = \tfrac{(l_1-s_1)^2}{l_1} + P(a) + f_{N_1,N_2}(r_{12}, m_{12}, z_{12}, b_{12}) + \tfrac{(l_2-s_2)^2}{2\times l_2} + P(a_2) \\ + f_{N_2,N_1}(r_{21}, m_{21}, z_{21}, b_{21}) \end{aligned} \tag{4}$$

Although estimating $P(a)$ for a general purpose computation would be difficult, because we are dealing with special purpose computations for assessing feasibility of local actions, it is possible to obtain good estimates, so long as local resource availability is known, which is in this case.

Equation 4 illustrates the coordination cost of a 2-node distributed primitive. This can be generalized to the n-node case as follows:

$$\begin{aligned} \Delta = \tfrac{(l_1-s_1)^2}{l_1} + P(a) + max(f_{N_1,N_i}(r_{1i}, m_{1i}, z_{1i}, b_{1i}) + \tfrac{(l_i-s_i)^2}{2\times l_i} + P(a_i) \\ + f_{N_i,N_1}(r_{i1}, m_{i1}, z_{i1}, b_{i1})) \end{aligned} \tag{5}$$

for i in $[2, n]$, where N_1 is the node with the master facilitator of the cyberorg. l_i is the length of scheduler cycle on node N_i, s_i is the time slice for which the facilitator agent on node N_i is scheduled, and $f_{N_i,N_j}()$ is the network communication cost of sending a message from N_i to N_j, which depends on network flow rate the cyberorg receives, number of messages to be processed before the specific message, the size of message to be sent, and the network bandwidth between the two nodes, for the path from N_i to N_j.

Optimistic Waits The master facilitator of an internally distributed cyberorg is responsible for making global decisions for the cyberorg, while slave facilitators of the cyberorg are free to make local decisions involving agents on their own nodes.

A global decision of a master facilitator may require modifying resources available to agents spread across multiple nodes. In order to guarantee that the corresponding actions associated with an n-node global decision will be performed successfully by time t on all involved nodes, a master facilitator must generate the decision by time t', so that $t' < t - \Delta$. However, some savings can be obtained by eliminating some communication. Particularly, if the master facilitator can calculate Δ without explicitly communicating with the slave facilitators, it can send requests to the slave facilitators to carry out their parts of the global action, with the knowledge that all actions will indeed succeed. Although this is not possible in general, if the the master facilitator receives periodic updates from the slaves about their locally available resources, along with promises to

maintain those availabilities for certain time intervals, the master may be able to assess feasibility of remote actions so long as the actions can be completed before expiration of the resource availability promises received from the slaves. Specifically, the master may send requests for coordinated actions, wait for the Δ it has independently calculated, and then assume that the actions successfully took place.

If the coordinated action itself is required in the future, the master facilitator may estimate the delay required for agreement on feasibility of the coordinated action in a similar manner. In this case, instead of waiting for each slave facilitator to acknowledge agreement, the master facilitator may be optimistic. In other words, the slave facilitators no longer have to send acknowledgments; they only report back if they find the action infeasible. The master facilitator, in turn, waits for Δ time for possible infeasibility reports, rather than wait for each slave to acknowledge. The master would be able to calculate this Δ if the promises of resource availability received from the slaves do not expire before the slaves finish assessing local feasibilities. Additionally, instead of waiting to be informed by the master of global agreement, the slaves too optimistically wait long enough to give the master a chance to inform them about possible cancellation of the coordinated action. Because the master facilitator calculates Δ prior to communication with the slaves, it can advise the slaves in the initial communication to wait for a period $\Delta + T_r$, where T_r is the time the master would take to report cancellation to them after it has received an infeasibility report from some slave. As a result, in the case when global agreement is achieved, all parties are ready for coordinated action after a delay of $\Delta + T_r$, without any need for communication after the initial requests from the master facilitator. Furthermore, any cancellations too are known by all parties by $\Delta + T_r$.

4 Implementing CyberOrgs

Our implementation of CyberOrgs is developed by extending Actor Architecture [11], which is a Java library and run-time system for supporting primitive agents. We extend Actor Architecture by adding two key components: *CyberOrg Manager* and *Scheduler Manager*.

A CyberOrgs platform is an instance of the system running on a single node, and CyberOrg Manager is the central component of each CyberOrgs platform. All resource control operations on a platform are carried out by the CyberOrg Manager. The results of such operations are sent to Scheduler Manager, which schedules all agents in the platform according to these results.

Algorithm 1 illustrates the algorithm of the Scheduler Manager. It schedules all agents in a loop, and each agent is executed for an amount of time which is allocated to it. After each scheduling cycle, the Scheduler Manager instructs the CyberOrg Manager to update the availability of resources for every cyberorg.

Network resources can be viewed as virtual links between two computers (or nodes) through which the connected computers may communicate with each other by exchanging data. Therefore, a cyberorg which owns network resources must be distributed between multiple nodes, which makes the cyberorg internally distributed.

Algorithm 1 Scheduling Algorithm

```
1: while true do
2:    if the length of the thread queue > 1 then
3:       get the first element from the front of queue;
4:       if the first element is the start flag then
5:          tell CyberOrg Manager to refresh resource records for every cyberorg;
6:       else
7:          schedule the thread for required time slice;
8:          if the thread is alive then
9:             insert it at the end of the queue;
10:         end if
11:      end if
12:   else
13:      sleep for some time;
14:   end if
15: end while
```

An internally distributed cyberorg may own CPU resources on multiple nodes. The CPU resource on a single node can be represented using a tuple, `(address, ticks, ticksRate)`. Here, `address` is the IP address of corresponding node, `ticks` is the total processor time (in milliseconds) the cyberorg can receive, and `ticksRate` stipulates the rate at which CPU resources can be received (e.g., in milliseconds per second).

In this prototype implementation, communication is abstracted as exchange of asynchronous messages.[4] Accordingly, network resource availability is abstracted as fixed-sized messages that can be sent within a unit of time. Therefore, network resources can be represented by `(link, flow, flowRate)`, where `link` identifies the the source and destination of the link, `flow` is the total number of messages that the cyberorg can send through the link, and `flowRate` specifies the number of messages that the cyberorg can send within a unit of time (e.g., per second), which indicates the rate of message flow.

To acquire these resources, a cyberorg negotiates contracts with other cyberorgs who own the resources. In addition to specifying the type of resource, a contract also stipulates the real-time interval (`time`) when the contract is in effect as well as the amount of eCash that the cyberorg must pay for the resources (`Price`). The price may be payable in full in advance (`type: 0`) or at regular intervals (`type: 1`).

Figure 4 shows an example contract. It applies to an internally distributed cyberorg with agents located on two nodes: N_1 with address "128.233.109.163" and N_2 with address "128.233.109.164". The cyberorg is to receive 1000 milliseconds of processor time on N_1, at the rate of 10 milliseconds per second, as well as 2000 milliseconds on N_2, at the rate of 5 milliseconds per second. In every second, the cyberorg is allowed

[4] Although we abstract over actual network bandwidth here by only accounting for messages, we have independently studied the effectiveness of fine-grained network resource control for cyberorgs. Preliminary results in this work show promise for effective fine grained control of network resource delivery [8].

CPU Resource
(“128.233.109.163”, 1000, 10) (“128.233.109.164”, 2000, 5)
Network Resource
((“128.233.109.163”, “128.233.109.164”), 10, 1) ((“128.233.109.164”, “128.233.109.163”), 15, 2)
Time
11:00:00 17:05:30
Price
1 5

Fig. 4. An Example Contract: CPU resources on two nodes; network resources connecting the processors in both directions; start and end time for when the contract is in effect; price (1 is the type of payment and 5 is the price in units of eCash)

to send 1 message from N_1 to N_2, and 2 messages from N_2 to N_1, as long as the total numbers of messages being sent in the two directions are less than 10 and 15 respectively. The contract takes effect at time 11:00:00 and expires at 17:05:30, and the cyberorg receiving the resources must pay 5 units of eCash per second, and the payment is to be made in installments.

Network resource accounting and control are achieved by cooperation between the CyberOrg Manager and the Scheduler Manager. Before sending out a message, the platform checks with the CyberOrg Manager, which checks for availability of network resources for the cyberorg requiring it. If there is enough resource, the message is sent out, and the remaining amount of corresponding type of network resource (the specific link) in the cyberorg is decremented. Otherwise, if enough network resource is not available to the cyberorg, the message is blocked until the required network resource becomes available.

As shown in Figure 5, an internally distributed cyberorg has agents located on different nodes, representing distributed parts of the cyberorg. Each part has its own local facilitator agent, which is responsible for making local decisions and receiving requests for primitive operations involving local agents. The master facilitator maintains global information of the cyberorg, and it alone is responsible for enforcing global decisions of the cyberorg by coordinating its actions with those of other (slave) facilitators. By default, the master facilitator is the facilitator located at the node on which the cyberorg's creation is originally requested requested. Slave facilitators, by themselves, only possess the resource knowledge of their own parts of the cyberorg, and a slave facilitator can autonomously make local decisions involving agents in its own part of the cyberorg.

At an internally distributed cyberorg's creation time, an initial contract is generated by the creating cyberorg. This contract contains information about resources available to the new cyberorg and the terms of their availability. The runtime system can examine the contract to obtain IP addresses of involved nodes. The main part of cyberorg – which holds the master facilitator – is created first on the node where the creation is

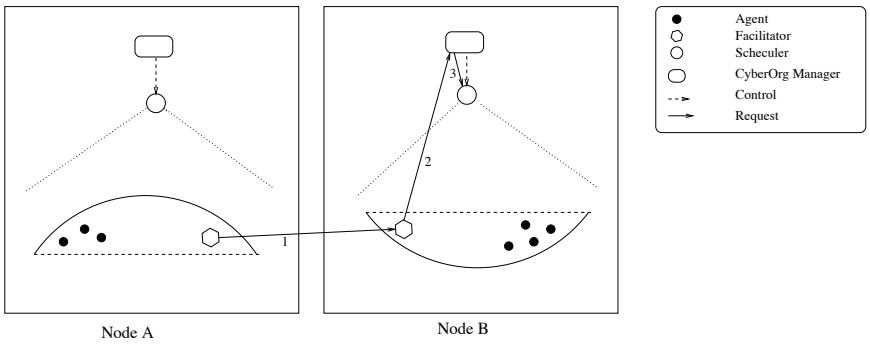


Fig. 5. Internally Distributed Cyberorg: master facilitator performs a primitive request on a remote node (1: master facilitator sends a primitive request to an involved slave facilitator on Node B; 2: slave facilitator tells CyberOrg Manager the requested primitive; 3: CyberOrg Manager controls Scheduler Manager to make changes on resource allocation)

invoked. Afterwards, "create partial cyberorg" requests are sent to other involved nodes, where parts of the cyberorg with slave facilitators are created asynchronously. When the creation is completed on the slave nodes, "creation done" messages are sent to the master facilitator, completing the creation when all replies have been received.

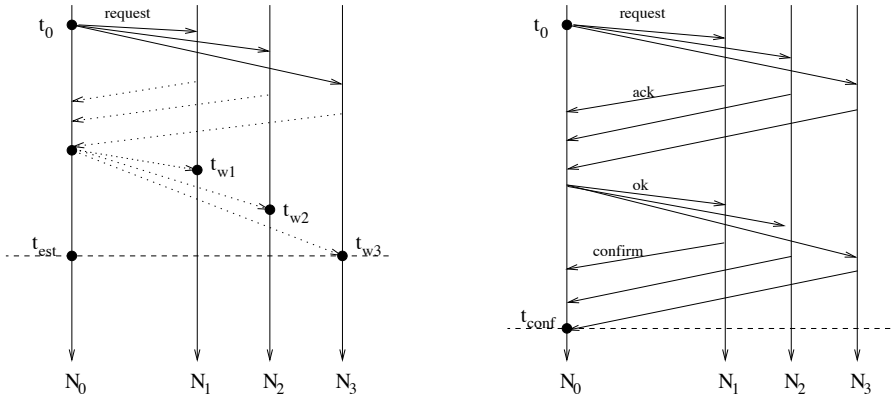
A distributed primitive operation invoked by an internally distributed cyberorg is implemented through coordination between master and slave facilitators. The master facilitator is responsible for analyzing the primitive request, identifying the nodes involved, and sending instructions to relevant slave facilitators to carry out local actions. On completion of their actions, the slave facilitators send reply messages to the master facilitator, indicating success. When the last reply message reaches the master facilitator, the distributed primitive operation is completed.

5 Experimental Results

A number of experiments were carried out to assess the effectiveness and scalability of this approach. We collected results on delays in completing distributed schedule update tasks involving up to 1500 agents distributed over networks of two and three processors, representing systems with 10^4 or more total number of agents. Specifically, we compared the delay in achieving group agreement on feasibility of success or failure of global updates to distributed processor schedules, when using and not using our approach of exploiting predictability of resource availability in cyberorgs.

We applied the approach to an implementation of CyberOrgs. In the first set of experiments – in the absence of resource availability information – we used the pessimistic approach of requiring a series of acknowledgments confirming that the requested updates can indeed be carried out at the required time. In the second set, we relied on knowledge of available resources to (optimistically) assume that the requests have been satisfied unless a failure message is received by a deadline. The two alternatives are depicted in Figure 6. Note that this is not a fair comparison because in the resource

unaware case, there is no guarantee of success of coordinated action until after the distributed actions have actually been attempted; nor is there a determination of failure, in which case a backtrack is required wherever the actions did happen to succeed. However, short of indicating that no comparison is possible, this appears to be a reasonable compromise.



a) Resource−aware approach

b) Resource−unaware approach

Fig. 6. Resource-aware approach vs. resource-unaware approach. (a) Dotted lines represent possible infeasibility reports only received when some slave finds coordination action to be infeasible; otherwise, no communication is required. t_{est} is the time by which the master as well as all slaves can assume global agreement on coordinated action. (b) t_{conf} is the time by which master facilitator knows that all slaves received knowledge of global agreement in time to attempt coordinated action.

The distributed task in our experiments involved coordinated update of the distributed schedule being enforced for delivering processor resources to up to 1500 agents of a cyberorg distributed across three physical processors. Each processor hosted up to 500 agents, scheduled by a local scheduler. Global update requests were received by the cyberorg's master facilitator. We carried out experiments to see the delay between when a request is received by the master facilitator, and when all parties are ready for coordinated action to be carried.

In the resource-unaware approach, the master facilitator sends requests for local updates to the remote (slave) facilitators, which report to the master about likely success or failure based on information of local resource availability. Note that without access to this information about local resource availability and ability to assess feasibility of local action given such constraints, it would be meaningless to plan on coordination action short of actually attempting the action; therefore, we chose to allow the competing approach with this knowledge. Another alternative would have been to allow the slave facilitators to construct an updated schedule and then report their ability or in-

ability to replace the active schedule at the requested time. In either case, if the master received positive reports from all slaves in good time, it could then instruct each slave to go ahead and carry out the actions. However, without knowledge of available network bandwidth, there is no way of ensuring that all slaves receive instructions to proceed with enough time remaining before the deadline to successfully carry them out. The last step, therefore, has to be each slave reporting back to the master, and the master sending instructions for backtracking in case the coordinated action has failed. We treat the point when the master knows of success or failure (not after backtracks have been completed) as the point when all parties are ready proceed. To summarize, despite significant communication, there is no way of predicting success of a coordinated action, short of actually attempting it, even when the distributed parties have the benefit of local resource information.

In comparison, in the resource-aware approach, slave facilitators periodically inform the master about their resource availability with promise of no change before an expiration time.[5] On receiving a new request for coordinated update, the master assesses its feasibility based on information about the updates as well as the slaves involved. Specifically, if some slaves will not have sufficient resources to carry out their parts of the coordinated action at the required time, the master summarily declines the request without ever communicating with the slaves; and if each slave will have sufficient resources to carry out the coordinated action at the required time, the master simply sends the requests, and prepares to carry out its part of the coordinated action at the required time. If, however, the master cannot make a summary determination – because the promises from slaves are expiring sooner than their resources would be required – the master assesses whether the slaves have enough resources to make local assessments of feasibility and report back by expirations of their promises. If so, the master sends the requests, and if no slave reports infeasibility by the time they should be able to (knowing their resources), the master assumes that all requests would be successful. If it does receive an infeasibility report, the action is cancelled. Of course, the slaves now have to be informed ahead of the time of coordinated action whether all slaves are ready to proceed. This too is handled optimistically. In the initial request, slaves are informed by the master about the time by which they would be informed if the coordinated action were not feasible for some of the slaves. The master calculates this time by adding to the time by which it would receive any infeasibility reports from the slaves, the time its own final report would take in arriving at the slaves, which in turn depends on the master's locally known network resource availability. The slaves too, in turn, guarantee that they will have enough processor resources to process the incoming final report from the master at the time when they were instructed to expect the report. If the slaves do not hear from the master by that time, they assume that all are ready for coordinated action, and proceed at the time, without requiring any information. In other words, if the coordinated action can be carried out by all parties at the required time, the only actual communication required is the requests sent by the master to all the slaves, following which, at specific times, each party knows that it is safe to proceed with the coordinated action at the required time. If the resource availability promises held by the master are

[5] It is assumed that the clocks are synchronized within some epsilon, which can be compensated by making conservative estimates

not sufficient to know if the slaves can report infeasibility of their local actions reliably, the master simply waits for the next resource updates from the slaves.

Figure 7 compares the delays described above for the two approaches for coordinated update of schedules for up to 1500 agents distributed across two and three physical nodes. Note that the number of agents effected by an update would typically be a fraction of the total number of agents in the system, meaning that the results apply to systems of at least 10^4 agents. The graph shows that significant savings in the delay for global agreement on coordinated action are achieved using the CyberOrgs based resource-aware approach. These savings are in addition to the savings achieve by avoiding attempting infeasible actions, which cannot be avoided in the resource-unaware approach, even when local resource information is available. Furthermore, the penalty of increasing number of agents linearly with the number of nodes is insignificant.

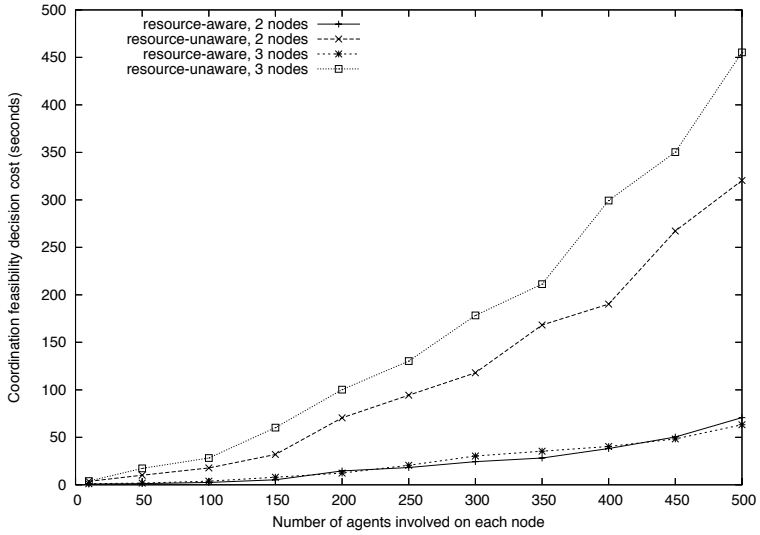


Fig. 7. Comparison of delay in achieving agreement on coordinated distributed schedule update.

6 Conclusion

Coordinating distributed multi-agent systems is a difficult problem because of unpredictability of network and processor resource availability. Our approach of encapsulating computational and communication resources in cyberorgs creates execution environments for distributed multi-agent systems with predictable resource availability. Coordination mechanisms can exploit this predictability by computing expected delays and using the information in decision making.

In this paper, we have presented a prototype implementation of cyberorgs distributed over multiple processors. We have shared our experience with coordinating distributed

scheduling of processor resources, where a number of local schedulers coordinate to enforce a global schedule for scheduling distributed processor resources.

Given the predictability of communication and processing delays in a system of cyberorgs, it is possible for the coordination mechanism to reason about whether or not a global scheduling change is feasible to enforce and to efficiently achieve global agreement on coordinated action. This achieves benefits in avoiding attempts of infeasible global actions. This approach also reduces communication overhead, which reduces the amount of time required in establishing that a coordinated distributed action is feasible, which in turn leads to enabling actions which would otherwise be infeasible. Experimental results show that the approach is effective and scalable.

Work is ongoing to extend these results to physical networks using approaches we have developed for reifying network resource control for systems of cyberorgs [8]. We are also examining the efficiency and effectiveness of this approach to support a wider class of coordination problems.

References

1. G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, Mass., 1986.
2. J.-M. Andreoli, P. Ciancarini, and R. Pareschi. *Research Directions in Concurrent Object-Oriented Programming*, chapter Interaction Abstract Machines, pages 257–280. MIT, 1993.
3. W. J. Bolosky, R. P. Fitzgerald, and J. R. Douceur. Distributed schedule management in the tiger video fileserver. In *Symposium on Operating Systems Principles*, pages 212–223, 1997.
4. A. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufman Publishers, San Mateo, California, 1988.
5. L. Gasser. DAI approaches to coordination. In N. M. Avouris and L. Gasser, editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 31–51. Kluwer Academic, 1992.
6. D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992.
7. C. Hewitt and P. de Jong. Open systems. In J. Mylopoulos, J. W. Schmidt, and M. L. Brodie, editors, *On Conceptual Modeling*, chapter 6, pages 147–164. Springer, 1984.
8. N. Jamali and C. Liu. Reifying control of multi-owned network resources. In *Proc. of the IPDPS Intl Workshop on High-Level Parallel Programming Models and Supportive Environments*, March 2007.
9. N. Jamali and X. Zhao. Hierarchical resource usage coordination for large-scale multi-agent systems. In T. Ishida, L. Gasser, and H. Nakashima, editors, *LNAI: Massively Multi-agent Systems I*, volume 3446, pages 40–54. Springer Verlag, 2005.
10. N. Jamali and X. Zhao. A scalable approach to multi-agent resource acquisition and control. In *Proc. of the Fourth Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS '05)*, pages 868–875, Utrecht, July 2005. ACM.
11. M. Jang and G. Agha. On efficient communication and service agent discovery in multi-agent systems. In *Proc. of the International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS '04)*, pages 27–33, Edinburgh, May 2004.
12. N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.
13. M. Lal and R. Pandey. A scheduling scheme for controlling allocation of CPU resources for mobile programs. *J. AAMAS*, 5(1):7–43, 2002.
14. C. Phillips, C. Stein, and J. Wein. Task scheduling in networks. *SIAM Journal on Discrete Mathematics*, 10(4):573–598, 1997.