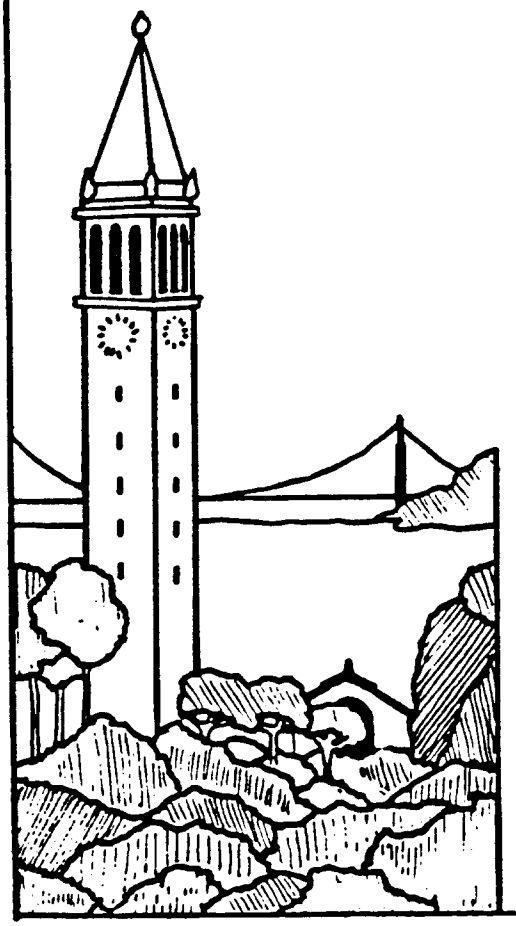
Using Expert Systems to Manage Distributed Computer Systems

Joseph Pasquale



Report No. UCB/CSD 87/334

January 1987
PROGRES Report No. 86.6

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Using Expert Systems to Manage Distributed Computer Systems

Joseph Pasquale

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720

Abstract

Expert systems can be used effectively to manage distributed computer systems which are based on decentralized control of shared resources. These distributed systems can exhibit high reliability and performance. Yet, designing such systems pose formidable problems. These problems involve real-time distributed decision-making where decision-makers do not know with full certainty the state of remote nodes. We identify what characteristics an intelligent decision-making agent must possess to successfully attack these problems, and argue that expert systems share these characteristics. We also present the architecture of an Expert Manager, an experimental system which uses expert system techniques to manage a distributed computer system.

This work was supported by an I.B.M. Fellowship and by the Defense Advanced Research Projects Agency (DoD), monitored by the Naval Electronics Systems Command under contract No. N00039-84-C-0089. The views and conclusions contained in this document are those of the author and should not be interpreted as representing official policies, either expressed or implied, of I.B.M., the Defense Research Projects Agency or of the US Government.

1. Introduction

A distributed computer system (or simply a distributed system) is a set of networked computers which not only communicate with each other, but typically share resources and sometimes work together to solve problems. Because they are comprised of multiple processors which are essentially compartmentalized, these systems can exhibit high performance and reliability relative to a single computer of equivalent combined processing power.

We still have a great deal to learn about how to tap the potential speed and reliability of distributed systems. One promising way is through *decentralized* control of shared resources in the distributed system, where all computers take part in making decisions about how resources should be allocated. Better performance is then realized by exploiting parallelism in decision-making, and multiple loci of control provide redundancy for better reliability.

Of course, decentralized control of resources in distributed systems poses formidable problems. Yet, we feel that these problems can be overcome through the use of expert system techniques. We agree with Stankovic, who states that major breakthroughs to achieve the full potential advantages of distributed computer systems "will necessarily use heuristics similar to those found in 'expert' systems." [1]

To illustrate the types of problems encountered in decentralized control, consider a society of computers where control is fully decentralized. These computers are willing to share resources and work together to solve problems. Specifically, consider what happens if one of the computers in this society becomes overloaded with work, while others are lightly loaded. It would be reasonable to offload some work to the lightly loaded computers. This technique is commonly referred to as *load balancing* or *load leveling*. Since there is no central controller, the overloaded computer must know or determine who is lightly loaded. Possible solutions to this state-distribution problem are: (1) to query other computers on demand; (2) to keep track of lightly loaded computers by periodic querying; (3) for lightly loaded computers to periodically broadcast to others the fact that they are lightly loaded.

The basic problem confronted by a computer in this society is that of not knowing the current state of remote computers, or more generally, *not knowing the current global state of the distributed system*. This same problem is even more pronounced in decentralized dynamic message routing algorithms, where local decisions concerning where to forward a message depend on knowledge of the state of remote nodes.

Each of the sample state-distribution methods presented above provides the overloaded computer with state information about other lightly loaded computers, but with varying degrees of delay, uncertainty and resource usage. The delay is the elapsed time between the point of realizing a decision must be made and the point of actually having adequate information to make that decision. Uncertainty arises in having to make decisions based on information which is not timely and is possibly noisy. Finally, resource usage refers to the cumulative effort and total resources required to communicate the state information. A first step toward intelligent distributed decision-making is to quantify these three parameters. A decision cost function may then be developed and evaluated for each potential state-distribution algorithm. However, this is a sizable task since the parameters are difficult to measure and are usually mutually dependent.

There is another basic problem which is more subtle than simply not knowing the current state of the system. A set of local decisions, which individually optimize global system performance, might be non-optimal when made jointly (i.e., at the same time). This is illustrated in the following example. Consider the society of computers mentioned above, except that many computers (rather than simply one) are now overloaded. Let us assume that they know exactly the instantaneous current load of remote computers, so that the problem of not knowing the global system state mentioned above is not relevant. Since they all know which remote computer is the least loaded, they might all decide to offload work to

the same remote computer, consequently overloading it. Thus, although each decision when considered by itself is well founded, when considered together they produce bad results. Solutions to this problem require some type of a priori cooperation between computers on decisions.

The objective of this paper is to informally describe how expert system techniques can help in addressing some of the problems raised. Section 2 will describe a model of a distributed computer system providing context for the rest of the paper. Section 3 will present why expert systems are relevant for managing distributed systems, section 4 will briefly describe current work in building such an expert system, and section 5 will present conclusions.

2. A Model of a Distributed System

The goal of this paper is to argue in an informal way how expert systems can be useful in solving distributed systems resource management problems. For this purpose, we present a simple, informal and qualitative model of a distributed system. The model implicitly assumes decentralized control and a willingness for cooperation on the part of decision-making entities. This allows us to focus on how to optimize performance purely through intelligent decision-making and coordination.

A distributed computer system can be modeled as a network of *managers* and *resources*. A manager receives computational requests from users, and then decides which resources should be used for those computations. A resource is an object needed to support a computation. Resources accept requests from managers to do work and return results when the work is done. Typically, many managers are simultaneously receiving requests for computations from users, and there are many choices of resources which will support any particular computation.

At any point in time, a resource is in one of a finite number of configurations, or *states*. The state of a resource is given by a vector describing many of the resource's characteristics (e.g., up/down, idle/busy, number of pending requests). Generally, the elements of this state vector are measures which imply a cost or penalty for using the resource. For instance, there would be a greater penalty associated with using a resource which is currently very busy than with a resource which is not busy.

We call the set of states of all resources in the distributed system the *global system state*. It would be highly desirable if all managers could know the global system state for all times. Typically though, at any single point in time, a manager is only interested in the current states of a subset of resources, specifically those that are eligible for the computation at hand. This subset of current states is called the *relevant global system state*, and is normally different for each manager.

Our assumptions about the network are that it is connected (i.e., not partitioned), and that there is a non-trivial cost associated with communication (including the communication of state information between resources and managers). These assumptions stress the importance of finding methods of decision-making and coordination which are based not solely on communication, but on doing any possible local computations which decrease the need for communication.

Given this model, the goal shared by all managers is to mutually decide on a set of resources of minimal (or at least low) cost to support a set of requested computations, subject to the following considerations:

1. The states of resources (and therefore their costs) are constantly changing. The minimal time to transmit state information from any resource to any manager is not constrained to be less than the time between state changes.

2. There is an additional non-trivial cost incurred by a manager which is proportional to the time and effort involved in simply making a decision.
3. Many managers may be making simultaneous decisions concerning the same resources.

The crucial element of this model which makes the problem difficult is the need for making decisions in real time which depend on information which is *inherently uncertain* to the decision-maker. Our purpose is then to illustrate that mechanisms offered by expert systems can be very useful to attacking this problem. We will describe what these mechanisms are, and why they are useful.

3. Characteristics of a Distributed Intelligent Resource Manager

Two basic problems in decentralized resource management were presented in section 1. Let us summarize them.

1. A manager generally does not know with complete certainty the global system state. Knowing the global system state is relevant to making good decisions concerning the allocation of resources.
2. Simultaneous decisions concerning the allocation of resources which individually optimize (minimize) some cost function may not mutually be optimal.

Let us consider in a qualitative way the characteristics of mechanisms which are needed to attack these problems. We list five properties these mechanisms should possess:

1. Ability to deal with uncertainty
2. Ability to react to changing situations
3. Capabilities for interpretation and inference
4. Ability to collaborate and coordinate
5. High performance

One will find that expert systems have many of the properties listed above. We now discuss each of these properties, and offer reasons for how expert systems might realize them.

3.1. Ability to Deal with Uncertainty

Given that a resource can change state faster than the time it takes to transmit state information to managers, managers will always be somewhat uncertain about the states of resources. This is a common characteristic of distributed systems management. Therefore, it is important that managers be able to deal with uncertain information as an integral part of their decision-making process. This means that a manager must be able to quantify the degree to which it believes the information it uses in making decisions, and to consider the consequences of its decisions as a function of the uncertainty of its information. Specifically, it must consider the benefits derived from its decisions if the information used is indeed true, and the penalties incurred if the information is false.

Expert systems have provided the context for much research in systems which must deal with uncertain information. In fact, some of the earliest expert systems such as MYCIN [2] made significant contributions in the use and expression of uncertainty. Although there is still much to learn about dealing with uncertain information, there are many methods of quantifying uncertainty, and of calculi for quantifying the combination of multiple sources of uncertain information. Among these are Bayesian posterior probabilities [3], certainty factors [4], fuzzy logic [5], the Dempster-Schafer theory of belief functions [6], and multi-valued logics [7]. Thus, expert systems research offers many mechanisms for expressing

and working with uncertain information.

3.2. Ability to React to Changing Situations

Given the dynamic nature of the distributed system model described, managers must be able to constantly monitor the states of resources and react based on this changing information. This form of control is commonly referred to as *data-driven*. Mechanisms are needed to monitor resources, communicate monitored information, and then react based on received information. Further, it is advantageous to find ways of inferring the changing state of resources, rather than simply communicating it. Clearly, there will be a need for both inference and communication.

Data-driven, or forward-chaining, control is common to many expert systems, which are most often programmed using rule-based languages. A rule is simply a situation-action pair; given the existence of the expressed situation, do the action. A program simply is a set of declarative rules with no explicit expression of control. Control is embodied in the expert system's inference engine, and can be either forward-chaining (data-driven) or backward-chaining (goal-driven), or a mixture. In the context of trying to determine the relevant global system state which is essentially a classification problem, we are most interested in forward-chaining control. This is a simple form of control (relative to backward-chaining) and can be programmed very efficiently. An efficient expert system language based on forward-chaining control is OPS5 [8].

Although data-driven control provides the right mechanism for making a manager reactive to changing situations, we do not mean to imply that goal-driven control is not desirable in the context of managing distributed systems. Consider cases where a manager has many different policies for different situations; goal-driven operation can be extremely useful. For example, one might consider different message routing policies for different network conditions [9]. To illustrate, a static routing policy which works fine for normal network conditions would be useless if part of the network were to be suddenly destroyed. In this situation, new routes would have to be found quickly, so a routing policy such as flooding would be more appropriate. A set of distributed expert systems, upon recognition of the new situation, would set out to find new routes as their new goal. Note that finding new routes is an instance of the general decentralized control of resources problem we have been considering all along.

3.3. Interpretive and Inferential Capabilities

It is not enough for a manager to simply react based on individual pieces of information. A manager needs the capability for combining many pieces of low-level information to form higher-level concepts, and react based on these concepts. This involves mechanisms for interpreting low-level measurement data, taking into account its uncertainty, and incorporating the interpreted data into a manager's knowledge database. These same mechanisms can also be used to infer higher-level knowledge from the current available knowledge. For example, a manager might infer that a resource is overloaded upon receiving low-level measurement data indicating the current number of pending requests. To infer this, the manager would have to combine this measurement data with other information indicating the resource's average capacity for processing requests. One of the main motivations behind using expert systems is in the power derived from combining of domain knowledge and new information to make inferences.

Rule-based languages provide natural support for interpreting information and for expressing inferences. In fact, a rule which expresses "IF it is believed (with at least certainty C) that situation S exists, THEN infer S' with certainty C'," is exactly an inference. As chains of inferences are built, high-level concepts are being derived from low-level ones.

3.4. Ability to Collaborate and Coordinate

To attack the problem of managers making decisions which mutually conflict, managers must be able to collaborate with each other and coordinate their decisions. This implies mechanisms for communication and a common language to describe current knowledge, goals and plans. To constrict the problem space to a manageable level, and to avoid large amounts of communication, it is reasonable that managers make long-term decisions on goals and policies, and agree on a number of plans or strategies for accomplishing these goals and for implementing the policies. Managers could also exchange abstract models of their decision-making processes to be used to predict each others decisions [10] [11]. By abstract models, we mean that exchanged models are simplified and retain only major distinctions between possible decisions in the way they conflict. This is to restrict the number of possible decisions to take into account.

An important special case arises where all managers share the same decision-making algorithm, as would be reasonable in decentralized resource control in a real distributed system. Specifically, for a given knowledge base, input (request for computations), and location in the distributed system, this algorithm produces a unique decision concerning the allocation of resources. Now, managers would not need explicit models of each other to predict each others actions. They need only consider what they themselves would do if they were at the other managers' locations. The problem is now reduced to a simpler, yet still difficult, problem of knowing what is in each others knowledge bases and inputs. Although it is difficult to predict inputs since these are determined by the actions of users, knowing what is in each others knowledge bases becomes a problem of arriving at a mutually consistent view of the global system state.

Research in team decision theory has contributed many ideas in the areas of multiple agent cooperation [12] [13], as has the field of distributed artificial intelligence (DAI). An early A.I. experiment in cooperative problem-solving system which emphasized dealing with distribution-caused uncertainty was done by Lesser and Erman in building a distributed version of the Hearsay-II speech understanding system [14].

3.5. High Performance

Since managers must make time-critical decisions, performance is an important issue. This requirement precludes using current algorithms which use linear programming techniques to optimally schedule a set of resources for a set of tasks. These algorithms involve gathering all information at a central processor, computing the optimal allocation, and then sending out instructions to carry out the work. Unfortunately, this method usually takes too long, and further uses centralized control which we have ruled out for reasons given in section 1.

Our goal is to develop mechanisms for making the best decisions in a restricted amount of time and in a decentralized manner. As already mentioned, to make good decisions a manager must have an accurate view of the global system state, hopefully a view consistent with other managers. The problem then is to converge on a correct view and then to make a decision, as quickly as possible, on how to allocate resources. We concentrate on how to quickly converge on a correct view of the global system state.

To achieve this goal, we propose the *hypothesize-and-test* problem-solving paradigm. Upon a manager's reception of new measurement data from remote resources, multiple hypotheses are generated which to varying degrees of certainty explain the data. One might see this as extending paths in the search space of possible global system states. Since performance is crucial to the success of a manager's decisions, the manager takes an active part in pruning this extended search space by testing hypotheses through experiments. For example, to test the hypothesis that a resource is not functional, a simple

experiment would be to send a request which causes the resource to reply promptly. This is commonly referred to as *probing*. If a reply is returned, the hypothesis can quickly be rejected.

Thus, high performance is gained by extending the search space using a highly efficient form of control, namely forward-chaining, and by taking an active part in pruning the search space of global system states. Note that we expect managers to decide on policies, and rely on services provided by an operating system, for instance, to carry out these policies (i.e., implement the mechanism). Consequently, this consideration of high performance is in the context of policy decision-making.

4. Architecture of an Expert Manager

We now describe an architecture for an *Expert Manager*. In our implementation, every decision-making node in the distributed system has an instance of an Expert Manager. At this early point in our research, we are concentrating most on resolving the problem of inferring the relevant global system state given partial and uncertain state information. Consequently, the goal of this architecture is to provide a framework for determining the global system state based on partial information. It also provides for active feedback, allowing actions to take place based on the perceived global system state.

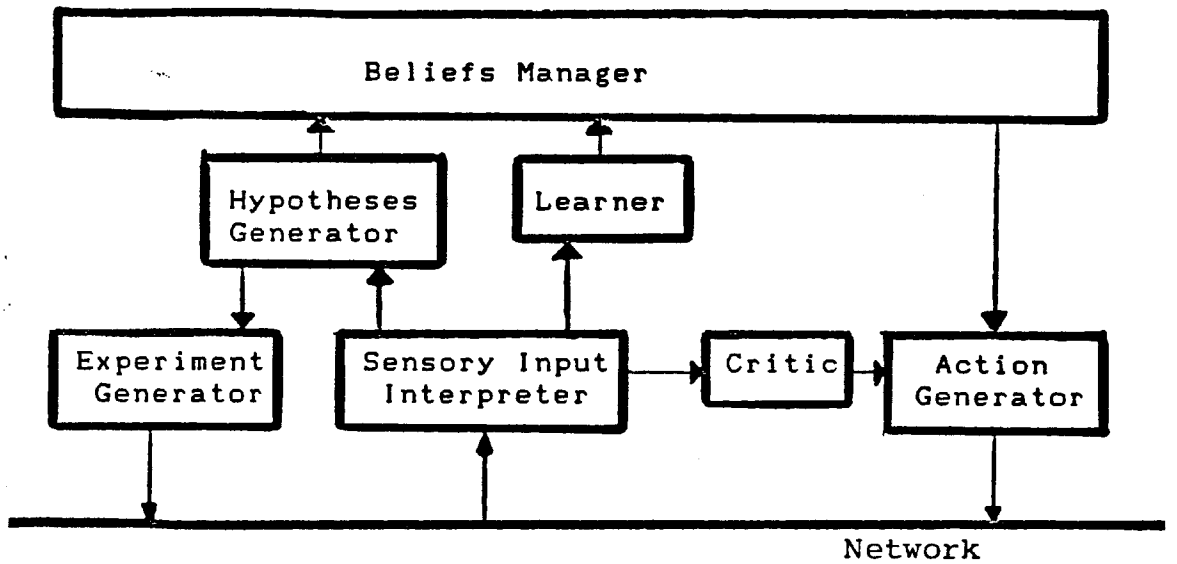


Figure 1

Figure 1 shows the modules making up an Expert Manager. The basic flow of information occurs as follows. There are four basic modules: Sensory Input Interpreter, Hypotheses Generator, Beliefs Manager, and Action Generator. Other modules will be discussed later.

Monitored state information is received by the Sensory Input Interpreter, which tries to makes sense of this uncertain raw sensory data. This may cause a number of hypotheses to be generated by the Hypotheses Generator, which try to explain the received information. If there is substantial evidence to support a hypothesis, it becomes a belief (or fact as far as the Expert Manager is concerned), to be managed by the Beliefs Manager. The Action Generator will take action if there are beliefs indicating the system is in a bad or undesirable state and it is known how to get back to a more desirable state.

There are three auxiliary modules: the Experiment Generator, Learner, and Critic. The Experiment Generator will start an experiment if there are competing hypotheses which require further evidence. The Learner allows for rudimentary learning by observing past trends. Finally, the Critic notes the actions of the Action Generator, monitors the effects, and then makes determinations as to whether the action was beneficial or not.

We now describe in more detail the function of each module.

4.1. The Sensory Input Interpreter

Data received over the network from remote sources is first read by the *Sensory Input Interpreter* (SII). The data is a set of measurements made at a resource which make up part or all of the relevant aspects of the resource's state. For example, for a resource such as a compute server, these measurements might include lengths of CPU process and I/O queues, and amount of free primary memory. For each measurement, two statistics are maintained: an average and a measure of variability, both weighted toward recent measurements. Finally, all measurements are time-stamped as to when the measurement took place.

The SII's primary task is to make a judgement as to how believable this information is by giving it an initial *belief factor* (BF), a number between 0 and 1 similar to a probability. This initial BF is a function of three variables: (1) the difference between the measurement and its predicted value, which is based on an extrapolation from the past history of measurements; (2) how old the measurement is; (3) the potential variability of the measurement.

A BF of a measurement will then decay in time, and the rate of decay is proportional to the variability of the measurement. We may think of the BF of a measurement as an indication of how useful that measurement is. The older a measurement is, the less useful it is. Moreover, the usefulness of a measurement which has a history of high variability degrades more rapidly than a measurement which has a history of low variability.

In summary, the Sensory Input Interpreter maintains a database of remote measurements, and assigns a Belief Factor BF to each measurement which changes in time based on age and variability.

4.2. The Hypotheses Generator

The *Hypotheses Generator* (HG) continually monitors the SII database and generates hypotheses which try to explain the measurements received (or why no measurements have been received for a period of time). These hypotheses are programmed using a rule-based language. The antecedent of a rule expresses what conditions need to be present for the rule to "fire." This would be a list of ranges for measurements, along with ranges for BFs, which must match the SII database. The consequent of a rule expresses either a single hypothesis or a set of competing hypotheses which explain the conditions of the antecedent. Hypotheses are also given BFs which are a measure of how much evidence supports them. These BFs also change in time based on new evidence or lack of evidence.

When a hypothesis's BF falls below a certain threshold, that hypothesis gets removed from the HG's database of current hypotheses. When a hypothesis's BF goes above a certain threshold, that hypothesis becomes eligible to become a belief. These low and high

thresholds are functions of the BFs of competing hypotheses. For example, the low threshold might be .2 plus or minus a quantity which depends on whether there are other competing hypotheses with very high or very low BFs. High thresholds work in a similar way using a base value of .8.

4.3. The Experiment Generator

The *Experiment Generator* (EG) simply tries to speed up the operation of converging on a single hypothesis whenever competing hypotheses exist. It is programmed using a rule-based language, where rule antecedents contain hypotheses, and the consequents contain actions. These actions are processes which get activated to perform experiments. Such experiments will produce evidence to confirm or reject the various competing hypotheses. For example, if there are two competing hypotheses, one stating that a resource is overloaded and another stating that the resource is down, an experiment might simply involve sending the resource a high-priority message asking for immediate response. If an response is received, the second hypothesis can easily be rejected. If no answer is received, both hypotheses must be entertained until other evidence arrives.

4.4. The Beliefs Manager

The *Beliefs Manager* (BM) maintains the database of what the Expert Manager believes to be true (i.e., facts). Specifically, the beliefs maintained by the BM express what the Expert Manager thinks the global system state is. Examples of beliefs would be what the connectivity graph of the network is, what the load of various resources are, or whether they are not operational or not. Different beliefs are expressed in different languages, depending on what is most natural. A number of different knowledge representations, such as simple lists, frames, networks, matrices, are currently recognized. Each belief is tagged with the type of representation language in which it is expressed. Thus, any module making use of the BM's database can understand a belief by inspecting the language representation type.

Beliefs are generated whenever the belief factor of a hypothesis reaches above some threshold. Note that this might cause a belief to be replaced if the hypothesis contradicts it. Beliefs are also generated through a learning mechanism, to be discussed later.

We are still working on finding a small but rich set of knowledge representation languages which express beliefs in the domain of distributed computer systems.

4.5. The Action Generator

The *Action Generator* (AG) is the module which provides feedback to the distributed system. The AG, like the Hypothesis Generator, contains rules. The antecedent of its rules expresses a set of beliefs, and the consequent expresses an action. If the set of beliefs are found in the belief database, the rule fires, causing the action to take place. The action will be the name of a process which gets activated when the rule fires. Processes used to carry out actions have a standard interface. This interface includes a way to start the process, stop it, and find out its status. In fact, an action will express the name of a process and which of these three functions to do.

The purpose of the AG is to notice that some aspect of the distributed system is not working correctly, and to take active measures to correct the problem. For example, the AG might notice that the load on the node on which it resides is too high, and may activate a process to offload work onto other nodes. Or, the AG might notice that a neighbor node is overloaded from network traffic. It would then activate a process which tells other nodes to route messages around the overloaded node.

4.6. The Critic

The *Critic* tries to hold the Action Generator accountable for its actions. It essentially takes note whenever a rule belonging to the Action Generator fires, and then monitors its effects by getting information from the Sensory Input Interpreter's database. Rules can then be graded on how beneficial their effects are, and how quickly they take place.

Implementing the Critic turns out to be difficult because it must somehow correlate actions with effects which take place possibly much later in time. This is commonly called the Assignment of Credit problem [15]. This remains an area of active research for us.

4.7. The Learner

The Expert Manager needs a mechanism whereby general learning or discovery of new facts can take place. At this time, we are experimenting with a very particular learning function. This involves the discovery of conditions which lead to bad states. The *Learner* is given knowledge of what good states are and what bad states are, each expressed as a set of beliefs. For example, a bad state would be simply the belief that the local node is overloaded. The Learner keeps a small and constantly updated history of measurements relevant to the bad states it was told about. When one of these bad states actually occurs, the Learner stores a snapshot of the current history. Given that these bad states occur many times and the Learner eventually acquires many snapshots, it looks for conditions which always exist before the bad state occurs. These are very useful discoveries to make, since beliefs are then generated which express conditions which potentially lead to bad states. The Action Generator can then be programmed to take evasive action when these conditions exist, thus avoiding bad states.

5. Conclusions

The potential gain in performance and reliability of distributed systems is yet to be realized. We believe that management of resources based on decentralized control is the right path to achieving this potential. Through the use of expert system techniques, specifically

(1) the ability to deal with the uncertainty of remote measurements as an integral part of problem-solving,

(2) the ability to infer high-level concepts such as hypotheses and beliefs from low-level measurement data, and

(3) the ability to converge quickly on views of the global system state by using a hypothesize-and-test paradigm,

we believe the formidable problems posed by real-time distributed decision-making can be overcome.

6. Acknowledgements

I would like to thank my advisor Professor Domenico Ferrari for his ideas and suggestions. His constant encouragement and valuable advice have made this project possible. I would also like to thank Barbara Bittel for reading and making suggestions on earlier versions of this paper. Finally, I thank Professor Lotfi Zadeh and the speakers of his weekly Expert Systems seminar for many thought-provoking discussions.

7. References

1. Stankovic, J.A., Distributed Computer Systems, *IEEE Transactions on Computers*, C-33:12 (1984) p. 1113.

2. Shortliffe, E.H., *Computer-based medical consultation: MYCIN*, American Elsevier, New York, 1976.

3. Pearl, J., On Evidential Reasoning in a Hierarchy of Hypothesis, *Artificial Intelligence Journal*, 28:1 (1986) pp. 9-16.

4. Shortliffe, E.H. and Buchanan, B.G., A model of inexact reasoning in medicine, *Mathematical Biosciences*, 23 (1975) pp. 351-379.

5. Zadeh, L.A., The role of fuzzy logic in the management of uncertainty in expert systems, *Fuzzy Sets and Systems*, 11 (1983), pp. 199-227.

6. Shafer, G., *A Mathematical Theory of Evidence*, Princeton University Press, 1976.

7. Gaines, B.R., Fuzzy and probability uncertainty logics, *Information and Control*, 38 (1978), pp. 154-169.

8. Brownston, L. et al., *Programming Expert Systems in OPS5*, Addison-Wesley, Reading MA, 1985.

9. Tanenbaum, A.S., *Computer Networks*, Prentice Hall, New Jersey, 1981.

10. Tenney, R.R. and Sandell, N.R., Structures for distributed decisionmaking, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11:8, (1981) pp. 517-527.

11. Tenney, R.R. and Sandell, N.R., Strategies for distributed decisionmaking, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11:8, (1981) pp. 527-538.

12. Ho, Y., Team Decision Theory and Information Structures, *Proceedings of the IEEE*, 68:6, (1980) pp. 644-654.

13. Tenney, R.R. and Sandell, N.R., Detection with Distributed Sensors, *IEEE Transactions on Aerospace and Electronic Systems*, AES-17:4, (1981) pp. 501-510.

14. Lesser, V.R. and Erman, L.D., Distributed Interpretation: A Model and Experiment, *IEEE Transactions on Computers*, C-29:12, (1980) pp. 1144-1162.

15. Minsky, M., Steps Toward Artificial Intelligence, in *Computers and Thought*, E.A. Feigenbaum and J. Feldman, eds., McGraw-Hill, New York, 1963, pp. 406-450.