An Organizational Self-Design Model for Organizational Change

Young-pa So and **Edmund H. Durfee**
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109
frege@caen.engin.umich.edu, durfee@caen.engin.umich.edu

Abstract

Organizational Self-Design (OSD) is a model of Organizational Change from the perspective of Distributed Artificial Intelligence (DAI). We present a top-down model of OSD in the context of Cooperative Distributed Problem Solving (CDPS). We emphasize the task environment as an important factor in determining an effective organizational structure. We provide an example of an analytical model of organization performance for an addition task using tree structure organizations, and show how such model can be used to endow a Distributed Network Mangement system with the capability to adapt to changing task environments via OSD.

Introduction

Our research is directed toward finding out the factors that affect the collective performance of a set of communicating autonomous agents engaged in cooperative problem solving, and our ultimate goal is to develop a scientific understanding of the cooperative process that can be embodied in computational mechanisms. Furthermore, because cooperating agents typically work in changing environments, it is crucial that the coordination mechanisms they employ are capable of adapting to changing circumstances. In particular, the structural aspect of a cooperative collection of agents—the agents' organization—must change over time as circumstances necessitate. In the field of Distributed Artificial Intelligence (DAI), such adaptive reorganization, when performed by members of the organization, has been called Organizational Self-Design (OSD) [Corkill, 1983, Corkill and Lesser, 1983, Gasser and Ishida, 1991].

Organizational Self-Design

An organization capable of OSD should have one or more members that can perform the following tasks:

1. *Monitor*: Monitor the organizational structure's effectiveness in directing organizational activities (including the OSD activities). A set of observable parameters that affect the performance of the organization, as well as the formula for computing the performance itself, must be defined. Also, the conditions under which reorganization will be considered must be defined which will typically involve a performance threshold.
2. *Design*: Identify new organizational structures appropriate to a new situation. For a design task, a way to generate alternative organizational structures for the current situation or for the projected future situations must be available. For the systematic generation of organizational structures, an organizational structure should be able to be specified using a set of dimensions. Some important dimensions include how the overall task is decomposed into a set of subtasks, how the subtasks are allocated to available agents, determining roles and communication structures among agents, how many agents are involved, and which resources are to be used and how they are to be shared by the agents.
3. *Evaluate and Select*: Evaluate possible organizations and select the best one. This involves evaluating each alternative (including the current one) using a performance measure and selecting the one which is estimated to give the best overall performance.
4. *Implement*: Implement (and execute) the new structure over the network while preserving the network's problem solving activities. Implementing the selected structure requires transfer of each task to the allocated agent.

The tasks of OSD can be done in a global, top-down manner [Corkill and Lesser, 1983] or in a local, bottom-up manner [Gasser and Ishida, 1991]. In the former approach, one powerful agent monitors the global performance and the local activities of the members, designs alternative organizations, selects the best, and imposes the restructuring of the organization on other agents. In the latter approach, OSD can occur at a local level where an agent will use what it

This work has been supported, in part, by NSF PYI Award IRI-9158473, and by a grant from Bellcore.

knows locally about the organization's performance in its neighborhood, design alternative activities for itself (load acquisition, shedding, or modification), select an alternative, and adopt it. This local change might in turn affect performance in the neighborhoods of others, causing a cascading of changes such that overall reorganization emerges from local choices.

These alternative views represent extremes of a continuum, and typical OSD will be influenced from both top-down and bottom-up factors. Common to all such alternatives, however, is the need to evaluate alternative reorganization proposals, whether they involve an individual or all individuals, in the context of what is known of the organization's circumstances. Because the lead individual in an organization that can be designed from the top-down would be expected to have the richest possible model of the organization's overall circumstances, the investigation described in this paper concentrates on the top-down model of OSD decision making.

An essential problem of OSD is to provide a general enough model of a task, an organization, and its performance. Thus, given a task, we would like to be able to generate the possible organizations to solve the problem, and evaluate each organization with the performance model. Since there may be different types of tasks and since the possible organization types as well as the performance of the organization will depend on the type of the task, we would like to classify the different organization structures in terms of the types of tasks each is well suited for.

Once we have such a predictive model of Task-Organization-Performance, the rest of the OSD tasks will be much easier. In sum, we see OSD as essentially involving a generate-and-test process.

Our Approach to OSD

Since dealing with all combinations of task types and organization types is exponentially hard, we initially concentrate on a few interesting task types and organization types. By parameterizing the model of the tasks and organizations on few variables, we hope to be able to enumerate the different possible tasks and organizations. And with the aid of objective performance measures which can apply to the different organizations for a given task, we hope to be able to compare and evaluate the performance of the various task-organization pairs.

Task Environment Model

The task we have initially considered is the addition of N numbers. The task of adding a set of numbers can be divided into several disjoint subtasks of adding a subset of numbers. The results of each subtask must be combined and the result must again be combined with other results until the final solution is synthesized. Thus, this task is representative of many tasks considered in CDPS which require decomposing a single task into subtasks and combining each result of the subtask into larger results and eventually into a single final result. For example, distributed interpretation tasks (e.g. DVMT [Durfee *et al.*, 1987]) involve combining local hypotheses to ultimately generate a global solution, and usually uses tree-like hierarchical organizations consisting of multiple layers of problem solving nodes, especially if each combination of results involves aggregation and/or abstraction of information.

However, a difference between the distributed interpretation task and the addition task is that, in the former, a node may have multiple possible interpretation tasks to choose from at a given time, and coordination between nodes in terms of selection of tasks is required in order to avoid redundant work and/or to construct a globally coherent interpretation.

The model of task environment we consider consists of agents who can communicate with one another and are able to do certain kinds of tasks. The unit task execution time and the unit message transmission time are considered as important environmental parameters that affect the performance of organizations. In particular, we determine the environmental conditions under which cooperation is effective for each task-organization pair.

Organization Model

Elements of an Organization. We think that the model of an organization is tightly related to the model of the task the organization is used for. A cooperative organization for a particular task consists of at least the following elements:

1. The set of tasks and subtasks to be done.
2. The set of agents participating in the organization.
3. An assignment of the tasks and subtasks to the participating agents.
4. A work flow structure which dictates the process by which the tasks and subtasks are to be distributed to the assigned agents and how partial results are to be synthesized.
5. Optionally, a set of resources aside from the agents, and a set of constraints on the usage of those resources, may apply to agents.

Organizational Structure as a Distributed Search Control Strategy. In the context of Cooperative Distributed Problem Solving (CDPS), an organizational structure can be seen as a way of specifying the domain-level coordination strategy. In particular, when a set of agents are to perform a distributed search task, the organizational structure specifies the decomposition of the search space among the agents and the way the overall search should be coordinated among the agents in terms of which agent should communicate what information to which agent when [Lesser, 1991].

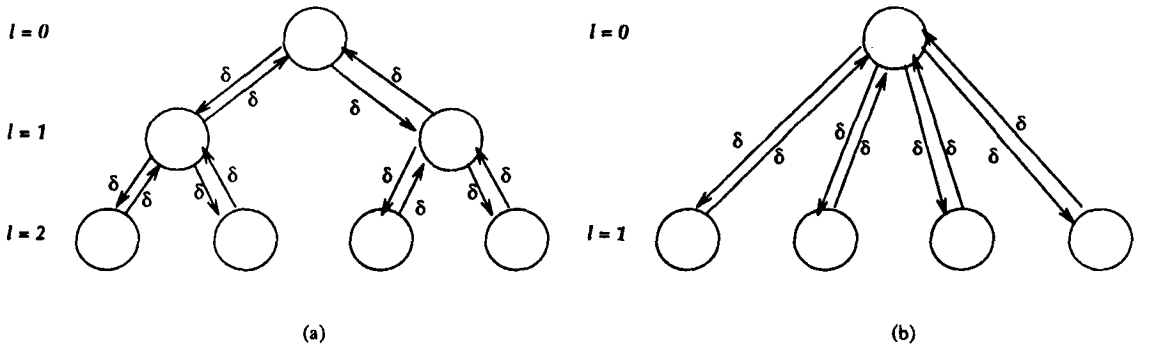

Figure 1: (a) 2-level binary tree. (b) one-level 4-ary tree

An analogy with a single-agent search process can be made for the distributed search case. The so-called *weak methods* of search can be seen as a set of domain-independent methods to determine the order of search. A corresponding set of *strong methods* can be defined as a set of domain-dependent methods which use domain-knowledge and heuristics appropriate for the domain to control the search process. Examples of strong methods include many heuristic- or knowledge-based control methods such as *minimax* and *alpha-beta pruning* methods for game trees, and *meta-rules* for rule-based systems.

An organizational structure in the context of CDPS can be seen as specifying the control method of the distributed search process [Corkill and Lesser, 1983], and therefore, we might be able to find the organizational structures corresponding to the weak and strong methods of distributed search. However, unlike single-agent search control which only deals with the problem of *when* to search *what* state, the multi-agent distributed search control must also address the problem of *who* will search what states when, and the possible coordination of parallel search processes in terms of what domain and control information will be communicated among which agents at what time.

Solving such a coordination problem can be characterized as a *search through a Behavior Space* where a point in the Behavior Space is specified by the 6-tuple of (who, what, when, where, how, why) [Durfee and Montgomery, 1991]. If a single agent is responsible for controlling the search behavior of all agents, it becomes a centralized coordination problem where the control agent searches the behavior space to find and allocate behaviors to the agents. On the other hand, when multiple agents participate, it becomes a distributed coordination problem which requires a distributed search of the behavior space by the multiple agents.

From the behavior space point of view, an organizational structure is an embodiment of long-term coordination knowledge which agents share, commit to, and follow. Therefore, OSD can be characterized as a search through the behavior space for a feasible or a best allocation of long-term behaviors among the agents.

As in single-agent search, we expect that certain (distributed) search control strategies will be better suited to certain kinds of search spaces. Thus, the characteristics of the search space (e.g. solution density, branching factor, etc.) will be a major factor in determining the appropriate organizational structure for the task.

Performance Model

In this section, we develop a performance model for the addition task and two kinds of tree organizations. Here, we use response time as the only performance measure. We are currently expanding our models to encompass other quantitative performance measures (e.g. throughput, system utilization, reliability, availability) as well as qualitative performance measures in the context of CDPS [Decker *et al.*, 1989].

We denote the problem size of the task by N. We assume that each agent or node is capable of storing and retrieving numbers and also capable of standard Arithmetic and Logical operations available in many computers. More specifically, we assume that addition is performed by using an accumulator to which the first number must be loaded. We assume that adding N numbers take $N\tau$ time units. In other words, a single addition operation takes 2τ time units. It is also assumed that the agents are capable of receiving and sending messages to any other agent. For simplification, we assume that any message to be sent can fit into a packet of fixed size, and that the communication delay between any two agents takes constant δ time units.

The two organization types we consider are a l-level binary tree and a one-level k-ary tree. The *height* of a rooted tree is the length of the longest path from the root to a leaf node. The *level* or *depth* of a node in a rooted tree is the length of the unique path from the root to that node. The *degree* of a node is the number of subnodes incident to it. Thus a l-level binary tree is

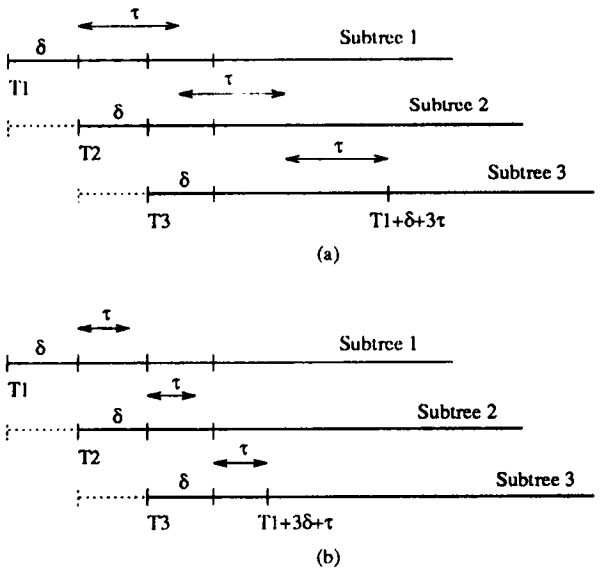

Figure 2: (a) $\delta \leq \tau$. (b) $\delta > \tau$.

a binary tree of height l, and a one-level k-ary tree is a tree of height one with the degree of the root equal to k. See Figure 1.

The performance measure of a given organization for a given task will be the time taken to complete the task. First, we note that the task of adding N numbers can be done by a single node. To have multiple agents cooperate on the task we use the following cooperation scheme. For l-level binary tree we assume that the problem size is 2^{l+1}. For one-level k-ary tree we assume that the problem size is $2k$. For task distribution and result synthesis, we assume that initially there is a single node that holds the entire task of adding N numbers. We call that node the root node.

We denote the time taken to complete the task of adding N numbers by a l-level k-ary tree by $T_k^l(N)$. When $N = k^{l+1}$, we drop the superscript and abbreviate $T_k^l(N)$ by $T_k(N)$. $T_0^0(N)$ denotes the performance of a single node on a task of size N, which is $N\tau$.

Definition 1 *A* complete k-ary tree *is a tree of which every internal node has degree k where an internal node is a non-leaf node.*

Definition 2 *A* balanced complete k-ary tree *is a tree of which all internal nodes of the same level have degree k.*

Definition 3 *A* general balanced complete tree *is a tree of which all nodes of the same level have the same degree.*

Note that, in a general balanced complete tree, the degree of internal nodes for different levels may differ. If we call a tree rooted on one of the nodes at level l a level-l subtree, we can see that for each level l in a general balanced complete tree, all level-l subtrees are of the same structure. The following lemma, which allows us to compute the performance (i.e. response time) of a general balanced complete tree organization for the addition task, allows us to derive performance equations for special kinds of general balanced complete tree organizations that are of interest to us; that is, the l-level binary tree and the one-level k-ary tree organizations.

Lemma 1 *Let D_l denote the time taken for a level-l subtree of a general balanced complete tree to complete its task. Let the degree of the root of that tree be k. Then, for $l \geq 0$,*

If $\delta \leq \tau$ then $D_l = D_{l+1} + 2\delta + k\tau$

If $\delta > \tau$ then $D_l = D_{l+1} + (k+1)\delta + \tau$

Proof. Let Δ_{l+1}^i denote the ith subtree of level $l+1$. Since the level-l subtree has degree of k, there will be k subtrees on level-$(l+1)$, i.e., from Δ_{l+1}^1 to Δ_{l+1}^k. Denote the subtask completion time of Δ_{l+1}^i by T_i, assuming that the time at which the level-l node assigns its first subtask to the first subtree is 0. Note that $T_1 = \delta + D_{l+1}$ since it takes δ to assign the task to Δ_{l+1}^1, and it takes D_{l+1} for Δ_{l+1}^1 to complete the assigned task.

Then, since subtasks are assigned sequentially, with each task assignment time δ, and since each subtask takes the same amount of time, $T_{i+1} = T_i + \delta$ for $1 \leq i \leq k-1$.

Figure 2 shows a timing diagram, where each δ after T_i represents the time taken for sending the result of Δ_{l+1}^i to the node in level l, and each τ represents one addition operation at a level l node after receiving the result from Δ_{l+1}^i.

If we define $\Im_k$ as the time at which the level l node receives the result of Δ_{l+1}^k and adds it to the partial sum,

(a) when $\delta \leq \tau$,

$$\Im_k = T_1 + \delta + k\tau \quad (1)$$

(b) when $\delta > \tau$,

$$\Im_k = T_1 + k\delta + \tau \quad (2)$$

Since $\Im_k$ represents the time taken to complete the task at level l by assigning k subtasks to k level-$(l+1)$ nodes (or subtrees), and receiving results from those nodes and adding up k numbers, we can see that $D_l = \Im_k$. But since $T_1 = \delta + D_{l+1}$, we get the following by substituting it into the above equations for $\Im_k$.

(a) If $\delta \leq \tau$ then,

$$\begin{aligned} D_l &= \Im_k \\ &= T_1 + \delta + k\tau \\ &= \delta + D_{l+1} + \delta + k\tau \\ &= D_{l+1} + 2\delta + k\tau \end{aligned}$$

(b) If $\delta > \tau$ then,

$$\begin{aligned} D_l &= \Im_k \\ &= T_1 + k\delta + \tau \\ &= \delta + D_{l+1} + k\delta + \tau \\ &= D_{l+1} + (k+1)\delta + \tau \end{aligned}$$

Task Size 2^{l+1}, l-Level Binary Tree Organization For simplification, we assume that $N = 2^{l+1}$ where l is the level or height of the binary tree. That is, we assume that each node in the binary tree adds two numbers. We assume that the root node divides the task of adding N numbers into two subtasks of adding $N/2$ numbers. The two subtasks are assigned to two other nodes sequentially. In this way, each subtask is divided in half and assigned to the next level down until the size of the subtask is 2. Since we assume the problem size is 2^{l+1}, a binary tree of l-levels will be sufficient and necessary. After the leaf nodes add the two numbers, the sum is propagated up to the node that assigned the subtasks. The node that receives the two numbers from one level down adds them and again propagates the partial sum up one level, and so on. When the root node finishes adding up the last two numbers the task is completed. Also, when subtasks are assigned down the tree, we ignore the time taken in each node to divide the received subtask into two equal subtasks since it can be considered as taking constant time and thus counted as a part of the communication delay.[1]

The binary tree is a special case when the degree of all internal nodes is 2. In such a case, using Lemma 1, $D_l = D_{l+1} + 2\delta + 2\tau$ when $\delta \leq \tau$. Since such a finite difference relation holds for each pair of levels, we get $D_0 = D_l + l \times (2\delta + 2\tau)$. Thus, for a l-level binary tree adding 2^{l+1} numbers according to our scheme, $D_l = 2\tau$ since all lth level nodes (i.e. leaf nodes) are assumed to add two numbers. Thus, the time taken to add $N = 2^{l+1}$ numbers using an l-level binary tree is:

If $\delta \leq \tau$ then,

$$\begin{aligned} T_2(N) &= D_0 \\ &= D_l + l \times (2\delta + 2\tau) \\ &= 2\tau + 2l(\delta + \tau) \\ &= 2l\delta + 2(l+1)\tau \\ &= 2\delta(log_2 N - 1) + 2\tau log_2 N \end{aligned}$$

If $\delta = \tau$ then,

$$\begin{aligned} T_2(N) &= 2\delta(log_2 N - 1) + 2\tau log_2 N \\ &= 2\tau(log_2 N - 1) + 2\tau log_2 N \\ &= 2\tau(2log_2 N - 1) \end{aligned}$$

If $\delta > \tau$ then,

$$\begin{aligned} T_2(N) &= D_0 \\ &= D_l + l \times ((2+1)\delta + \tau) \\ &= 2\tau + l(3\delta + \tau) \\ &= 3l\delta + (l+2)\tau \\ &= 3\delta(log_2 N - 1) + \tau(log_2 N + 1) \end{aligned}$$

[1] This assumption cannot hold in DAI domains where task decomposition itself is a non-trivial task that can consume unpredictably large amounts of time.

We can determine the conditions under which the binary tree structured organization is effective for the task by comparing its performance with single node performance. More specifically, we want to know the conditions under which $T_2(N) < T_0^0(N)$. Knowing that $T_0^0(N) = N\tau$ we can derive such conditions. Our results are shown in Table 1.

Task Size $2k$, One-Level k-ary Tree Organization In this case, there is only one-level of k nodes to which the task addition of $2k$ numbers is distributed. Each of the k nodes will add 2 numbers and return the result to the root node.

We model this organization by using Lemma 1 with $T_k^1(N) = D_0$, $D_1 = 2\tau$, and $N = 2k$.

If $\delta \leq \tau$ then,

$$\begin{aligned} T_k^1(N) &= D_0 \\ &= D_1 + 2\delta + k\tau \\ &= 2\tau + 2\delta + (N/2)\tau \\ &= 2\delta + \frac{N+4}{2}\tau \end{aligned}$$

If $\delta = \tau$ then,

$$\begin{aligned} T_k^1(N) &= 2\delta + \frac{N+4}{2}\tau \\ &= 2\tau + \frac{N+4}{2}\tau \\ &= \tau(4 + \frac{N}{2}) \end{aligned}$$

If $\delta > \tau$ then,

$$\begin{aligned} T_k^1(N) &= D_0 \\ &= D_1 + (k+1)\delta + \tau \\ &= 2\tau + ((N/2)+1)\delta + \tau \\ &= \frac{N+2}{2}\delta + 3\tau \end{aligned}$$

From the above equations, we can derive the conditions under which cooperation via a single level k-ary tree organization is effective in a similar way to what we did for binary tree organization. That is, by comparing $T_k^1(N)$ with $T_0^0(N)$.

Table 1 summarizes our results, using the following definitions.

Definition 4 *Let $\gamma = \frac{\tau}{\delta}$ where τ is the unit task execution time, and δ is the unit message transmission time. We call γ the* task environment granularity.

Definition 5 *A task environment is of* Coarse Granularity *when $\gamma > 1$ (i.e. $\delta < \tau$). A task environment is of* Medium Granularity *when $\gamma = 1$ (i.e. $\delta = \tau$). A task environment is of* Fine Granularity *when $\gamma < 1$ (i.e. $\delta > \tau$).*

Organization ($\gamma = \tau/\delta$)	l-level binary tree ($N = 2^{l+1}$)	one-level k-ary tree ($N = 2k$)
Coarse Granularity ($\gamma > 1$)	$T_2(N) = 2\delta(log_2 N - 1) + 2\tau log_2 N$ $T_2(N) < T_0^0(N) \Longleftrightarrow N > 12$ or $4 < N \leq 12$ and $\gamma > \frac{2(log_2 N - 1)}{N - 2log_2 N}$	$T_k^1(N) = 2\delta + \frac{N+4}{2}\tau$ $T_k^1(N) < T_0^0(N) \Longleftrightarrow N \geq 8$ or $4 < N < 8$ and $\gamma > \frac{4}{N-4}$
Medium Granularity ($\gamma = 1$)	$T_2(N) = 2\tau(2log_2 N - 1)$ $T_2(N) < T_0^0(N) \Longleftrightarrow N > 12$	$T_k^1(N) = \tau(4 + \frac{N}{2})$ $T_k^1(N) < T_0^0(N) \Longleftrightarrow N > 8$
Fine Granularity ($\gamma < 1$)	$T_2(N) = 3\delta(log_2 N - 1) + \tau(log_2 N + 1)$ $T_2(N) < T_0^0(N) \Longleftrightarrow$ $N > 12$ and $\frac{3(log_2 N - 1)}{(N-1) - log_2 N} < \gamma < 1$	$T_k^1(N) = \frac{N+2}{2}\delta + 3\tau$ $T_k^1(N) < T_0^0(N) \Longleftrightarrow$ $N > 8$ and $\frac{N+2}{2(N-3)} < \gamma < 1$

Table 1: Performance of Tree Organizations for Addition Task.

Interpretation of the Result

We can see that the performance of a given type of organization depends both on the size of the task and on the granularity of the task-organization configuration. For instance, our model shows that adding four numbers using a tree organization gives worse performance than done sequentially at one node. Tree organizations outperform single node performance for increased task size since, in our model, the organization size grows with problem size, but even this tendency is conditional on the speed of the communication links relative to the unit task execution rate of processors. Thus, for fine granularity, binary tree organizations usually outperform single nodes but only if the granularity is above some bound. That is, for fine granularity, if the communication delay is too large relative to the unit task execution time, it may still be better to execute the task at single node.

Our results confirm our intuition that cooperative distributed problem solving using task-organizations are better than centralized problem solving as long as the task is big enough (thus exploiting the benefits of parallelism) and communication is fast enough relative to computation. However, intuitions are limited in providing precise predictive knowledge. By quantifying relationships between various factors affecting the performance of organizations, our model provides predictive knowledge that can be used in organizational self-design.

We can use the model to determine not only the conditions under which cooperation is effective, but also the conditions under which one organization is better than the other. We can think of the l-level binary tree organization as representing a tall-thin hierarchical organization, and the one-level k-ary tree organization as representing a short-fat hierarchical organization. Thus, by determining the conditions under which $T_2(N) < T_k^1(N)$, we can gain intuition on the conditions under which tall thin hierarchical organizations outperform short-fat hierarchical organizations.

We find that, for Coarse Granularity task environments,

$$T_2(N) < T_k^1(N) \Leftrightarrow (N < 4) \vee (N \geq 26) \vee [(8 < N < 26) \wedge (\gamma > \frac{log_2 N - 2}{N/4 - log_2 N + 1})]$$

$$T_2(N) > T_k^1(N) \Leftrightarrow (4 < N \leq 8) \vee [(8 < N < 26) \wedge (1 < \gamma < \frac{log_2 N - 2}{N/4 - log_2 N + 1})]$$

For Medium Granularity task environment,

$$T_2(N) < T_k^1(N) \Leftrightarrow (N < 4) \vee (N \geq 26)$$

$$T_2(N) > T_k^1(N) \Leftrightarrow (4 < N < 26)$$

For Fine Granularity task environment,

$$T_2(N) < T_k^1(N) \Leftrightarrow (N < 4) \vee (N \geq 26) \vee [(16 < N < 26) \wedge (\gamma < \frac{N/2 - 3log_2 N + 4}{log_2 N - 2})]$$

$$T_2(N) > T_k^1(N) \Leftrightarrow (4 < N \leq 16) \vee [(16 < N < 26) \wedge (\frac{N/2 - 3log_2 N + 4}{log_2 N - 2} < \gamma < 1)]$$

Thus, in general, tall-thin hierarchies outperform short-fat hierarchies when problem size is sufficiently large (i.e. above some bound). However, for some problem sizes (e.g. $N = 24$, $l \cong 5$, or $k = 12$), short-fat hierarchies may outperform tall-thin ones if the granularity of the task environment is neither too large nor too small.

Evaluation of the Model

The assumptions made in the model were:

1. Uniform processing rate for all nodes.
2. Uniform transmission rate for all links.
3. Uniform packet size for all messages.
4. Number of nodes grows as problem size increases.
5. Task decomposition takes negligible constant time.

The distinguishing feature of this model compared to other models of distributed hierarchical problem solving such as in [Montgomery and Durfee, 1992] is that this model takes into account the effect of task assignment overhead. Although parallel asynchronous communication is not uncommon, in many applications synchronous communication such as TCP/IP is common, and when we think of human organizations,

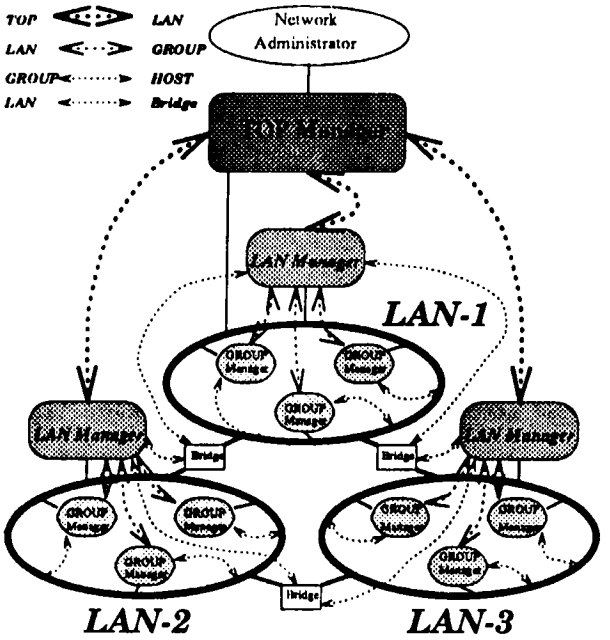

Figure 3: Overview of DBB Environment.

due to the biological limitation of a single agent, task assignment to other agents is often a sequential process. However, the assumption here that the task assigner has to wait for an acknowledgement before s/he can start assigning the next task to another individual may be unrealistic since people often do continue doing other things after sending off a task but before it arrives at the destination.

Although we used addition as our problem, the model can apply equally well to any divisible task of size N, where N is the number of subtasks each of which takes equal time to perform it.

Application to Intelligent Distributed Network Management

Our previous work on Distributed Big Brother (DBB) [So and Durfee, 1992a, So and Durfee, 1992b] focused on integrating various techniques of DAI to implement an organizational structure which serves as an infrastructure for intelligent distributed computer network management. DBB had a limited capacity for OSD in the sense that it can recover from failure of one or more nodes in the structure and maintain the organizational structure in a way that preserves the overall functionality of the organization.

One simple task of DBB was to have management agents gather and abstract information about the hosts in the network. The response time performance (information recency) depends on the size of each task (N_i - the number of hosts to poll), the unit task execution time (τ), and the task assignment rates (σ), as well as the transmission delay (δ). Therefore, if the DBB agents are to not only recover from failure but also to adaptively reconfigure themselves as the task and resource environment changes, they must be able to continuously monitor the changes in the environment (e.g. values of N, τ, δ) and determine whether a change in the organizational structure (e.g. number of levels, span of control) can lead to better performance. In order to do that, there must be a way to generate the possible changes in the organization, and a way to evaluate each possible organization given the current set of environmental parameter values or the predicted future values of those parameters.

We have applied the analytical method presented in this paper to model the performance of DBB, and our preliminary results successfully predict the response times of DBB under various task configurations.

Conclusions

Obviously there are many shortcomings to the analytical model of task, organization, and performance presented above. We list some of them in the following.

1. The kind of task considered were those that were decomposable to independent subtasks. That is, the subtasks had no dependency relationships. Since coordination of behaviors of agents are needed largely due to the dependency of one's task to another's, the current model does not cover many interesting coordination types and the corresponding organizational structures. Although the subtasks had to be temporally coordinated in the sense that an agent can only execute a subtask when it receives one from another agent, *during the execution* of subtasks agents need not temporally coordinate their activities. Also, there were no *constraints* as to the product or output of two or more subtasks, or more precisely, such constraints are implicit in the subtask decomposition.
2. The kind of task considered had no uncertainty. Each task to be performed by each agent or node was fully and completely specified such that there can be no ambiguity as to what each agent was supposed to do. In other words, each agent has no choice but to do what it is supposed to do. Thus, agents generally do not make decisions. Alternatively, if we view a "task" as a set of tasks where a subset of them needs to be done, an agent generally will have to decide which subset of the tasks it will work on at a given time (as in the DVMT [Durfee *et al.*, 1987]).
3. There was no environment with which the agents had to interact in order to accomplish their tasks besides the other agents from and to which tasks and results are passed. There was no environment or world as an independent variable affecting the performance of the agents and the organization. This has relation to the type of task considered which in

our case does not depend on any aspects of the environment. That is, no information about the environment is used in the execution of the task. The only input-elements and output-elements of each agent are the task and result. The agents neither independently gather information from the world or from others nor actively change the world. For example, in a package delivery task, agents must sense the location of the packages as well as the location of other agents, and when they move around the spatial area picking up and delivering the packages, they are changing the environment they share. Such tasks, which require agents to interact with a changing environment, are more complex, and need a more sophisticated model of task, organization, and performance.

4. There was no *conflict* among agent activities because there was no resource sharing. That is, subtasks were independent as to the required resources to accomplish the subtasks. Since agents had no alternatives to choose from when executing their tasks, they did not have to consider the possible *inconsistencies* of outputs resulting from their choices.

5. We have only looked at cases where the number of agents can be increased indefinitely as the problem size increases. In realistic settings, there may be certain bounds as to the number of agents that can be employed to organize and carry out the organizational task. In general, we should also look for the best organization for a given task when there are resource bounds.

6. The type of organization considered was rigid and reflects the static characteristic of the task and the environment. When we begin to add complexities, dynamicities, and uncertainties to the task and environment, the corresponding organization structure is expected to be more flexible in order to be efficient. Also, such a task environment is expected to force the agents to be more sophisticated in order to be efficient, possibly requiring them to learn from past experience and/or plan for the future.

7. We only considered one kind of performance measure, namely the response time. Other measures such as reliability and system utilization should be considered and compared. We expect tradeoff relationships between different performance measures for various task-organization configurations.

We are currently developing a more general analytical model of task, organization, and performance. By elucidating the factors that are involved in determining the performance of an organization, we are working toward implementing a computational organization which can automatically reorganize as the environment changes. Such a model can also help in understanding human organizations, and thus may contribute to organization science in general.

References

Daniel D. Corkill and Victor R. Lesser. The use of meta-level control for coordination in a distributed problem solving network. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–756, Karlsruhe, Federal Republic of Germany, August 1983. (Also appeared in *Computer Architectures for Artificial Intelligence Applications*, Benjamin W. Wah and G.-J. Li, editors, IEEE Computer Society Press, pages 507–515, 1986).

Daniel David Corkill. *A Framework for Organizational Self-Design in Distributed Problem Solving Networks*. PhD thesis, University of Massachusetts, February 1983. (Also published as Technical Report 82-33, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, December 1982.).

Keith S. Decker, Edmund H. Durfee, and Victor R. Lesser. Evaluating research in cooperative distributed problem solving. In Michael N. Huhns and Les Gasser, editors, *Distributed Artificial Intelligence*, volume 2 of *Research Notes in Artificial Intelligence*. Pitman, 1989.

Edmund H. Durfee and Thomas A. Montgomery. Coordination as distributed search in a hierarchical behavior space. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), December 1991. (Special Issue on Distributed AI).

Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, C-36(11):1275–1291, November 1987. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 268–284, Morgan Kaufmann, 1988.).

Les Gasser and Toru Ishida. A dynamic organizational architecture for adaptive problem solving. In *Proceedings of the National Conference on Artificial Intelligence*, pages 185–190, July 1991.

Victor R. Lesser. A retrospective view of FA/C distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), December 1991. (Special Issue on Distributed AI).

Thomas A. Montgomery and Edmund H. Durfee. Search reduction in hierarchical distributed problem solving. In *Proceedings of the 1992 Distributed AI Workshop*, February 1992.

Young-pa So and Edmund H. Durfee. Distributed big brother. In *Proceedings of the Eighth IEEE Conference on Artificial Intelligence for Applications*, 1992.

Young-pa So and Edmund H. Durfee. A distributed problem solving infrastructure for computer network management. *International Journal of Intelligent and Cooperative Information Systems*, 1(2), 1992.