Intelligent Decentralized Control in Large Distributed Computer Systems

A DISSERTATION
SUBMITTED IN PARTIAL SATISFACTION OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN COMPUTER SCIENCE
IN THE GRADUATE DIVISION
OF THE UNIVERSITY OF CALIFORNIA, BERKELEY

by
Joseph Carlo Pasquale
April 1988

Copyright © 1988
by
Joseph Carlo Pasquale

Intelligent Decentralized Control in Large Distributed Computer Systems

By

Joseph Carlo Pasquale
B.S. (Massachusetts Institute of Technology) 1982
M.S. (Massachusetts Institute of Technology) 1982

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

in the

GRADUATE DIVISION

OF THE

UNIVERSITY OF CALIFORNIA, BERKELEY

Approved: .... April 22, 1988
Chairman Date

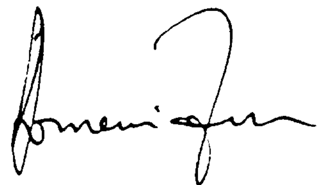
.... 4/18/88

.... 4/13/88

.... 4/19/88

....

ABSTRACT



In very large distributed computer systems, there are significant problems when one considers decentralization of control amongst agents managing resources. Probably the most difficult is that agents must make good fast coordinated decisions based on uncertain and differing views of the global system state. Our thesis is that despite such problems, effective decentralized control systems can be built based on a set of seven design principles which we describe. We also apply these principles to the problem of decentralized load balancing, and provide results based on trace-driven simulation experiments.

Our approach is *knowledge-based,* by which we mean that an agent will make use of heuristics and domain-specific knowledge about the behavior of itself and other agents to make good decisions. A powerful technique we present is one that agents use to quantify the uncertainty of information they have, and, based on these quantifications, to make better decisions. Agents adapt their decisionmaking to changing conditions by observing the system at infrequent (to minimize communication overhead) and opportune times, and then relying on their inference capabilities between observations. To minimize the occurrence of mutually conflicting decisions, we introduce a technique called *SPACE/TIME Randomization,* which provides implicit coordination of agents and requires minimal communication. The solutions we present are based on a combination of extensions of decision theoretic techniques and artificial intelligence techniques.

ACKNOWLEDGEMENTS

I have been most fortunate in having Domenico Ferrari as my thesis advisor; he has been exceptional. Without his guidance and support, this dissertation would not exist. His intellectual honesty and integrity has been a source of inspiration, and his warmth as a human being, boundless. I will treasure all that he has taught me, and I hope to continue to learn from him. I truly admire him.

I would like to thank Lotfi Zadeh for all his advice, guidance, and stimulating conversation. I thank Ronald Wolff for introducing me to stochastic processes. I also thank Luis-Felipe Cabrera who provided me with ideas, motivation, and encouragement.

I thank my family for their love and support. And, I thank Barbara; she has been my source of strength.

I greatly appreciate the support given to me by IBM for the fellowship they provided, by DARPA contract N00039-84-C-0089, and by DEC for their donation of a VAX 8600 on which I carried out my experiments.

To Mama and Papa

TABLE OF CONTENTS

1 Introduction 1
1.1 Motivations for Decentralized Control Systems 2
1.2 The Problems of Decentralized Control 4
1.3 Formula for a Solution 5

2 Past Related Work 6
2.1 Foundations: Decision Theory 6
2.2 Decentralized Control Applications 8
2.3 Review of Two Significant Past Studies 11

3 A Formal Model for Decentralized Control 14
3.1 Model Requirements 14
3.2 The Formal Model 14
3.2.1 The Agent Model 15
3.2.2 The Influence Model 17
3.3 Utility and Objective 18
3.4 Theoretical and Practical Limits 22
3.5 Summary of Model Requirements 24

4 Principles and Techniques 26
4.1 Knowledge-based Solution 27
4.2 Knowledge Abstraction 29
4.3 Uncertainty Quantification 33
4.4 Directional Heuristics 35
4.5 Information Age Integration 37
4.5.1 Choosing the Right Abstract State Space 38
4.5.2 Abstract State Spaces with Slow Transition Rates 40
4.5.3 Decisions and Utility 42
4.6 Frugal Communication 49
4.6.1 Informal Analysis of Local Loss 49
4.6.2 Decision Quality Loss Function 56
4.6.3 Observations and Simplifications 59
4.6.4 Approximations 61
4.6.5 Communication Loss Function 65
4.7 SPACE/TIME Randomization to Avoid Resonances 69
4.7.1 Problem Formalization 70

4.7.2 The SPACE/TIME Randomization Technique 71
4.8 Towards a Unifying Framework for Intelligent Agent Design 73
4.8.1 Observe-Reason-Act Structure 73
4.8.2 Architectural Framework 75
4.8.3 The Observation/Interpretation Structure 76
4.8.4 Actions at Different Levels of Abstraction 78
4.8.5 How the Framework Incorporates Our Principles 79
4.9 Summary of Principles 80

5 Decentralized Load Balancing 82
5.1 Formal Description 82
5.2 Abstract State Space 82
5.3 Domain-specific Knowledge 85
5.4 Designing State-transition Models 86
5.4.1 Load Levels and Degree of Variability 86
5.4.2 The Agent State-transition Model 88
5.5 Measures for Comparing the Desirabilities of Agents 90
5.5.1 Computing an Agent's State Utility 91
5.5.2 Conditional Utility with Respect to Job Offloading 92
5.5.3 Expected Utility 94
5.5.4 Conditional Expected Utility 94
5.6 Making Rational Decisions 97
5.6.1 Problem Decision Rules 97
5.6.2 SPACE/TIME Randomization for Load Balancing 99
5.6.3 Decisionmaking Processes for Load Balancing 103
5.6.4 Communication Decision Rule 105

6 Experiments 110
6.1 Experimental Load Balancing 110
6.2 Experimental Setup 112
6.2.1 Processor Simulation Model 112
6.2.2 Job Activity 115
6.2.3 Job Movement Between Machines 116
6.2.4 Operating System Overhead 116
6.2.5 Input Trace Description 118
6.2.6 Validation of the Simulator 119
6.3 Constants for State Transition and Utility Models 122
6.3.1 Abstract State Space 122
6.3.2 Load Level and Degree of Variability 124
6.3.3 State Transition Probability Matrix 126
6.3.4 Utility Models 129
6.3.5 Efficient Utility Computations 137

6.3.6 Efficient Payoff Computations 138
6.4 Experimental Results 139
6.4.1 Types of Job Placement Strategies 140
6.4.2 Accounting for Costs 141
6.4.3 Number and Types of Experiments 142
6.4.4 Results 143

7 Summary and Conclusions 151
7.1 Summary 151
7.2 Conclusions 152

Bibliography 154

CHAPTER 1

INTRODUCTION

A distributed computer system [Davi81] [Ensl78] with decentralized resource control [Abra80] [Jens78] is a collection of *agents* which reside on a (geographically) distributed set of computers and which control *resources* so that *work* can be carried out in an integrated fashion. In this dissertation, we investigate the problem of designing such systems. A key feature is that control of all resources is, to varying degrees, distributed amongst the agents, i.e., decentralized. The goal is to find a way for the agents to coordinate their actions to maximize some index of system performance. (Our main interest lies in performance optimization.)

The systems we are exploring have special characteristics. Every resource *belongs* to a particular agent, that is, it is *directly controlled* by that agent. An agent may also accept requests from other agents for the use of its resources. So, in a sense, all agents have an *indirect control* over all resources, by having (varying degrees of) access to them. Although agents can act *autonomously,* we are interested in making them *cooperate*. In fact, we limit the scope of this research to cooperative systems where agents will not act maliciously. The goal is to determine *how* to make agents cooperate effectively, given that they are willing to do so.

The systems of interest have a large number of agents and resources, at least tens, but more likely hundreds or even thousands of them. This means that, at any point in time, it is likely that there will be many simultaneous requests for resources, and that there will be many resources from which to select.

We are concerned with an agent's main activity, *decisionmaking*. Agents must decide when to make use of resources, and which resources are most beneficial at that time. Consequently, agents are interested in the *states* of resources, which they can determine by observation through *communication*. Since communication takes time, all received state information is delayed by transmission time. This situation is acceptable because the state information which agents use to base their decisions does not have to be perfect. In fact, bad decisions are tolerable, so long as they do not occur very often. This is in contrast to database control problems, where data integrity requirements are very stringent.

A major constraint is that the agents have real-time response requirements. The time it takes to make a decision, along with its expected consequences, is a cost which *must* be taken into account. This rules out solutions which involve mathematical programming or exhaustive searches. Agents must make reasonably good decisions in a reasonably short amount of time, where what is "reasonable" depends on the problem.

Why is it important to consider decentralized control systems? The current trend of computer systems is toward larger numbers of loosely connected processing nodes [Ande87]. Much progress has been made in addressing the *communication* problem in such systems, specifically, the establishment of network protocols for information transmission [Tane80]. This has heightened *the potential* for the effective sharing of all resources by all agents (for which there still remains much work in establishing adequate protocols). To realize this potential, we need to develop good solutions for the *coordination* problem, specifically for the harmonious interaction of all agents in the sharing of all the resources.

There are significant problems when one considers decentralization of control. Probably the most difficult is that agents have limited knowledge about the state of other agents and resources. They must make decisions based on partial information, which is often out-of-date and sometimes incorrect. Further, communication is not free. Agents must be prudent in deciding when to request for updated state information. Agents must also work within real-time constraints. They must realize the tradeoffs between the quality of a decision and the time it takes to achieve that quality.

In developing solutions for these problems, the following philosophy will be adopted. Agents will seek to *infer* state information about resources and other agents rather than relying solely on explicit communication, which will be done *frugally*. These inferences will be based on knowledge of behavioral models of the resources and of the other agents. These models are either acquired by the agent's ability to observe and summarize past behavior, or by the initial programming of a human expert, or by both methods. Coordination will also be achieved through implicit communication (e.g., information sharing and inference) rather than relying solely on explicit communication. The reader will note that a major tenet of this philosophy is to *replace communication with local computation* (e.g., inferencing) whenever possible. This is of critical importance when the systems under consideration are very large, and the number of messages required for global communication (e.g., broadcasting) is unreasonably high, causing excessive overhead.

1.1. Motivations for Decentralized Control Systems

Let us consider more deeply the question of why the study of decentralized control systems is important. In particular, why should decentralized control be preferred to a centralized scheme? We will argue that the potential benefit of decentralized over centralized control is so great that it outweighs the difficult problems it poses. And yet, these problems have caused many designers to resort to centralized schemes, as they lacked methods for dealing with them. So, we shall now review the potential benefits of decentralized control. Afterwards, we will consider in detail what are the corresponding problems.

Complexity Management

Consider a large distributed computer system, one with hundreds or thousands of agents. It is clear that the global control of such a system in real time is too complex for any single agent. There are just too many objects to monitor and control simultaneously. Since a single controlling agent must receive information from and send commands to all other agents, bottlenecks are likely to occur on the communication paths leading to this agent. Division of labor amongst many cooperating agents is one way of managing this complexity. Ideally, control is distributed so that each agent accepts part of the burden of control and contributes to the effective global control of the system. Moreover, control-communication paths are short and uniformly distributed throughout the system, minimizing the likelihood of communication bottlenecks.

Speed

When control is distributed among many agents, multiple decisions are made in parallel, which offers a potential increase in system performance. Control decisions can be made *near* the objects they are to affect; thus, communication of state information to the decisionmaker and communication of commands travel short distances, thereby without appreciably reducing speed (i.e., system throughput and responsiveness).

Reliability

A major reason for designing distributed systems is the reliability they offer. By using distribution and redundancy of control, single points of failure which are characteristic of centralized control systems are avoided. The potential benefits in speed discussed above offer the possibility of using more sophisticated reliability (fault detection and recovery) algorithms.

Scalability

In a *fully* decentralized control scheme, all agents may follow the same control algorithm; consequently, the distribution of control is symmetric amongst all the agents. Scaling the system up to comprise a larger number of agents is indeed simpler than if control were distributed asymmetrically amongst the agents. Asymmetric distributions of control (e.g., master/slave relationships) force an a priori grouping of agents, which must necessarily change when more agents are added. A symmetric distribution of control imposes no a priori grouping. Instead, natural groupings will evolve out of necessity (i.e., an agent will be likely to impose more control on objects in its immediate vicinity, than on those which are distant).

Autonomy

Finally, an important positive characteristic of decentralized control is that agents can act autonomously. Each agent is in full control of itself, and need not necessarily rely on other agents. Control is then shared, not by imposition, but rather because it is in each agent's interest to do so. This is important in considering present

(and, even more, future) distributed systems, whose parts are often owned by different organizations. These organizations are generally not willing to give up power over their machines, but find it reasonable to share power because it is in their best interest to do so.

1.2. The Problems of Decentralized Control

Although decentralized control offers complexity management, speed, reliability, scalability, and autonomy, there are associated problems which have no apparently simple solutions; rather, they seem insurmountable.

Consider the mind-set of an agent taking part in a decentralized control scheme. It is summed up succinctly in the phrase:

think globally, act locally.

An agent's control decisions should be based on the global system state. In turn, an agent's local decisions, taken as a component of all the concurrent decisions made by all participating agents, will affect the global system state. Clearly, these decisions should be harmonious, and not mutually destructive.

And so, we encounter the first fundamental problem of decentralized control: *each agent is uncertain about the current global system state.* Information about the global system state is distributed. Since pieces of such information take varying degrees of time to be received by any single agent, no agent can ever know, with complete certainty, the current global system state. At best, an agent can determine a past global state, but not the current one. And yet, agents must base their decisions on what they believe to be the current global state.

There is a second fundamental problem of decentralized control: *each agent is uncertain about the current actions of all other agents.* Since an agent does not know what other agents believe, it cannot predict what they will do. And yet, a goal of all the agents is to make harmonious decisions.

Finally, as if these problems were not difficult enough, there is the constraint imposed on all agents that they make *fast* decisions. The time used to make decisions has a cost, and must be minimized. These decisions, which are complex because they must take decentralization into account, are generally more costly in time than decisions under a centralized control scheme.

In summary, agents must make decisions based on global state information of which they are uncertain; agents are uncertain of each other's actions and yet must act harmoniously; agents are under time pressure to make their decisions, which must therefore be not just good, but also fast. Thus, we are challenged with the question: can the benefits of decentralized control really outweigh the difficulties? Our thesis is that this question can be answered in the affirmative. In the next section, we will outline a formula for the solution, which is the main subject of this dissertation.

1.3. Formula for a Solution

Underlying the solution we adopt to deal with the problems presented above is the idea that, in all real systems, there are *patterns* in their behavior. If only these patterns could be recognized and encoded, they could be used for predicting future events. In the case of decentralized control systems, knowledge of these patterns and knowledge of recent-past state information could be used to infer the current system state.

The solution we present is a *knowledge-based* solution, by which we mean that an agent will make use of heuristics and domain-specific knowledge about the behavior of itself and other agents to make good decisions. A powerful technique we present is one the agents can use to quantify the uncertainty of information they have, and, based on these quantifications, to make better decisions. Finally, agents adapt their decisionmaking to changing conditions by observing the system at infrequent (to minimize communication overhead) and opportune times, and then relying on their inference capabilities between observations. The solutions we present are based on a combination of extensions of decision theoretic techniques and artificial intelligence techniques.

We proceed as follows. In Chapter 2, past work in related areas is discussed. In Chapter 3, a formal model of decentralized control is presented. In Chapter 4, the principles and techniques for dealing with the problems of decentralized control are introduced. In Chapter 5, we discuss the application of these principles to a particular decentralized control problem, namely, *load balancing*. In Chapter 6, we present the results of load balancing experiments to illustrate the feasibility of our solutions. In Chapter 7, we summarize our conclusions.

CHAPTER 2

PAST RELATED WORK

In this chapter, we first explore the theoretical foundations upon which this dissertation is based. Next, we discuss a number of areas of research which are closely related to our work. Finally, we review two relevant experimental studies which were sources of inspiration for this dissertation.

2.1. Foundations: Decision Theory

According to Blackwell and Girshick, "decision theory applies to statistical problems the principle that a statistical procedure should be evaluated by its consequences..." [Blac54]. This principle is rooted in Neyman and Pearson's theory of hypothesis testing [Neym33], and was extended to all statistical problems by Wald [Wald50].

Decision theory attempts to provide a model for making *optimal* decisions based on *uncertain* information. The uncertainty of information is quantified by a probability measure on the possible values of variables, and optimality depends on the evaluation of possible outcomes, as modelled by utility theory. The structure of a decision theoretic problem is to select a decision which maximizes the expected utility over all possible consequences.

One of the problems with a decision theoretic formulation is that the probability distributions of all random variables are assumed to be known by all decisionmakers. Most often, this is not the case. In fact, some of these distributions may not even exist. Consequently, approximate forms of reasoning have been developed, such as Zadeh's fuzzy reasoning and possibility theory [Zade77] [Zade79], and the Dempster-Shafer theory of beliefs [Shaf76].

Game theory

Decision theory is really a special case of the earlier *theory of games,* first introduced by Borel [Bore21], developed and generalized by von Neumann [Neum28], and then appearing in the definitive work by von Neumann and Morgenstern, *Theory of Games and Economic Behavior* [Neum47]. Game theory focuses on the problem of dealing with *adversaries,* and predicting their actions in the absence of communicated data. The structure of a game theoretic problem is similar to that of a decision theoretic problem, except that there are now *multiple* decisionmakers (usually, but not necessarily, two), and the nature of possible information uncertainty lies in the inability to perfectly predict the outcomes of moves (i.e., of decisions) because they are governed by *chance,* or because they are the moves of other decisionmakers, which

cannot be anticipated (a constraint imposed by the structure of the problem), or both. A concise treatment of game theory appears in [Blac54], and a more in-depth survey and analysis appears in [Luce57].

Team decision theory

Team decision theory, first proposed by Marschak as an outgrowth of organization theory [Mars55], and further developed by Radner [Radn62], combines game theory's notion of multiple decisionmakers and decision theory's notion of optimal decisionmaking based on uncertain information and utility. Team decision theory stresses the *distributed* nature of the decisionmakers, in that the availability of information is governed by constraints placed on their modes of interactions (e.g., limitations on communications and observations). Decisionmakers often have different but correlated information about underlying system dynamics. Decisionmakers are part of a team: they are usually not adversarial (as is often, but not always, the case in game theoretic problems) in that they work together to solve a single problem. Thus, there is a need for *coordinated* actions to realize a positive payoff.

Witsenhausen [Wits68] describes a team-theoretic decision problem as having five ingredients:

(1) *states of nature*: a vector of random variables;

(2) *observations*: functions of the states of nature;

(3) *decision variables*: functions of the observations;

(4) *strategy*: decision rules which must be determined by design;

(5) *loss criterion*: a function of decisions and states of nature.

A team decision problem is to select decisions which minimize the expected loss over the possible states of nature. Consequently, the goal is to find strategies for decisionmakers which carry out this minimization of loss. In [Ho80], Ho presents a concise tutorial on the team decision problem with examples and a survey.

Distributed knowledge

The theories presented so far are all *information* based: decisions and their quality are based on the value of information, its reliability, and its availability. But there is also the problem of how distributed agents acquire *knowledge*. An agent's knowledge is not only concerned with the states of nature, but, because of the multiplicity of agents and because they do not share a common memory (i.e., they are distributed), it is also concerned with what other agents know. Halpern and Moses explore this in [Halp84], where they develop the notion of knowledge hierarchies and *common knowledge* in distributed environments. If a proposition p is common knowledge, then p is known by all agents, and all agents know that all agents know p, and all agents know that all agents know that all agents know p, and so on *ad infinitum*. They show that common knowledge cannot be attained solely through communication. This is important, as we will show that a *perfect* decentralized

control system, one where all agents always make the best decisions, cannot be built if the current states and current actions of all agents are not common knowledge.

2.2. Decentralized Control Applications

We now consider different areas of research concerned with decentralized control problems. Within each area, we will discuss a number of research studies which influenced our research.

Control-theoretic techniques

Research on control-theoretic techniques for decentralized control have focused on optimization based on *decomposition*. A problem is decomposed into subproblems with interactions modelled by *interaction variables*. In general, interaction is limited due to constrained cooperation (by design) amongst the individual subproblem-solvers in order to make the analysis tractable.

In [Jarv75], Jarvis presents an informative survey of adaptive control optimization based on such decomposition techniques. He discusses a number of methods, including gradient, correlation, random, stochastic automata, fuzzy automata, pattern recognition, and mixed strategies. In [Sand78], Sandell, Varaiya, Athans and Safonov present a fairly complete survey (up to 1978) of decomposition techniques for decentralized and hierarchical control, and mathematical methods for analyzing large scale systems. For a tutorial on distributed control theory based on decomposition techniques, see [Lars79].

Many of the algorithms based on decomposition techniques sequentially solve a (possibly infinite) number of subproblems, whose solutions are expected to converge to the optimal solution for the main problem. In [Cohe78], Cohen discusses the principles of such "decomposition-coordination (two-level) algorithms."

Another problem of interest is that of *data fusion*, the combining of data from a distributed set of sources. In a decentralized control system, each agent may be both a source of data (which is sent to other agents) and a combiner of data (which is received from other agents); consequently, data fusion potentially occurs at each agent. In [Tenn81c], Tenney and Sandell present an extension of detection theory (see [VanT68]) to problems requiring distributed sensors, which has applications to data fusion. See [Draz78] for another data fusion application.

Formal models

In [Tenn81a], Tenney and Sandell present a formal model for distributed decisionmaking agents. This is a refinement of the general team decision model found in [Ho80], in that they transform the notion of state-space into one that is more distributed, and knowledge of agent states is more compartmentalized. In essence, by restricting the scope of each decisionmaker's knowledge of the underlying system dynamics, the search for optimality becomes more tractable. In a companion paper [Tenn81b], they present a formal model for distributed decisionmaking coordination

strategies based on communication, prediction, and abstraction. They analyze how information uncertainty is reduced by considering various types of communication and different organizational structures.

Casavant and Kuhl present a formal model of distributed decisionmaking in [Casa86], based on graph and finite automata theory. Their model is novel in that they explicitly account for message-passing. They also give an example of how the model can be applied to the problem of load balancing.

Distributed problem solving

Distributed problem solving (DPS) is the activity of multiple agents seeking to solve a single problem in a cooperative but decentralized fashion. The agents are generally high-performance loosely-coupled semiautonomous computers. The focus of DPS research has been on developing effective methods of interaction for agents. Specifically, this includes techniques for cooperation and coordination through selective information sharing with limited amounts of communication. A collection of papers describing the current state of research in this area can be found in [Huhn87].

Early work in DPS research was done by Victor Lesser and Lee Erman [Less78], which led to their model for distributed interpretation in [Less80]. We will review this work in the next section. In [Less81], Lesser and Corkill describe an approach for structuring distributed processing systems which they call *functionally accurate/cooperative*. Agents in these systems cooperate and function effectively despite inconsistent views of state information due to distribution-caused uncertainty.

In related research, Corkill and Lesser discuss the problem of attaining *global coherence* [Cork83] through meta-level control for the coordination of agents. The *network organizational structure* paradigm is presented, which specifies the control of information and the relationships between nodes. Specifically, this paradigm is concerned with allocation of tasks such that local activity within agents is encouraged, and the need for inter-agent communication is reduced. In [Less83], Lesser and Corkill describe an application of these techniques to the problem of vehicle monitoring with information collected by a geographically distributed network of sensor nodes.

In [Smit81], Smith and Davis also describe frameworks for DPS cooperation. In particular, they discuss the contract net protocol (see also [Smit80] and [Davi83]), one of the major cooperation paradigms developed by DPS research. In the contract net paradigm, agents *negotiate* and make contractual agreements over tasks to be carried out. This differs significantly from other forms of cooperation in that agents can back out of the negotiation process at any time, rather than, say, submitting to majority rule as in voting paradigms.

In [Stee86], Steeb, McArthur, Cammarata, Narain, and Giarla discuss cooperation strategies for distributed problem-solving agents by analyzing organizational policies (task decomposition and assignment), and information-distribution policies (the nature of inter-agent communication). They present a framework for DPS, and its

application to the problem of air fleet control. See also [Camm83] for a more detailed discussion of cooperation strategies for distributed air-traffic control.

Genesereth, Ginsberg, and Rosenschein propose in [Gene85] a different approach to cooperation, using a metaphor for communication based on game theory. Specifically, agents with differing goals try to produce better results than other agents working on the same problem. In this approach, agents interact less, and conflict dominates their activities.

Finally, in [Gins87], Ginsberg identifies the conflict between agents pursuing purely local optimizations and the need for cooperation and coordination. He shows the desirability of the *common rationality assumption* (equipping agents with matching decision procedures). Under this assumption, one can precisely characterize the communication needs of agents in the context of cooperation and coordination.

Job scheduling

We end this section with a discussion of decentralized approaches to job scheduling, since we will apply in Chapters 5 and 6 our techniques to this problem to demonstrate their feasibility. Although a great deal of research has been carried out on the job scheduling problem (see [Casa88] for a taxonomy and a survey of job scheduling in distributed computer systems), we focus on a small number of select works which were relevant to our research in that they based their solutions on decision-theoretic techniques or distributed problem-solving paradigms.

In [Chou82], Chou and Abraham propose a decision-theoretic approach to the problem of assigning tasks to processors which have different speed and reliability characteristics. Although their approach was static (i.e., it did not respond to changing loads), and it assumed the availability of a priori information about tasks (which is generally not the case in practice), it was the first to make explicit use of decision theory and attain good theoretical results.

In [Malo84], Malone, Fikes and Howard considered task allocation using the contract net protocol. In their simulation experiments, they used the metaphor of a marketplace in which the bids represented estimates by the bidding nodes of when they could complete the processing of a task. Although they also had to assume the availability of a priori information about task execution requirements, they were able to achieve good performance with low communication overhead.

Stankovic has conducted significant research in decentralized job scheduling. In [Stan84b], he presents three adaptive algorithms for decentralized scheduling which assume no a priori knowledge about jobs. His simulation results show that decentralized algorithms exhibit stable behavior and improve performance at modest cost. In [Stan85], he successfully applies Bayesian decision theory to job scheduling. We will review this research in the next section.

2.3. Review of Two Significant Past Studies

We now consider two studies which were particularly relevent to this dissertation. The first, by Victor R. Lesser and Lee D. Erman, describes an experiment based on a model for distributed interpretation which allows for inconsistent and incomplete views of the global system state [Less80]. The second, by John A. Stankovic, reports on a simulation study of decentralized job scheduling based on an application of Bayesian decision theory [Stan85].

Distributed Interpretation

Lesser and Erman [Less80] present a model of distributed processing which explicitly deals with the problems of distribution-caused uncertainty and errors in control, data, and algorithm. In their study, they apply this model to the problem of *distributed interpretation.* They define an interpretation system as a transformer of a set of signals from some environment to higher level descriptions of objects and events in the environment. A *distributed* interpretation system is one where sensors for signal reception are widely distributed, interpretation requires data from multiple sensors, and must operate in a distributed manner since communication of all information to a centralized interpreter is undesirable (often due to real-time response constraints, limited communication bandwidth, and reliability).

Their model is based on interpretation techniques found in knowledge-based artificial intelligence systems, which use the problem-solving paradigm of solution search through the incremental aggregation of partial solutions. These techniques handle, *as an integral part of the interpretation process,* the existence of uncertainty in the input data (e.g., due to noisy channels), and the possibility of incorrect and incomplete knowledge. In particular, they use speech understanding, based on the Hearsay-II system [Erma80], as the vehicle for experimentation with their model.

Their distributed system consists of three nodes, each of which is a functionally complete Hearsay-II system with access to one segment of the speech data of an utterance. The nodes generate a single unified interpretation of an utterance by communicating and resolving, in a cooperative and competitive fashion, partial tentative interpretations based on their local views. Each node includes a number of knowledge-sources, which are independent modules containing separate areas of knowledge, such as acoustics, phonetics, syntax, and semantics. A hypothesize-and-test problem-solving paradigm is used iteratively to arrive at partial solutions. Consequently, control is decentralized: it is asynchronous and data-directed, and synchronization is obviated by the self-correcting nature of information flow between knowledge sources. Their primary goal in the decomposition of the problem was to minimize internode communication relative to intranode processing.

The results of this research are significant because of the techniques it proposes, which we believe can be used as general structures for solving decentralized control problems. These include:

(1) structuring a problem-solving activity as a distributed, incremental, asynchronous, and opportunistic process, in that solution paths are investigated in the order of how promising they are, based on the reliability (degree of certainty) of information;

(2) minimizing internode communication by exchanging high level, abstract information, which is more succinct and more readily usable than low-level information (e.g., raw inputs);

(3) arriving at the solution by incremental aggregation, which allows the system to be self-correcting in spite of incorrect and incomplete information.

Decentralized Job Scheduling

In this work, Stankovic applies a heuristic based on Bayesian decision theory to decentralized job scheduling [Stan85]; the main advantage of this approach is that the heuristic dynamically adapts to the quality of state information. He considers a relatively small distributed system of five nodes, where each node is a separate decision-making agent, with the addition of two *monitor* nodes (only one is used, the other is a backup), to compute the information necessary for the heuristic.

An agent's decision as to whether jobs should be offloaded depends on its view of the load conditions on all nodes. An agent's view is based on periodic state information updates from all other agents. Agents periodically send state updates to the monitor node, which computes probability distributions and tables of "maximizing actions," indicating the utility of all possible actions for all possible views. These tables are periodically sent back to the agents. Consequently, an agent's job scheduling function is to consult its table of maximizing actions: given its view of the global state, the best action (the one with the maximal utility) is selected.

Simulation experiments were carried out where a number of characteristics and their effects on performance were studied, such as: the state information update period and the period for calculating maximizing actions; the mix of job arrival rates; the delay in the subnet; the job scheduling interval; the loss of monitor nodes. The results of these simulation experiments were analyzed and then compared with results of analytic models which provided theoretical upper and lower bounds on average job response time. Stankovic showed that his heuristic method based on Bayesian decision theory not only performed well and incurred low overhead, but was robust given the presence of out-of-date state information. The main limitation of this approach was in the centralization of the probability distribution and utility computations: although reliability is maintained by having a backup monitor node, the likelihood of bottlenecks occurring on communication paths leading to the active monitor node increases with the size of the distributed system.

Observations and Summary

We observe several common underlying themes in both approaches to the activity of multiple agents working in a decentralized fashion cooperatively to solve a

problem. The first observation is the importance of handling uncertainty of information *as an integral part* of the decisionmaking process. The second observation is the *preference* for using imperfect and uncertain information in decisionmaking over possibly better information which could only be acquired at a significantly higher cost. The third observation is the avoidance of explicit synchronization by a somewhat implicit coordination through information sharing. Consequently, errors will occur, *but they are tolerated* by providing for self-correction.

We consider the ability to quantify and explicitly account for the goodness of information on which decisions will be based to be of great importance in decentralized control systems. When systems become very large (with hundreds or thousands of agents), the problem of communication overhead and the importance of agents relying on local views of the global state are dramatically accentuated. Although both of the studies referred to above considered a very small number of agents, we hypothesize that the three observations we made concerning their approaches only become more relevant when larger systems are considered. One of our goals is to extend their results by explicitly considering the following problems, and developing techniques for solving them:

(1) how decentralized control can be structured in *large* distributed systems;

(2) how state information can be effectively shared;

(3) how not only the value of information, but also its *age*, can be incorporated in decisionmaking.

CHAPTER 3

A FORMAL MODEL FOR DECENTRALIZED CONTROL

The purpose for creating and presenting a formal model for decentralized control is mainly to establish a language for describing the objects of interest and their relationships. Our goal is to describe, in precise terms, what the limitations are in decentralized control applications. There exist other models, which focus on different levels of abstraction, and have correspondingly different goals [Ho80] [Tenn85a] [Tenn85b] [Casa86]. This model was devised not to replace these, but rather to focus on the particular level of abstraction of interest here. In fact, this model freely borrows elements and builds on the ideas developed from these existing models, in particular from that of Tenney and Sandell [Tenn85a].

3.1. Model Requirements

The model should fulfill the following requirements.

- It should comprise the following basic objects of interest: machines which carry out work; work, to be carried out; and information, to be communicated and used in making decisions.
- It should allow for the system's distributed nature.
- It should capture the basic notion of system activity *over time*.
- It should allow machines to be both autonomous, having direct control over themselves, and cooperative, allowing them to coordinate their activities.
- It should allow for the quantification of *preferences* for the various courses of action a machine can decide to take.
- It should provide for objectives for machines to attain.

At the conclusion of this chapter, we will review whether these requirements are indeed satisfied.

3.2. The Formal Model

A decentralized control system can be modeled as a directed graph **G**. Nodes **N** represent agents, and links **L** represent inter-agent *influences*. Formal definitions of these elements of a model are as follows:

$\mathbf{G} = (\mathbf{N}, \mathbf{L})$ $\mathbf{N}$ = set of nodes, $\mathbf{L}$ = set of links

$\mathbf{N} = \{A_i\},\ 1 \leqslant i \leqslant N$ A_i = agent i, $A_i \in \mathbf{A}$,
$\mathbf{L} = \{Z_{ji}\},\ 1 \leqslant j \leqslant N,\ 1 \leqslant i \leqslant N$ Z_{ji} = influence of A_j on A_i

As mentioned briefly in Chapter 1, a distributed computer system with decentralized resource control contains the following objects: agents, resources, and work. Each resource belongs to, or is owned by, or is *directly* controlled by, one and only one agent. With no loss of generality, we will limit each agent to own at most one resource in the model. Consequently, an agent directly controls the resource it owns, if it owns any. (Note that an agent may indirectly control a *remote* resource by sending requests to the corresponding remote agent, and these requests may be accepted or denied.) An agent which does not own a resource is limited to indirect control of some other resources. Since for each resource there exists an agent, agents may subsume the activity of resources, and therefore only agents are explicitly modeled. When an agent wants to control a remote resource (indirectly), we will simply say that the agent makes a request to the agent owning that remote resource. Resources will no longer be mentioned explicitly. We first consider the modeling of agents, and later that of inter-agent influences.

3.2.1. The Agent Model

The model of an agent is a structure with eight components (i.e., an 8-tuple). The first component is the agent's *state,*

$$x_i(t) \in \mathbf{X}_i = \{\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \cdots\}.$$

The agent's state is a time-dependent variable, whose values are chosen from a local state space $\mathbf{X}_i$. The value of an agent's state is of general interest. Remote agents will base their decisions on whether to make requests based on what they believe this value to be.

Each agent has its own source of tasks which are units of *work* to be carried out by the agent, which are generated and submitted (e.g., by users):

$$s_i(t) \in \mathbf{W} = \{\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \cdots\}.$$

The variable $s_i(t)$ is the locally generated work arriving at the agent at time t, and takes on values from the system-wide work space $\mathbf{W}$. Each member of $\mathbf{W}$ represents an atomic element of work. There is a special symbol, $\mathbf{w}_0$, that represents the null element of work. If at time t no work arrives at agent A_i, then $s_i(t)$ takes on the value $\mathbf{w}_0$. Since this work is locally generated, we call $s_i(t)$ A_i's *generated work.*

An agent's generated work will be considered as a stochastic process over time. At any point in time, the value taken on by $s_i(t)$ is selected from the distribution

$$P_{i_{\mathbf{W}}}\ (s_i(t),\ t),$$

defined over the work space **W**. In general, this distribution is time-dependent.

An agent is affected by other agents through what are called *influences*. Directed links (discussed below) in the graph G represent influences; all incoming links to a node represent all the external influences directly affecting an agent. This is modeled explicitly as part of an agent's structure:

$$z_i(t) = (z_{1i}(t),\ z_{2i}(t),\ \ldots,\ z_{Ni}(t)).$$

The variable $z_{ji}(t)$ is the influence of A_j on A_i. The *total influences* on agent A_i are represented by the vector $z_i(t)$.

Influences come in two varieties: *work* and *information*. A work influence and an information influence are given respectively by the vectors:

$$w_i(t) = (w_{1i}(t),\ w_{2i}(t),\ \ldots,\ w_{Ni}(t));$$

$$k_i(t) = (k_{1i}(t),\ k_{2i}(t),\ \ldots,\ k_{Ni}(t)).$$

Agent A_i's work influence at time t is made up of all the work requests (made by remote agents) which *arrive* at A_i at time t. We call $w_i(t)$ A_i's *transferred work* at time t. The information influence on A_i at time t is any information about the state of the system originating from agents in **A** and *present* in A_i at time t. We call $k_i(t)$ A_i's information concerning the system state. Note that, after some time t, say, $t+1$, $w_i(t+1)$ consists of only the work transferred between the times t and $t+1$ (or of null work elements if no work is transferred), whereas $k_i(t+1)$ will be the *same* as $k_i(t)$, *unless* new state information has arrived causing it to change. Thus, $k_i(t)$ is persistent, whereas $w_i(t)$ is not. Precise definitions of z_{ji}, w_{ji}, and k_{ji} will be given in the next section.

A most important concern about agents is their decisionmaking capability. At any point in time, an agent can make a decision, which is represented by the decision variable

$$d_i(t) \in \mathbf{D}_i = \{\mathbf{d}_0, \mathbf{d}_{i_1},\ \mathbf{d}_{i_2},\ \cdots\}.$$

Each point in space $\mathbf{D}_i$ is a distinct decision unique to agent A_i, except for $\mathbf{d}_0$, which represents the null decision, and is common to every agent's decision space. Decision-making is what allows an agent to (partially) control future values of its state, and, to a lesser degree, the future values of the global system state.

Each agent has a next state function f_i, where

$$x_i(t+1) = f_i(x_i(t), d_i(t)).$$

The next state is a function of the current state, and of the current decision an agent makes.

Finally, each agent has a decision rule, or *strategy*,

$$d_i(t) = \gamma_i(z_i(t), s_i(t)).$$

An agent's strategy takes into account influences (transferred work and state information), and generated work, and produces a decision. Note that since decisions are a function of influences and generated work, the next state $x_i(t+1)$ may be thought of as a function of the current state $x_i(t)$, of the current influences $z_i(t)$, and of the currently generated work $s_i(t)$. Ultimately, the agent decides how it will *allow* itself to be affected by influences and generated work. To emphasize this, the next state is defined solely as a function of the current state and of the current decision.

In summary, the model M_i of agent A_i is given by the 8-tuple:

1. $x_i \in \mathbf{X}_i = \{\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \cdots\}$	$x_i(t) = A_i$'s state
2. $s_i \in \mathbf{W} = \{\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \cdots\}$	$s_i(t) = A_i$'s generated work
3. $P_{i\mathbf{W}}: \mathbf{W} \times \mathrm{N} \rightarrow [0,1]$	$P_{i\mathbf{W}}(s_i(t), t)$, work distrib.
4. $w_i = (w_{1i}, w_{2i}, \ldots, w_{Ni})$	$w_i(t) = A_i$'s transferred work
5. $k_i = (k_{1i}, k_{2i}, \ldots, k_{Ni})$	$k_i(t) = A_i$'s global state info.
$z_i = (z_{1i}, \ldots, z_{Ni})$, $z_{ji} = (w_{ji}, k_{ji})$	$z_i(t) = \mathbf{A}$'s influence on A_i
6. $d_i \in \mathbf{D}_i = \{\mathbf{d}_0, \mathbf{d}_{i_1}, \mathbf{d}_{i_2}, \cdots\}$	$d_i(t) = A_i$'s decision
7. f_i: $\mathbf{X}_i \times \mathbf{D}_i \rightarrow \mathbf{X}_i$	$f_i(x_i(t), d_i(t))$, next state
8. γ_i: $((\mathbf{W}, \mathbf{X}_1), \mathbf{W}) \times \cdots \times ((\mathbf{W}, \mathbf{X}_N), \mathbf{W}) \rightarrow \mathbf{D}_i$	$\gamma_i(z_i(t), s_i(t))$, decision rule

It will be useful to consider vectors of the structures presented above for the entire system, i.e., for all agents. This will be done by simply dropping the agent's subscript. For example, the *global* system state is $x(t) = (x_1(t), \ldots, x_N(t))$, the *global* decision is $d(t) = (d_1(t), \ldots, d_N(t))$, and so on.

3.2.2. The Influence Model

The links of graph G, *labeled* Z_{ji}, represent inter-agent influences; the *variable* z_{ji} models the influence of A_j on A_i. As previously mentioned, influences are of two varieties: work and information.

Work influence, or work transferred from A_j to A_i is given by the variable

$$w_{ji}(t) \in \mathbf{W} = \{\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \cdots\}.$$

The value of $w_{ji}(t)$ is a point in the space $\mathbf{W}$.

Information influence, or A_i's information concerning A_j's state, is given by the variable

$$k_{ji}(t) \in \mathbf{X}_j = \{\mathbf{x}_{j1}, \mathbf{x}_{j2}, \cdots\}.$$

The value of $k_{ji}(t)$ is a point in A_j's state space $\mathbf{X}_j$. Together, $w_{ji}(t)$ and $k_{ji}(t)$ make up A_j's influence on A_i at time t, which is given by the variable

$$z_{ji}(t) = (w_{ji}(t),\ k_{ji}(t)).$$

What determines the values of $w_{ji}(t)$ and of $k_{ji}(t)$? Since $w_{ji}(t)$ represents a transfer of work from A_j to A_i, A_j must have made a decision, some time *before* t, for the request to be present at agent A_i at time t. This relationship between the *arrival* of transferred work and the *decision* which caused the work to be transferred is given by the *work function* g_{ji},

$$w_{ji}(t) = g_{ji}(d_j(\tau)),\ \ \tau < t.$$

The time τ will generally depend on factors such as the transmission time between A_j and A_i, message processing delays, and so on. For our purposes, this value τ need not be made precise. We are only interested in the fact that the decision by A_j must be made at some time τ preceding the reception of the transferred work $w_{ji}(t)$ by A_i.

Similarly, A_i's information concerning the state of A_j is based on a past communication by A_j to A_i, decided upon by A_j. This relationship between the arrival of information and the decision which caused the information to be sent is given by the *information function* h_{ji},

$$k_{ji}(t) = h_{ji}(d_j(\tau)),\ \ \tau < t.$$

Again, the important relationship is $\tau < t$. In particular, during the time interval $(\tau,\ t]$, a state change in A_j can occur. Thus, $k_{ji}(t)$ may not accurately reflect the state of A_j at time t. We will explore the ramifications of this shortly.

Summarizing the influence structures, link Z_{ji} in $\mathbf{L}$, which represents the influence of agent A_j on A_i, is defined by the following 4-tuple:

1. $w_{ji} \in \mathbf{W} = \{\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \cdots\}$ — $w_{ji}(t)$ = work transferred from A_j to A_i
2. $k_{ji} \in \mathbf{X}_j = \{\mathbf{x}_{j1}, \mathbf{x}_{j2}, \cdots\}$ — $k_{ji}(t)$ = A_i's state information about A_j
3. g_{ji}: $\mathbf{D}_j \rightarrow \mathbf{W}$ — $g_{ji}(d_j(\tau))$, work function, $\tau<t$
4. h_{ji}: $\mathbf{D}_j \rightarrow \mathbf{X}_j$ — $h_{ji}(d_j(\tau))$, information function, $\tau<t$

3.3. Utility and Objective

All agents have a goal or objective. It is useful then to develop a notion of *utility*, something for agents to maximize as their objective. Utility is defined to be a real-valued function of an agent A_i's state,

$$u_i(x_i(t)) \in \mathrm{R},\ \ 1 \leqslant i \leqslant N$$

The utility function $u_i(x_i(t))$ maps each state of A_i to a real number. The magnitude of a state's utility provides a measure of the degree of preference for it relative to other states.

We also want a notion of *global* utility, which we denote by

$$u(x(t)) \in \mathrm{R}.$$

$u(x(t))$ maps each global state $x(t) \in \mathbf{X}$ to a real number, again indicating the degree of preference of $x(t)$ over other global states.

Utility has been defined as a measure of preference of a state at a single point in time. Often, it is more interesting to consider the utility of a particular *sequence* of states. Therefore, we extend the utility function to map *distinct sequences of states* to real numbers.

A sequence of states for A_i between times t and $t+\tau$ is denoted by

$$x_i(t,t+\tau) = (x_i(t),\ x_i(t+1), \ldots,\ x_i(t+\tau)),\quad \tau > 0,$$

and denote the utility of $x_i(t,t+\tau)$ by

$$u_i(x_i(t,t+\tau)).$$

The utility of a state sequence might be defined by a simple formula, such as for instance

$$u_i(x_i(t_0,T)) = \sum_{t=t_0}^{T} u_i(x_i(t)).$$

More likely, however, it will be some complex function possibly not separable as in the example above, of all the states in the sequence.

This extension will also apply to global utility. The utility of the global state sequence $x(t,\ t+\tau)$ is

$$u(x(t,\ t+\tau)).$$

We are now ready to consider possible *objective* functions, which, like utility functions, can be defined as local or global. The local objective function for each agent is

find $d_i \in \mathbf{D}_i$ which maximizes

$$J_i(t+1) = \mathrm{E}[u_i(x_i(t+1))].$$

Thus, the goal of each agent in this case is to select a decision which will maximize the *expected* local utility of its next state. It is an expectation because an agent's next state is a function of a random variable, the generated work $s_i(t)$.

For local state sequences, we would have

find $d_i \in \mathbf{D}_i$ which maximizes

$$J_i(t+1,t+\tau) = \mathrm{E}[u_i(x_i(t+1,t+\tau))],\ \tau \geqslant 1.$$

In this formulation, an agent makes a decision at a particular point in time which maximizes the expected utility of states for an interval of time in the near future. Note that during this interval, the agent is free to make more decisions. Thus,

although an agent's decision is made *accounting for* multiple possible future states, at each future state the agent can correct for unpredictable events (e.g., due to the stochastic nature of the system) by making more decisions. Consequently, we call this objective a *step-wise probabilistic* optimization.

For the global state case, the objective is:

find $d \in \mathbf{D}$ which maximizes

$$J(t+1) = \mathrm{E}[u(x(t+1))].$$

For the global state sequence case, the objective is:

find $d \in \mathbf{D}$ which maximizes

$$J(t+1,t+\tau) = \mathrm{E}[u(x(t+1,t+\tau))],\ \tau \geqslant 1.$$

Some assumptions must be made about the knowledge and activity of our agents. The first is that the only items of information that agents do not know or cannot perfectly predict are those based on the non-deterministic elements of the system model. For example, this would include what new work from the agent's private source of work would appear in the future. Other items which are by nature *static*, such as the agents' state spaces $\mathbf{X}_1, \cdots, \mathbf{X}_N$, can be known by any or all agents, since, it would be *possible* to endow each agent with such information at the beginning of time. For example, generated work $s_i(t)$ is selected from the distribution $P_{i\mathbf{W}}$. Although an agent cannot predict a future value of $s_i(t)$, it may indeed know the distribution(s) of the types of generated work arriving.

Finally, it is assumed that everything that can be known at the beginning of time is *common knowledge*. The concept of common knowledge [Halp84] is defined constructively as follows. Consider a *propositional modal logic*, with propositions, formulas formed from propositions closed under negation and conjunction, and modal operators $K_1, \cdots, K_N$. The formula $K_i p$ has the semantics "agent A_i *knows* proposition p." Define Ep as the disjunction $K_1 p \wedge \cdots \wedge K_N p$, meaning "everybody knows p." Extend this definition so that $E^1 p = Ep$ and $E^{i+1} p = EE^i p$. $EE^i p$ means "everybody knows $E^i p$," thus, $E^2 p$ means "everybody knows that everybody knows p." Finally, define $Cp = E^1 p \vee E^2 p \vee \cdots$, meaning "$p$ is common knowledge."

These assumptions allow us specifically to focus on purely cooperative systems, which are most interesting when there is a common goal to be achieved, i.e., all agents try to achieve a single global objective. Consequently, our interests lie in determining *how* to achieve coordination amongst agents in the best practical way possible, given their *willingness* to cooperate.

It is interesting to consider the theoretical question of whether the global objective of maximizing the utility of the next state or next state sequence can be achieved. Therefore we end this section with the following theorem.

Theorem: The global objective, "find $d \in \mathbf{D}$ such that $J(t+1)$ is maximized," cannot be guaranteed unless $x(t)$ and $z(t)$ are common knowledge.

Proof: Let $d^* = (d_1^*, \cdots, d_N^*)$ be the global decision which maximizes $J(t+1)$. If $x(t)$ and $z(t)$ are common knowledge, then every agent A_i has the same information to determine d^*, can make the decision d_i^*, and can assume correctly that every other agent will do the same. (If there is more than one possible d^*, then, by convention, the first d^* computed is selected, assuming each agent tests all $d \in \mathbf{D}$ in the same order.) Of course, if any agent A_i does not know $x(t)$ or $z(t)$, it cannot uniquely determine d^* (and therefore, cannot determine d_i^*, because this decision depends on the decision of every other agent). Note that the theorem says that $x(t)$ and $z(t)$ must not only be known by all agents, but that $x(t)$ and $z(t)$ must be common knowledge. Consider the case where $E^2x(t)$ and $E^2z(t)$ are true, (i.e., everyone knows that everyone knows $x(t)$ and $z(t)$), but $E^kx(t)$ and $E^kz(t)$, for all $k > 2$, are false. An agent A_i might say of some other agent A_j: "I know A_j knows $x(t)$ and $z(t)$, but if A_j does not know that I know this, A_j may not make the decision I expect; I must account for this in the decision that I make." Since A_i cannot count on A_j making d_j^*, it may be statistically better for A_i to make a decision other than d_i^*.

More precisely, let $M \geqslant 1$ be the maximum value such that

$$[A] \quad (E^kx(t) \vee E^kz(t)) \text{ is } \mathit{true} \text{ for all } k \leqslant M.$$

If M exists (i.e., $M < \infty$), then for any agent A_i,

$$[B] \quad K_iK_j(E^{M-1}x(t) \wedge E^{M-1}z(t)) \text{ is false for some } j \neq i,$$

but

$$[C] \quad K_j(E^{M-1}x(t) \wedge E^{M-1}z(t)) \text{ is true,}$$

where [B] and [C] follow directly from [A]. (If $M=1$, then $E^{M-1}x(t)$ and $E^{M-1}z(t)$ simply become $x(t)$ and $z(t)$ respectively, in both statements [B] and [C].) As $d_i(t)$ and $d_j(t)$ are mutually dependent (in particular, the value of $d_i(t)$ will depend on whether $d_j(t)$ equals d_j^*), and $d_j(t)$ will generally depend on [C], then, since A_i does not know [C] (from [B]), there may exist a decision d_i' *which A_i considers statistically better* than d_i^* (i.e., the expected consequences of $d_i(t) = d_i'$, are better (the expected future state utility is greater) than the expected consequences of $d_i(t) = d_i^*$, *over all possible decisions* $d_j(t)$, but not as good as the consequences of $d_i(t) = d_i^*$ and $d_j(t) = d_j^*$). But if $d_i(t) = d_i'$, $J(t+1)$ is not maximized, and therefore the objective is not realized.

Note that if $x(t)$ and $z(t)$ are not common knowledge, but are known by all agents, then if every agent A_i behaves such that it computes d^* and simply selects the ith component as its decision $d_i(t)$, then this would achieve the objective! Where is the trick? It lies in the *implicit assumption* that agents behave in the manner just described, specifically, that they do not try to second-guess other agents but that they all follow the same procedure, and that this *rational behavior* is common knowledge.

Note that this assumption is not an unreasonable one; it could easily be built into every agent's decision procedure. (This assumption is similar in spirit to Ginsburg's "common rationality assumption" [Gins87].) However, it still requires that $x(t)$ and $z(t)$ are known by all agents, which generally *is* an unreasonable assumption.

3.4. Theoretical and Practical Limits

In the previous section we saw that unless the current global state $x(t)$ and the current global influence $z(t)$ are common knowledge, (or that unless $x(t)$ and $z(t)$ are known by all agents whose rational behavior is common knowledge), agents in a decentralized control system cannot perfectly achieve a global objective. It is most unlikely that in a real system, because of communication delays and random processes, either the current global state or the current global influences could be simply known by any agent, let alone be common knowledge. This is why the problems posed by decentralized control systems have been so formidable. And yet, there must be ways of structuring such systems so that global objectives can be achieved at least near-optimally; human organizations seem to be able to do this (although their time constants are much longer).

To explore possible techniques for solution, we must clearly understand the limits of what can be done theoretically, and practically. Let us start with some general observations about the model and its implications for coordinating agents.

- **Observation 1:** Each agent has *direct* control over itself, and only itself.
  An agent's next state $x_i(t+1)$ is directly affected by that agent's decision $d_i(t)$ through its next-state function $f_i(x_i(t),\ d_i(t))$. Thus, each agent takes part in determining the global system state in a decentralized control system, but does that by local actions only.
  Conclusion 1: If a global objective is to be achieved, *coordination of agents is a necessary condition.*

- **Observation 2:** Agents can *allow* other agents to *influence* them.
  Although an agent's next state $x_i(t+1)$ is directly affected only by that agent's decision $d_i(t)$, that agent can be indirectly affected by other agents' decisions. Each agent can potentially affect remote agent states through communication.
  Conclusion 2: *Coordination is possible, but only through limited communication.*

- **Observation 3:** Each agent cannot predict, with complete certainty, its future states.
  An agent's decision is a function of its influences and its generated work, $d_i(t) = \gamma_i(z_i(t),\ s_i(t))$. Since $s_i(t)$ is a random variable, the agent cannot predict what its future actions, and therefore what its future states, will be. (Of course, an agent can always decide to ignore its inputs, but this would generally go against realizing reasonable objectives.)
  Conclusion 3: *Coordination is limited by random events,* and therefore agents cannot affect the global state deterministically.

- **Observation 4**: Each agent has *limited indirect control* over other agents. The future states of remote agents cannot be directly affected by an agent; that agent can only influence remote agents by communication.
  Conclusion 4: Since any communication is not instantaneous, *coordination is limited by communication delays,* limiting the control that agents have collectively over the global state.

In summary, we see that, to achieve a global objective in a decentralized control system, coordination is a necessary condition, and is possible (to a limited degree) in our model through communication. But there are limitations to coordinating agents since communication is delayed, and agents cannot predict the future because they must respond to random events.

These observations and conclusions lead to what we call the *two fundamental problems of decentralized control:*

1. **No agent can know with certainty the current global state.**
2. **No agent can know with certainty the current actions of remote agents.**

Knowledge is attained either by computing it or by receiving it. (We can consider innate knowledge to be received at the beginning of time.) Thus, the appropriate question is whether an agent can obtain certain knowledge about the global system state and remote agent actions either by computation or by communication.

Problem 1 results from the fact that, due to finite communication bandwidth, the global *state* $x(t)$ cannot be communicated to all agents instantaneously. And, since inputs to the system are stochastic, the global state $x(t)$ cannot be computed exactly from previous information.

Problem 2 results from the fact that, due to finite communication bandwidth, the global *influence* $z(t)$ cannot be communicated to all agents instantaneously. And, since inputs to the system are stochastic, the global influence $z(t)$ cannot be computed exactly from previous information.

It is clear that Problem 1 is indeed a fundamental problem; Problem 2 is more subtle, but just as critical, and indeed a fundamental problem too. In fact, Problem 2 leads to the corollary that agents cannot simply optimize local objective functions and expect a global objective function to be optimized. This is best illustrated in the following example.

Consider the decentralized load balancing problem. Agents (e.g., computers) receive jobs from outside sources and must determine where the job should be executed so that the average time a job spends in the system is minimized. Clearly, a bad situation would be one where one agent has much work to do, while other agents are idle. A good load balancing decision rule would be one where an agent, if it has many jobs pending, offloads new jobs to less loaded agents. Say the state of each agent is characterized by its load, the amount of work it has on hand. Let us now

assume that Problem 1 is actually *not* a problem: all agents know the global system state, they all know what each other's load is. Now, if each agent used the locally optimal decision rule "offload a new job to the *least* loaded agent," notice what may happen. At any point in time, there is one least loaded agent (assume there are no ties). It is possible that new jobs arrive at each agent simultaneously, and they all decide to offload to the least loaded agent, whose identity they all know perfectly. This agent will therefore get swamped with work, turning what were locally optimal decisions into a global disaster. A globally optimal decision would somehow cause these jobs to be distributed over a number of agents so that no agent is overloaded. This is difficult to do, exactly because of Problem 2, since agents do not know their counterparts' current decisions because of the unpredictability of inputs.

There are some difficult practical problems also, such as that of measuring the utility of a state in a real system. Just recognizing its own current state may be a problem for an agent, not to speak of the state of a remote agent. Another problem is the implementation of how an agent selects a decision which even approximately can achieve the objective. This problem is aggravated by real-time constraints: the time spent deciding is a cost which must be taken into consideration. Ultimately, the best decision rules will be those that consider the tradeoff between decision quality and decisionmaking cost.

3.5. Summary of Model Requirements

Let us review the requirements fulfilled by the model we have introduced in the previous sections.

- The basic objects of interest modeled are: machines (called agents) which carry out work; work, to be carried out; and information, to be communicated and used in making decisions.
- The system's distributed nature is modeled. The model allows for a set of control points or decisionmakers (the agents). It also allows for coordination through inter-agent influences.
- The basic notion of system activity *over time* is modeled. Agents progress through a sequence of states. These states can be described, and agents can communicate their states to each other.
- Machines are modeled as both autonomous and cooperative. Each agent is capable of controlling itself. Although it has direct control over itself, it can influence remote agents. Furthermore, agents cooperate by allowing other agents to influence them.
- Utility, the quantification of *preferences* for the various courses of action, is modeled. States and sequences of states can be ordered from least to most preferable, and the degree of preference of each can be described quantitatively.

- Objectives for agents to attain are modeled. This involves maximizing expected utility, and dealing with the problems of delays in inter-agent influencing, i.e., in communication of information and transfer of work.

The model focuses solely on how to achieve cooperation efficiently, assuming a willingness to cooperate amongst agents. We note again that it does not include the possibility that they may be adversaries, or that agents behave irrationally.

CHAPTER 4

PRINCIPLES AND TECHNIQUES

In this chapter, we present a number of principles to guide the design of large decentralized control systems, and we describe a variety of general techniques (i.e., problem-solving methods based on our design principles), many from different disciplines, aimed at attacking the problems posed by decentralized control. These principles and techniques are presented in broad high-level terms, and their implementation will depend greatly on the application at hand. In the following chapters, we look at an application of these principles and techniques to a relevant decentralized control problem facing operating system researchers and designers today, namely that of *load balancing.*

The reader should be convinced by now that decentralized control poses formidable problems where the search for "perfect" solutions may simply be futile. But, of course, this should not stop us from searching for *approximate* solutions, solutions which come close to achieving what perfect solutions would provide. After all, doing *nothing* (e.g., opting for centralized control solely because decentralized control seems too hard) may be much worse than adopting an approximate solution to the decentralized control problem. The central point of this research is that we can do better than simply giving up. The question is, how?

We should first realize that other disciplines have had to deal with similar problems. Perhaps something could be learned from them. It should be encouraging to observe that decentralized control systems, such as those where the agents are humans, work, and work reasonably well; what is the secret of their success?

Since the problems mainly stem from decisionmaking with incomplete knowledge, the following disciplines come to mind as potentially helpful: probability theory, statistical inference, decision theory, game theory, and a number of subareas within artificial intelligence which are mentioned below. Probability theory teaches us how to model stochastic behavior. Statistical inference teaches us how to make predictions about future events using knowledge of the frequency of past events, and of patterns within sequences of past events. Decision theory, along with utility theory or the theory of preferences, shows us how to make the right decisions given probabilistic information. In fact, since we are interested in *multiple* decisionmakers, the subfield of *team* decision theory gives us insight about the structure and properties of our problems. Game theory tells us how to make optimal *probabilistic* decisions (especially where optimal *deterministic* decisions do not exist), taking into account multiple decisionmakers.

Research in artificial intelligence has also given us a wealth of knowledge about using heuristics, dealing with uncertainty, searching large solution spaces guided by constraints imposed by the problem, reasoning about distributed knowledge, reasoning about the activities of multiple agents, and distributed problem solving. The development of these techniques has been driven by the desire to solve extremely hard problems, and by the recognition that these problems, although they cannot be solved perfectly, can be solved in some near-optimal fashion.

Each of the disciplines discussed above have had an impact in our choice of design principles and the problem-solving techniques based on these principles. In this chapter, we will propose seven principles:

- knowledge-based solution;
- knowledge abstraction;
- uncertainty quantification;
- directional heuristics;
- information age integration;
- frugal communication;
- SPACE/TIME randomization.

The first seven sections of this chapter are devoted to these design principles. In the eighth section, we present a framework for intelligent agent design which is based on our principles. Finally, the chapter concludes with a summary section.

4.1. Knowledge-based Solution

The first principle of designing large decentralized control systems is to construct a *knowledge-based* solution. By this, we mean the adoption of a particular philosophy, one that applies multiple pieces of *case-specific knowledge* rather than a single unified model, to solve the problem.

In general, when one sets out to solve a problem, a model of the problem is formulated, which tries to capture its essential features. That model is then analyzed, manipulated, and eventually "solved." If the model is a good one, the results obtained will also apply to the original problem, which in effect is then also solved. Of course, this assumes that a model can be developed, but this is not the always the case. Some problems are so difficult that we cannot reduce them to single sufficiently accurate models. This is generally true of real-world problems where control is decentralized amongst multiple agents.

Although a single general model cannot be devised, it is often the case that multiple special-case models, each applying to a specific set of circumstances the agents would find themselves in, can be constructed. If such a set of special-case models existed, we could solve the problem, perhaps not in the general case, but certainly in some limited set of cases. This set of special-case models, or, more generally, this

case-specific knowledge, would allow us to get at least an approximate solution, which may be much better than nothing at all. In fact, this is the idea behind *expert systems* [Haye83], which have been very successful in problem domains where there is little structure and no single concise model can be devised. (In the case of expert systems, case-specific knowledge is often in the form of rules: "under the circumstance x, deduce y.")

In our case, we make a great effort to identify any case-specific knowledge about the specific decentralized control problem under investigation, and then use that knowledge to make better control decisions. What kind of case-specific knowledge might this be?

Consider a decentralized control system's activity in terms of state sequence realizations: $x(t)$, $x(t+1)$, $\cdots$, $x(t+T)$. Such a sequence would characterize the system's activity in the time interval $[t, t+T]$. If we were modeling a real system, we would generally expect there to be perceivable dependencies between successive states. For example, if the system is in state $x(t) = \mathbf{x}$, we would expect the probability that the next state is $x(t+1)$ to depend on this state $\mathbf{x}$. Although a single distribution which does not change over time may not exist for $(x(t+1) | x(t))$, we might be able to identify a number of conditional distributions, each one applying under a set of special conditions.

If the system can recognize these special conditions and therefore know which conditional distribution to apply, then it can make statistical inferences about the current state, or even future states, given the knowledge of a past state. Consequently, knowledge which is specific to a particular system and to a particular situation, such as the structure of time-domain state dependencies indicated above, would be used to make educated guesses and predictions about the current system and its situation.

Case-specific knowledge of the structure of *space-domain* as well as time-domain state dependencies is of value. Some states are *close* to each other in the sense that they form a group in which they all share a set of interesting features (e.g., their utilities are nearly the same). It would be useful if such groups of states could be identified. This knowledge would allow the system to abstract a large number of low-level states, perhaps too large to manage, into a small number of more meaningful and manageable high-level states. This reduction could have a great impact on the efficiency of decisionmaking, especially in decisions which are based on what *possible* states the system is in, as we will see later. This reduction would also have a great impact on communication efficiency since, communicating high-level (instead of low-level) state knowledge increases the bandwidth of information flow, leading to less frequent communication of messages.

In any knowledge-based solution, there is always the question about how the knowledge is represented. There are three types of knowledge we are interested in: *states, inter-state* relationships, and *uncertainty.* We must answer the following

questions: what aspects of the real system do states model, and how are they encoded? What do inter-state relationships model, and how are they encoded? What does uncertainty about states and their relationships mean, and how is it modeled and encoded?

As we will see in the next section, for problems of decentralized control, states will generally represent ranges of the values of system variables, which measure some relevant aspect (e.g., the amount of pending work) of each node. They may be encoded in terms of a statistic about these state variables (e.g., the average value of the range), or as abstract symbols representing higher-level concepts (e.g., the node is overloaded with work). Inter-state relationships can represent causal properties between states, or they can represent properties of similarity or dissimilarity between states. These relationships can often be conveniently encoded in terms of rules, such as

$$\textbf{if } \textit{the current state is } \mathbf{x}_{i_k} \rightarrow \textbf{then } \textit{the next state is } \mathbf{x}_{i_l}.$$

As for uncertainty, it is the topic of Section 4.3.

4.2. Knowledge Abstraction

The second design principle is *knowledge abstraction,* and is concerned with how an agent organizes knowledge so that it is most useful. By knowledge abstraction, an agent transforms low-level information into higher-level symbols which are more directly useful to the agent. The low-level information is derived from a state space as given by the model in Section 3.2,

$$\mathbf{X}_i = \{\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \cdots\}.$$

$\mathbf{X}_i$ is the state space of agent A_i. Although $\mathbf{X}_i$ is a local state space, this discussion applies to the global state space as well.

Each state in the low-level state space identifies a possible configuration of an agent at a given time instant. For example, if the agent is a computer, then its low-level state uniquely identifies a possible configuration of values in all memory locations, registers, and so on. Knowing an agent's state at this level of detail is typically unnecessary, and most likely unmanageable (as the possible number of states is extremely large).

Now define an *abstract* state space simply as a set of symbols

$$\mathbf{Y}_i = \{\mathbf{y}_{i_1}, \mathbf{y}_{i_2}, \cdots\},$$

where each element of $\mathbf{Y}_i$ is a non-empty subset of $\mathbf{X}_i$,

$$\mathbf{y}_{i_k} \subseteq \mathbf{X}_i. \tag{4.1}$$

The subscript i indicates that this is the abstract state space of agent A_i, and subscript k enumerates each state within the abstract state space. In general, an agent may maintain a number of different abstract state spaces for different purposes. We

will denote by $y_i(t)$ an abstract state variable taking on a value from the abstract state space $\mathbf{Y}_i$ at time t. (By convention, variables are printed in italic type, and constants are printed in boldface type.)

Defining an abstraction, denoted by a symbol $\mathbf{y}_{i_k}$, of a number of low-level states $\{\mathbf{x}_{i_p}, \mathbf{x}_{i_q}, \mathbf{x}_{i_r}, \cdots\}$, means that the agent finds it more *convenient* to view any of these low-level states in terms of a single abstract state $\mathbf{y}_{i_k}$. The features which distinguish each of the low-level states may be unnecessary for the functions of the agent.

Abstraction allows an agent to *compress* its knowledge into essential parts: a small number of parts whose differences are interesting to the agent. This reduction in the number of objects an agent has to keep track of is important for three reasons:

(1) the agent's *knowledge base,* the store where it keeps its knowledge (e.g., its data), and its *rule base,* the store where it keeps its rules (e.g., its code), is finite;

(2) the time to search the knowledge base and rule base increases as number of objects in it increases;

(3) communication between agents can take place at the *right* level of abstraction, that is, at whatever the agents decide is most useful (e.g., most efficient).

Points (1) and (2) have to do with the practical space and time constraints in the implementation of an agent's knowledge base and rule base. The agent's knowledge base is the store where associations of state symbols and what their values are believed to be, are kept. The agent's rule base is the store where associations of conditionals and actions are kept. The conditionals are tests about whether a state symbol has one of a number of given values (e.g., is $y_i \in \{\mathbf{y}_{i_k}, \mathbf{y}_{i_l}, \mathbf{y}_{i_m}\}$?). Very simply, the less items an agent has to manage, the less space it will need to store these items, and the less time it will use, on the average, in searching for an item. Thus, points (1) and (2) say that abstraction allows agents to make use of their own space and processing time more efficiently.

Point (3) has to do with the practical time constraints in the implementation of inter-agent *communication.* Time spent communicating (e.g., constructing a message and sending it, transmitting it over a network, receiving it, and interpreting its contents) is a cost an agent must take into account. Thus, just the compression of information due to abstraction reduces send, transmit, and receive times. Also, the receiving agent will ultimately use the information in some way; it may have to translate the information to a more abstract form. If the information was already communicated in the abstract form in which it will be used, then the time that would be lost in translating it does not have to be consumed. Often, the sending agent will already have the information needed by a remote agent in the desired abstract form, thus requiring no extra abstracting by the receiver.

Practically speaking, an agent never even considers the low-level state space $\mathbf{X}_i$, as it is much too large, and its level of detail is unnecessary. An agent's knowledge

base and rule base will only contain abstract states of current interest. In theory, since each abstract state $y_i \in \mathbf{Y}_i$ is a collection of low-level states (from (4.1)), there exists some mapping of (generally many) low-level states in X_i to each y_i. Yet it would be unreasonable in practice to implement such a mapping as its specification would be too large.

Ultimately, an agent needs to identify what the current abstract state of interest is. Rather than implementing a mapping of low-level states to abstract states to do this, the agent uses an *indicator*, denoted by $I(x_i)$, $x_i \in \mathbf{X}_i$. An indicator is some readily accessible portion of the low-level state, such as the value of a single memory location in a computer or a small set of instructions which compute a value, which will map to the abstract state that the low-level state would map to:

$$\exists g \text{ such that } x_i \in y_i \rightarrow g(I(x_i)) = y_i,\ x_i \in \mathbf{X}_i,\ y_i \in \mathbf{Y}_i$$

In a sense, the indicator *is* the abstract state identifier.

As an example, an indicator $I(x_i)$ may be a memory location storing a single integer which is a function of the machine's state. Depending on whether this integer is above or below a threshold $\mathbf{T}$, one of two abstract states, $\mathbf{y}_{i_{hi}}$, or $\mathbf{y}_{i_{lo}}$, which comprise the state space $\mathbf{Y}_i$, is *the current* abstract state of the machine. Thus, the agent's rule base would include the following two rules:

$$I(x_i) < \mathbf{T} \rightarrow y_i := \mathbf{y}_{i_{lo}}$$

$$I(x_i) \geqslant \mathbf{T} \rightarrow y_i := \mathbf{y}_{i_{hi}}$$

This operation *classifies* the low-level state information, summarized by the indicator, into a higher level of abstraction, so that it is more convenient to use. (For instance, the firing conditions of other rules may depend solely on whether $I(x_i)$ is above or below the threshold, and not on its actual value.) These types of rules could be constrained to a fixed format (e.g., check if the state is within an interval, given by a lower and upper bound), and would then be compiled for efficiency.

Note that the existence of an indicator implies that the agent can *compute* the abstract state of interest and encode it as part of the low-level state. For some abstract state definitions, however, this cannot be done efficiently. As an example, suppose we wanted to characterize the state of a very large memory holding a constantly changing number of objects, which in general are uniformly distributed throughout the memory. Suppose also that we did not have available a running count of the number of objects in existence; if we wanted to know exactly how many objects were in the memory at a given point in time, we would have to count them at that time. Let us also assume that the memory can hold a maximum of 2 million objects, and that we want to determine whether the memory is more than half filled, or less than half filled.

Consider an indicator which computes the total amount of used memory by *counting* the total number of objects. The state is then determined by comparing the

total count to the number 1 million. Counting every object though might be too time consuming. A better indicator could make use of the fact that the objects are distributed uniformly about memory, and therefore count the number of objects in a small area. The state is then determined by comparing the count to half the number of objects that could fit in the small area. From a practical standpoint, this second indicator would work well. But it would not have the property of *always* indicating the correct abstract state: more than half filled, or less than half filled.

What can we say about the second indicator? We can say that there is some probability that it indicates the correct abstract state, and that, if the probability is high, it may serve our needs. In the decentralized control systems of interest here, this is certainly the case. Recall that we want agents which make good *fast* decisions. Probabilistic indicators will allow bad decisions to be made once in a while, but these bad decisions might be tolerable as long as they occur infrequently (unless of course they could never be tolerated, such as when human lives depend on them). What is important is that these indicators allow fast determination of that abstract state.

Since agents will use indicators rather than low-level states to determine the abstract state, it is useful to view the relationships between indicator, low-level state, and abstract state a little differently. Let us say that the *correct* state is the abstract state $y_i \in \mathbf{Y}_i$ implied by indicator $I(x_i)$. Thus, in this new formulation it is the *low-level state* x_i for which there is a *probability* in its correspondence to y_i:

$$0 \leqslant P(x_i \in y_i) \leqslant 1. \tag{4.2}$$

Consequently, we call $\mathbf{Y}_i$ a *probabilistic abstraction* of $\mathbf{X}_i$. This formulation of abstract states as sets of which low-level states have a probability of membership closely parallels Zadeh's notion of *fuzzy sets* [Zade65] in that fuzzy set elements have a degree of membership. The formulations differ though in that, unlike the relationship given by (4.2), the degree of membership of a fuzzy set element is neither random nor statistical in nature. This is not to say that fuzzy set theory is not applicable to the principle of knowledge abstraction: *on the contrary,* it is extremely useful in the human classification of states, even when the states are based on our probabilistic indicators. For example, a probabilistic indicator of the busyness of a computer is its CPU job queue length. A human may describe *classes* of busyness, such as *idle, not-too-busy, busy, very-busy, overloaded,* (which collectively one might call a *fuzzy abstraction* of the low-level state-space) in terms of fuzzy sets of values of the CPU job queue length.

Probabilistic and fuzzy abstractions are very useful in systems which must deal with uncertainty, as is the case of decentralized control systems. All state information an agent has about remote agents will necessarily be uncertain in nature, since the agent cannot acquire this information instantaneously. The relevant question, though, is not whether the information is true or false (note that this question cannot be answered by the agent), but rather, "to what degree is the information true?" Note that this is really the case in all complex decisionmaking systems. For example,

in a computer operating system, a decision is never made based on exact knowledge of the actual low-level state (e.g., all the values in memory, in registers, and so on), but rather on some indicator (e.g., the average CPU queue length) which in some sense captures an interesting feature shared by a group of low-level states. The same is true in economic systems: the level of employment may be considered an indicator of the health of the economy. Of course, an increase in this indicator does not *always* mean the economy is growing, and vice-versa, but its value does provide a rough estimate of the growth. It is useful exactly because it is a concise piece of information, but has a non-negligible correlation to the underlying economic growth.

4.3. Uncertainty Quantification

The third design principle is *uncertainty quantification,* the accounting for and the quantification of underlying system uncertainties. In decentralized control systems, an agent's uncertainty about the system's state and about actions of other agents is a fundamental problem, as we have seen. The main points we made in Chapter 3 are that: first, uncertainty exists; second, it is *the* problem at the root of the difficulties in building decentralized control systems; finally, it cannot be ignored, it cannot be *assumed* to be out of the problem, it will not go away. Consequently one of our major focuses is on defining uncertainty, on how to quantify it, and on how to use it to make good control decisions.

First, what does uncertainty mean? The notion of uncertainty characterizes an agent's *beliefs* about propositions (e.g., what state the system is in) dealing with itself and its environment. Although a proposition, as an entity in itself, is either true or false, an agent may believe *to varying degrees* that it is true or false. This is because what a belief says about the state of the world, and what the state of the world actually is, need not be the same. Quantifying uncertainty simply means defining a measure of confidence for a belief.

Artificial intelligence researchers have recognized the value of qualifying information with confidence measures ever since the pioneering work of MYCIN [Shor76], which used *certainty factors* [Shor75] to express uncertainty of propositions. Today, there are many other methods in use: Bayesian probability methods [Pear86]; the Dempster-Shafer theory of belief functions [Shaf76]; fuzzy logic [Zade83]; other multi-valued logics [Gain78]. These methods differ in how uncertainty is represented, such as the point probabilities used by Bayesian methods, the intervals of uncertainty used by Dempster-Shafer theory, and the linguistic truth-values used by fuzzy logic.

We will argue that point probabilities meet our needs. As our goal is to find ways of building decentralized control systems where agents make good *fast* decisions, accounting for uncertainty will help in making good decisions, and its *implementation* will determine whether fast decisions can be made. Note that we are not necessarily interested in the development of a representation and of calculi for uncertainty; however, it is the realization that uncertainty is an integral part of decentralized control decisionmaking which is central to our work.

We choose to represent the measure of an agent's uncertainty as a *conditional probability density function* (cpdf) over the space of possible beliefs, which are about the states of remote agents. It is conditional on past state information, and on how old the information is:

$$p(y_i(t) \mid y_i(t-\tau),\tau).$$

This expression is simply the probability that the agent's abstract state is $y_i(t)$, given that the state τ time units in the past was $y_i(t-\tau)$.

For notational convenience, we will denote this probability by

$$p(y_i(t) \mid y_i(t-\tau)),$$

leaving out the second τ since the parameter $t-\tau$ implies the age of the state information, as long as the reader understands that, in general, this probability explicitly depends also on the age of the information. There will be some instances where the dependency on the information's age is not obvious; in those cases, it will be written explicitly.

An agent can use the cpdf to make decisions which generally assume knowledge of the current state. Knowledge of the cpdf also allows us to compute the *expected* utility of the current (and, by simple extension, of the future) states:

$$E[u(y_i(t)) \mid y_i(t-\tau)].$$

Note that point probabilities fit our needs for several reasons. First, probability theory and statistics are disciplines which are well established and well understood. Second, most events of interest in a distributed system occur with high frequency. Most of the cpdfs we will use can be built by an expert who has observed the system, or by the system after observing itself in real time. Although frequency data may not exist for all points (e.g., for all states), we have observed in our experiments that, if the abstract state space is properly defined, the resultant cpdfs have enough structure that these unknown points can be acquired by interpolation (or extrapolation) of other known points. The cpdfs can then be stored for efficient access as a 3-dimensional array, addressed by integers k, l, and m, where k and l are the indexes of $\mathbf{Y}_i$ selecting $\mathbf{y}_{i_k}$ and $\mathbf{y}_{i_l}$, and $m \in \{0,1,2,...,N\}$ represents an interval of time $(m\mathrm{T}, (m+1)\mathrm{T})$, where T is a fixed period over which the cpdf changes insignificantly, if at all. Expectations can also be tabulated, statically if the cpdfs are assumed not to change, or whenever the cpdfs are modified. (There is an implicit assumption that, if the cpdfs change, they change slowly in time, relative to period T. We shall address this issue shortly.)

Since one of our goals is to integrate uncertainty in decisionmaking, we can make use of decision theory and utility theory, as they are based on point probabilities (which is another advantage of using them). A central question we have to answer relates to how a scheme based on decision theory which will operate efficiently can be implemented.

4.4. Directional Heuristics

The fourth design principle is the reliance on *directional heuristics*. In our formal description of decentralized control, we defined the objective function in terms of the utility of states. An agent makes the decision (a component of the global decision) which it believes will maximize the expected utility of the next global state, or of the next sequence of global states. As we saw in Section 3.3, the objective can be formulated as:

find $d \in \mathbf{D}$ which maximizes

$$J(t+1, T+\tau) = E[u(x(t+1, t+\tau))],\ \tau \geqslant 1.$$

The reason for maximizing the *expected* utility is that agents do not generally know the next global state, or the next sequence of global states, but presumably they know which states are possible, along with their respective probabilities. Note that it is not enough for each agent to know the conditional probability density function (cpdf) of the current state of every other agent,

$$p(x_i(t) \mid x_i(t-\tau_i)),\ 1 \leqslant i \leqslant N,$$

since in general, state transitions of multiple agents are dependent. Rather, agents would have to know a joint cpdf of the form

$$p(x_1(t),\ x_2(t),\ \cdots,\ x_N(t) \mid x_1(t-\tau_1),\ x_2(t-\tau_2), \ldots,\ x_N(t-\tau_N)).$$

Constructing a solution which requires each agent to know this joint cpdf would not be at all realistic. In fact, the existence of a single cpdf valid for all times in real systems is doubtful, to say the least. And, even if agents had access to such a function, computing expected values over the global state space for a number of points in time would probably take an excessively long time. Agents need a better mechanism for achieving the objective.

In dealing with such problems, one approach is the development of an approximate solution, one which uses a *directional heuristic* to *guide* the selection of a decision. The idea is simply to find decisions which will at least tend to *increase* the expected utility of states in the near future. Thus, our very ambitious global objective is converted to something more reasonable, like

find $d_i \in D_i$ such that

$$E[u(y(t+\tau))] > E[u(y(t))],\ \tau \geqslant 1.$$

Note that the states of interest are no longer low-level global states, but abstract global states, and also that we only care about the positive *direction* of change in the expected utility of a future state. This is a common technique used in artificial intelligence applications, and is often referred to as "hillclimbing" [Wins84].

We therefore define a general heuristic for approximating the objective (in terms still not accessible to agents, but this will be corrected with further development of

these ideas). The new objective can be summarized as:

find $\gamma \in \Gamma$ that minimizes the following *stepwise loss function:*

$$L(t) = L_d(d(t),x(t)) + L_c(\Delta k(t),x(t)) + L_e(\gamma,x(t),k(t)) + L_r(s(t),x(t),d(t))$$

The strategy or decision rule γ (which belongs to the space Γ of all strategies) will be sensitive to four types of loss, given by the four terms in the loss function $L(t)$.

$L_d(d(t),x(t))$ is the loss due to decision quality degradation of $d(t)$, given state $x(t)$. For each possible value of $x(t)$, there are good decisions and there are bad ones. Ideally, the decision rule should produce a good decision for a given $x(t)$. L_d is a measure of how far off the decision made is from the best possible decision, due to uncertainty about the global state $x(t)$. Note that agents will never base their decisions on $x(t)$, but rather on an abstract state $y(t)$. A robust decision rule will still select good decisions (i.e., decisions which are close to the optimal one, thus making L_d small) even though what is believed based on $k(t)$, and what is true (i.e., $x(t) \in y(t)$ or $x(t) \notin y(t)$), may be different. How agents infer $y(t)$ from $k(t)$ will determine L_d.

$L_c(\Delta k(t),x(t))$ is the loss due to *communication overhead,* given the *change* in information $\Delta k(t)$, and given state $x(t)$. When there is no communication, i.e., $\Delta k(t) = 0$, there is no communication overhead, and therefore L_c is zero. When there is communication, there will be a change in information, i.e., $\Delta k(t) \neq 0$, and L_c will depend on how much communication took place (the magnitude of $\Delta k(t)$), and under what conditions (the value of $x(t)$). There is a subtle tradeoff between L_d and L_c: decisions depend on inter-agent influences, which correspond to work requests and information. If L_c is minimized by not communicating, then the decision function will be using out-of-date information, potentially causing bad decisions and increasing L_d. Or, L_d could be minimized by making sure that the global system state is known with high certainty by all agents through a great deal of inter-agent communication. Good decisions could then be made, but the communication overhead incurred may be unacceptable. Clearly, the goal is to find the amount of communication between these extremes which allows fairly good decisions to be made but does not create a great deal of overhead. We will address this tradeoff shortly.

$L_e(\gamma,x(t),k(t))$ is the loss due to time spent *evaluating* the decision rule. There may be many decision rules which provide very good decisions, but these decisions take an unreasonably long time to compute, as in mathematical programming or exhaustive search solutions. L_e is a function of the decision rule (thus, L_e is a functional), the information influence $k(t)$ available to it, and the state $x(t)$.

Finally, $L_r(s(t),x(t),d(t))$ is the loss due to *random* effects because of the stochastic nature of the system. The quality of decisions in distributed decisionmaking is necessarily limited by the random input stream of generated work $s(t)$, since this randomness makes the system nondeterministic. Consequently, decisions must take the unexpected into account; these decisions are ones which are designed to work well

under many situations, but are not as good as decisions based on certain information.

Note that the first two terms, L_d and L_c, are relatively *more* sensitive to system dynamics (i.e., how the global state changes over time) than the last two terms, L_e and L_r. This is because once a decision rule has been selected, its efficiency can be analyzed under worst-case or average-case conditions, and L_e will be sufficiently well characterized to allow the evaluator to know whether the decision rule is good or not. Also, L_r is totally a function of the stochastic nature of the system, something which cannot really be controlled. Consequently, focus is placed on L_e and L_r during in the *design* phase of the solution (e.g., when algorithmic efficiency is evaluated; when attempting to guard against improbable but possible worst case situations; and so on), while L_d and L_c will play a more active role in the agent's dynamic decisionmaking activity.

In summary, there are four characteristics for a good decision rule γ which seeks to minimize the total loss $L(t)$. One is that γ must provide a good decision based on reasonable indicators of $x(t)$. Furthermore, it should require little communication, to be traded off with the quality of information needed to make good decisions. It should require little computation; γ must be efficiently computable. And it should be robust, to account for the randomness of the inputs.

The next major task is to convert this notion of loss into functions which are computable by agents using the limited and uncertain knowledge they possess about the global state. Before we can do this, we must consider how to quantify the quality of decisionmaking based on aging information. This is the subject of the next section.

4.5. Information Age Integration

The fifth design principle is *information age integration.* How does an agent make the "best" decision using information which is not current? An agent's decision rule produces a decision based on the agent's current knowledge, which is based on communicated information, which in general will be old since communication cannot occur continuously or instantaneously. This simple fact tells us that knowledge *which does not quickly become outdated* is most desirable to an agent. Thus, in designing an agent's knowledge space, the following four goals should be achieved:

(1) given a decision space $\mathbf{D}_i$, the level of abstraction of states should be chosen as one which allows efficient selection of decisions from $\mathbf{D}_i$;

(2) abstract state spaces where states change *slowly* in time, relative to inter-agent communication delays, should be sought;

(3) a measure of an agent's *confidence* in these state abstractions, as a function of their "age," ought to be developed;

(4) these state abstractions and their confidence measures should be incorporated as an integral part of decisionmaking.

How to achieve these goals will usually require a careful study of the particular application at hand. We will consider each goal in the abstract here, and look at a concrete realization of these goals for a specific application in the following chapters.

4.5.1. Choosing the Right Abstract State Space

First, an abstract state space, *appropriate for the decision space,* must be defined. Consider the decision space

$$\mathbf{D}_* = \{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \ldots, \mathbf{d}_K\}.$$

For simplicity, we will assume that every agent has the same decision space $\mathbf{D}_*$. What abstract state space would give the appropriate level of state differentiation so that the selection of a decision is meaningful and efficient? In the previous section we discussed the notion of a loss due to decision quality degradation, $L_d(d(t),x(t))$. The loss L_d is a measure of the distance between the selected decision and the best possible decision. Given $x(t)$, the next-state function $f(x(t),d(t))$, and the utility function $u(x(t))$, it should be possible, in theory, to determine for every state what the best decision is. Thus, we could partition the state space $\mathbf{X}$ into K parts, each corresponding to a decision which is best given that the state is in that partition. Figure 4.1 shows an example of such a partitioning where K=5.

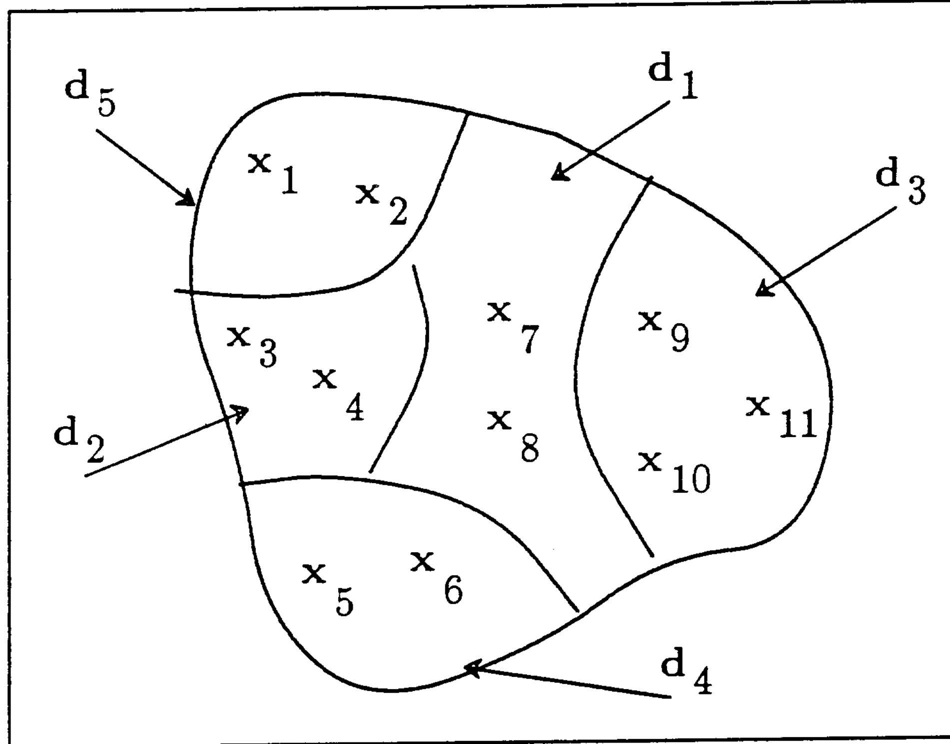


Figure 4.1. State Space Partition

In general, we can expect the state space $\mathbf{X}$ to be extremely large, not only in absolute terms, but also relative to the decision space $\mathbf{D}$. Thus, a particular decision may potentially be the best for many low-level states, and such a many-to-one mapping would eventually have to be implemented. It would be unreasonable to expect each low-level state to be listed, followed by the decision to be taken, such as in a set of rules like

$$\begin{array}{cc} \mathbf{x}_1 \rightarrow \mathbf{d}_5 & \mathbf{x}_2 \rightarrow \mathbf{d}_5 \\ \mathbf{x}_3 \rightarrow \mathbf{d}_2 & \mathbf{x}_4 \rightarrow \mathbf{d}_2 \\ \mathbf{x}_5 \rightarrow \mathbf{d}_4 & \mathbf{x}_6 \rightarrow \mathbf{d}_4 \\ \mathbf{x}_7 \rightarrow \mathbf{d}_1 & \mathbf{x}_8 \rightarrow \mathbf{d}_1 \\ \mathbf{x}_9 \rightarrow \mathbf{d}_3 & \mathbf{x}_{10} \rightarrow \mathbf{d}_3 \\ \mathbf{x}_{11} \rightarrow \mathbf{d}_3 & \end{array}$$

Even if implemented with a data structure which could be efficiently searched, the number of rules would take up too much space. More important, agents would never deal directly with low-level states; rather, they would use an indicator whose values correspond to abstract states. Therefore, what is the proper abstract space?

Very simply, the proper abstract space would be one which allowed a differentiation of low-level states similar to the partition imposed by the decision space. Thus, abstract space $\mathbf{Y}$ should have the property that, if $x \rightarrow d$, and $x \in y$, then $\forall z \in y$, $z \rightarrow d$, where $x, z \in \mathbf{X}$, $y \in \mathbf{Y}$, and $d \in \mathbf{D}$. This says that the low-level states which make up a single abstract state should all imply the same decision. Otherwise, the abstract space does not differentiate low-level states properly.

It would be ideal if such an abstract space could systematically be constructed. In practice, however, the partitioning of the low-level state space must be studied, with the goal of finding underlying similarities among the states within a partition. An indicator which captures these similarities must then be found. As we discussed in Section 4.2, a perfect indicator, one which *always* maps the low-level state to the correct abstract state, may not exist. Fortunately, our requirements are soft enough to allow indicators that *have a high probability* of selecting the correct abstract state. In the end, the indicator will *define* the abstract state space to be used. It will be a good indicator if the abstract state space it defines reasonably differentiates low-level states into groups which reflect the partitioning of the low-level state space by the decision space.

In practice, the decision space, the abstract state space, and the indicator, are often not difficult to design since our main focus is on decentralized resource control problems. The decision space is made up of decisions mainly to transfer work as well as information between agents, in such a way as to obtain a higher level of performance than if no work or information transfers were allowed, i.e., if agents were totally isolated. The abstract state space generally characterizes the agent's (for the

local space) or the system's (for the global space) capacity to do work; each state represents a different amount of *available capacity*, or, from the opposite viewpoint, *pending work*. An indicator is simply an index of the amount of pending work, such as a queue length or the utilization factor of some major resource controlled by the agent.

4.5.2. Abstract State Spaces with Slow Transition Rates

Goal (2) in Section 4.5 was to select an abstract state space where states change slowly in time, *relative to inter-agent communication times*. Decentralized control implies affecting a system's global activity at a level controllable by a *distributed* set of agents. An agent can only have an effect on a remote part of the system by (implicitly or explicitly) communicating something to it. Thus, the speed at which the system *responds* to control commands (which are the consequences of decisions) issued by agents is constrained, to a large degree, by communication times. We saw in the last section that it is the decision space that *drives* the definition of the abstract state space. The whole purpose behind obtaining state information is to make informed decisions. Therefore, if states change rapidly (i.e., multiple times during a single communication time interval between agents), then the level of activity captured by the state changes is too detailed for adequate decentralized control.

In fact, one can argue that states must change slowly, so that a communication time interval is a fraction of the time between state transitions, otherwise the communication necessary to update state information in each agent would generate excessive overhead. Of course, if we can take advantage of state transition dependencies, so that an inference about a future state can be made based on past information, this will help. But it will also help if we can design the abstract state space in such a way that state transitions occur as slowly as possible, while maintaining the property of differentiating low-level states for good decisionmaking.

One design guideline that follows from these considerations is that states should be selected so that their minimum duration is larger than the communication time interval. For example, consider the state transition sequence in Figure 4.2.

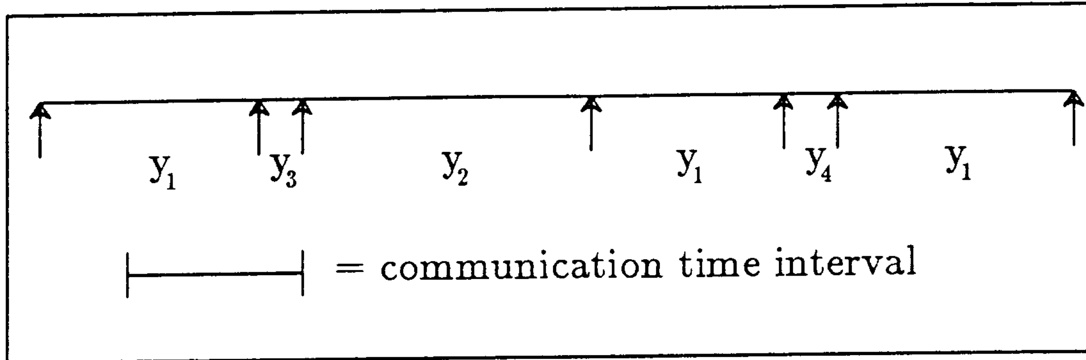


Figure 4.2. State Transition Sequence

The time spent in states $\mathbf{y}_3$ and $\mathbf{y}_4$ is significantly smaller than the communication time interval. Thus, the activity which they represent cannot be effectively controlled by a remote agent, nor can a remote agent be effectively influenced by this level of activity. One solution would be simply to disregard them, and consider the previous state still to be in effect. This is similar to applying a low-pass filter, which selectively removes high frequency components from a signal.

As previously mentioned, state spaces for decentralized control applications typically capture the notion of available capacity for doing work; for instance, each state might correspond to a different degree of pending work (the greater the amount of pending work, the less the available capacity). Say this degree was quantified by a real-valued measure $M(y_i) \in \mathrm{R}$ defined on the state space $\mathbf{Y}_i$. We can then divide time into small intervals, of equal duration T (where T is much smaller than *any* inter-state transition time), *sample* the state at the end of each such time interval, and produce a time-series of real numbers derived from the states. Time-series techniques equivalent to low-pass filters such as those that produce moving-average or autoregressive processes, can then be applied.

The N-order moving average of $M(y_i(t))$ is defined as:

$$m_i(n) = \sum_{k=0}^{N} \omega_k \cdot M(y_i((n-k) \cdot \mathrm{T})), \tag{4.3}$$

and the N-order autoregression on $M(y_i(t))$ as:

$$r_i(n) = M(y_i(n \cdot \mathrm{T})) + \sum_{k=1}^{N} \omega_k{}' \cdot r_i(n-k). \tag{4.4}$$

In (4.4), n is the time interval index, and the coefficients ω_k, $\omega_k{}'$, $0 \leqslant k \leqslant N$, are constants. $m_i(n)$ or $r_i(n)$ could then be maintained by each agent.

We can now define a *new* abstract state space $\mathbf{Y}_i'$, which corresponds to a partition of the real numbers into non-overlapping intervals, with each interval centered about the points defined by $M(y_i)$, for each $y_i \in \mathbf{Y}_i$. The abstract state implied by $m_i(n)$ or $r_i(n)$ would be the one which maps to the interval (defined by the partition) containing $m_i(n)$ or $r_i(n)$. Thus, this abstract state space $\mathbf{Y}_i'$ has a close correspondence to the original space $\mathbf{Y}_i$, and yet it has the desirable property that states of small duration are *filtered* out.

Two final points are in order. The first is that, in practice, the measure $M(y_i)$, $y_i \in \mathbf{Y}$, is unnecessary; the indicator $I(x_i)$, $x_i \in y_i$, can be conveniently used in the moving-average or autoregression formula instead. The second point is to realize that $\mathbf{Y}_i'$ is a *probabilistic* abstraction of $\mathbf{X}_i$. As long as the agent's decision rules take into account the probabilistic nature of the state information, good decisions can be made, and communication overhead is reduced.

We are now ready to explore exactly how such probabilistic information is accounted for in decisionmaking. This will allow us to achieve our goal of integrating aging information in the decisionmaking process.

4.5.3. Decisions and Utility

We have concentrated on the characteristics of the abstract state space; let us now focus our discussion on decisionmaking and utility. In decentralized resource control problems, agent decisions have to do with transferring work and information, and the abstract states have to do with an agent's capacity to accept work. Consider a decision to transfer work. By this, we mean that an agent sends to another agent a *request* to do work. This will require sending a message to communicate the request, along with any data necessary to carry out the work. The type of message and data that is communicated will depend on the application. For instance, in network routing, the message forwards an information packet, and the data is the information packet to be forwarded; in load balancing, the message is to execute a job, and the data is the job itself (or at least the name of the job, assuming the remote agent has a copy of it) plus associated data files.

In general, we must consider three aspects for a decision to transfer work:

(1) when to transfer work;

(2) what work to transfer;

(3) to which agent to transfer work.

Consideration (1) is driven either by the input (e.g., the arrival of a packet, or a job), or by a perceived change in conditions, causing an agent, which has pending work not transferred in the past, to transfer some of it (e.g., store-and-forwarding of packets, or job migration). In the first case, it is *the environment* which causes the triggering of the work transfer. But in the second case, it is the agent which must detect the change in the environment. This requires a decision of *when* the agent

should observe the environment; this decision should be judicious, since there will most likely be a cost in observing.

Consideration (2) *is* something the agent must decide, but the decision about what to transfer should not depend on the states of remote agents (assuming no special dependencies between agents and work). In fact, this consideration depends mostly on the application at hand rather than on peculiar properties or requirements of decentralized control. For example, in load balancing, where at a given point in time there are potentially many jobs within an overloaded machine to select for offloading, the job to offload may depend on the job characteristics, like the expected remaining execution time, and not on remote agent characteristics.

Consideration (3), to which agent to transfer work, is a decision which is driven by the agents themselves, and the basis for the decision, unlike (2), should depend on the states of remote agents. We will concentrate first on the *selection* decision, i.e., to which agent to transfer work. In Section 4.6, we will consider the *observation* decision, i.e., when to observe the states of remote agents (e.g., when to communicate with remote agents). As for consideration (2), it is best to deal with this in the next two chapters which focus on a load balancing application.

How do we construct a good decision rule for selecting to which agent to transfer work (which includes the possibility of keeping the work locally), given past state information about remote agents? This state information will generally have the form:

$$(y_1(t-\alpha_{1i}),\ y_2(t-\alpha_{2i}),\ \ldots,\ y_N(t-\alpha_{Ni}))$$

This is the information about remote agents that some agent A_i will have, namely $k_i(t)$, except that, rather than low-level states as defined in the model of Section 3.2, it is made up of remote agent abstract states. The value α_{ji}, $1 \leqslant j \leqslant N$, represents the *age* of the information about agent A_j's state, known by A_i. (Maintaining our convention that the ordering of subscripts, in this case j followed by i, corresponds to the *direction of influence*, then it is A_j that influences A_i concerning the age of information about itself.) α_{ji} will increase with time until a communication from agent A_j is received, at which time α_{ji} is set to the transmission time between the remote agent and the receiving agent (which can be derived either from the timestamp on the message, i.e., the time the message was transmitted by the remote agent, assuming synchronized clocks, or simply by using a precomputed expected value). Later, we will discuss when is the *best* time for this communication to take place. For now, let us concentrate solely on how to use aging state information in the remote agent selection decision.

Let

$$\Theta = \{\theta_0, \theta_1, \cdots, \theta_{10}\}$$

be the agent's abstract state space, where the higher the state subscript, the greater the degree of pending work the agent has when in that state. State θ_0 represents the

state where there is no work pending, and state θ_{10} represents the state where there is a maximal amount of work pending (and therefore the agent cannot receive any additional work). For simplicity of exposition, assume a homogeneous set of agents, so that this abstract space is the same for *all* agents (i.e., $\mathbf{Y}_i = \Theta,\ 1 \leqslant i \leqslant N$).

Given this abstract state space, and noting that each agent has state information with different ages about other agents, how is the selection decision made? The agent which must select where to transfer work could simply disregard the age of its information, and select the agent with the "best" state, the one corresponding to the least amount of pending work. This would, of course, be naive, but it serves the purpose of pointing out the pitfall of not taking age of information into account. The problem is that the state information about a remote agent, if it is very old, may have no bearing on its real current state, leading to a bad decision (i.e., the utility of the future state goes down rather than up). What we need is a quantification of *how much bearing* past information has on the current situation.

One thing that could be done is to provide every agent A_i with the set of conditional probability density functions (cpdf) $p(y_j(t) \mid y_j(t-\alpha_{ji}))$, one cpdf per value of α_{ji}, for a large range of (discrete) values. We call this set a *family* of cpdfs. (If the system of agents were not homogeneous, an agent would need a *separate* family of cpdfs for each remote agent. Also, assume for the moment that the cpdfs depend only on age α_{ji}, and not on the time t.) If an agent A_i knew a past state value $y_j(t-\alpha_{ji}) = \theta_k$, it could then determine the probability of each state in Θ being the *current* state.

At this point, it may seem reasonable for A_i to compute the *mean amount of pending work,* using the expected value of $M(y_j(t))$ given $y_j(t-\alpha_{ji})$:

$$E[M(y_j(t)) \mid y_j(t-\alpha_{ji})] = \sum_{\theta \in \mathbf{Y}_j} M(\theta) \cdot p(y_j(t) = \theta \mid y_j(t-\alpha_{ji}))$$

A_i can then compute $E[M(y_j(t)) \mid y_j(t-\alpha_{ji})]$ for all remote agents A_j, $1 \leqslant j \leqslant N$, and consider the agent with the minimum mean amount of pending work to be the optimal agent for transferring work. Unlike the previous approach, this takes aging information into account, but still has serious problems which are illustrated by the following example.

Say the family of cpdfs $p(y_j(t) \mid y_j(t-\alpha_{ji})=\theta_5)$, for $\alpha_{ji} \in \{0, 20, 50, \infty\}$, looks like that shown in Figure 4.3.

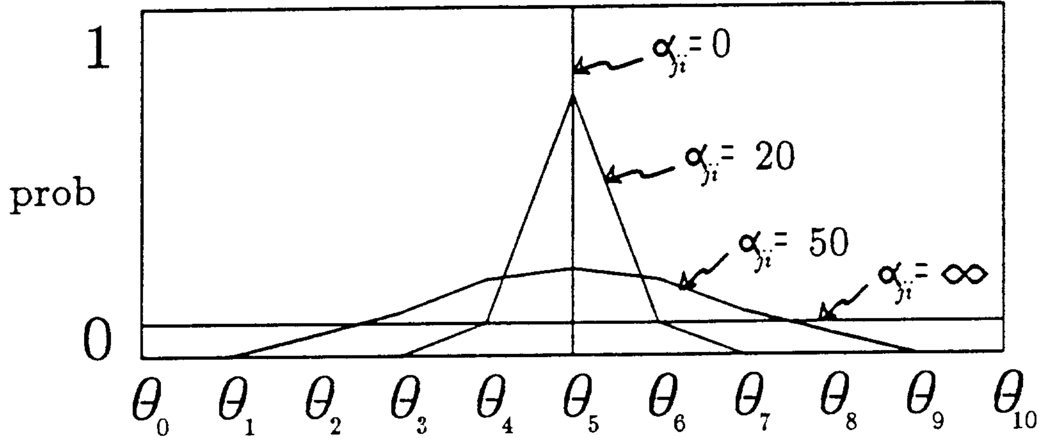


Figure 4.3. Example of Condition Probability Density Function

To ease the visualization of each of the cpdfs, they are shown as continuous functions, even though they are really discrete functions, with probability mass defined by the the height of each curve at the points $\theta_1, \ldots, \theta_{10}$. The cpdfs illustrate a reasonable hypothesis: as information ages, the number of *possible* states the remote agent can be in increases.

For the purposes of our example, the actual values of the probabilities in Figure 4.3 are not important, but the shape of each curve is. In particular, as the information age increases, the probability mass in the cpdf spreads symmetrically about $y_j(t-\alpha_{ji}) = \theta_5$. Therefore, computing the mean amount of pending work produces $M(\theta_5)$, regardless of the information's age α_{ji}. Yet, it is also a reasonable hypothesis that, say, the negative *consequences* of transferring work to a remote agent A_j, if it is in, say, state θ_{10} (which means it already has reached its maximum capacity for pending work), outweigh the positive consequences if A_j is in, say, state θ_0. Using the mean amount of pending work for agent comparison ignores this asymmetry in state utility.

What we need is a measured evaluation of the *consequences* of transferring work to an agent A_j whose state is $y_j \in \mathbf{Y}_j$. We call this real-valued quantification *the state utility of agent* A_j,

$$u_j(y_j) \in \mathrm{R}, \quad y_j \in \mathbf{Y}_j$$

Utility is a measure (defined on the agent's state space) which corresponds to the performance index to be optimized (e.g., average response time, average throughput).

For example, Figure 4.4 illustrates a likely state utility function for an agent from our previous example, whose state space is Θ.

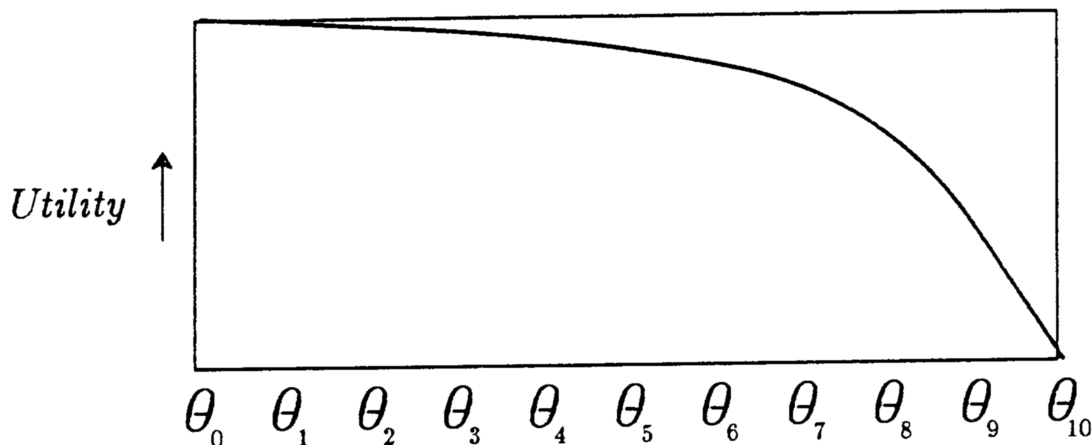


Figure 4.4. Example of State Utility Function

The utility function indicates that, as the state number increases (i.e., as the amount of pending work increases), not only does utility decrease, but the *rate* of decrease increases, meaning that the severity of the negative consequences of transferring work to an agent in state θ_k increases with k. Thus, transferring work to an agent A_j whose state is believed to be θ_5 with probability 1, or transferring work to another agent A_k whose state is believed to be either θ_0 or θ_{10}, each with probability .5, are two very different options, with the former being preferable. The concept of state utility allows us to encode this difference.

Once $u_j(y_j)$ is defined, we are still left with the problem that since an agent A_i does not know with certainty the state of agent A_j, it cannot know with certainty the state utility of A_j. But now, A_i can use the family of cpdfs to compute the *expected state utility*, given past state information,

$$E[u_j(y_j(t)) \mid y_j(t-\alpha_{ji})] = \sum_{\theta \in \mathbf{Y}_j} u_j(\theta) \cdot p(y_j(t) = \theta \mid y_j(t-\alpha_{ji})) \tag{4.5}$$

This expectation tells us two important things:

(1) how desirable remote agents are, relative to each other, as destinations of work to be transferred;

(2) how the *value* of information changes as a function of age.

Item (1) is related to the selection decision, and (2) is related to the observation decision.

We are now ready to consider the question of how $E[u_j(y_j(t)) \mid y_j(t-\alpha_{ji})]$ behaves with increasing age α_{ji}. Let us continue the discussion of our example, with $y_j(t-\alpha_{ji}) = \theta_5$. Afterwards, we will generalize. First consider the extreme values $\alpha_{ji}=0$, and α_{ji} approaching ∞. When $\alpha_{ji} = 0$, all the probability mass in the cpdf is

at θ_5; therefore, using formula (4.5), the expected utility is

$$u(\theta_5) \cdot p(y_j(t) = \theta_5 \mid y_j(t-\alpha_{ji}) = \theta_5) = u(\theta_5).$$

As α_{ji} approaches ∞, the probability mass in the cpdf is uniformly spread over each possible state. Therefore, using (4.5) again, the expected utility will be

$$\frac{1}{10}\sum_{k=0}^{10} u(\theta_k).$$

Finally, the higher the value of α_{ji}, the wider the *even* spread of the cpdf about state θ_5. Therefore, we can think of the cpdf as *selecting* equal parts of the utility function values to the left and to the right of $u(\theta_5)$; how much of this function it selects grows with α_{ji}. Since $u(\theta_{5+k})$ decreases more rapidly than $u(\theta_{5-k})$ increases (with k), we must conclude that the expected utility *decreases* when the age of state information increases, as shown in Figure 4.5.

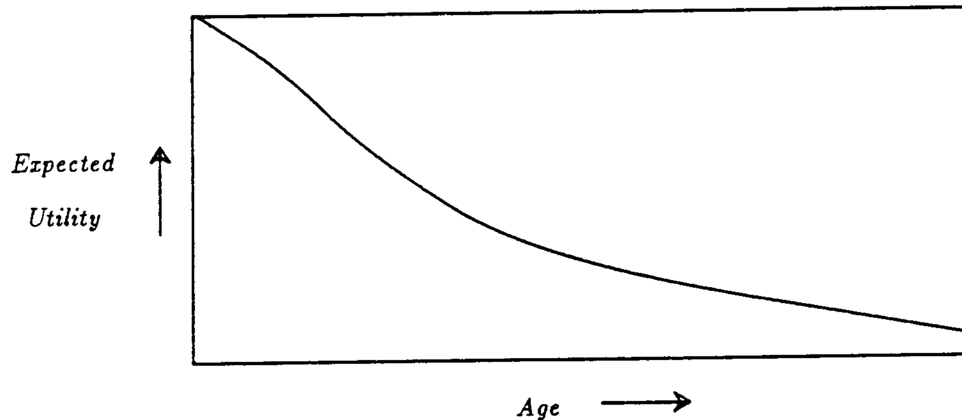


Figure 4.5. Expected Utility as Information Ages

The astute reader will have noticed that our example was chosen very carefully: the cpdf was defined as an *even* function about the given state θ_5. Say that we condition on some other state, and that the cpdf about this state is not symmetric. This may lead to situations where the expected utility *does increase* with aging information. Could it possibly make sense that the expected utility of an agent's current state should increase with the aging of the past information on which the expectation is based?

Again, let us consider an extreme value, but this time, for the past state on which the expectation is conditioned in (4.5). Suppose that agent A_i knows that agent A_j is *currently* in state θ_{10}, i.e., $y_j(t) = \theta_{10}$. Then the current expected utility is the minimal possible utility. Now consider what happens after some time α_{ji} goes by: how should A_i view agent A_j state $y_j(t+\alpha_{ji})$? A_i would reason that at worst,

$y_j(t+\alpha_{ji})$ is equal to $y_j(t) = \theta_{10}$, But it is also possible that A_j's state has changed, and since there are no worse states than θ_{10}, it could have only changed for the better. Therefore, it is reasonable that the expected utility of A_j's state has increased with time, so that at time $t+\alpha_{ji}$, it is a *statistically better* choice for A_i to transfer work to A_j than it was at time t.

If the utility function has a shape as indicated in Figure 4.4, and the family of cpdfs have shapes as indicated in Figure 4.3, the shape of the expected utility function based on (4.5) is illustrated in Figure 4.6.

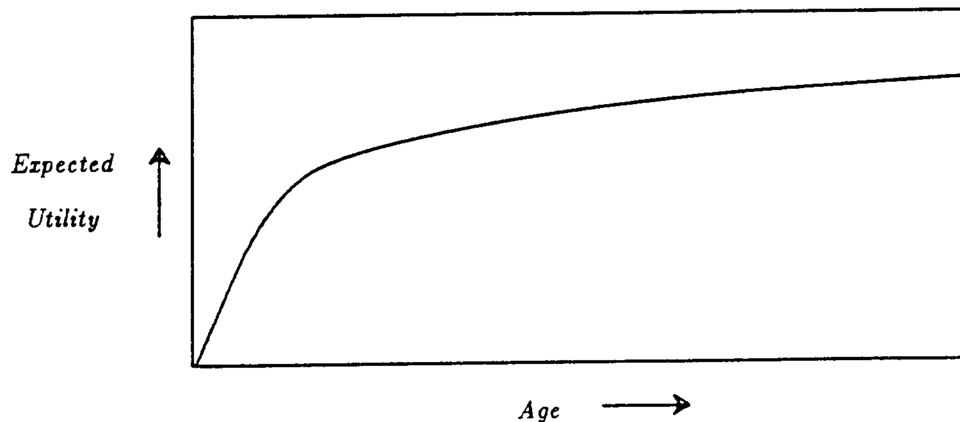


Figure 4.6. Increasing Expected Utility with Age

Note that, in both examples of the behavior of the expected utility as the age of information increases, it was implicitly assumed that the cpdf depended on age α_{ji}, but not on absolute time t. But it would be unreasonable to expect the same family of cpdfs to model the state transitions of a real system accurately *for all times*, although perhaps multiple families of cpdfs, each being an accurate model for different periods of time, could be found. If an agent could know which cpdf family is the correct model for any period of time, it could simply use the techniques presented so far. Of course, we must next answer the question: "how does an agent *know* which cpdf to use?" This will be answered in the next section, where we discuss the *observation* decision, "when should an agent observe the system?"

In summary, the expected utility of remote agent states gives an agent, who wants to transfer work, a way of comparing the merits of remote agents as possible destinations. This expected utility allowed past state information to be incorporated in an agent's decisionmaking. Although an agent may be uncertain about whether the past state information correctly reflects the current state, by quantifying this uncertainty using conditional probability density functions, and using state utility, an informed decision can be made.

4.6. Frugal Communication

To observe the state of a remote agent, an agent must obtain information from it through direct or indirect communication, which takes time. Further, communication cannot go on continuously since it contributes to overhead. Thus, any information an agent is sent about the state of a remote agent will experience a necessary and non-trivial *delay.* Delay is the key factor contributing to an agent's uncertainty about the global system state. (Of course, there may be other factors, such as noisy channels, inaccurate measurements, and so on, but these are secondary with respect to delay. In fact, while there are methods such as error correction and repetitive sending to solve these other problems which, incidentally, will tend to increase delay, we can never completely eliminate delay.)

The question then becomes: *when* does an agent communicate state information with remote agents, i.e., when does an agent observe the system? Clearly, there is a tradeoff between communicating too often, thereby causing a great deal of overhead, and communicating too infrequently, thereby making bad decisions due to out-of-date information. We shall now analyze the characteristics of this tradeoff.

4.6.1. Informal Analysis of Local Loss

For illustrative purposes, let us focus on only two distinct agents, A_i and A_j, in the distributed system; A_i is the observing agent or the *observer,* and A_j is the *observed* agent. The observer keeps track of the other's state by communication updates and by inference between updates. In particular, A_i keeps track of the last communicated value of A_j's state, and the time that value was known to be true. (A_j can send the time it recorded its state, along with the value, and for simplicity, we will assume that both agents' clocks are synchronized). When we speak of any loss function, it is of a function computed by the observer.

Recall from Section 4.4 that the loss due to degradation in decision quality was represented by the function $L_d(d(t),x(t))$, and the loss due to communication overhead was represented by the function $L_c(\Delta k(t),x(t))$. These are *global* functions: $L_d(d(t),x(t))$ is the global loss which occurs when the global (or collective) decision $d(t)$ is made and the global low-level state is $x(t)$; $L_c(\Delta k(t),x(t))$ is the global loss which occurs when there is a global change in information $\Delta k(t)$ and the global low-level state is $x(t)$. For now, we would like to focus our attention on *local* losses: the losses an agent experiences directly, due to degradation in the quality of *its own* decisions, and for the overhead it incurs due to *its own* communications. Also, rather than using the low-level state as a parameter, we will use the abstract state. The local loss functions for agent A_i will be denoted by $L_d(d_i(t),y_i(t))$, and $L_c(\Delta k_i(t),y_i(t))$. Our notation implicitly distinguishes between local and global loss functions by whether parameters are local or global variables. For example, $L_d(d_i(t),y_i(t))$ is a local loss since d_i and y_i are local variables of A_i, whereas $L_d(d(t),y(t))$ is a global loss since d and y are global variables. (In general, $L_d(d_i(t),y_i(t))$ and $L_c(\Delta k_i(t),y_i(t))$ will *not* capture the *total* local loss experienced by

A_i, since the local loss functions ignore influences by other agents. We will have to correct for this later when we generalize our analysis. For now, we will simply assume that an agent's local losses depend only on its local variables.)

These local losses are further refined by analyzing their behavior as functions of α_{ji}, the age of A_i's information about A_j, and T_{ji}, the period of communication between A_i and A_j (which can vary over time). Although α_{ji} does not explicitly appear as a parameter of $L_d(d_i(t), y_i(t))$, it is the primary variable affecting the quality of decision $d_i(t)$. The primary variable affecting $\Delta k_i(t)$ in $L_c(\Delta k_i(t), y_i(t))$ is the period of communication T_{ji}. In our analysis, we will explore the relationship between T_{ji} and α_{ji}. (We order the subscripts j followed by i in T_{ji} because it is A_j that provides information to, and therefore *influences*, A_i as to what value T_{ji} should have.)

It will be convenient to consider different representations of the loss functions L_d and L_c, with their explicit parameter being either α_{ji} or T_{ji}. We will denote the appropriate representation by superscripting either L_d or L_c with either (α) or (T). For instance, to make a statement about decision quality loss as a function of aging information, we will use the notation $L_d^{(\alpha)}(\alpha_{ji})$. Other combinations will become clear as we proceed. If we are making a *general* statement about decision quality loss or communication overhead loss, we will continue simply to use L_d or L_c. (Note that, since A_i needs to keep track of only a single remote agent, namely A_j, α_{ji} and T_{ji} appear in the loss functions as scalar values. Later, when we generalize these functions, the age parameter will be a vector α_i representing the various ages of A_i's state information about all other agents, as in the *global* loss $L_d^{(\alpha)}(\alpha_i)$, and the period parameter will be a vector T_i representing the various periods of communication between A_i and all other agents, as in the *global* loss $L_c^{(T)}(T_i)$.)

L_d depends inversely on the quality of the information used as a basis for a decision; as the information gets better, the decisionmaking gets better, and the loss L_d goes down. Since an agent's information is about the *past* states of remote agents, and this information is used to predict their current states, we can say that the quality of information, and therefore the quality of decisionmaking, decreases monotonically with the *age* of the information. Thus, the loss $L_d^{(\alpha)}(\alpha_{ji})$ *increases* monotonically with α_{ji}. $L_d^{(\alpha)}(\alpha_{ji})$ should eventually flatten as α_{ji} approaches infinity; as the age of state information gets very large, the information becomes useless, since it offers no clue about the current state. When this point is reached, further aging implies no difference in the usefulness (or uselessness) of the information.

Therefore, we may conclude that $L_d^{(\alpha)}(\alpha_{ji})$ is characterized by a curve with the following properties:

(1) it is positive and monotonically increasing;

(2) its first derivative asymptotically approaches zero.

For example, we expect $L_d^{(\alpha)}(\alpha_{ji})$ to have the general shape shown in Figure 4.7.

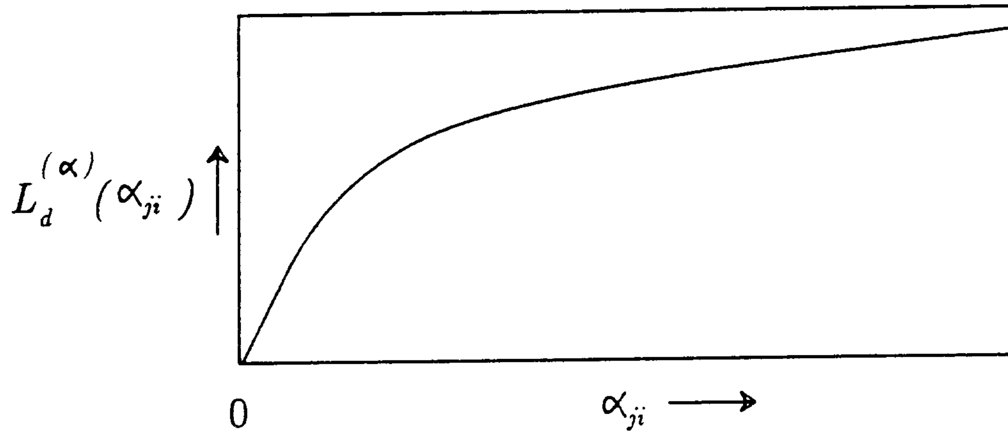


Figure 4.7. Decision Quality Loss vs. Information Age

Let us assume that A_i communicates on a periodic basis with A_j, i.e., that there is a definite inter-communication period between agents. Let this period be T_{ji}. (Later, we will relax the assumption by allowing T_{ji} to vary.) Given periodic updates, the age of information about a remote agent is a saw-toothed function of time, $age(t)$, like that shown in Figure 4.8.

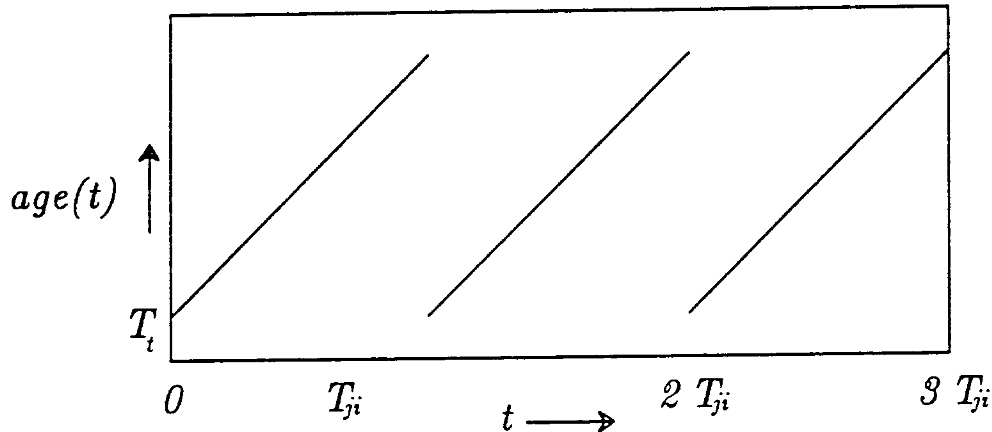


Figure 4.8. Age of Information with Periodic Communication

T_t, the transmission time, is the minimal age of information from agent A_j. $Age(t)$ increases linearly with t until receipt of new information, *replacing* the old information, causes the age to begin at T_t again.

On the basis of the assumption we made about how the decision quality loss function $L_d^{(\alpha)}(\alpha_{ji})$ varies with the age of information, we can now determine how $L_d^{(\alpha)}(age(t))$ varies in time, assuming a communication period of length T_{ji}. This is illustrated in Figure 4.9.

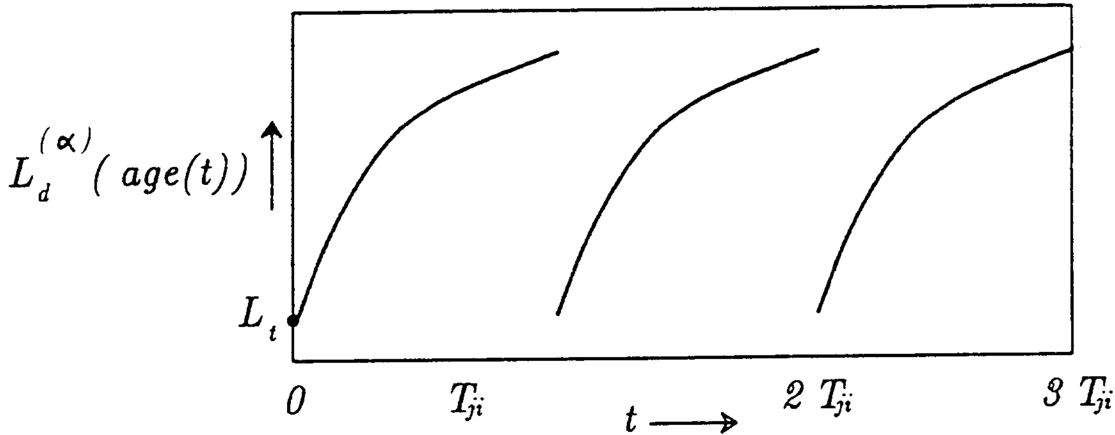


Figure 4.9. Decision Quality Loss with Periodic Communication

The quantity L_t in Figure 4.9 identifies the minimum decision quality loss. Note that, since the minimum age for information in the diagram of Figure 4.8 is T_t, the decision quality loss is at least

$$L_t = L_d^{(\alpha)}(age(n \cdot T_{ji})) = L_d^{(\alpha)}(T_t), \quad n \in \{0,1,2,...\}.$$

Thus, there will *always* be some positive loss since information is not received instantaneously.

We can now compute the decision quality loss as a function of the communication period T_{ji}, which is a time-average of $L_d^{(\alpha)}(\alpha_{ji})$, given by the formula

$$L_d^{(T)}(T_{ji}) = \frac{1}{T_{ji}} \int_{T_t}^{T_{ji}+T_t} L_d^{(\alpha)}(a)\,da. \tag{4.6}$$

The general shape of this function is similar to that of $L_d^{(\alpha)}(\alpha_{ji})$, except that it increases more slowly. The reason for this is simple: $L_d^{(\alpha)}(T_{ji})$ is the decision quality loss due to using information which is T_{ji} time units old; $L_d^{(T)}(T_{ji})$ is the average loss in the quality of decisionmaking due to the use of information which is *between* T_t *and* $T_p + T_t$ units old, with the assumption that $T_t \ll T_p$. This assumption is reasonable: the transmission time between two agents should be much less than the communication period.

Now let us consider the communication overhead loss, L_c. Communication overhead decreases monotonically with the period T_{ji}. For very small periods, we expect a large amount of overhead, and therefore a large loss. In fact, as the period approaches zero, the loss goes to infinity since there will be no time to do useful work. As the period approaches infinity, the rate of communication goes to zero, so the loss should go to zero. Therefore, we may conclude that $L_c^{(T)}(T_{ji})$ is characterized by a curve with the following properties:

(1) it is monotonically decreasing;

(2) $\lim_{\tau \to 0} L_c^{(T)}(\tau) = \infty$ and $\lim_{\tau \to \infty} L_c^{(T)}(\tau) = 0$.

For example, we expect $L_c^{(T)}(T_{ji})$ to have the general shape shown in Figure 4.10.

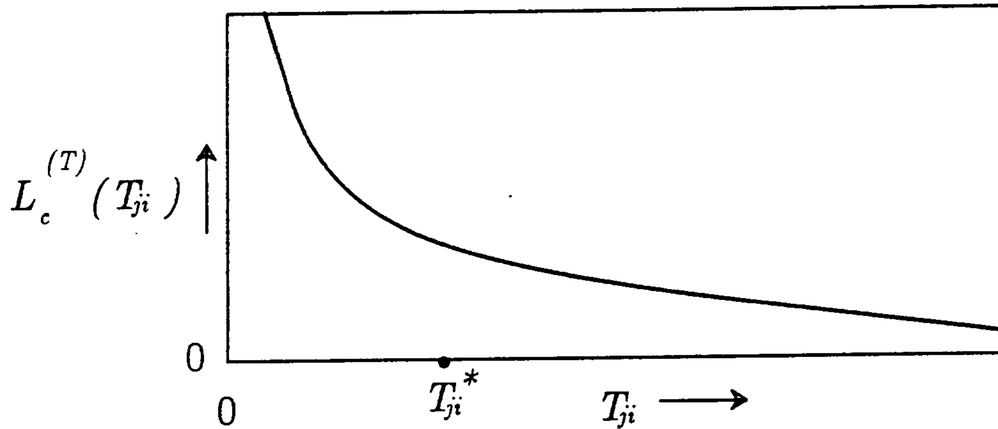


Figure 4.10. Loss due to Communication Overhead

In Figure 4.10, we have identified a specific value for the period T_{ji}, T_{ji}^*, which we refer to as the *optimal* communication period between the two agents A_i and A_j. Recall that our goal is to determine when an agent should observe the system. To do this, we need to explore how to determine T_{ji}^*.

We have two expressions, $L_d^{(T)}(T_{ji})$ and $L_c^{(T)}(T_{ji})$, which characterize the decision quality loss as a function of period T_{ji}, and the loss due to communication overhead as a function of T_{ji}, respectively. The sum of these functions gives the total loss due to degradation in decision quality and to communication overhead. The goal is then to find the minimum of this sum; the corresponding period T_{ji}^* is the optimal communication period.

What insights about the existence of T_{ji}^* can we can draw from an informal analysis of the general shapes of $L_c^{(T)}(T_{ji})$ and $L_d^{(T)}(T_{ji})$? Since we are looking to

minimize $L_c^{(T)}(T_{ji}) + L_d^{(T)}(T_{ji})$, we need to solve the equation,

$$\frac{d}{dT_{ji}}\left[L_c^{(T)}(T_{ji}) + L_d^{(T)}(T_{ji})\right] = 0. \tag{4.7}$$

If this equation has a solution (i.e., there exists a value for T_{ji} for which this equation is true), that would constitute a minimum point for $L_c^{(T)}(T_{ji}) + L_d^{(T)}(T_{ji})$. If there are multiple solutions (multiple local minima), we want the one which produces the global minimum value. To simplify our discussion, we will assume that there is only one minimum point, and therefore, there is a single solution to (4.7). (Note that if (4.7) has a solution, it must be at a minimum point, as the sum $L_c^{(T)}(T_{ji}) + L_d^{(T)}(T_{ji})$ *cannot* have a maximum point. This is because $L_c^{(T)}(T_{ji})$ is infinite when T_{ji} is 0.)

Let us consider the conditions for which (4.7) will or will not have a solution. Consider the first case:

$$\left|\frac{dL_c^{(T)}(T_{ji})}{dT_{ji}}\right| > \left|\frac{dL_d^{(T)}(T_{ji})}{dT_{ji}}\right| \qquad T_{ji} \leqslant \tau$$

$$\left|\frac{dL_c^{(T)}(T_{ji})}{dT_{ji}}\right| < \left|\frac{dL_d^{(T)}(T_{ji})}{dT_{ji}}\right| \qquad T_{ji} > \tau$$

In this case, when the communication period is below some threshold τ, the rate at which communication overhead loss decreases is greater than the rate at which decision quality loss increases. Above this threshold, the opposite is true. Under these conditions, $L_c^{(T)}(\tau) + L_d^{(T)}(\tau)$ is a minimum value, and therefore $T_{ji}^* = \tau$. This is illustrated in Figure 4.11.

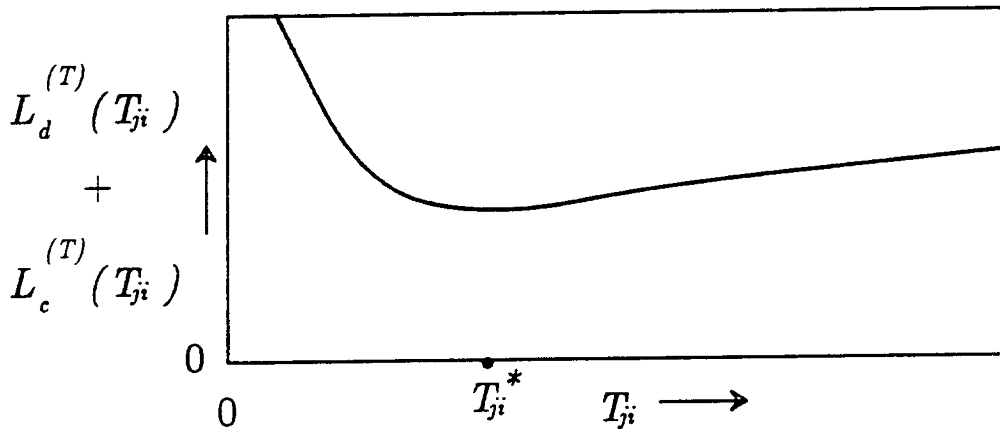


Figure 4.11. Case 1: Sum of Losses with Minimum Point

Summarizing case 1, since there is a global minimum point for the total loss, the period T^*_{ji} is the optimal communication period between A_i and A_j. That is the point where the tradeoff of degradation in decision quality due to aging information and overhead due to communication is optimized.

Now consider the second case: the sum does not have a minimum point, as shown in Figure 4.12.

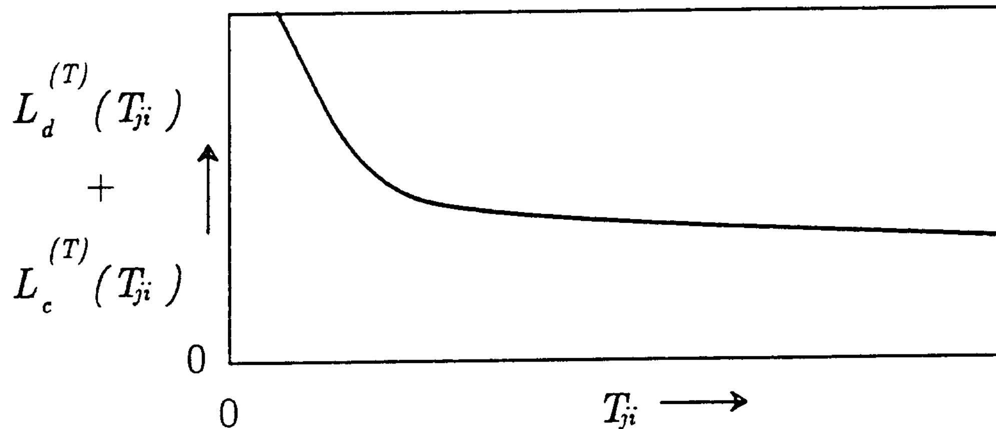


Figure 4.12. Case 2: Sum of Losses with no Minimum Point

This condition implies that, the longer the period of communication T_{ji}, the smaller the loss incurred by an agent. Therefore, in this case, it is simply better not to communicate at all!

Case 2 arises in distributed environments where states change rapidly relative to communication time. By the time an agent receives communicated state information from a remote agent, that information is useless since the state has changed many times during the transmission. The new state may depend very little on the past state which was communicated. In such a case, it makes sense not to communicate at all, since communication provides no useful information but does add overhead. Agents should instead base decisions on the limiting probability distributions of remote agent states, assuming that they exist and are known (e.g., the system's steady state behavior can be modeled and analyzed).

The more interesting situation is case 1, where there is some optimal period T^*_{ji} for communication. Note that T^*_{ji} may vary, depending on how the loss functions vary with time. Thus, for any decentralized control application, these loss functions must be determined so that the communication period T^*_{ji} can be computed dynamically.

We make the final observation that $L_d^{(T)}(T_{ji})$ is a slow changing function, as it is a time-smoothed average of $L_d^{(\alpha)}(\alpha_{ji})$. Thus, the sensitivity of the optimal communication period

$$T_{ji}^{*} = \min_{T_{ji}}[L_d^{(T)}(T_{ji}) + L_c^{(T)}(T_{ji})]$$

to variations in $L_d^{(T)}(T_{ji})$ over relatively short time intervals is small. The point is that we can approximate $L_d^{(\alpha)}(\alpha_{ji})$ and know that, as long as the best communication period $\tilde{T}_{ji}$, as produced by this approximation, is close to the theoretical optimal value T_{ji}^{*}, the actual difference between the loss using period $\tilde{T}_{ji}$ and the loss using period T_{ji}^{*}, will be small.

Up to this point, we have considered the tradeoff between decision quality and communication overhead in general terms to gain insight about when communication between agents should take place. We now analyze the loss functions themselves in detail, first formalizing them, identifying their parameters, and then considering how they may be evaluated efficiently through the use of approximations.

4.6.2. Decision Quality Loss Function

We begin by considering the *global* decision quality loss function. Our goal is to develop a formula for it, using the formalism presented in Chapter 3. This formula will be complete, but unusable due to its complexity. We then bridge the gap between theory and practice by providing a much simpler approximate formula for the local loss.

For any global state $x(t)$, there is a best collective decision $d^{*}(t)$ that will optimize some global objective function. But agent A_i does not know $x(t)$; rather, it has its own view of the global state, which is $k_i(t)$. Therefore, A_i makes decision $d_i(t) = \gamma_i(z_i(t), s_i(t))$ (see Section 3.2 for definitions of these variables), which is part of the collective decision $d(t) = (d_1(t), \ldots, d_N(t))$. To compare the quality of decision-making, it would be desirable to quantify the *goodness* of decision $d(t)$ relative to the best possible $d^{*}(t)$. One way of doing this is to evaluate and compare the *consequences* of these two decisions.

Given state $x(t)$, decision $d(t)$ causes the next state $x(t+1) = f(x(t), d(t))$, and decision $d^{*}(t)$ causes the next state $x^{*}(t+1) = f(x(t), d^{*}(t))$: the difference between the consequences of $d(t)$ and of $d^{*}(t)$ may be defined as the expected difference *in the utilities* of the respective next states,

$$E[u(x(t+1)) - u(x^{*}(t+1))],$$

or as the expected difference in the utilities of the respective next sequence of states,

$$E[u(x(t+1, t+\tau)) - u(x^{*}(t+1, t+\tau))], \quad \tau > 1.$$

This difference is the *loss due to degradation in decisionmaking quality,* and is due to agents not knowing the global state $x(t)$, and having different, possibly conflicting,

views of the global state.

Although this definition of loss makes sense, no agent can compute it, since it knows neither the current global state, nor the current actions performed by the other agents. Note that we did assume that every agent knows the decision rules γ_j for all j, which is reasonable in systems where agents are willing to cooperate, i.e., in the systems of interest here.

What agents can do is to compute *expected* losses over all possible global states, influences, and private inputs, conditioned on the information they have. To develop a formula for this, we need some additional notation. Recall from Section 3.2 that a decision d_i is based on the decision rule $\gamma_i(z_i, s_i)$, where z_i is the influence of other agents on A_i, and s_i is A_i's generated work. Influence has two components: information influence and work influence. A_i's information influence variable k_i, which contains information about the states of other agents, takes on values from the global state space,

$$\mathbf{X} = \mathbf{X}_1 \times \mathbf{X}_2 \times \cdots \times \mathbf{X}_N.$$

Define the *global* information influence variable k (with no subscript) which contains the information influence of every agent,

$$k = (k_1, k_2, \cdots, k_N),$$

and takes on values from the set

$$\mathbf{X}^N = \mathbf{X} \times \mathbf{X} \times \cdots \times \mathbf{X},$$

where there are N copies of $\mathbf{X}$. If x is a *particular* global state (i.e., $x \in \mathbf{X}$), then the product x^N is $x \times x \times \cdots \times x$, where there are N copies of x. Let

$$k = x^N$$

represent the situation where every agent's local information influence is set to the same global state, namely x. Similar to information influence, A_i's work influence variable w_i, which contains the transferred work from all other agents, takes on values from the global work space,

$$\mathbf{W} = \mathbf{W}_1 \times \mathbf{W}_2 \times \cdots \times \mathbf{W}_N,$$

and each $\mathbf{W}_i$ is the work space of agent A_i. Define the global work influence variable w, which contains the work influence of every agent, as

$$w = (w_1, w_2, \cdots, w_N);$$

w takes on values from the product

$$\mathbf{W}^N = \mathbf{W} \times \mathbf{W} \times \cdots \times \mathbf{W},$$

where there are N copies of $\mathbf{W}$. Finally, define the global generated work variable s, which contains the newly generated work (not the transferred work) arriving at each agent,

$$s = (s_1, s_2, \cdots, s_N),$$

and takes on values from $\mathbf{W}$.

Continuing with the development of a formula for decision quality loss, agent A_i can compute the expected maximum global utility $U^*(\alpha_i)$, defined as follows:

$$U^*(\alpha_i) = \sum_{x \in \mathbf{X}} \sum_{w \in \mathbf{W}^N} \sum_{s \in \mathbf{W}} u(f(x, \gamma((w, x^N), s)))p(x, (w, x^N), s \mid k_i)). \tag{4.8}$$

In (4.8), we are taking the expected value of the utility of the next state, over the global state space, over the global work influence space, and over the global generated-work space. Since $k = x^N$, every agent's view of the global state is the same, and the value of the global state as viewed by everyone (which, in general, is not the same as the actual global state) is x. (Note that this *does not* automatically imply that x is *common knowledge* (see Section 3.5). That x is common knowledge must be assumed separately.) For each value of x, $f(x, \gamma((w, x^N), s))$ will produce the *best* possible next state in terms of utility (since the decision is based on perfect knowledge of the global state); therefore, $U^*(\alpha_i)$ is the maximum expected utility based on k_i. Note that $k_i(t)$ will equal $(x_1(t-\alpha_{1i}),\ x_2(t-\alpha_{2i}), \ldots,\ x_N(t-\alpha_{Ni}))$, where α_{ji} is the age of the most recent communication from agent A_j to A_i. Consequently, we use the vector of ages

$$\alpha_i = (\alpha_{1i}, \alpha_{2i}, \ldots, \alpha_{Ni})$$

as the parameter to U^*.

Computation of (4.8) assumes that every agent A_i knows the conditional probability density function (cpdf) $p(x, (w, x^N), s \mid k_i)$, which, in general, is an unreasonable assumption. We will elaborate shortly on how this cpdf can be simplified to be used in a real system.

The expected maximum utility $U^*(\alpha_i)$ is then compared with $U(\alpha_i)$, simply the expected utility, defined as follows:

$$U(\alpha_i) = \sum_{x \in \mathbf{X}} \sum_{w \in \mathbf{W}^N} \sum_{\kappa \in \mathbf{X}^N} \sum_{s \in \mathbf{W}} u(f(x, \gamma((w, \kappa), s)))p(x, (w, \kappa), s \mid k_i) \tag{4.9}$$

Again, we are taking an expected value over the global state space, over the global work influence space, and over the global generated work space, *but also over the global information influence space* (κ is varied over this space). In contrast to d^*, the decision d is based on *imperfect* knowledge of the true global state. This will usually produce a suboptimal decision, and consequently a suboptimal next state in terms of utility. (More precisely, the next state will be *statistically* suboptimal, due to the stochastic nature of the distributed system. For example, it is possible that, for a particular set of inputs for each agent (e.g., new work arrivals), which cannot be predicted beforehand, what is a suboptimal decision in general, is actually optimal for this specific case.)

There are two noteworthy properties regarding the relationship between $U^*(\alpha_i)$ of (4.8) and $U(\alpha_i)$ of (4.9). The first property is that

$$U(\alpha_i) \leqslant U^*(\alpha_i),$$

and therefore $U(\alpha_i) - U^*(\alpha_i)$ is a non-positive value, indicating an expected *drop* in utility. This is because, for given values of x, w, and s, $u(f(x,\gamma((w,\kappa),s)))$ is maximum when $\kappa = x^N$, i.e., when every agent has the same global state information. Assuming the optimal case, where for all possible values of x, w, and s, the probability that $\kappa = x^N$ is 1, then the value of $U(\alpha_i)$ is maximal, and is exactly $U^*(\alpha_i)$.

The expected drop in utility is what we define as the *expected decision quality loss*

$$L_d^{(\alpha)}(\alpha_i) = U^*(\alpha_i) - U(\alpha_i). \tag{4.10}$$

This is our first approximation to the exact global decision quality loss $L_d(d(t),x(t))$.

The second property is that

$$\Big[U^*(\alpha_i) - U(\alpha_i)\Big] \textit{ increases with } \alpha_i,$$

as this difference should if it is to model decision quality loss. The reason is obvious: as age increases, the probability of an agent successfully predicting the current global state goes down, which increases the probability of making bad decisions. Indeed, the decision rule γ should be constructed so that the probability of selecting bad decisions grows as slowly as possible with the age of information.

4.6.3. Observations and Simplifications

In theory, an agent can determine the expected decision quality loss $L_d^{(\alpha)}(\alpha_i)$ using (4.10). However, this cannot be done *while maintaining our goal of fast decisionmaking.* (This is in contrast to the quantity we are trying to approximate, $L_d(d(t),x(t))$, which cannot be computed, not because it would be computationally inefficient, but because agents cannot know $x(t)$ and cannot theoretically compute $d^*(t)$.)

Recall the *complete and exact* global loss function as presented in Section 4.4:

$$L(t) = L_d(d(t),x(t)) + L_c(\Delta k(t),x(t)) + L_e(\gamma,x(t),k(t)) + L_r(s(t),x(t),d(t))$$

We have concentrated on L_d and L_c because they will change dynamically, whereas L_e will be relatively constant once the decision rule γ is defined, and similarly L_r is a fixed loss due to the inherent stochastic nature of the system. Requiring an agent to compute $L_d^{(\alpha)}(\alpha_i)$ as defined above would make L_e extremely large, and consequently the total loss would become extremely large. Since we want to minimize the total loss, we need cheap ways to determine L_d and L_c dynamically during system operation so that L_e is kept low.

To make $L_d^{(\alpha)}(\alpha_i)$ cheap to compute, and yet a reasonable approximation to $L_d(d(t),x(t))$ so that we can attain our ultimate goal of determining the vector of near-optimal communication periods $T_i^* = (T_{1i}^*, T_{2i}^*, \ldots, T_{Ni}^*)$, we will now identify the information we need, and the first order effects of L_d.

As was made clear in the informal analysis of local loss presented in Section 4.6.1, we are interested mainly in the *shape* of L_d, in particular its rate of increase relative to the rate of decrease of L_c, so that a minimum point in the sum of losses can be found. There is the underlying assumption that such a minimum point exists, otherwise it would not make sense for agents to communicate at all, contradicting the purpose of computing these losses. The major factor contributing to the shape of L_d is the cpdf $p(x(t)|\ k_i(t))$. Specifically, the rate at which the probability mass spreads from the known past state to other states provides a good indicator of the rate at which $L_d^{(\alpha)}(\alpha_i)$ rises. The faster this spread occurs, the more uncertainty there is in the current state based on past information, and therefore the faster the rise in the loss due to decisionmaking quality.

$U(\alpha_i)$ and $U^*(\alpha_i)$ in (4.8) and (4.9) depend on not only the global state variable x, but also the work and information influence variables, w and k, and the generated work s. We will argue that the effects of w and s on the shape of L_d are secondary to that of $p(x(t)|\ k_i(t))$ for *most* situations, as the *rates* at which work is transferred and new work arrives are expected to be much smaller than the inter-agent communication rate (otherwise, as we argued in Section 4.6.1, many state changes would take place within one communication period, which goes against our assumptions). Likewise, the effects of most components of the global information variable k are secondary in that most decisions are not based on the entire global state, but rather on a small set of local states, particularly that of the decisionmaking agent and some specific remote agent (e.g., if the decision is to transfer work, the state of the remote agent who is to receive the work is of utmost interest).

Yet, there are some situations where w, s, and especially k, can have a major effect on the shape of L_d. For some small collection of views which agents possess about each other, the resulting decisions based on these views might conflict to such a high degree that they will cause the system to go into very undesirable global states, i.e., those with very low utility. In fact, this may result not only because of differing views, but also from some collections of work transfers or new work arrivals. A typical example of this is when all agents happen to transfer work to a single agent, which was viewed by each one as being the most desirable agent to whom work should be transferred. The problem is that each agent does not expect that every other agent also sees this single agent as the most desirable. (Actually, this may be the result *no matter* whether agents have the same or differing views; the point is that, whatever these views were, they led to all agents finding the same single agent as the most desirable destination of work.) From a single agent's perspective, its decision to transfer work to what it considers to be the most desirable destination agent may make complete sense, except for the situation where *every other agent* comes to the *same unexpected conclusion* (i.e., unexpected by each single agent). We call such a situation a *resonance*, and will deal with it separately in Section 4.7.

4.6.4. Approximations

The purpose of the discussion above is to provide guidelines for constructing approximations based on simplifying (4.10) which is itself an approximation for $L_d(d(t), x(t))$. We now offer four simpler approximations. Each successive approximation depends on a smaller number of factors, making it easier to compute, but potentially introducing more error. Selection of the best approximation will depend on the application, and particularly on the size of the distributed system and the degree of dependence among the actions of agents.

First Approximation

The first approximation distinguishes between primary and secondary variables in (4.8) and (4.9), replacing summations over secondary variables with average values, and it considers only abstract states in $\mathbf{Y} = \mathbf{Y}_1 \times \mathbf{Y}_2 \times \cdots \times \mathbf{Y}_n$, rather than low-level states.

$$\overline{U}^*(\alpha_i) = \sum_{\psi \in \mathbf{Y}} u(\, f(\psi,\ \gamma((\overline{w},\ \psi^N)), \overline{s}))\ p(\psi |\ k_i) \tag{4.11}$$

$$\overline{U}(\alpha_i) = \sum_{\psi \in \mathbf{Y}} u(\, f(\psi,\ \gamma((\overline{w},\ k_i^N), \overline{s}))\ p(\psi |\ k_i) \tag{4.12}$$

Thus, the first approximation for $L_d^{(\alpha)}(\alpha_i)$ is,

$$L_d^{(\alpha)}(\alpha_i) \approx \overline{U}^*(\alpha(_i) - \overline{U}(\alpha_i) \tag{4.13}$$

Formulas (4.12) and (4.13) differ from (4.8) and (4.9) respectively in that all factors in the formulas deemed secondary in the discussion above, namely work influence w, information influence k, and newly generated work s, have been replaced by *expected* values. Thus, $\overline{w}$ is the expected amount of work to be transferred, which may be a static value based on an analysis of complete past histories, or a value dynamically recomputed on the basis of an analysis of recent past histories. Similarly, $\overline{s}$ is the expected amount of newly generated work. For information influence, we use ψ^N (i.e., every agent knows the same global state on which decisions are based) in (4.11), and we use k_i^N (i.e., agent A_i believes all agents share the same view, in particular the view of A_i, but this may be different from what the global state really is) in (4.12). Again, this ignores the problem of resonances where conflicting decisions are made, arising from special combinations of values for w, s, and k, but we solve this problem separately. Our main focus here is to establish a cheap way of determining the shape of $L_d^{(\alpha)}(\alpha_i)$, so that eventually a good communication period can be determined.

If the distributed system is very large, the loss approximation defined (4.13) is still too time-consuming to compute because of the range of values over which the summation variable ψ will vary. For example, if there are one hundred agents, and the size of each agent's state space is two, ψ will range over 2^{100} possible values. Thus, the approximation is useful only if the size of the distributed system is small and each agent's state space is small. Otherwise, we must simplify our formulas

further.

Second Approximation

In the second approximation, we will consider the loss due to information aging on an *agent-by-agent* basis, as opposed to the previous approximation which was on a system-wide basis and accounted for all agents simultaneously. The loss is still global, however: it represents a degradation in decision quality for the entire distributed system. In particular, this approximation is the global loss contributed by the consequences of decisions made by an agent A_i which directly affects a single remote agent A_j.

$$\overline{U}^{*}_{D_{ij}}(\alpha_i) = \frac{1}{\beta} \sum_{\delta_i \in D_{ij}} \sum_{\psi \in \Gamma_i(\delta_i)} u(\, f(\,\psi,\, \mathbf{d}_0^{i-1} \cdot \delta_i \cdot \mathbf{d}_0^{N-i}))\; p(\psi |\, k_i) \tag{4.14}$$

$$\overline{U}_{D_{ij}}(\alpha_i) = \frac{1}{\beta} \sum_{\delta_i \in D_{ij}} \sum_{\psi \in \Gamma_i(\delta_i)} u(\, f(\,\psi,\, \mathbf{d}_0^{i-1} \cdot d_i \cdot \mathbf{d}_0^{N-i}))\; p(\psi |\, k_i) \tag{4.15}$$

Thus, the second approximation for $L_d^{(\alpha)}(\alpha_i)$ is,

$$L_d^{(\alpha)}(\alpha_i) \;\approx\; \overline{U}^{*}_{D_{ij}}(\alpha_i) - \overline{U}_{D_{ij}}(\alpha_i) \tag{4.16}$$

D_{ij} is the set of all the decisions in agent A_i's decision set which are considered to have a direct effect on agent A_j. (Since these are decisions with which A_i will influence A_j, in the subscript i is followed by j.) Thus, $D_{ij} \subseteq \mathbf{D}_i$, and, if $\delta_i \in D_{ij}$, then $g_{ij}(\delta_i) \neq \mathbf{w}_0$. Recall from Section 3.2 that g_{ij} is the *work function*, mapping decisions by A_i to work w_{ji} appearing at agent A_j, and that $\mathbf{w}_0$ denotes "no work."

$\Gamma_i(\delta_i)$ is the set of all global states such that if $k_i \in \Gamma_i(\delta_i)$, A_i would make decision δ_i. Recall that a decision is a function of work transfers, state information, and generated work. Thus, from another viewpoint, Γ_i is γ_i^{-1}, the inverse of A_i's decision rule, for given values of w_i and s_i.

Finally, recall that $\mathbf{d}_0$ is the *null* decision, meaning that the agent decides to simply do nothing. Then the product $\mathbf{d}_0^{i} \cdot \delta_i \cdot \mathbf{d}_0^{N-i}$ is the set of decisions where agents A_j, for $j < i$, make the null decision, agent A_i makes decision δ_i, and agents A_k, for $k > i$, make the null decision. The product $\mathbf{d}_0^{i} \cdot d_i \cdot \mathbf{d}_0^{N-i}$ has the same meaning, except that A_i makes the usual decision d_i based on k_i, rather than δ_i which is an element of D_{ij}.

Let us analyze $\overline{U}^{*}_{D_{ij}}(\alpha_i)$ and $\overline{U}_{D_{ij}}(\alpha_i)$ in (4.14) and (4.15). First, they are *global* expected utilities since the next state function f is global (this is in contrast with the next two approximations, which make use of *local* expected utilities). Second, the expectations are only over global states that would trigger decisions affecting A_j. In fact, since the expectations are only over these states, a normalization factor $1/\beta$ is necessary, where

$$\beta = \sum_{\delta_i \in D_{ij}} \sum_{\psi \in \Gamma_i(\delta_i)} p(\psi | k_i) = Prob(\ y \in \bigcup_{\delta_i \in D_{ij}} \Gamma_i(\delta_i)).$$

β is the probability that the current global state y is a member of the set of possible states which would trigger any decision in D_{ij}.

The loss approximation defined by (4.16) is useful because it considers global decision quality loss on an agent-by-agent basis, with the assumption that a decision has a direct effect on only one remote agent. Given this, along with loss due to communication overhead on an agent-by-agent basis, the optimal communication period between each pair of agents can be determined. Thus, A_i can determine the frequency of communication between itself and every remote agent A_j, for all $j \neq i$. Also, the number of states to consider may be significantly less than the entire global state space, which was a problem with the previous approximation.

Unfortunately, (4.16) requires knowledge of the global next state function f, and of the cpdf $p(\psi | k_i)$, which are generally unavailable. Also, although the size of D_{ij} will be small, the number of global states in $\Gamma_i(\delta_i)$ may still be quite large.

Third Approximation

The third approximation considers local losses, the losses of the decisionmaker A_i and the remote agent A_j affected by the decision, rather than an overall global loss. First, we need some additional definitions.

Let $c(u_i, u_j)$ be a real-valued averaging function of u_i and u_j, which are real numbers representing local utilities of the decisionmaking agent A_i, and of some remote agent A_j. For example, $c(u_i, u_j)$ may be the arithmetic mean $(u_i + u_j)/2$, or the geometric mean $(u_i \cdot u_j)^{1/2}$, or the root mean square $(u_i^2 + u_j^2)^{1/2}$.

If ψ is a global state, let $[\psi]_j$, the jth component of ψ, be the local state of A_j corresponding to ψ. Let $\overline{w}_j$ be the expected work transferred to A_j; let $\overline{\kappa}_j$ be the expected information A_j has about the system; let $\overline{s}_j$ be the expected generated work arriving at A_j. Note that these are expected values as viewed (i.e., computed) by A_i. Most likely, they will be time-dependent, but it is assumed that they change very slowly, and can conveniently be communicated when necessary with minimal overhead.

Finally, recall that $g_{ij}(d_i)$ is the work function indicating what work will appear at A_j based on the decision d_i made by A_i. Let $\overline{w}_{ij}$ be the average work transferred from A_i to A_j, and let $\dfrac{\overline{w}_j \cdot g_{ij}(d_i)}{\overline{w}_{ij}}$ be the same as the $\overline{w}_j$ defined above, except that its ith component, the work transferred to A_j from A_i, is $g_{ij}(d_i)$. (Consequently, we are replacing $\overline{w}_{ij}$ with $g_{ij}(d_i)$ in $\overline{w}_j$.) We are now ready to introduce the third approximation.

$$\overline{u}^{*}_{D_{ij}}(\alpha_i) = \tag{4.17}$$

$$\frac{1}{\beta}\sum_{\delta_i \in D_{ij}}\sum_{\psi \in \Gamma_i(\delta_i)} c(u_i(f_i(y_i,\delta_i)),\ u_j(f_j([\psi]_j,\gamma_j((\frac{\overline{w}_j \cdot g_{ij}(\delta_i)}{\overline{w}_{ij}},\overline{\kappa}_j),\overline{s}_j)))) \ p(\psi|\ k_i),$$

$$\overline{u}_{D_{ij}}(\alpha_i) = \tag{4.18}$$

$$\frac{1}{\beta}\sum_{\delta_i \in D_{ij}}\sum_{\psi \in \Gamma_i(\delta_i)} c(u_i(f_i(y_i,d_i)),\ u_j(f_j([\psi]_j,\gamma_j((\frac{\overline{w}_j \cdot g_{ij}(d_i)}{\overline{w}_{ij}},\overline{\kappa}_j),\overline{s}_j)))) \ p(\psi|\ k_i).$$

Thus, the third approximation for $L_d^{(\alpha)}(\alpha_i)$ is,

$$L_d^{(\alpha)}(\alpha_i) \approx \overline{u}^{*}_{D_{ij}}(\alpha_i) - \overline{u}_{D_{ij}}(\alpha_i) \tag{4.19}$$

The main difference between the third approximation given by (4.19) and the second approximation given by (4.16) is that the next state functions are local rather than global in (4.17) and (4.18). Further, we consider a combination of the local state utilities of A_i and A_j, not the global state utility. Notice that, since A_i's state is known with certainty (since A_i is computing these utilities), the local current state of A_i used in the next state function f_i is y_i, A_i's true local state. Similar to the second approximation, the expectations are taken over all global states which could trigger decisions affecting A_j by A_i. Thus, the local state of A_j used in its next state function f_j is $[\psi]_j$. The second parameter of f_j is A_j's decision, computed by using its decision rule γ_j with expected values, except for the transferred work which depends on A_i's decision. This assumes that the global state ψ, which triggered A_i to make its decision, is still in effect up to the time when A_j receives any transferred work. (Receiving work does not mean that A_j has been affected by the work transfer in any significant way, at least according to the design of A_j's abstract state space, which is much coarser than its low-level state space $\mathbf{X}_j$. In fact, when A_i does get affected, it will have changed state because *it decided* to either accept the work, or to transfer it somewhere else.)

Fourth Approximation

Using (4.19) as the loss approximation still poses the problem of knowing the global state transition probabilities, $p(\psi|\ k_i)$. This leads us to the fourth approximation, which is based on the expected local state utilities of A_i, taken over all possible local states of A_j (since it is A_j's state of which decisionmaker A_i is uncertain):

$$\overset{*}{\overline{u}}_{ji}(\alpha_{ji}) = \tag{4.20}$$

$$\sum_{\psi_j \in \mathbf{Y}_j} c(u_i(f_i(y_i, d_i | \psi_j), \; u_j(f_j(\psi_j, \gamma_j((\frac{\overline{w}_j \cdot g_{ij}(d_i | \psi_j)}{\overline{w}_{ij}}, \overline{\kappa}_j), \overline{s}_j)))) \; p(\psi_j | k_{ji})$$

$$\overline{u}_{ji}(\alpha_{ji}) = \tag{4.21}$$

$$\sum_{\psi_j \in \mathbf{Y}_j} c(u_i(f_i(y_i, d_i)), \; u_j(f_j(\psi_j, \gamma_j((\frac{\overline{w}_j \cdot g_{ij}(d_i)}{\overline{w}_{ij}}, \overline{\kappa}_j), \overline{s}_j)))) \; p(\psi_j | k_{ji})$$

Thus, the fourth approximation for $L_d^{(\alpha)}(\alpha_i)$ is,

$$L_d^{(\alpha)}(\alpha_{ji}) \approx \overset{*}{\overline{u}}_{ij}(\alpha_{ji}) - \overline{u}_{ij}(\alpha_{ji}) \tag{4.22}$$

Let $(d_i | \psi_j)$ denote the decision similar to d_i given by decision rule $\gamma_i((w_i, k_i), s_i)$, except that the state information k_i has its jth component substituted with ψ_j. Thus, in (4.22) we are approximating the expected loss in utility of A_i's and A_j's next abstract states, not by considering all possible decisions which will affect A_j, but only decisions triggered by A_j's possible states. The conditional probability density of each possible state ψ_j given a past state k_{ji} is $p(\psi_j | k_{ji})$. (Recall that $k_i(t)$ equals $(y_1(t-\alpha_{1i}), ..., y_N(t-\alpha_{Ni}))$; therefore, $k_{ji}(t) = y_j(t-\alpha_{ji})$, which is used in the formula.) We use this fourth approximation in our load-balancing experiments to compute decision quality loss as a function of aging information.

This concludes our discussion about approximating the loss caused by decision quality degradation due to aging information. It is interesting to note that, in our quest to simplify L_d in order to reduce L_e, which is the loss due to evaluating the decision rule, we are effectively increasing L_r, the loss due to random effects. A reason for this is the statistical nature of our approximations, which are expectations over multiple variables. Although it is beyond the scope of this dissertation to quantify L_e and L_r, we have introduced them to focus on the tradeoff between the ease and speed of computing the decision rule, and the degree of error such a computation produces.

4.6.5. Communication Loss Function

Assume for a moment that the time devoted to the act of communicating, i.e., to the construction of messages to be transmitted and to the interpretation of messages received, is simply wasted time. Ultimately, we would like to say that, if an agent wastes a certain percentage of its time, there will be a known or measurable degradation in performance, which will manifest itself as a loss in future state utility.

Of course, although communication overhead represents a loss in processing time and therefore a decrease in utility, it is expected that the fruits of communicating with other agents (i.e., the value of new information), will cause an increase in utility larger than the decrease due to overhead. When this is the case, it pays to

communicate.

Thus, the purpose of quantifying loss due to communication overhead is to determine the benefit vs. cost tradeoff of communication. For some rate of inter-agent communication, the combined losses due to communication overhead and to the declining value of aging information are at a minimum. The goal is to find that minimum point, and to communicate at the corresponding frequency between each pair of agents.

Each agent must quantify its communication overhead as a function of its frequency of communication. If an agent A_i communicates with a remote agent A_j, let

$$F_{ji} = \textit{frequency of communication of } A_i \textit{ with } A_j.$$

In terms of intercommunication periods, let

$$T_{ji} = \frac{1}{F_{ji}}$$

be the period of communication between A_i and A_j. Define the *average* time between all communications by A_i as

$$\overline{T}_i = \frac{1}{\sum_{j=1}^{N} F_{ji}}. \tag{4.23}$$

Note the difference between $\overline{T}_i$ defined by (4.23), and T_i, which is the *vector* of intercommunication periods $(T_{1i}, T_{2i}, \ldots, T_{Ni})$ between A_i and A_j, for all j. We alluded to T_i in Section 4.6.1, and make use of it in this section. The communication overhead loss for agent A_i will be a function of $\overline{T}_i$ since it depends solely on the *total rate* of communication, without regard for the relative magnitudes of individual communication rates between A_i and any particular remote agent. Thus, the communication overhead loss is denoted by

$$L_c^{(T)}(\overline{T}_i).$$

On the other hand, the decision quality loss does depend on the relative ages of information about remote agents on which A_i will base its decisions, so the individual intercommunication periods cannot be combined, but must remain as a vector:

$$L_d^{(T)}(T_i).$$

Let $\overline{h}_i$ be the average overhead in CPU time to support a single communication between A_i and another agent. Thus, $\overline{h}_i/\overline{T}_i$ is the fraction of cycle time (on the average) spent by A_i communicating, and a necessary constraint is that $\overline{h}_i/\overline{T}_i < 1$, otherwise all time is spent communicating and nothing else gets done.

We need to determine how $\overline{h}_i/\overline{T}_i$ affects the next state function, and ultimately how it affects the expected utility of future states $E[u(x(t+1,\tau+1))]$, for some $\tau \geqslant 1$. The reason for introducing the fraction of time $\overline{h}_i/\overline{T}_i$ is that one can often

conveniently express a loss of some measure of performance (one that relates to the abstract state space, and consequently to utility) in terms of this fraction. For example, if half of an agent's time is spent communicating, its effective rate of processing tasks drops by at least a factor of two, and therefore the total time to complete a task is at least doubled.

Our objective is to find values for periods T_{ji}, for all j, which minimize

$$L_c^{(T)}(\overline{T}_i) + L_d^{(T)}(T_i).$$

Achieving this objective raises several problems, especially when the distributed system has a large number of agents. First, all the T_{ji}'s must be varied simultaneously to find their optimal values; it may be difficult finding a way of doing this efficiently. Second, the loss function $L_d^{(T)}(T_i)$ will change dynamically, depending on the rate at which the global system state changes. Third, as pointed out in Section 4.6.4, it is often the case that the best an agent can do is to compute an approximation to L_d, such as the pair-wise decision quality loss between two agents given by (4.22), from which it can compute $L_d^{(T)}(T_{ji})$ using (4.6).

In light of these formidable problems, we propose the following method for determining the update period between two agents. This solution is based on preallocating a maximum communication bandwidth between every pair of agents, and then minimizing a local rather than global loss function.

The first step is to recognize that, for small values of $\overline{T}_i$,

$$L_c^{(T)}(\overline{T}_i) \gg L_d^{(T)}(T_i).$$

From (4.23), it is possible that $\overline{T}_i$ is small even though some T_{ij} is large (this is true if there is some T_{ik}, $k \neq j$, which is small). Yet, L_c can be much greater than L_d since $L_c^{(T)}(\overline{T}_i)$ goes to infinity as $\overline{T}_i$ approaches zero, whereas, $L_d^{(T)}(T_i)$ has an upper bound for large values of any component T_{ij} of T_i (see Section 4.6.1). This simply means that when the frequency of communication is very high (with *any* remote agent), the loss due to communication overhead will dominate the loss due to decision quality degradation. For small values of $\overline{T}_i$ we will approximate the sum of L_c and L_d with just L_c, ignoring the relatively small contribution of L_d.

Next, we determine the minimum value of $\overline{T}_i$ such that $L_c^{(T)}(\overline{T}_i)$ is tolerable. We will call this value $T_{i_{MIN}}$, and the corresponding frequency

$$F_{i_{MAX}} = \frac{1}{T_{i_{MIN}}}.$$

$F_{i_{MAX}}$ corresponds to A_i's maximum bandwidth used by A_i for communicating with all remote agents (i.e., the sum of the communication frequencies between A_i and every other agent must be at most $F_{i_{MAX}}$).

This maximum bandwidth is now divided into per-agent maximum bandwidths, $F_{ji_{MAX}}$, $1 \leqslant j \leqslant N$. Of course, this distribution need not be a uniform allocation; rather,

the amount allocated for a particular remote agent will depend on the expected needs for communication with that agent. For example, an agent will expect to communicate more frequently with a neighboring agent than with a very distant agent; thus, more bandwidth should be preallocated for communication with neighboring agents.

To achieve such a distribution, we must define a measure of desirability on a per-agent basis, as determined by agent A_i. Let σ_{ji} be A_i's measure of desirability for A_j. The definition of σ_{ji} will depend on the application. For example, a definition based on proximity is,

$$\sigma_{ji} = \frac{1}{distance(A_i, A_j)},$$

where $distance(A_i, A_j)$ the expected transmission time from A_i to A_j. Or, σ_{ji} may be based on the relative computing power of the agents.

Once the σ_{ji} are determined for all j, let

$$\sigma_i = \sum_{j=1}^{N} \sigma_{ji}.$$

Now, we can simply allocate maximum per-agent bandwidths as follows:

$$F_{ji_{MAX}} = F_{i_{MAX}} \cdot \frac{\sigma_{ji}}{\sigma_i}.$$

Once the $F_{ji_{MAX}}$'s are determined, an iterative algorithm such as the following can be used to determine the T_{ji}'s.

1. for all j, let $T_{ji} = 1/F_{ji_{MAX}}$
2. for all j, in order of decreasing σ_{ji}, find $T_{ji} \leqslant 1/F_{ji_{MAX}}$ that minimizes

$$L_c^{(T)}(\overline{T}_i) + L_d^{(T)}(T_{ji}) + \sum_{k=1, k \neq j}^{N} L_d^{(T)}(T_{ik})$$

Step 1 initializes the T_{ji}'s to their minimum values. Step 2 considers each T_{ji} in order of importance (based on σ_{ji}), and may increase its value to minimize the objective function. In minimizing the objective function, only T_{ji} is allowed to vary; the T_{ik}'s are kept constant.

The T_{ji}'s are then recalculated by A_i whenever there is a change in the A_j's cpdf, $p(y_j(t) \mid y_j(t-\alpha_{ji}))$ (which A_j must explicitly communicate to A_i). There are several optimizations possible here. In step 2, a recalculation of every T_{ik} need not occur. If the change in A_j's cpdf is such that the probability mass spreads more slowly over the state space with increasing α_{ji}, then T_{ji} can be increased, and any T_{ik} which is not at its minimum value $1/F_{ik_{MAX}}$ can now decrease (to improve decision quality). Again, the order of checking each T_{ik} should be based on decreasing values of σ_{ik}. If the change in A_j's cpdf is such that the probability mass spreads more rapidly over the state space with increasing α_{ji}, then T_{ji} should be decreased, unless

it was already at its minimum value in which case no recalculation is necessary. Thus, recalculation only takes place when necessary (as opposed to doing it, say, periodically). We use static preallocation of bandwidth in the experiments described in Chapter 6.

A potential improvement with respect to static preallocation of bandwidth is always to keep in reserve some bandwidth for dynamic allocation. Thus, let

$$F_{i_{MAX}} = F_{i_{SMAX}} + F_{i_{RSRV}},$$

where $F_{i_{SMAX}}$ is the maximum bandwidth allocated statically, and $F_{i_{RSRV}}$ is reserved bandwidth allocated dynamically. Also, let

$$F_{ji_{SMAX}} = F_{i_{SMAX}} \cdot \frac{\sigma_{ji}}{\sigma_i}, \text{ and } F_{ji_{RSRV}} = F_{i_{RSRV}} \cdot \frac{\sigma_{ji}}{\sigma_i}.$$

The T_{ji}, for all j, are determined statically in a manner similar to the one described above, except that $F_{ji_{SMAX}}$ is used in step 1 rather that $F_{ji_{MAX}}$. Reserved bandwidth from $F_{ji_{RSRV}}$ is allocated in step 2 whenever T_{ji} needs to be decreased. Again, the order of allocation is based on σ_{ji}.

4.7. SPACE/TIME Randomization to Avoid Resonances

The seventh design principle is to use *SPACE/TIME randomization to avoid resonances*. In Section 3.4, we stated the second fundamental problem of decentralized control: an agent is uncertain about remote agent actions. Even if all agents followed the same decision rules, an agent could not predict the actions of a remote agent since it would not know that agent's view of the system state, nor would it not know that agent's influences. The situation where the concurrent local decisions made by all agents (which make up the global decision) are mutually conflicting, thereby causing the system to go into an undesirable global state, is called a *resonance* in the system.

The decisions that can cause resonances have to do with work transfer. If the decentralized control problem is, for example, load balancing, work transfer means offloading processes from one machine to another. If the decentralized control problem is, for instance, network routing, work transfer means sending messages from one machine, through intermediate machines, to a specific destination machine. One characteristic of large distributed systems is that, in general, there are *many* potential candidates for work to be transferred, e.g., many less-loaded machines for load balancing, many low traffic routes for network routing, simply because there are many machines and many routes to start out with. There may be a *best* candidate, but many others can be nearly as good.

To avoid resonances, a number of solutions might be proposed. For example, the agents could coordinate their actions by making agreements on how to act. This might involve taking votes, establishing contracts, and so on. In general, this may

involve several rounds of communication among the agents, which incurs a cost in time that we are generally not willing to pay.

Our solution is based on agents making decisions which take the possibility of remote agent actions into account without explicitly communicating. Given its view of the global system state, an agent can make reasonable inferences about the possible actions of remote agents. Resonances can be avoided by randomly selecting one of the many good candidates. Assuming that all agents randomize, and they will since our model of agents assumes they are cooperative and not adversarial, the chances for mutually conflicting decisions are reduced. We call this *global coordination by implicit communication* since the coordination occurs based on past state observations, inferences, and pre-established conventions programmed into the agents. Since explicit communication is minimized, the time frame for making decisions is minimized, and efficiency is likely to be higher.

4.7.1. Problem Formalization

Let us first formalize the problem, and define what a resonance is in terms of utility loss. Given it's current information $k_i(t)$ about the system state, and input $s_i(t)$ indicating newly generated work entering the system, an agent A_i must decide where this new work must be performed. Thus, there will generally be multiple work-transfer type decisions, which we will denote by a set of δ_k's, each representing an agent A_i's decision to transfer work to a particular agent A_k. This set of possible decisions will also include the *null* decision, $\mathbf{d}_0$, which means to keep the work locally (either for performing it or for possible transfer at a later time).

To select which decision is best, the expected consequences of each decision must be considered; this can be expressed as expected utilities of probable future states, or by a directional heuristic providing an expected positive change in utility. With a directional heuristic, a measure of the consequence of the decision δ_k is given by a real number Δ_k, which is the change in utility expected if agent A_i selects the decision to transfer work to A_k. Using game theory terminology, Δ_k is the decisionmaking agent's *payoff* for selecting the decision δ_k, which will have to be maximized by A_i.

A major problem with this formulation is that any realistic payoff Δ_k will depend not only on A_i's selection of δ_k, but also on the decisions possibly made by *all* other agents at that time. We have referred to this *collective* decision as $d(t)$. Thus, Δ_k generally depends on $d(t)$, with the restriction that $d_i(t) = \delta_k$. The problem is that the number of possible values $d(t)$ can take on is very large, and further that an agent must know $k(t) = (k_1(t), k_2(t), \cdots, k_N(t))$ (along with a number of other things) to compute $d(t)$. So, for all practical purposes, the true value of payoff Δ_k cannot be known.

We call Δ_k's dependence on δ_k a *direct* dependence, and the dependencies on decisions by other agents *indirect* dependencies. Consider what would happen if A_i simply ignored indirect dependencies, and computed a payoff $\tilde{\Delta}(\delta_k)$, a function based

solely on δ_k. Assuming that the indirect dependence of any one particular $d_j(t)$, $j \neq i$, on Δ_k is small, then either the combined effects of all the indirect dependencies will be negligible so that

$$\tilde{\Delta}(\delta_k) \approx \Delta_k,$$

or they will be additive or multiplicative such that the difference between $\tilde{\Delta}(\delta_k)$ and Δ_k is large. Define a *resonance* as the condition where

$$\tilde{\Delta}(\delta_k) > 0 \;\; AND \;\; \Delta_k \ll 0.$$

Thus, a resonance is where a seemingly optimal decision, based on an agent's local information, becomes a global disaster due to indirect effects. We are then faced with the problem that either an agent attempts to take into account indirect dependencies, making the loss due to evaluating the decision rule L_e large, or the indirect dependencies are ignored, risking to make the loss due to decision quality degradation L_d large.

We base our solution to this problem on an assumption about the likelihood of resonances: this assumption is that of all the possible collective decisions made by all agents at a given point in time, only a small fraction of them cause resonances. But the probability that a decision will be selected from this small fraction of decisions is high enough, and the consequent payoffs are bad enough, that care must be taken to avoid selecting such decisions.

Our solution then is to use the payoff $\tilde{\Delta}(\delta_k)$, which depends solely on the local decision but ignores resonances, and to build *into the decision rule* a method for avoiding resonances. This method must be cheap so that the decision rule is easy to evaluate.

4.7.2. The SPACE/TIME Randomization Technique

We now present a technique, which we call the *SPACE/TIME randomization*, for distributed decisionmaking, whose goal is to minimize the occurrence of resonances, and yet achieve coordination with minimal communication. With respect to work-transfers, an agent would determine which decision, given state information about remote agents (along with the associated measures of uncertainty), will produce the highest payoff. First, in defining the payoffs, use a payoff $\tilde{\Delta}(\delta_k)$ which depends solely on the decision δ_k. Then, let the decision rule produce *all* the decisions δ_p, δ_q,..., whose payoffs, $\tilde{\Delta}(\delta_p)$, $\tilde{\Delta}(\delta_q)$,..., are positive. Included also is the null decision d_0 (regardless of its payoff), which indicates no transfer of work. (Making the null decision provides an agent with a mechanism for *delaying* the transfer of work, as we will see shortly.) Thus, the set of *possible decisions* is

$$D_\delta = \{\delta_p, \delta_q, \ldots, d_0\}$$

Once D_δ is determined, one decision must be selected. Since agents are to act rationally, the best decision to select is the one which will produce the highest payoff.

But the computed payoffs are only approximations to the real payoffs in that they ignore indirect dependencies, thus increasing the probability of a resonance. A resonance will occur when a large number of work-transfer decisions (made independently of each other) cause work to go to an unexpectedly small number of agents (due to their ignored dependencies: they all select the "best" agent, which may be the same for all decisionmaking agents). To minimize this problem, a decision is *randomly* selected from D_δ.

The randomization is accomplished by building a probability distribution over the possible decision set in the following manner. Let p_k be the probability that decision δ_k with payoff $\tilde{\Delta}(\delta_k)$, is selected, for each δ_k in D_δ. Also, let p_{d_0} be the probability that the null decision is selected.

Define

$$p_k = (1 - p_{\mathrm{d}_0}) \cdot \frac{\tilde{\Delta}(\delta_k)}{\sum\limits_{\delta_j \in D_\delta - \mathrm{d}_0} \tilde{\Delta}(\delta_j)}, \quad \text{for } all\ \delta_k \in D_\delta, \tag{4.24}$$

and define

$$p_{\mathrm{d}_0} = f_{\mathrm{d}_0}(D_\delta, k_i(t)). \tag{4.25}$$

where f_{d_0} is a function which produces a probability, which will be discussed shortly.

Note that the sum of (4.24) and (4.25) indeed comprise a complete probability distribution:

$$\sum p_k + p_{\mathrm{d}_0} = 1.$$

The probability of choosing decision δ_k is proportional to the decision's payoff relative to the other decisions. Thus, the "best" agent is not always selected. Rather, one of a number of "good" agents is selected, with the idea that multiple decisionmakers spread work amongst these agents in accordance with their degree of goodness.

This works well, provided the size of D_δ is not small and the number of agents which are likely to make work-transfer decisions is not large. What if this is not the case? In this situation, we would like only a fraction of agents which desire to transfer work to actually do so. The fraction should be large enough to take advantage of agents which can accept work, but small enough to avoid a resonance. Thus, there should be a mechanism for some agents to abstain from transferring work; this is why the null decision $\mathbf{d}_0$ is part of the decision set D_δ.

Since we are seeking a solution where agents need not have explicitly to coordinate their actions to decide who should send and who should not, we resort again to randomization. An agent will select the null decision with probability p_{d_0}. It is defined above as the function f_{d_0}, whose range is [0,1], and is based on the decision-making agent's decision set D_δ and the current state information $k_i(t)$ (A_i is the

decisionmaker). Although the details of f_{d_0} are application dependent and therefore left open, we offer the following guidelines for its construction. It should decrease when the size of D_δ increases, since a larger number of possible decisions imply a higher probability of the spreading of work, and therefore a lower probability that a resonance will occur. On the other hand, it should increase as the number of agents likely to transfer work increases, as may be inferred from $k_i(t)$, since this implies a higher probability that a resonance will occur. A more complicated f_{d_0} would also account for the distribution of payoffs in the decision set (e.g., if the payoffs are all nearly equal, a resonance is less likely than if one decision has a very large payoff, and the others have very small ones), and for the degree of uncertainty of state information (e.g., the more uncertain is the state information, the harder it is to predict the probability of a resonance, and consequently a conservative approach would be to increase p_{d_0}). An example expression for f_{d_0} and how it is derived is given in Chapter 5.

We now see that, to minimize the occurrence of resonances, a decisionmaker selects its work-transfer decision by randomizing over the *space* of possible recipient agents when it selects a decision δ_k. The decisionmaker randomizes over *time* when it makes the null decision, in a sense holding onto its work, possibly to transfer it at a later time. The technique has the advantage of requiring no explicit coordination; rather, agents implicitly coordinate by inferring what they can about the global state through infrequent communication, and by acting on the common knowledge that all agents will SPACE/TIME randomize their work-transfer decisions.

4.8. Towards a Unifying Framework for Intelligent Agent Design

We now propose the organization of our principles of decentralized control design into a single unifying framework which imposes a structure on the reasoning and decisionmaking processes of each agent. In particular, the framework facilitates the management a hierarchy of beliefs and actions. We call this a framework for *intelligent agent design.*

4.8.1. Observe-Reason-Act Structure

The operation of a classical control system is generally based on the simple and logical *observe-act* loop, shown in Figure 4.13. The controller observes the environment through sensors, and then may issue a command (i.e., take action) to affect the environment; this is done repeatedly. Actions are based directly and solely on observation; consequently, this assumes that the environment is sufficiently *observable.* The notions of *observability* and *controllability* are the central concepts of classic control system design. (For a discussion of these and related concepts, see [Padu74].)

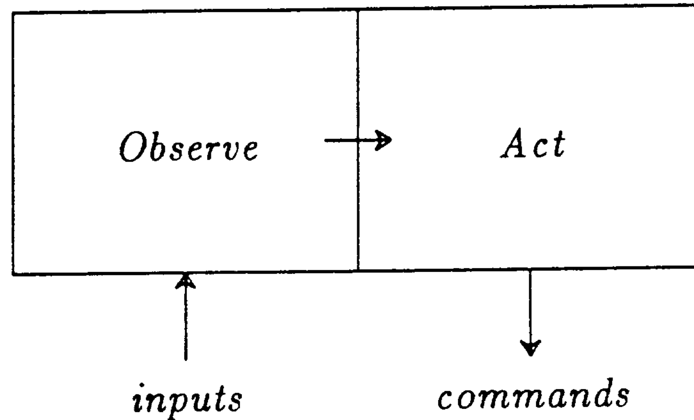


Figure 4.13. Observe-act control structure.

A decentralized control system with a large number of controllers (the agents, in our terminology) suffers from severe observability and controllability limitations. Consider again the two fundamental problems of decentralized control. Problem 1 is that no agent knows with complete certainty the current global system state, and Problem 2 is that no agent knows with complete certainty the current actions of other agents. It is interesting to note that Problem 1 has to do directly with observability, and Problem 2 with controllability; this confirms our intuition that they are *fundamental* problems.

Returning to the classical controller observe-act loop, we see that this simple framework will not be sufficient for our purposes since both global observations and global actions are uncertain. What is needed is the ability for agents to *reason* about observations and about actions.

It then becomes reasonable to extend the classical control framework to an observe-*reason*-act loop, as shown in Figure 4.14. After an agent has observed the system, it can reason about the implications of the observations concerning the system state, and then take action based on this reasoning. This reasoning is necessary since, although the agent may not be able to observe everything, it may be able to *hypothesize* (which is a reasoning process) on the basis of the observations it has been able to make. Indeed, such a process will require agents to have powerful processors and large memories, requirements which were not economically satisfiable in the past, but are today, and will be even more in the future.

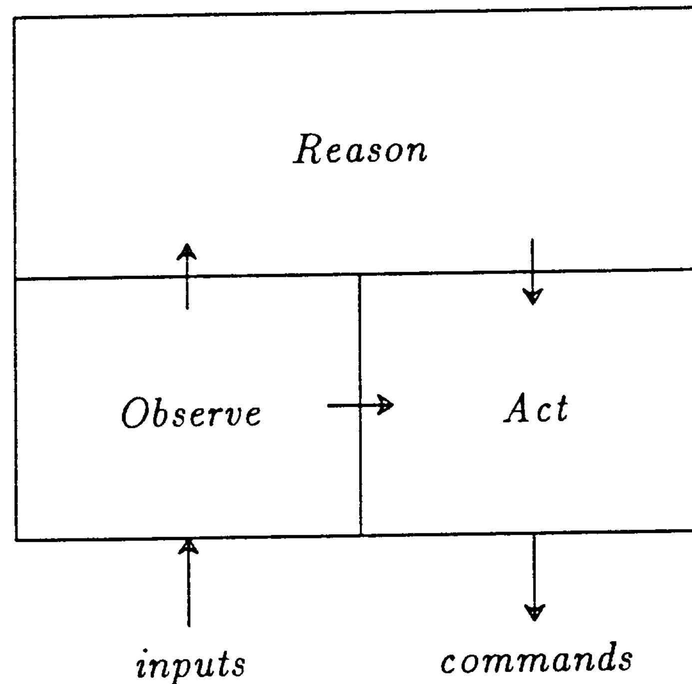


Figure 4.14. Observe-reason-act control structure.

To be sure, "reasoning" does not necessarily imply a great cost in terms of resources, but does imply a costly, time-consuming process. As efficiency is certainly a major design objective, how can we propose to perform the reasoning step, which is a potential bottleneck, in the middle of an agent's control loop? The key here is to realize that not all actions need to be based on "reasoned observations." Therefore, we do not want a hard design constraint stating that, in order for an agent to take actions, it *must* do some type of reasoning. The solution is to have a framework which allows reasoning to take place, but does not preclude a simple observe-act control path for fast reactions.

4.8.2. Architectural Framework

The framework for an agent's control architecture must capture the observe-reason-act control loop as discussed above. It must have components addressing each of the three (observe, reason, and act) processes. Further, the interfaces between these components must reflect their functional relationships. Such a framework is displayed in Figure 4.15.

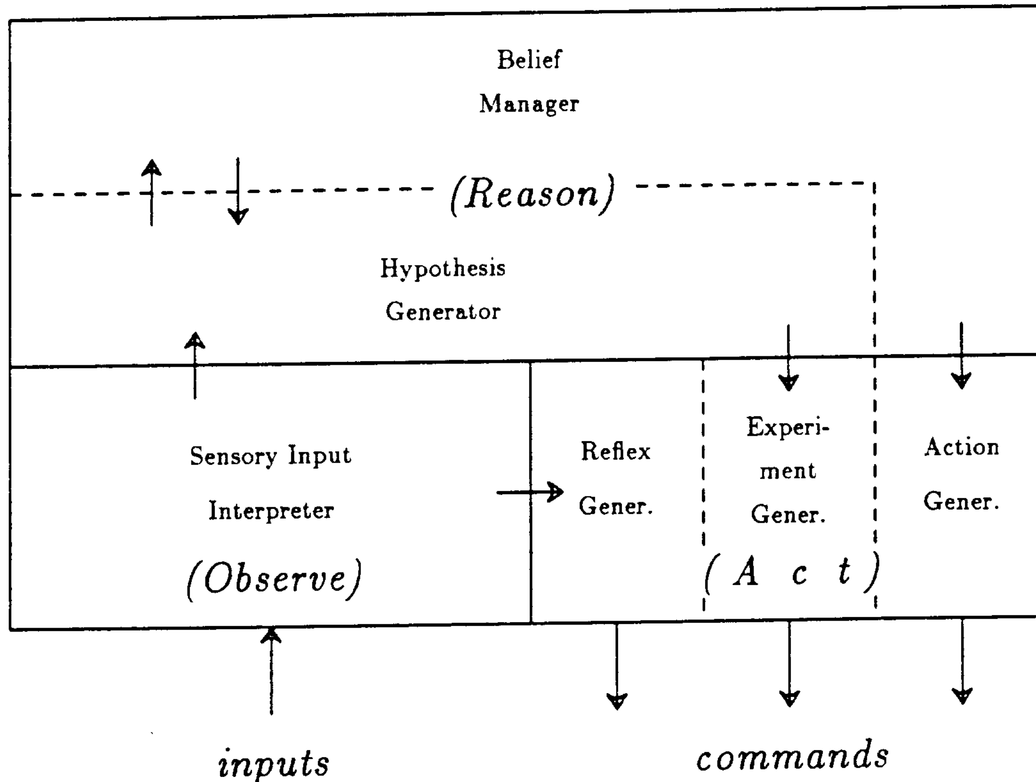


Figure 4.15. Agent framework.

Notice that the framework indeed has an observe-reason-act structure: observations enter the *Sensory Input Interpreter*, reasoning may then take place between the *Hypothesis Generator* and the *Belief Manager*, and finally actions are issued by the *Action Generator*, the *Experiment Generator*, or the *Reflex Generator*. Before going on, the word "generator" in these logical blocks is used in the sense that their main function is to produce something after a potentially elaborate processing of their inputs, similar to the way a *code generator* produces machine code based on the intermediate results of the processing of a higher-level program description. These generators may simply produce results through a simple table-lookup, or by applying a set of rules. The word "generator" is *not* meant to mislead the reader into thinking that some form of *creativity*, with all the nebulous connotations of this term, is required in these logical blocks.

4.8.3. The Observation/Interpretation Structure

The blocks called in Figure 4.15 the Sensory Input Interpreter, the Hypothesis Generator, and the Belief Manager, make up the *observation/interpretation* structure. A useful way to view these blocks is as a stratification of information inside an agent.

Raw input signals (i.e., observations) enter at the bottom, and through an interpretive process are transformed into information which rises to higher levels, representing progressively higher degrees of abstraction. What form this information takes on, and the definition of the appropriate degrees of abstraction, will depend on the application.

Abstraction is useful because an agent's observations give only partial information and indirect clues as to the real state of the distributed system. Sometimes, these observations may even be misleading (e.g., the received state information from a remote agent may not correspond to that agent's current state). Yet, this information may imply a *set* of possible states, which constitute a *single* abstract state. This limits what real states the agent needs to consider. The agent can decide to get more information to reduce further the number of possible real states. At some point, the agent's decision becomes insensitive to the distinction between these possible real states, and so it can be made without any further ado, *even though the real state is not known with complete certainty.* Thus, the production and the maintenance of this stratification of information allow actions to take place at different degrees of abstraction; actions can be triggered by changes occurring at the different levels of reasoning.

Let us now look at each logical block in the framework in more detail. The Sensory Input Interpreter accepts inputs from the environment. These inputs derive either from the agent's private source of work ($s_i(t)$), or from communications with other agents ($z_i(t)$). We use the name "sensory input interpreter" to denote the act of sensing followed by a simple act of interpretation. Although more generally the term "sensing" brings to mind probes sensing physical phenomena such as light or temperature changes, our sensing is purely information-based. (There is nothing, however, that precludes using our framework for agents which interact more directly with a physical environment). The simple interpretation of input may consist of updating counts or statistics, detecting threshold crossings, or modifying a probability that represents a measure of confidence about some piece of state information.

Up one level, the Hypothesis Generator produces hypotheses (e.g., about possible states). The reception of a new observation from the Sensory Input Interpreter, and the interpretation of this observation in light of current beliefs, will cause the production of a particular set of hypotheses, where each hypothesis has an associated probability, a measure indicating the chances that the hypothesis is true. These hypotheses *explain* the new observation in terms of its implications about the system state; in fact, these hypotheses may be regarded collectively as a single super-state comprising all the system states it implies along with a probability distribution function defined over these possible states. The probabilities are defined by the past behavior of the system (e.g., by the frequency counts of past states). Note, however, that new hypotheses do not necessarily imply that new observations have been received; in fact, the *lack* of a new observation (e.g., of an expected message from a remote agent which does not arrive) may cause the generation of new hypotheses.

Hypotheses go through a process of acceptance or rejection. This process is realized through the modification of probabilities assigned to all hypotheses. As a hypothesis gains more support, its probability is raised. Similarly, as a hypothesis loses support, its probability is lowered. When a hypothesis's probability rises above a high threshold (e.g., .8), it becomes a belief, and all competing hypotheses are rejected. A number of experimental systems have been built which use different methods for modifying credibility ratings for hypotheses based on the reception of new pieces of evidence, such as MYCIN [Shor75], Prospector [Duda76], Distributed Hearsay II [Less80], AL/X [Reit81], and SPERIL [Ishi81], to name a few.

The uppermost level of Figure 4.15 contains the Belief Manager. In it reside hypotheses which have been accepted over competing hypotheses which were eventually rejected. Beliefs must be *managed* in the sense that they must be stored in a database, and they must be mutually consistent (i.e., there can be no contradictory beliefs). Thus, every time a new belief is created, the Belief Manager must verify that it does not contradict existing beliefs. If there are contradictory beliefs, they must be removed from the database. This activity is commonly called *truth maintenance* [Doyl79].

4.8.4. Actions at Different Levels of Abstraction

The framework in Figure 4.15 provides for different types of actions at different levels of abstraction that an agent may take, based on observation and reasoning. An action at a particular level of abstraction is triggered by a change in information occurring at that same level of abstraction in the observation/interpretation structure.

At the lowest level is the Reflex Generator; it produces actions which are triggered by changes occurring in the Sensory Input Interpreter. As the name implies, reflexes are quick responses to a simple first-level interpretation of new observations, such as the detection of a threshold crossing. These actions are not based on reasoning, thus establishing a simple and fast observe-act control path.

At the next higher level is the Experiment Generator, actions which are triggered by the Hypothesis Generator. When a set of competing hypotheses are generated, it is desirable for the agent to determine which is the correct one, or, at least to reduce their number. The agent can passively wait for new information to arrive, or it can take an active role by generating a simple test designed to rule out some of the hypotheses. These tests are preprogrammed and attached to their corresponding set of competing hypotheses so that, when that set is proposed, the test is invoked. The test may cause actions at remote agents, through the sending of messages. This causes new observations (or the lack of expected observations), providing further information for modifying the probabilities of the competing hypotheses. The combination of the Hypothesis and Experiment Generator implement what is commonly called the *hypothesize-and-test* problem-solving paradigm. (See [Less80] for an application of this paradigm to the problem of distributed interpretation.)

At the highest level is the Action Generator, triggered by the Beliefs Manager. At this level, calculated actions based on the agent's reasoning capabilities take place. These are high-level heavy-weight decisions, in the sense that they have wide-ranging effects, and therefore are not to be made lightly. From the point in time particular inputs are received, an action at this level will occur after a relatively much longer time than a reflex. Although reflexes occur on a frequent basis, and their effects are quickly perceived and short-lived, the effects of high-level actions may not fully occur or be perceived until a much later time, but are expected to be long-lasting. These actions may be viewed as implementing global goals.

4.8.5. How the Framework Incorporates Our Principles

As the reader may have already noticed, our principles for constructing approximate solutions for decentralized control are present within our framework, whose purpose is to unify them.

The framework encourages a *knowledge-based solution* through the use of rules that transform information into more abstract information (i.e., control flowing up the left-hand side of the framework), or rules by which new information causes different types of actions (i.e., control flowing down the right-hand side of the framework).

The application of *knowledge abstraction* is present in the observation/interpretation structure. Information is received by the Sensory Input Interpreter, causing higher-level abstractions to be generated by the Hypothesis Generator and the Beliefs Manager.

Quantification of uncertainty initially takes place at the Sensory Input Interpreter, where an observation is given an initial measure of confidence. This measure may depend on a number of factors, such as the transmission time between sender and receiver, and the presence of noisy channels. The generation of hypotheses also involves uncertainty quantification. For example, in computing a conditional expected utility (i.e., a belief about the desirability of a remote agent), a number of possible states (i.e., hypotheses) are considered with appropriate probabilities. In other cases, it is desirable to consider the hypotheses more carefully by generating experiments to acquire more evidence before arriving at a belief. Again, this involves modifying a measure of confidence. Note that, in all cases, the principle of *integrating information aging in decisionmaking* is also involved, as the probabilities of hypotheses will generally change over time due to the system's dynamic nature.

The use of *directional heuristics* is made manifest in the reason-act part of the framework. In particular, the separation of actions into three different types is a direct consequence of the principle of using directional heuristics. Consequently, different actions can be triggered by different levels of reasoning depending on their expected tendencies (which may be positive at some levels but negative at others) to increase utility.

The principle of *frugal communication* is present in the framework's emphasis on *reasoning* about observations, and creating hypotheses and beliefs, before acting. The idea is that hypotheses, and, to a greater degree, beliefs, are valid for longer periods of time than observations; consequently, what can be inferred by hypotheses and beliefs replaces the need for extensive communication. When information becomes too uncertain, causing the generation of a large number of low-confidence hypotheses by the Hypothesis Generator, it is the Experiment Generator which causes a request for a state update from a remote agent. On the other hand, when a belief about the local state changes (which may cause a number of hypotheses indicating that other agents have incorrect information), either experiments can be generated (e.g., to inquire whether remote agents do indeed have incorrect information), or a high-level action can be generated (e.g., broadcasting the new information because of the assumption that remote agents have incorrect information).

Finally, the principle of *SPACE/TIME randomization* is used to raise the efficiency of hypothesis generation and belief management. It limits the number of hypotheses which must be considered by minimizing the possibility of mutually conflicting decisions. Since this also limits the possibility for what an agent may consider a contradiction (e.g., a locally optimal decision creates a global disaster) the rate at which truth maintenance of beliefs takes place is reduced.

4.9. Summary of Principles

In this chapter, we presented a set of seven design principles for constructing approximate solutions to decentralized control problems. The principles are as follows:

- **Adopt a knowledge-based solution**: incorporate all special-case knowledge about the problem as an integral part of the decisionmaking process.
- **Apply knowledge abstraction**: summarize information into a form which can be utilized and communicated more efficiently.
- **Quantify uncertainty**: explicitly account for information uncertainty in decisionmaking.
- **Use directional heuristics**: select decisions based on their tendencies to increase utility.
- **Integrate information aging in decisionmaking**: condition expected state utility on the age of information.
- **Communicate frugally**: communicate only when the cost of the consequences of using out-of-date information in decisionmaking exceeds cost of communication overhead.
- **Avoid resonances using SPACE/TIME randomization**: randomize over the space of good decisions and over the time during which these decisions can be made to avoid mutually conflicting decisions between agents.

We also presented a framework for the design of intelligent agents, which is based on our principles.

CHAPTER 5

DECENTRALIZED LOAD BALANCING

In this chapter, we consider the general problem of balancing the load over multiple computers. Although there are many formulations of the load-balancing problem (see [Casa88] for a survey), we will limit ourselves to ones where control is decentralized; all computers take part in making load-balancing decisions. This problem is used as a vehicle for demonstrating the application of the principles and techniques outlined in the previous chapter. We will use the formalism developed in Chapter 3 to describe the problem in more precise terms. The reader may wish to refer to the model summary in Section 3.2 during the following discussion. This chapter is limited to a discussion of applying our methods to the *general* problem of load balancing. In the next chapter, we will present results of experiments for a particular load-balancing environment.

The chapter is organized as follows. In Section 5.1, we give a brief description of the load balancing problem using the formalism developed in Chapter 3. In Section 5.2, we discuss the design of an abstract state space appropriate for load balancing and, how the decision space affects the design. In Section 5.3, we present types of domain specific knowledge which are useful. In Section 5.4, we discuss the design of state transition models. In Section 5.5, we present measures for comparing the desirabilities of agents. Finally, in Section 5.6, we discuss the rational decisionmaking process of a load-balancing agent.

5.1. Formal Description

An *agent* $A_i \in \mathbf{A}$ is a *computer system* supporting the execution of *jobs*. A job is some finite amount of *work* $w \in \mathbf{W}$. For each job it receives, an agent A_i makes a *load-balancing decision* $d_i \in \mathbf{D}_i$ to execute the job itself, or to transfer it in a network to some other agent A_j. The global objective is to minimize the average time a job is delayed during its execution due to its contention with other jobs seeking execution and due to its possible network transmission from one agent to another if offloaded. We will make this objective more precise after defining the abstract state spaces of agents.

5.2. Abstract State Space

We mentioned in Section 4.2 that, in general, an agent's low-level state space $\mathbf{X}_i$ is never used directly in decisionmaking. Rather, an abstract state space, where each state represents a (possibly large) number of low-level states, is needed. The construction of this abstract state space is driven by the needs of decisionmaking. It must

discriminate the possible situations which could trigger any of the agent's possible decisions. Thus, let us identify these possible decisions.

We distinguish between two *types* of decisions an agent will make: *problem* decisions and *communication* decisions. Problem decisions are those which have to do specifically with the particular problem at hand, in this case, load balancing. The decision to execute a job locally, or to transfer it to another agent, is a problem decision. An agent's decision rule γ_i pertains to problem decisions.

Communication decisions are those having to do with maintaining global state information for each agent. The decision to send an update of the local state to remote agents, and the decision to inquire about a remote agent's state are communication decisions. These decisions are common for agents taking part in any type of decentralized control (although parameters such as the update period will be problem specific). Thus, they are dealt with separately, using a mechanism independent of the problem decision rule γ_i.

Given the stated load-balancing decisions, an agent must be able to discriminate among the following situations. It must recognize whether it is overloaded or not, to decide whether it should execute a job locally. It must recognize whether remote agents are underloaded or not, to decide where to transfer a job if it is overloaded. Thus, we see there are at least three necessary abstract states: *overloaded, underloaded,* or neither, which will be called *normal.*

How are these abstract states recognized? Ultimately, an agent must be able to reflect on some aspect of its low-level state, and quickly determine the correct abstract state. Such a readily accessible portion of the low-level state was defined as an *indicator,* $I(x_i)$. A good indicator of load for load balancing is the *number of jobs ready for execution* (see [Ferr86] for an investigation of load indices for load balancing). Thus, we will require agents to maintain in their primary memory, for quick access, the current value of this number, which we call the *instantaneous number of ready jobs,* denoted by R_i.

The instantaneous number of ready jobs was chosen as an indicator for a number of important reasons. First, since it is a single quantity, it allows differentiation between the necessary abstract states using the following simple rules:

if $\overline{R}_i < \mathrm{R}_u$ then *underloaded*
else if $\overline{R}_i > \mathrm{R}_o$ then *overloaded*
else *normal*

R_u and R_o are experimentally determined thresholds, and $\overline{R}_i$ is a time-average over recent past values of R_i (to be discussed shortly). Second, it has relationships to communication overhead and expected job delay, two quantities whose importance will become apparent below. These relationships are easily determined, and will be used for predictive purposes. Third, a simple stochastic model describing how R_i changes over time, also to be used for prediction, can be conveniently realized. Fourth, and

most importantly, there is strong theoretical support under assumptions that are not unreasonable that R_i is a good indicator of load [Ferr86] (which is also supported by experimental evidence [Ferr88]).

As presented so far, the abstract state $y_i(t)$ could simply be defined as the value of $I(x_i(t))$, which equals $R_i(t)$; i.e., the current state could just be the current instantaneous number of ready jobs. This would not work well though in a system where R_i changes rapidly. A common scenario is that R_i fluctuates rapidly about a more slowly changing fundamental variation. (We have observed this in our experiments described in Chapter 6. Similar observations are reported in [Ferr88] and [Zhou87].) It is this fundamental variation which is of most interest.

One way of extracting this fundamental variation is to consider the sequence $R_i(t), R_i(t+T_s), R_i(t+2T_s), \cdots$, where T_s is the sampling period for $I(x_i)$. Note that $I(x_i)$ can change at the rate at which x_i, the low-level state, changes. Even if it were possible, monitoring every change in $I(x_i)$ would be unnecessary, hence the reason for sampling. T_s should be small enough that the Nyquist criterion is satisfied, i.e., that $1/T_s$ is greater than twice the base frequency of R_i. Standard Fourier analysis techniques can be used to determine this frequency. (Note that a very high base frequency would indicate a poor choice of indicator.)

If this sequence is considered as a time series $R_i(n)$, $R_i(n+1)$, $R_i(n+2), \cdots$, where n is the index of the sampling period, filtering techniques can be applied to remove very high frequency components, leaving only the fundamental components. In Section 4.5.2, we discussed the use of a moving average and autoregression techniques for filtering. We will use the following simple autoregressive model (based on (4.4)):

$$\overline{R}_i(n) = \omega \cdot \overline{R}_i(n-1) + (1-\omega) \cdot R_i(n), \tag{5.1}$$

where ω is a constant between zero and one. This has the effect of averaging samples of the time series of the instantaneous number of ready jobs over the recent past. How much weight is given to the recent past is determined by the value of ω.

Define the *number of ready jobs* $B_i(n)$, for time period n as the closest integer to $\overline{R}_i(n)$,

$$B_i(n) = ROUND(\overline{R}_i(n)) \tag{5.2}$$

Finally, the value of agent A_i's local state at time t is simply the number of ready jobs during time period n which contains t,

$$y_i(t) = B_i(n), \quad nT_s \leqslant t < n(T_s+1). \tag{5.3}$$

Therefore, the abstract state space $\mathbf{Y}_i$ is the set of non-negative integers, up to some maximum $B_{\max}$. The size of $\mathbf{Y}_i$ is limited because of the finiteness of the memory of real machines. If agents are heterogeneous, they can have different values for $B_{\max}$. We will assume a homogeneous set of agents to simplify our discussion; therefore, they will all have the same value for $B_{\max}$. Thus,

$$\mathbf{Y}_i = \{0, 1, 2, \ldots, B_{\max}\}, \text{ for } all\ i.$$

For purposes of exposition, we have so far ignored the distinction made in Section 4.5.2 between the state space $\mathbf{Y}_i$ and the *measure* $M(y_i)$, $y_i \in \mathbf{Y}_i$, defined over the same state space. We now make that distinction precise. The current abstract state $y_i(t)$ can take on, as its value, any of the *state identifiers* in $\mathbf{Y}_i$. When necessary, we will use the symbol θ to refer to one of these state identifiers. Each of these states also has a *numerical value*, defined by the measure $M(\theta)$. M simply maps state θ to the integer value it represents. When necessary, we will use the symbol β to refer to one of these integers.

Summarizing, the current abstract state $y_i(t)$ is θ, where $M(\theta) = \beta$ is agent A_i's number of ready jobs, as defined by (5.1) and (5.2). The abstract state space $\mathbf{Y}_i$ is the set of symbols representing all the possible values of B_i, which are the non-negative integers up to some maximum value $B_{\max}$.

Since the symbols of the state space and the values of the measure defined over that space have a one-to-one correspondence, there is no need to burden the reading of this text by maintaining this distinction, so long as the reader understands that it does exist. In the few cases where the distinction must be made, the reader will be alerted.

5.3. Domain-specific Knowledge

To conquer uncertainty, agents are given knowledge specific to the domain of load dynamics. This knowledge is gathered by observing variables of interest, such as B_i, and noting specifically how they change *over time*. Most of this knowledge can be acquired offline, applying time-series analysis to past histories of these variables.

Knowledge can take on many forms, which depend on the problem. Our analyses will produce some knowledge in the form of a number of models of load dynamics. An agent can use these models to predict what state the system is in, given past state information. Since it is desirable that the reliability of the predictions be quantifiable, probabilistic models are used. These models are convenient since they encode very economically a great deal of knowledge about agent activity.

Unfortunately, not all knowledge can be expressed succinctly in terms of simple models. For example, when an agent receives a job, it must make a prediction about how much processing time the job will need. This can be done by consulting a table, which has recorded the amount of processing time that similar jobs needed in the past. This table is really a set of special-case *rules,* each one stating: if this is a job of type x, then it will need y units of processing time. Consequently, information which is unstructured, and cannot be generalized as a parametric model, can always be expressed as a set of special-case rules.

Finally, we note that knowledge can be represented by *rules which indicate which models to use.* Thus, from a large set of models, where each model applies to a

limited situation, a selection can be made based on rules, which test which situation applies. Even this form of knowledge is used in our load-balancing study.

5.4. Designing State-transition Models

An agent's abstract state space is defined to be the set of non-negative integers, up to some maximum value, and each state represents a possible value of the number of ready jobs in the agent. Given the value of a past state, an agent can predict a future state using a *state-transition model.* Our model will be probabilistic rather than deterministic; that is, it will provide for a set of *possible* future states, with assigned probabilities. These probabilities quantify the reliability of the model's predictions.

We will assume that state changes occur at discrete points in time, with some period T as the discretization period. (This period T need not necessarily be the same as the sampling period T_s, although it is convenient when T is a multiple of T_s. To simplify our discussion, we will assume that $T = T_s$.)
When we speak of agent A_i going through a sequence of states θ_0, θ_1, θ_2, starting at time t, we mean that $y_i(t) = \theta_0$, $y_i(t+T) = \theta_1$, $y_i(t+2T) = \theta_2$. In Section 4.5.2, we discussed the relationship between the abstract state space and the period T, and saw that the larger this period T, the smaller the rate of state change. A large T is desirable for many reasons, most importantly that uncertainty grows more slowly in time if the rate of state change decreases. The slower the increase in uncertainty, the greater the confidence in inferences about the current state based on past state information, and the lesser the need to update state information through communication.

A particularly useful model is one which has the *Markovian* property: the probability distribution of the next state $y_i((n+1)T)$, given the past states $y_i(0) = \theta_0$, $y_i(T) = \theta_1, \ldots, y_i((n-1)T) = \theta_{n-1}$, and the current state $y_i(nT) = \theta_n$, depends *only* on the value of the current state, not on those of past states. A first-order Markov model can be conveniently represented as a matrix of one-step transition probabilities P, with the n-step transition matrix given by P^n. This allows one to derive a set of possible states, along with their probabilities, given any past state, which is exactly what our agents must be able to do.

5.4.1. Load Levels and Degree of Variability

Load balancing deals with the natural fluctuations of the loads, and their consequent imbalances among different agents, by redistributing them. These fluctuations are caused by the unpredictable increases and decreases in the rate of job arrivals, and the unpredictable amount of work each job brings in. We make the following assumption, based on empirical evidence from our experiments described in Chapter 6, about the pattern of load fluctuations: the *load* does not change in a continuous fashion; rather, it remains constant, at some *load level* for an unpredictable interval of time, after which it changes to a new load level, where it then remains constant for

another interval of time until the next change. (See Figure 6.6 in Chapter 6 for an illustration of this behavior.) The lengths of time over which these changes take place are very short relative to the intervals of time the load remains constant. This is a good example of *domain-specific* knowledge, of which our agents will take advantage.

Note that the load level and the fundamental component of the variation in the number of ready jobs as defined in Section 5.2 are *not* the same. The purpose of identifying the fundamental component was to remove high-frequency components from the time series of instantaneous number of ready jobs by taking a *short-term* average. As for the notion of load level, it embodies the idea of *plateaus* within the *long-term* fluctuation of load. Their relationship manifests itself as the relatively continuous movement of the fundamental component of variation about a temporarily *fixed* load level. (See Figure 6.6 in Chapter 6 for an illustration of the relationship between the fundamental component and the load level.)

To make this more concrete, we now pose it in terms of how an agent's abstract state changes. The time behavior of *load* B_i will be characterized by two quantities: a *long-term average* of B_i, denoted by L_i; and the average difference between B_i and L_i, denoted by V_i. The idea is that L_i represents the load level, which should remain constant until a significant change in load takes place, and V_i is a measure of the *degree of variability* of B_i about L_i.

Let $MA_L(n)$ be a moving average (see (4.3)) of the past N_L loads,

$$MA_L(n) = \sum_{k=0}^{N_L} \omega_k B_i(n-k) \tag{5.4}$$

and let $MA_V(n)$ be a moving average of absolute *differences* between B_i and L_i for the past N_V loads,

$$MA_V(n) = \sum_{k=0}^{N_V} \omega_k{'} \mid B_i(n-k) - L_i(n-k) \mid . \tag{5.5}$$

The symbols ω_0, ω_1,..., $\omega_0{'}$, $\omega_1{'}$,..., represent weights of constant value, which are to be determined by design and experimentation.

Let H_L be the minimum *significant* change in B_i, according to some definition of significance. Similarly, let H_V be the minimum *significant* change in the variation of B_i about the load level. Finding good values for H_L and H_V will depend on experimentation. Good values for H_L and H_V will cause L_i and V_i to change slowly (e.g., with a period much larger than period T), and they will both represent long-term averages about which $B_i(n)$ and $B_i(n) - L_i(n)$, respectively, fluctuate.

We approximate L_i by rounding the value of $MA_L(n)$ to the nearest multiple of H_L. To avoid fluctuations of L_i when $MA_L(n)$ varies closely about the midpoint between multiples of H_L, hysteresis is to be added. We thus define L_i as a function of n as follows:

$$L_i(n) = \begin{cases} ROUND(MA_L(n),\ H_L) \text{ if } MA_L(n) > L_i(n-1) + H_L/2 + h \\ ROUND(MA_L(n),\ H_L) \text{ if } MA_L(n) < L_i(n-1) - H_L/2 - h \\ L_i(n-1) \qquad\qquad\qquad \textit{otherwise} \end{cases} \tag{5.6}$$

To compute V_i, recall that it represents a measure of B_i's variation *about* L_i. When L_i changes (i.e., when $L_i(n) \neq L_i(n-1)$), a *new* value for V_i must be established, regardless of its previous value. Let $N_c(L_i(n))$ be the number of time periods during which L_i has not changed (i.e., $L_i(n) \neq L_i(n-N_c(L_i(n))-1)$ but $L_i(n) = L_i(n-k)$ for $0 \leqslant k < N_c(L_i(n))$). Thus, in computing the moving average $MA_V(n)$, N_V is the minimum of N_L, the number of past states used in the MA_L moving average computation, and $N_c(L_i(n))$, the amount of time L_i has remained constant.

Consequently, we will define $V_i(n)$ as follows:

$$V_i(n) = \begin{cases} ROUND(MA_V(n),\ H_V) \text{ if } MA_V(n) > V_i(n-1) + H_V/2 + h \\ ROUND(MA_V(n),\ H_V) \text{ if } MA_V(n) < V_i(n-1) - H_V/2 - h \\ V_i(n-1) \qquad\qquad\qquad \textit{otherwise} \end{cases} \tag{5.7}$$

5.4.2. The Agent State-transition Model

With this characterization of the load in terms of a load level and a degree of variability in mind, we can now design an agent's state transition model. The model should allow an agent A_i to predict the possible states of another agent A_j, based on A_i's most recent reception of information about A_j.

Recall from Section 3.2 that $k_{ji}(t)$ represents A_i's current *state* information about A_j. Rather than sending each other the value of their number of ready jobs $B_i(n)$, our agents will exchange their load levels and degrees of variability, i.e., $L_i(n)$ and $V_i(n)$. This information is in a sense more valuable than simply the value of $B_i(n)$, as that value is true for a single point in time, but may change very quickly (within one period T). The load level and the degree of variability change much more slowly, and therefore do not need to be communicated as often. Consequently, the prediction of the current state based on the load level and on the degree of variability is likely to be more effective than if it were based simply on information about the most recent state.

Thus, we define

$$K_{ji}(t) = (L_j(n-a_{ji}),\ V_j(n-a_{ji})),$$

where $t=nT$, $\alpha_{ji} = a_{ji}T$, and $t - \alpha_{ji}$ is the time A_j recorded this information and subsequently sent it to A_i. Thus, α_{ji} is the age of this information in units of time, and a_{ji} is the same age in terms of time periods.

The agent's state-transition model should say, given $K_{ji}(t)$, what possible states A_j is currently in, and what the respective probabilities are. This model should have the following properties:

(1) it should be Markovian;

(2) given that $L_j(n) = \lambda$ has not changed in the interval of time periods $[n-a_{ji},\ n]$, the probability that $B_j(n)$ is greater than λ should equal the probability that $B_j(n)$ is less than λ. More generally, we would like $E(B_j(n) \mid L_j(n-a_{ji}) = \lambda) = \lambda$;

(3) the variance of $(B_j(n) \mid L_j(n-a_{ji}))$, should grow with increasing $V_i(n)$, and with increasing a_{ji}.

The actual state transition model is given by

$$p(B_j(n) = \beta \mid L_j(n-k) = \lambda,\ V_j(n-k) = v) = [P_v^k]_{\lambda\beta}, \tag{5.8}$$

where $[P_v^k]_{\lambda\beta}$ is the element in row λ and column β of the *k−step* state transition probability matrix P_v^k, P_v is the *one-step* transition matrix of size $B_{\max} \times B_{\max}$, defined by

$$P_v = \begin{bmatrix} \frac{(1+\rho_v)}{2} & \frac{(1-\rho_v)}{2} & 0 & 0 & \cdot & 0 & 0 \\ \frac{(1-\rho_v)}{2} & \rho_v & \frac{(1-\rho_v)}{2} & 0 & \cdot & 0 & 0 \\ 0 & \frac{(1-\rho_v)}{2} & \rho_v & \frac{(1-\rho_v)}{2} & \cdot & 0 & 0 \\ 0 & 0 & \frac{(1-\rho_v)}{2} & \rho_v & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 & \cdot & \rho_v & \frac{(1-\rho_v)}{2} \\ 0 & 0 & 0 & 0 & \cdot & \frac{(1-\rho_v)}{2} & \frac{(1+\rho_v)}{2} \end{bmatrix}. \tag{5.9}$$

This model has many good qualities: it is simple; it has the properties described above; it can be stored efficiently; and it can be evaluated efficiently (or pre-evaluated and stored as a table, for quick lookup afterwards). The parameter ρ_v is the probability that A_j will remain in the same state after one transition (except for states 0 and $B_{\max}$, for which the probability of remaining in these states after one transition is $(1 + \rho_v)/2$). It is a decreasing function of V_j (hence the subscript v),

$$\rho_v = f(V_j), \quad \frac{d\rho_v}{dV_j} < 0 \tag{5.10}$$

Thus, the greater the degree of variability, the greater the probability of moving to a higher or lower state, and therefore the lesser the probability of remaining in the same state. The exact relationship between ρ_v and V_j is determined experimentally.

We call this model the *steady-state response* model, in that it bases its prediction on the fact that the remote agent's load *is assumed to be at the same load level* since the reception of the load level information. Thus, the agent is in a steady state: to remain at the same load level, the load, determined by the arrival rate of jobs from its private source and from agents transferring jobs, and the mix of job types, has essentially remained the same.

An agent also needs a model for predicting *what would happen* if it offloaded a job to a particular agent. Given such a prediction, it can calculate whether utility will go up or down, and thus make a rational decision. For load balancing, our agents use the following simple model (so simple it can be stated as a single rule):

if a job is offloaded to A_j, L_j will increase by 1.

We call this model the *transient response* model in that it is used to predict a future state based on the expectation that the remote agent's load level *will change*. This is in contrast to the steady-state response model, where the load level remains constant. Note that there is an implicit assumption that an offloaded job will have a significant effect on the receiving agent's load level, specifically, that the load level will go up by one. This implies that, in general, the types of jobs that are offloaded are those that will execute for a relatively long time, long enough to affect the long-term average of B_i.

5.5. Measures for Comparing the Desirabilities of Agents

Load-balancing decisions require an agent, that wants to offload a job, to be able to tell which remote agent, if any, is best for receiving the job. In general, the remote agent with the "best" state, i.e., the state representing the presence of the smallest number of ready jobs, is not necessarily the one to which jobs should be offloaded. The main reason, illustrated in Section 4.5.3, is due to the uncertainty of information about these remote agents, and the nonlinear relationship between the measure of an agent's state and its utility. This results in situations where, for example, a remote agent with a reported load level of 4, which was reported 1 second ago, may be more desirable as the destination than a remote agent with a reported load level of 0, but which was reported 20 seconds ago. There is also the factor of distance: an agent which is far away is generally less desirable than a close one. Thus, we need a way of measuring the utility of an agent, given the uncertainty of the information about its number of ready jobs and its distance.

5.5.1. Computing an Agent's State Utility

Let us first consider the utility of the states of an agent. The objective is to minimize job delay; thus, we will define utility to be a measure of negative expected job delay, given the agent's state which represents its number of ready jobs. (Utility is a quantity to be maximized, and job delay is to be minimized, hence the reason for equating utility with *negative* job delay.)

Let $elap(w, \beta)$ be the total elapsed time of job w, given β, which is the time-averaged number of ready jobs during the lifetime of job w.

$$elap(w, \beta) = total\ elapsed\ time.$$

In general, $elap(w, \beta)$ increases with increasing β, for a given job w. Since we assume a homogeneous set of agents, the *elap* function is the same for all of them.

A job's *execution* time, the total time spent by an unloaded machine executing it, is

$$elap(w, 0) = total\ execution\ time,$$

since there are no other jobs to interrupt job w. Thus, the factor by which job w's elapsed time increases when there are a total of β jobs (including w) contending for the machine is

$$\frac{elap(w, \beta)}{elap(w, 0)}, \tag{5.11}$$

which we call the *stretch* factor. We define the quantity $\mu(\beta)$ as the negative stretch factor. The value of $\mu(\beta)$ reaches its maximum (which equals 0) when $\beta = 0$, and decreases with increasing β. We now determine its shape.

To construct $\mu(\beta)$, we must have a model of how jobs get control of the CPU. In particular, this model must tell us how much time is spent executing all jobs, and how operating system overhead is accounted for. Suppose that there are β compute-bound jobs, and that each takes the same amount of time $elap(w,0)$ to complete if any one of them were placed in an empty system. But since there are β of them, we would expect them to take at least $\beta \cdot elap(w,0)$ to complete, given pure processor-sharing scheduling. Thus, we have

$$elap(w, \beta) \geqslant \beta \cdot elap(w, 0).$$

This would be an equality if there were no overhead, which is never the case in real systems.

Let us use a very simple model for overhead. Let

$$f_{overhead} = \frac{\beta}{\beta_{\max}} < 1 \tag{5.12}$$

be the fraction of time attributed to overhead functions by the CPU. It is a linear function of the number of ready jobs: the more jobs it has to schedule for execution,

the more overhead. $\beta_{\max}$ is a constant representing the number of jobs which causes the CPU to spend *all* its time scheduling, therefore never getting any work done. (Note that $\beta_{\max}$ must be greater than $B_{\max}$, the maximum number of ready jobs, otherwise an agent might get into an undesirable state where it would never get any work done.)

Given a number β of ready jobs, the fraction of time the CPU can spend executing jobs is

$$f_{executing} = 1 - \frac{\beta}{\beta_{\max}}. \tag{5.13}$$

Using these models, we can now say that a job w, in a system where the number of ready jobs is β, would take the following amount of time to complete:

$$elap(w, \beta) = \frac{\beta \cdot elap(w, 0)}{1 - \beta/\beta_{\max}}. \tag{5.14}$$

The function $\mu(\beta)$ then becomes,

$$\mu(\beta) = \frac{-\beta}{1 - \beta/\beta_{\max}}. \tag{5.15}$$

Given $\mu(\beta)$, we define the *local utility* of agent A_i as,

$$u_i(\theta) = \mu(\beta), \;\; M(\theta) = \beta, \text{ for } all \; i, \tag{5.16}$$

where A_i is in state θ, and $M(\theta)$, the measure of state θ, is the numerical value β. Since our agents are homogeneous, they all share the same local utility function, given by $\mu(\beta)$.

Note that utility is not a linear function of the agent's state. This has a profound effect on decisionmaking when remote agents' states are not known with complete certainty. We explore this in more detail in Chapter 6.

5.5.2. Conditional Utility with Respect to Job Offloading

An agent's state utility indicates its ability to execute jobs, measured in terms of (the negative of) the expected increase in the job's elapsed time relative to its execution time. This measure, as defined by (5.15), depends solely on the jobs *already present* at that agent, and contending for the CPU.

Consider now the problem of agent A_i, who is trying to determine which agent, if any, is the best destination for a job w. Let us assume for a moment that A_i knows, with complete certainty, the state of every other agent, so that it can compute $u_j(y_j)$, for all j. Further, assume that A_i is the only agent making an offloading decision, so that a resonance is impossible. Would the best destination agent for w be the one with the maximal utility?

The problem here is that A_i must compute the utility of a *future* state of A_j, that which will occur *if* job w is offloaded there. This computation depends on

knowledge which only A_i has, namely that w is being considered for offloading to A_j. Therefore, A_j's future state utility is conditioned on knowledge possessed by the agent who is computing the utility. The difference between this *conditional* utility and the previous *absolute* utility is that conditional utility depends on the viewpoint of the computing agent, whereas absolute utility does not.

There is also the less subtle problem of accounting for network delays. If the delay in time due to the network transmission of a job from one agent to another is a significant factor, as is most likely if the agents are distributed over a large geographic area, conditional utility will be affected. This is because utility was defined as the negative stretch factor. Previously, this quantity was solely due to CPU contention; network delay, is now a factor to be taken into account. The network delay will generally be a function of the job size (including the size of data files associated with the job), and of the distance between the agents.

Thus, agent A_i is no longer just interested in A_j's current state. It is interested in its future state, conditioned on the probability of offloading a job to it, and conditioned on the time it takes to transmit the job to it. We now formalize this notion of conditional utility.

Assume that the function $netdelay_{ij}(w)$, i.e., the time job w is delayed due to network transmission from A_i to A_j, is known. The total elapsed time from the arrival of w at A_i to the end of w's execution on A_j (which has β ready jobs just before w's arrival) is,

$$netdelay_{ij}(w) + elap(w, \beta{+}1).$$

The conditional state utility of agent A_j (whose state is θ) from A_i's viewpoint, given that A_i will offload job w to A_j, is

$$u_{ji}(\theta, w) = -\frac{netdelay_{ij}(w) + elap(w, \beta{+}1)}{elap(w, 0)}, \quad M(\theta) = \beta,\ \beta < B_{\max}. \tag{5.17}$$

The ratio of a job's network delay to its execution time is denoted by

$$\eta_{ij}(w) = -\frac{netdelay_{ij}(w)}{elap(w, 0)}. \tag{5.18}$$

Extending definition (5.17) to include $\beta = B_{\max}$, and expressing $u_{ji}(\theta, w)$ in terms of definitions (5.15) and (5.18), the conditional state utility is defined by

$$u_{ji}(\theta, w) = \begin{cases} \eta_{ij}(w) + \mu(\beta{+}1), & \beta < B_{max},\ M(\theta) = \beta, \\ -\infty & \beta = B_{\max}. \end{cases} \tag{5.19}$$

Since after the transfer of w there will be $\beta{+}1$ jobs present at A_j, we use $\mu(\beta{+}1)$, assuming A_j's ready job number, β, is less than $B_{\max}$. If $\beta = B_{\max}$, a job offloaded to A_j could not be accepted, and would be returned. We consider this to have the lowest possible utility, $-\infty$. Note also that $\eta_{ij}(w)$ is independent of A_j's state. Therefore, conditional utility differs from absolute utility by a constant, but this

constant depends on information known only to the computing agent.

Unfortunately, the conditional utility function forces agents to know, or at least reasonably predict, the execution time a job will need (to compute $\eta_{ij}(w)$). The *net-delay* function must also be known, but this is less difficult to approximate as it is often a simple linear function of the distance between agents and of the total job size.

5.5.3. Expected Utility

So far in this chapter, we have assumed that agents know each other's states, and therefore can compute each other's utilities. Of course, this is an unreasonable assumption, as has been emphasized in the previous chapters. It is true that, in general, agent A_i does not know with complete certainty that, say, the state of A_j is θ. But A_i can compute the probability of this, based on past information about A_j,

$$p(y_j(t)=\theta \mid K_{ji}(t)). \tag{5.20}$$

Using the usual convention that $t = nT$, $\alpha_{ji} = a_{ji}T$, and $M(\theta) = \beta$, this probability can also be expressed as

$$p(B_j(n)=\beta \mid L_j(n-a_{ji}), V_j(n-a_{ji})). \tag{5.21}$$

Consequently, A_i can compute the *expected utility* of the state of A_j, based on past information $K_{ji}(t)$, as follows:

$$E[u_j(y_j(t)) \mid K_{ji}(t)] = \sum_{\theta \in \mathbf{Y}_j} u_j(\theta) \cdot p(y_j(t)=\theta \mid K_{ji}(t)). \tag{5.22}$$

Since the probabilities in (5.20) and (5.21) are equivalent, (5.22) can be expressed as

$$E[u_j(y_j(t)) \mid K_{ji}(t)] = \sum_{\beta=0}^{B_{\max}} \mu(\beta) \cdot p(B_j(n)=\beta \mid L_j(n-a_{ji}), V_j(n-a_{ji})). \tag{5.23}$$

If A_i's information about A_j, $K_{ji}(t)$, is that $L_j(n-a_{ji}) = \lambda$, and $V_j(n-a_{ji}) = v$, then in terms of the state-transition matrix P_v defined by (5.9), the expected utility is

$$E[u_j(B_j(n)) \mid \lambda, v, a_{ji}] = \sum_{\beta=0}^{\beta_{\max}} \mu(\beta) \cdot [P_v^{a_{ji}}]_{\lambda\beta}. \tag{5.24}$$

Using the expected utility, an agent can make comparisons between remote agents using information which is, to varying degrees, uncertain.

It is interesting to see the effect of expected utility as a function of the age of the state information. We will explore this behavior in Chapter 6.

5.5.4. Conditional Expected Utility

We now extend our formulation of the expected utility to include conditional expected utility. This requires computing the weighted sum of the conditional utilities for each possible state, weighted by the probability that an agent is in that state. In formula (5.19), which defines conditional utility, if the probability that the agent is

in state $B_{\max}$ is non-zero, the entire conditional expected utility will be $-\infty$. Consequently, we will first define the conditional expected utility given that the agent is *not* in state $B_{\max}$.

Let B represent the case that A_j is in state $B_{\max}$,

$$\mathrm{B} = \{B_j(n) = B_{\max}\},$$

and let $\overline{\mathrm{B}}$ represent the negative of B, i.e., the case that A_j is not in state $B_{\max}$. Let p_{B} be the probability of B occurring, given $K_{ji}(t)$:

$$p_{\mathrm{B}} = P(\ \mathrm{B}\ |\ K_{ji}(t)).$$

Define the conditional expected state utility of A_j from A_i's viewpoint, given that A_i will offload job w to A_j, and that A_j is not in state $B_{\max}$, as follows:

$$E[u_{ji}(y_j(t),w)\ |\ K_{ji}(t),\overline{\mathrm{B}}\ ] = \frac{1}{1-p_{\mathrm{B}}} \sum_{\theta \in \mathrm{Y}_j - B_{\max}} u_{ji}(\theta,w) \cdot p(y_j(t)=\theta\ |\ K_{ji}(t)), \tag{5.25}$$

which equals

$$\frac{1}{1-p_{\mathrm{B}}} \sum_{\beta=0}^{B_{\max}-1} (\eta_{ij}(w)+\mu(\beta+1)) \cdot p(B_j(n)=\beta\ |\ K_{ji}(t)). \tag{5.26}$$

Since the expectation sums over values 0 through $B_{\max}-1$ for β, it must be divided by the probability that $\beta \neq B_{\max}$, which is $1 - p_{B_{\max}}$.

If A_i's information about A_j, $K_{ji}(t)$, is that $L_j(n-a_{ji}) = \lambda$, and $V_j(n-a_{ji}) = v$, then, in terms of the state-transition matrix P_v, the conditional expected utility is

$$E[u_{ji}(B_j(n),w)\ |\ \lambda,v,a_{ji},\overline{\mathrm{B}}\ ] = \frac{1}{1-[P_v^{a_{ji}}]_{\lambda B_{\max}}} \sum_{\beta=0}^{B_{\max}-1} (\eta_{ij}(w)\ +\ \mu(\beta+1)) \cdot [P_v^{a_{ji}}]_{\lambda\beta}.$$

Since $\eta_{ij}(w)$ does not depend on β, it can be taken out of the summation, and we get

$$E[u_{ji}(B_j(n),w)\ |\ \lambda,v,a_{ji},\overline{\mathrm{B}}\ ] = \frac{1}{1-[P_v^{a_{ji}}]_{\lambda B_{\max}}} \left(\eta_{ij}(w)\ +\ \sum_{\beta=0}^{B_{\max}-1} \mu(\beta+1) \cdot [P_v^{a_{ji}}]_{\lambda\beta} \right).$$

What if $B_j(n)$ does equal $B_{\max}$ (i.e., B is true)? How is this incorporated into the conditional expected utility function? This will depend on the job offloading policies, which can vary for different load balancing environments. For example, one possible policy is that, if A_i offloads a job to A_j who already has the maximum number of jobs it can support, A_j will indicate to A_i that it cannot accept the job. A_i can then consider another agent for offloading it, or can retain it for local execution. To simplify matters, if we assume that A_i will keep the job, the conditional expected utility given $B_j = B_{\max}$ is,

$$E[u_{ji}(B_j(n),w) \mid K_{ji}(t), \mathrm{B}\,] = \eta_{ij}(w) + \mu(B_{\max}). \tag{5.29}$$

By offloading a job to A_j when A_j is in state $B_{\max}$, the *state utility of* A_j is only affected in that time was wasted shipping w. Since A_j does not accept w, its actual state is unaffected. (This policy can be improved by making A_i first send a *request* to A_j, and then making A_i wait for a confirmation for acceptance from A_j, rather than shipping the entire job first. In fact, A_j would reply with an update concerning L_j and V_j, so that A_i could make a decision based on better information. To keep our discussion simple, we will not use these improved policies. Once all the models are defined using the simple policy, it is not difficult to extend them to incorporate the improvements.)

Note that, although A_j's state which is $B_{\max}$ is unaffected if w is mistakenly offloaded to it, the state of *some other agent*, the one who eventually gets the job for execution, most certainly will be affected with a decrease in utility.

We can now give a complete expression for the conditional expected utility:

$$E[u_{ji}(B_j(n),w) \mid \lambda,v,a_{ji}\,] = \sum_{\beta=0}^{B_{\max}-1} (\eta_{ij}(w) + \mu(\beta+1))\cdot[P_v^{a_{ji}}]_{\lambda\beta} + (\eta_{ij}(w) + \mu(B_{\max}))\cdot[P_v^{a_{ji}}]_{\lambda B_{\max}}.$$

After some simplification, we get

$$E[u_{ji}(B_j(n),w) \mid \lambda,v,a_{ji}\,] = \eta_{ij}(w) + \sum_{\beta=0}^{B_{\max}-1} \mu(\beta+1)\cdot[P_v^{a_{ji}}]_{\lambda\beta} + \mu(B_{\max})\cdot[P_v^{a_{ji}}]_{\lambda B_{\max}}. \tag{5.31}$$

Finally, we define the conditional expected utility of A_i, given that it offloads job w to A_j, which may or may not be able to accept it. Let

$$E[u_{ii}(y_i(t),w) \mid K_{ji}(t)] = (1-p_{\mathrm{B}})\cdot u_i(\beta_i) + p_{\mathrm{B}}\cdot u_i(\beta_i+1),\ y_i(t) = \beta_i. \tag{5.32}$$

A_i's conditional expected state utility is based on its local state not changing if A_j is *not* in state $B_{\max}$, but that it will change (i.e., its number of ready jobs will increase by one) if A_j is in state $B_{\max}$.

In terms of P_v, $L_j(n-a_{ji}) = \lambda$ and $V_j(n-a_{ji}) = v$, this expectation can also be expressed as

$$E[u_{ii}(B_i(n),w) \mid \lambda,v,a_{ji}\,] = (1-[P_v^{a_{ji}}]_{\lambda B_{\max}})\cdot\mu(B_i(n)) + [P_v^{a_{ji}}]_{\lambda B_{\max}}\cdot\mu(B_i(n)+1).$$

In summary, agents can use conditional expected utility as a measure for comparing the desirabilities of remote agents with respect to offloading decisions. This measure accounts for the state of a remote agent, including the fact that such information is uncertain, and for the network delays between agents.

5.6. Making Rational Decisions

How do agents make rational decisions? It will depend on the types of decisions agents make, i.e., on whether they are problem decisions, which address the transfer of jobs to balance the load, or communication decisions, which address the maintenance of an agent's view of the global state. We are now ready to answer this question for each case. We also consider a method for avoiding resonances, designed specifically for the load-balancing problem.

5.6.1. Problem Decision Rules

First, we will consider the construction of an agent's problem decision rule. Recall from Section 3.2 that an agent's decision d_i is given by the decision rule γ_i,

$$d_i(t) = \gamma_i(z_i(t),\ s_i(t)).$$

The decision rule for agent A_i is a function of all *influences* z_i affecting it, and the newly generated work s_i arriving at A_i. The influences are of two types, transferred work and information. In terms of load-balancing, A_i's decision rule is a function of the current job arrivals, including those transferred from other agents as well as new ones, and of A_i's information about other agents, $K_i(t)$. The result of the decision rule is a *load-balancing decision:* whether a given job should be offloaded, and, if so, where.

It is the arrival of a job that triggers an agent into making a load-balancing decision, invoking the decision rule. We will assume that only *new* jobs (not those which have already been transferred) can be offloaded. More specifically,

(1) a job cannot be transferred more than once;

(2) if a job is to be transferred, the transfer must occur *before* the job has executed;

(3) if a job is transferred to an agent which cannot accept the job, the job is executed locally.

Note that these limitations are by choice, and not imposed by our formal model. Having them accentuates the consequences of each load-balancing decision that an agent makes, since later "corrections" are not allowed. For example, if the decision is to offload a job to a particular remote agent, and that agent determines this was not a good decision, it cannot correct it by further offloading the job to another agent. Similarly, if the decision is to execute a job locally, and later it is found that it would have been better to have offloaded it, we cannot correct it by later offloading.

An agent A_i has a *decision space* $\mathbf{D}_i$. Since our agents are homogeneous, they will have the same decision spaces. Thus, the decision space for A_i is

$$\mathbf{D}_i = \{\mathbf{d}_0, \delta_1, \delta_2, \cdots, \delta_N\},$$

where decision δ_k means "offload to agent A_k," for any $1 \leqslant k \leqslant N, k \neq i$; if $k=i$, δ_k is the decision to execute the job locally. Decision $\mathbf{d}_0$ is the null decision, meaning that no decision is made, and the job is kept locally but not executed (but may be executed at

a later time when another decision is made). Note the difference between the A_i's *decision variable* d_i, which can take on any value in $\mathbf{D}_i$, and an actual *decision value* d_0 or δ_k for $1 \leqslant k \leqslant N$, which is a particular element of $\mathbf{D}_i$.

In order to make a rational decision, agents must be able to predict what will happen to global utility if a particular decision is made. The decision which produces a maximal positive change of global utility is the best decision. In Chapter 4, we discussed the problems of computing global utility; approximations were proposed, of which we now make use. The central idea was to convert a global optimization into local optimizations, reducing the chance of resonances (caused by the conversion) by a separate mechanism. We now present the design of an agent's decision rule. The form of the decision rule is the same for every agent, again due to the homogeneity of agents.

Agent A_i's local optimization considers the change in utility for each possible decision A_i can make, ignoring all decisions being made concurrently by other agents. The *payoff* for $d_i = \delta_j$, the decision to offload job w to A_j made by A_i, is defined as

$$\tilde{\Delta}(\delta_j, w) = E[u_{ji}(y_j(t), w) \mid K_{ji}(t)] + E[u_{ii}(y_i(t), w) \mid K_{ji}(t)] + \sum_{k \neq i, j} E[u_k(y_k(t)) \mid K_{ki}(t)].$$

It is the sum of the utilities of the states in which every agent is expected to be after the decision is made. For every agent except A_j and A_i, the assumption is that the state will be the same as what it currently is; therefore, we compute the expected utility $E[u_k(y_k(t)) \mid K_{ki}(t)]$ of the current state based on the past information. (See Section 5.5 for definitions of the expectations used in the formula for $\tilde{\Delta}(\delta_j, w)$.)

A_j is expected to change its state if it is not already in state $B_{\max}$; its number of ready jobs is expected to increase by one since it is to receive a new job for execution. Therefore, we use $E[u_{ji}(y_j(t), w) \mid K_{ji}(t)]$, the conditional expected utility of the state A_j will be in after A_i offloads job w to it, defined by formula (5.31).

A_i, the decisionmaking agent computing the payoffs, must compute its own state utility taking into account the fact that, if the job being offloaded is not accepted by A_j, it must be executed locally. Therefore, we use $E[u_{ii}(y_i(t), w) \mid K_{ji}(t)]$, defined by formula (5.32).

When a new job w arrives at agent A_i, A_i computes the payoffs $\tilde{\Delta}(\delta_j, w)$ for all $1 \leqslant j \leqslant N$. It then creates a set of decisions

$$D_\delta(w) = \{\delta_k \colon \tilde{\Delta}(\delta_k, w) \geqslant \tilde{\Delta}(\delta_i, w),\ \delta_k \in \mathbf{D}_i\}.$$

$D_\delta(w)$ is the set of all decisions which A_i can make, that have a payoff greater than or equal to that of the decision to execute job w locally. It will then make a randomized selection, to minimize the probability of a resonance, from $D_i^*(w)$. That will be A_i's decision concerning job w.

5.6.2. SPACE/TIME Randomization for Load Balancing

To avoid the situation where agents that have a job to transfer all select the minimally loaded agent and cause a resonance, it makes sense to randomize selections over the *space* of all good candidates as job destinations. In the situation where there are too few good candidates for the number of jobs to be transferred by all agents, some agents will delay their transfer for a random amount of *time*. We are now ready to describe how this is done specifically for the load-balancing problem.

An agent A_i, about to make a load-balancing decision, must first determine if it should delay its decision. This will depend on the likelihood of a resonance, based on A_i's current information about the load levels of other agents,

$$[K_i(t)]_L = (L_1(n-a_{1i}),\ L_2(n-a_{2i}),\ \cdots,\ L_N(n-a_{Ni})).$$

Agent A_i will determine the number of jobs it expects all agents are capable of accepting, and the total number of jobs it expects will be transferred by all agents. We will make use of the thresholds defined earlier, R_u and R_o. Recall that, if an agent's average number of ready jobs is below R_u, the agent is considered *under-loaded*, and if an agent's average number of ready jobs is above R_o, the agent is considered *overloaded*.

Let C_k, the *job capacity* of A_k, be the expected number of jobs A_k is capable of accepting from other agents:

$$C_k = \max(0,\ R_u - L_k(n-a_{ki})),\quad 1 \leqslant k \leqslant N.$$

If A_k's load level is below R_u, then A_k has, on the average, room for $R_u - L_k(n-a_{ki})$ jobs.

Let C be the total job capacity of all agents:

$$C = \sum_{k=1}^{N} C_k.$$

Let J_k, the *job overflow* of A_k, be the expected number of jobs A_k has to offload,

$$J_k = \max(0,\ L_k(n-a_{ki}) - R_o).$$

If A_k's load level is above R_o, then A_k has, on the average, $R_u - L_k(n-a_{ki})$ jobs to possibly offload.

Let J be the total job overflow of all agents,

$$J = \sum_{k=1}^{N} J_k.$$

Given C and J, should an overloaded agent offload now, or wait? We want the agent to make this determination quickly, without having to consult other agents. Therefore, the agent must consider the likelihood that many jobs might be sent to a small number of agents. To determine this, we consider the following graph matching

problem.

Given a graph with two sets of vertices,

$$\mathbf{J} = \{j_1, j_2, \cdots, j_J\},$$

and

$$\mathbf{C} = \{c_1, c_2, \cdots, c_C\},$$

where the size of **J** is J, the total job overflow, and the size of **C** is C, the total job capacity, let a *matching* m be a set of edges where, for each edge, one vertex is from **J**, and the other vertex is from **C**, and each vertex in **J** is incident to at most one edge. A matching need not include all vertices in **J**, or in **C**. Let **M**(**J**, **C**) be the set of all possible matchings given **J** and **C**. Figure 5.1 contains an example of a matching.

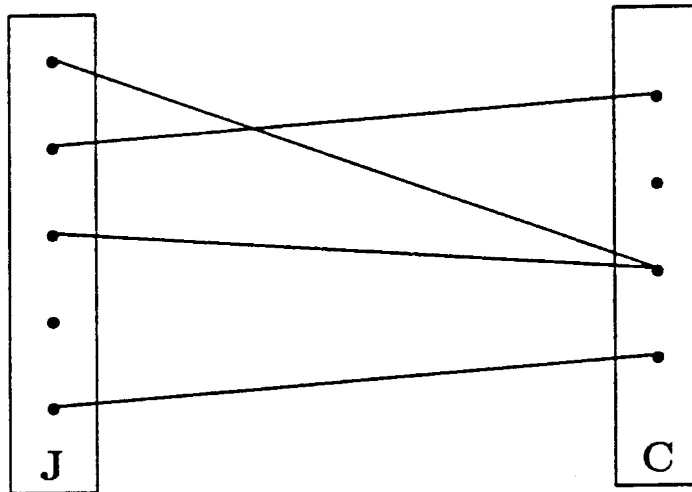


Figure 5.1. Example of a matching.

A matching has the *at-most-one* property if the degree of any vertex in **C** is at most one. Let $\mathbf{M}^+(\mathbf{J}, \mathbf{C})$ be the set of all possible matchings that have the *at-most-one* property. Figure 5.2 contains an example of a matching with the at-most-one property.

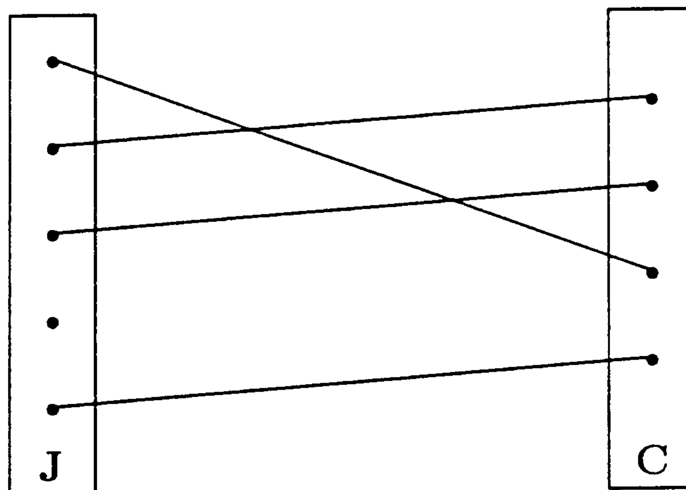


Figure 5.2. A matching with the at-most-one property.

Relating this to the load-balancing resonance problem, **J** represents all the jobs that can be offloaded, and C represents all the *slots* available at all agents for those jobs. A matching is simply a mapping of jobs to slots. A matching with the *at-most-one* property is a mapping where no slot gets more than one job, a desirable situation in avoiding resonances.

A vertex $j \in \mathbf{J}$ is *assigned* in matching m if m contains an edge (j,c), $c \in \mathrm{C}$. Consider the following algorithm for building a matching $m \in \mathbf{M}$. For each $j \in \mathbf{J}$, randomly decide whether it should be assigned in m or not, with probability

$$P(j \textit{ is assigned}) = \rho.$$

The value of ρ will be defined shortly.

If j is to be assigned, then randomly select $c \in \mathrm{C}$ such that each vertex in C has an equal likelihood of being chosen, and let (j,c) be an edge in m.

If matchings are created in this fashion, then the probability that exactly n vertices of **J** are assigned in the matching (or, equivalently, that the matching will have exactly n edges) is

$$p(|m| = n) = \begin{cases} \binom{J}{n} \rho^n (1-\rho)^{J-n} & n \leqslant C \\ 0 & n > C \end{cases} \tag{5.35}$$

If a matching has exactly n assignments, what is the probability that the matching has the *at-most-one* property? The total number of possible matchings given n assignments is C^n. The total number of matchings which have the *at-most-one*

property, given n assignments, is simply the number of permutations made up of n vertices, chosen from the C vertices in C.

$$P(C,n) = \frac{C!}{(C-n)!}, \quad 0 \leqslant n \leqslant C.$$

Therefore, the probability that a matching with n assignments has the *at-most-one* property is simply

$$\frac{P(C,n)}{C^n} = \frac{C!}{(C-n)!C^n}, \quad 0 \leqslant n \leqslant C.$$

If we now consider m_ρ^+ to be a random variable, representing a matching from M^+(**J**, **C**) created using the algorithm described above, we can determine the *expected* number of assignments $E(|\, m_\rho^+ \,|)$,

$$E(|\, m_\rho^+ \,|) = \sum_{n=0}^{\min(J,C)} n \binom{J}{n} \rho^n (1-\rho)^{J-n} \frac{P(C,n)}{C^n}. \tag{5.36}$$

Now, the value of ρ which maximizes $E(|\, m_\rho^+ \,|)$ can be found by differentiating the expectation in (5.36) with respect to ρ, setting the derivative equal to 0, and solving for ρ:

$$\frac{dE(|\, m_\rho^+ \,|)}{d\rho} = \sum_{n=1}^{\min(J,C)} n \binom{J}{n} \frac{P(C,n)}{C^n} \rho^{n-1} (1-\rho)^{J-n-1} (n - J\rho) = 0. \tag{5.37}$$

As an illustration, consider the case where there are $J > 1$ jobs, but only one slot $C = 1$. Substituting for C in the derivative, we get,

$$\sum_{n=1}^{\min(J,1)} n \binom{J}{n} \frac{P(1,n)}{1^n} \rho^{n-1} (1-\rho)^{J-n-1} (n - J\rho) = 0,$$

which, after simplification, yields

$$J(1-\rho)^{J-2}(1-J\rho) = 0.$$

Solving for ρ, we get

$$\rho = \frac{1}{J}.$$

Intuitively, it makes sense that an agent should decide to offload a job with probability $1/J$ if it knows that there are $J-1$ other potential senders, and only one slot available. In fact, this is precisely what happens in a distributed system of computers connected by an Ethernet [Metc76]. If J computers wish to transmit a packet at the same time, each should randomly decide to do so with probability $1/J$ to minimize the probability of collisions.

A table of optimal ρ values, given J and C, can then be constructed and made available to each agent. Whenever an agent has a job to offload, it first computes J

and C using its local information, looks up ρ, and makes a randomized decision. If the job is not to be offloaded, then the agent can either elect to execute it locally, or to wait until a later time to make the decision again. Since there are always new jobs arriving, we choose always to begin execution of the job locally, and then randomize the decision concerning the new job that arrives next.

We can now express the SPACE/TIME probability distribution for selecting a decision. Recall from the previous section that an agent determines

$$D_\delta(w) = \{\delta_k \colon \tilde{\Delta}(\delta_k, w) \geqslant \tilde{\Delta}(\delta_i, w),\ \delta_k \in \mathbf{D}_i\},$$

the set of all decisions which A_i can make with a payoff greater than or equal to that of the decision to execute job w locally. The probability of choosing local execution has already been determined above:

$$p(d_i = \delta_i) = \rho.$$

Using formula (4.24), the probability of choosing decision δ_j, $j \neq i$, is

$$(1-\rho)\frac{\tilde{\Delta}(\delta_j, w)}{\sum\limits_{\delta_k \in D_\delta(w), \delta_k \neq \delta_i} \tilde{\Delta}(\delta_k, w)}.$$

In summary, when an agent receives a new job, it first determines randomly whether the job should be offloaded or not. This random decision is based on its perception of how many jobs will be offloaded by all agents, and how much capacity there is for accepting jobs by all agents. The agent uses this to determine the optimal offloading probability, the one that maximizes the expected number of jobs offloaded while avoiding resonances. If it is decided to offload the job, then one of the multiple remote agents which are good candidates is selected randomly, with the probability of selecting an agent proportional to its relative payoff in utility.

5.6.3. Decisionmaking Processes for Load Balancing

We can now describe the various decisionmaking processes of a load-balancing agent. An agent's decision procedure, which gets evaluated when a new job arrives and whose result is a load-balancing decision, has four distinct phases:

(1) situation evaluation;

(2) job evaluation;

(3) destination evaluation;

(4) SPACE/TIME randomization.

In the first phase, *situation evaluation,* the agent considers whether a load-balancing decision is actually necessary, given its beliefs about the global system state. In particular, the agent considers its own local state. If the agent is lightly loaded (i.e., the number of ready jobs is below some threshold R_u), there is no reason to spend time evaluating the rest of the decision procedure since the decision to run the job locally is

a good one, and can be made quickly. If the agent is not lightly loaded, it considers the states of remote agents. If *all* remote agents are heavily loaded (i.e., their numbers of ready jobs are above some threshold R_o), then again, there is no reason to spend time evaluating the rest of the decision procedure since the decision to offload anywhere is a bad one. Consequently, the decision to run locally is best, and can be made quickly.

Assuming the situation warrants further evaluation of the decision procedure, the process enters the second phase, *job evaluation*. Specifically, it is determined whether the job is a good candidate for offloading. For example, the longer the job's expected execution time, the more desirable the job is for offloading. Although in general such information is not explicitly available, it may be possible to infer it based on an analysis of past behavior, such as on the previous execution times of the same job, or on those of similar type jobs.

The third phase is *destination evaluation*, where remote agents are considered as possible destinations for the job. In this phase, hypotheses are made concerning the expected improvement in utility, and are based on local information about the states of remote agents. We have called this expected improvement the *payoff* of a decision to transfer a job to a particular agent. Agents with positive payoffs make up the space of possible destinations, used by the next phase.

Finally, the fourth phase is *SPACE/TIME randomization*, where the destination for the job is actually selected. This selection is based on randomizing over the space of possible candidates, and possibly randomizing over a future time interval (as described in Section 5.6.2), so that resonances are avoided.

Figure 5.3 summarizes the phases of an agent's load-balancing decision procedure.

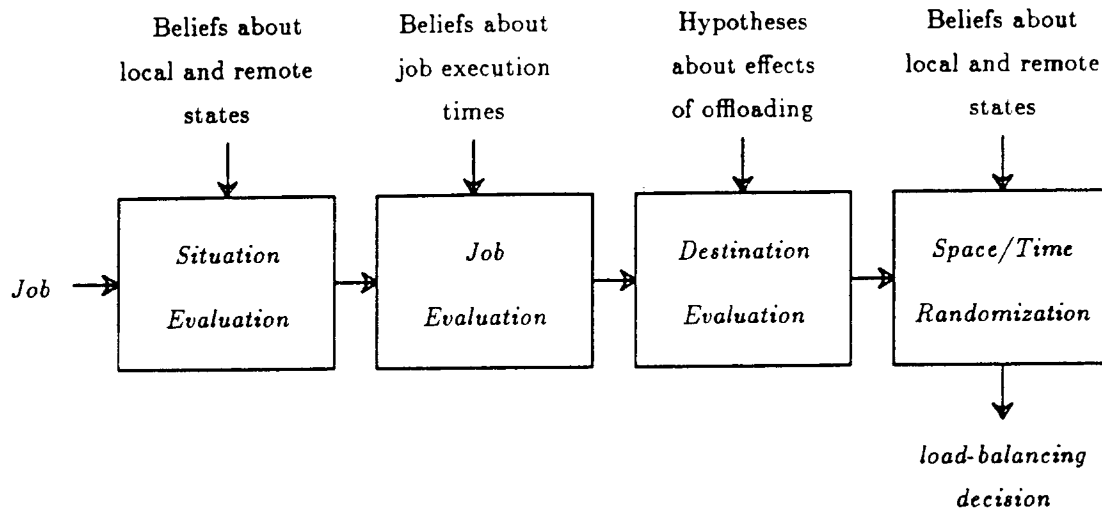


Figure 5.3. Phases of a Load-Balancing Decision Procedure.

5.6.4. Communication Decision Rule

Agents must decide *when* to communicate in order to update each other's state information. In Section 4.6, we described the process of determining when to *inquire* about a remote agent's state. This inquiry occurred when the sum of the loss due to communication overhead and the loss due to degradation in decision quality (due to aging information) was at a minimum.

For load balancing, agents will use a two-way cooperative communication protocol (see Figure 5.4): an agent will *request* a state update from a remote agent when its information about that remote agent becomes too uncertain; an agent will voluntarily *offer* a state update to a remote agent when it believes the quality of the decisions made by that remote agent will significantly improve with updated information.

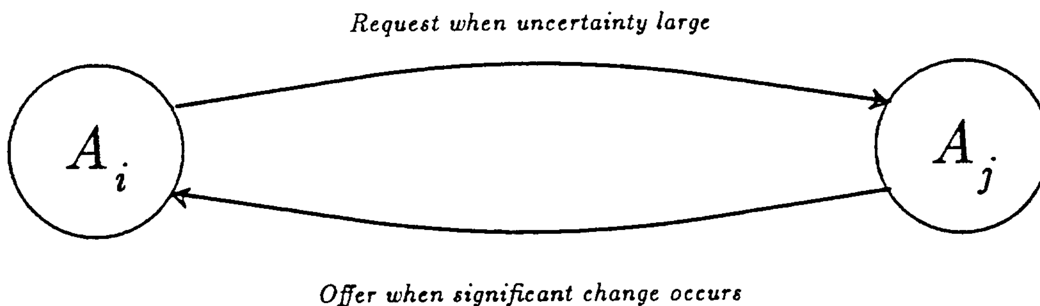


Figure 5.4. Two-way cooperative protocol for state updating.

When should an agent request an update from a remote agent? To answer this, we first need a loss function for communication overhead. Assume that agent A_i sends or receives a message (that is, *communicates*) with average period T_{ji}. It does not matter whom A_i communicates with, it simply matters that a communication takes place every T_{ji} time units. Assume also that every communication incurs a fixed amount of overhead in time, T_c (see Figure 5.5). We can then say that the fraction of time A_i wastes due to communication overhead is T_c/T_{ji}.

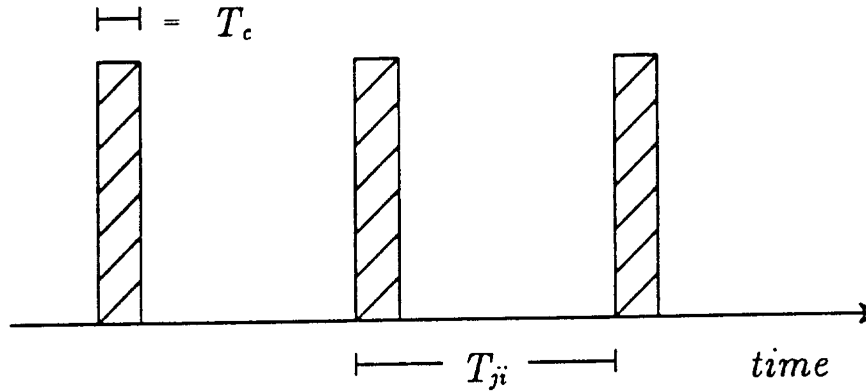


Figure 5.5. Communication overhead over time.

From (5.12), the fraction of time that an agent whose number of ready jobs equals β wastes due to job scheduling overhead is $\beta/\beta_{\max}$. Combining the two sources of overhead, we can say that the *loss* due to communication overhead, given the presence of job scheduling overhead, is the agent's state utility without communication overhead, minus the agent's state utility including communication overhead:

$$L_c^{(T)}(T_{ji}) = \frac{-\beta}{1-\beta/\beta_{\max}} - \frac{-\beta}{1-\beta/\beta_{\max} - T_c/T_{ji}}. \qquad (5.38)$$

Simplifying, we get

$$L_c^{(T)}(T_{ji}) = \frac{\beta \cdot C/T_{ji}}{(1-\beta/\beta_{\max})(1-\beta/\beta_{\max} - T_c/T_{ji})}. \qquad (5.39)$$

Checking extreme values, we see, as expected, that as the communication overhead $f_c = T_c/T_{ji}$ approaches zero, the loss goes to zero:

$$\lim_{f_c \to 0} L_c^{(T)}(T_{ji}) = 0,$$

and, as the communication overhead approaches $1 - \beta/\beta_{\max}$, the loss goes to infinity:

$$\lim_{f_c \to 1-\beta/\beta_{\max}} L_c^{(T)}(T_{ji}) = \infty.$$

We use this result to determine the maximum communication bandwidth for updating state information, given the maximum loss an agent is willing to allow. Then, a portion of this bandwidth is assigned to communication with every remote agent. We have described how this is done in Section 4.6.5.

We also need a loss function for degradation in decision quality due to aging information. We will use the pair-wise approximation based on A_i's local state, and A_j's possible local states, using formulas (4.20), (4.21), and (4.22) which were developed in Section 4.6.4.

Consider an agent A_i's decision to offload a job to remote agent A_j. Let β_i be A_i's number of ready jobs, and let A_i's information about A_j's load level be λ. Say that $\overline{w}$, a job of average *size*, arrives at A_i. Let

$$O_{ij}(\beta_i,\beta_j) = \mu(\beta_i) + \eta_{ij}(\overline{w}) + \mu(\beta_j+1), \tag{5.40}$$

represent the utility of A_i and A_j if $\overline{w}$ is *offloaded* from A_i to A_j, and let

$$R_{ij}(\beta_i,\beta_j) = \mu(\beta_i+1) + \mu(\beta_j), \tag{5.41}$$

represent the utility of A_i and A_j if $\overline{w}$ is *retained* by A_i.

If A_i offloads $\overline{w}$ to A_j, the sum of A_i's state utility and the conditional expected state utility of A_j from A_i's viewpoint will be

$$\overline{u}_{ji}(a_{ji})|_{\,offload} = E[u_{ii}(B_i(n),\overline{w}) \mid \lambda,v,a_{ji}] + E[u_{ji}(B_j(n),\overline{w}) \mid \lambda,v,a_{ji}],$$

which, by definition (5.40), equals

$$\overline{u}_{ji}(a_{ji})|_{\,offload} = \sum_{\beta=0}^{B_{\max}-1} O_{ij}(\beta_i,\beta)[P_v^{a_{ji}}]_{\lambda\beta} + O_{ij}(B_{\max},\beta_i)[P_v^{a_{ji}}]_{\lambda B_{\max}}. \tag{5.42}$$

Notice that, if A_j has a maximum number of ready jobs, then it cannot accept A_i's job, and A_i must consider the consequences of retaining the job.

If A_i does not offload $\overline{w}$ to A_j, the sum of A_i's state utility and the conditional expected state utility of A_j from A_i's viewpoint will be

$$\overline{u}_{ji}(a_{ji})|_{\,retain} = u_i(B_i(n)+1) + E[u_j(B_j(n)) \mid \lambda,v,a_{ji}],$$

which, by definition (5.41), equals

$$\overline{u}_{ji}(a_{ji})|_{\,retain} = \sum_{\beta=0}^{B_{\max}} R_{ij}(\beta_i,\beta)[P_v^{a_{ji}}]_{\lambda\beta}. \tag{5.43}$$

In fact, A_i will base its decision to offload or not on which of these two utilities is greater. Therefore, we can say that A_i will make the decision which maximizes the pair-wise conditional expected utility of A_i and A_j,

$$\overline{u}_{ji}(a_{ji}) = \max\left(\overline{u}_{ji}(a_{ji})|_{\,offload}\ ,\ \overline{u}_{ji}(a_{ji})|_{\,retain}\right). \tag{5.44}$$

We now need to calculate the maximum possible sum of pair-wise conditional expected utilities, $\overline{u}^*_{ji}$, based on making best decision if A_j's state *were known* by A_i. This is given by the formula,

$$\overline{u}^*_{ji}(a_{ji}) =$$

$$\sum_{\beta=0}^{B_{\max}-1} \max\left(O_{ij}(\beta_i,\beta),\ R_{ij}(\beta_i,\beta)\right)[P_v^{a_{ji}}]_{\lambda\beta} + R_{ij}(\beta_i, B_{\max})[P_v^{a_{ji}}]_{\lambda B_{\max}}. \tag{5.45}$$

The difference between $\overline{u}_{ji}(a_{ji})$ and $\overline{u}^*_{ji}(a_{ji})$ is that the former is the maximum of state utilities which are consequences of an offload or retain decision. This decision is based on the *expected utility* of A_j's state. The latter is the *expectation* of the *maximum of state utilities* which are consequences of an offload or retain decisions, *considered for each possible state* of A_j.

Therefore, A_i's loss function for degradation in decision quality due to aging information (based on formula (4.22), see Section 4.6.2) about A_j is

$$L_d^{(\alpha)}(a_{ji}) = \overline{u}^*_{ji}(a_{ji}) - \overline{u}_{ji}(a_{ji}).$$

This loss is a function of the age of information; we need to express it as a function of period T_{ji}. By (4.6) in Section 4.6.1,

$$L_d^{(T)}(T_{ji}) = \frac{1}{T_{ji}} \int_{T_t}^{T_{ji}+T_t} L_d^{(\alpha)}(\alpha_{ji}).$$

We need the discrete time version of $L_d^{(T)}(T_{ji})$. Let

$$N_{ji} = \left\lceil \frac{T_{ji}}{T} \right\rceil,$$

and let

$$N_t = \left\lceil \frac{T_t}{T} \right\rceil,$$

where T_{ji} and N_{ji} is the continuous and discrete time communication period respectively, where T_t and N_t is the continuous and discrete average transmission time respectively, and T is the continuous time state-transition period. Then,

$$L_d^{(T)}(N_{ji}) = \frac{1}{N_{ji}} \sum_{N_t}^{N_{ji}+N_t} L_d^{(\alpha)}(a_{ji}) \cdot T. \tag{5.46}$$

Now that we have the communication loss function and the decision quality loss function, we can compute the period T^*_{ji} for which their sum is at a minimum, which

is the best period for communication.

Finally, an agent will voluntarily *offer* a state update to a remote agent when it believes the quality of the decisions made by that remote agent will significantly improve. When should this happen?

We have assumed all along that an agent A_i can infer A_j's state based on knowledge of A_j's load level L_j, and A_j's state transition probability matrix, $P_v^{a_{ji}}$, which is derived from the measure of variability V_j. Therefore, when L_j or V_j change, A_j must broadcast their new values to all interested agents (which we have assumed to be *all* agents). Note that A_j knows the values of L_j and V_j with complete certainty, and, when they change, A_j knows that all other agents have old information which must be updated.

This method imposes an acceptable overhead since the load level and the degree of variability change very slowly. (If they did not change slowly, other variables would have to be identified which changed slowly in time, while providing enough information to a parameterized model so that statistical inferences could be made between updates.)

In summary, we have described a two-way cooperative protocol between pairs of agents, A_i and A_j. When A_i's information about A_j becomes too uncertain, A_i sends a request to A_j for an update. When A_j senses a change in its load-level or degree of variability, it volunteers the new information to other agents. The period between update requests is that which minimizes the sum of two loss functions: the loss from communication overhead, and the loss from degradation in decision quality due to aging information.

CHAPTER 6

EXPERIMENTS

We now discuss load balancing experiments in which we use our principles and techniques for intelligent decentralized control. The goal is to demonstrate the application of these techniques and principles, and verify their feasibility. The chapter is organized as follows. We will first summarize and analyze the load balancing problem in more concrete terms than in the previous chapter. Next, we describe the experimental setup and the approach, which includes a validation study of the simulator. We then provide experimentally determined values for the parameters of the models developed in Chapter 5. Finally, we present the results of the experiments.

6.1. Experimental Load Balancing

In Chapter 3, we presented a formal model that ignored the distinction between a machine (or computer system), which supports the execution of jobs, and an agent residing on the machine, which makes decisions pertaining to a decentralized control problem. We now do need to make this distinction, because in our experiments the machine is simulated, but the agent is real. With this in mind, we shall summarize the main ideas behind load balancing.

The load balancing problem centers around dynamically assigning jobs to machines so that some job performance index, such as the average response time, is optimized. The important characteristics of the problem are: each machine has its own job stream; an agent on each machine decides whether a job should either execute locally, on the machine owning the job stream it came from, or remotely, but in this case the job must be explicitly sent to a specific remote site; job arrival times and service times are *not* known in advance to the agent (even though the inter-arrival and service-time distributions may be predicted using past information).

We focus on the decentralized *source-initiated* form of the problem; i.e., when a job arrives at a machine, the agent residing on that machine must make a *load balancing decision* of whether to execute the job locally or remotely, and, if remotely, where. This is in contrast to the *receiver-initiated* scheme, where agents *request* work from other agents. There was no reason to select one scheme over the other. Either would have satisfied our goal, which was to determine the feasibility of our techniques, and not necessarily to find the best scheme for load balancing. (See [Eage86] for a comparative analysis of the two schemes.)

How can load balancing optimize the average response time of jobs? A job's lifetime can be divided into a number of time intervals, where for each interval the job is

characterized in one of three ways:

(1) job dependent execution;

(2) job dependent sleeping;

(3) system dependent waiting.

Thus, during the job's lifetime, the job is either executing, or sleeping (i.e., doing nothing) for an interval of time which is dependent on the job's characteristics, or waiting for a time which is dependent on external system factors. For example, if a job cannot execute because there are other jobs which must also execute on the same machine, this is a system dependent factor and therefore the job is classified as waiting. On the other hand, if a job cannot execute because it must wait for input from a user, this is a job dependent factor, and therefore the job is classified as sleeping.

The sleep/wait distinction is made for the following reason: wait time can be affected by load balancing, sleep time cannot. A good load balancing scheme will minimize wait time, but it cannot affect sleep time. (We are taking a very idealized view of the separation between wait and sleep times. In a real system, sleep time and wait time are generally not independent of each other. For example, the user's input speed, and the input itself, in an interactive program can be affected by sluggish response time.)

So if a goal of load balancing is to minimize job lifetimes, and only the wait time component of the lifetime can be affected, then it is the wait times which must be minimized. Factors contributing to wait times are queueing delays at the CPU, disks, or network communication channels, and network transfer times when offloading. If good load balancing decisions are being made, jobs will be assigned to machines in such a way that contention for these resources is spread more uniformly over time.

The decision rules that agents will execute are *stochastic replicated decision functions* [Stan85]. They are stochastic because decisions will be probabilistic, due to the fact that important aspects of the job streams are not known in advance. They are replicated because all agents use the same algorithm, control is fully decentralized, and jobs can execute on any machine.

There are a number of important reasons why we chose load balancing as a vehicle for illustrating the usefulness of our methods. Load balancing is a relevant research topic in itself, worthy of investigation. The operating systems community has recognized the importance of location-independent process (i.e., job) design, so that load balancing of jobs is feasible [Powe83]. Also, load balancing has been found to be effective [Zhou87], although the question remains as to what is the best way to do load balancing. Further, current distributed systems research is focusing on the design of very large distributed systems [Ande87], where the potential for resource sharing, and in particular, processor sharing, is great. Finally, one can create controlled meaningful simulation experiments for load balancing, given the availability of job trace data, and one can verify the realism of a simulated environment, something

we have made a great effort to do.

6.2. Experimental Setup

The environment of our load balancing experiments is a simulated distributed system of DEC VAX/780 machines running the Berkeley Unix operating system. These machines are connected by a point-to-point network whose topology is randomly created for each experiment with the constraint that each machine has, on the average, three neighbors.

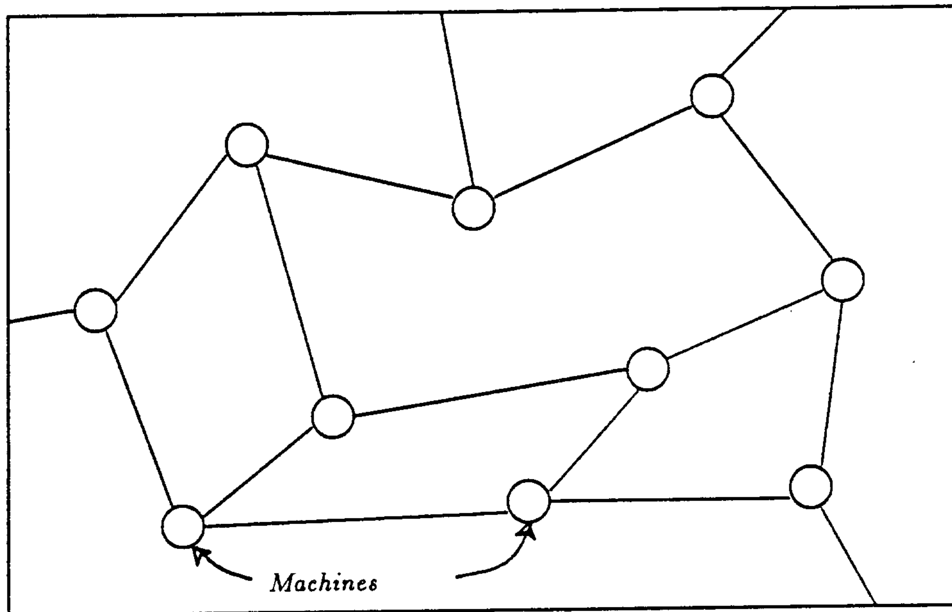


Figure 6.1. Example of a network of machines.

Each machine individually simulates its own job activity (i.e., the scheduling and movement of jobs among a number of servers). The network is simulated in the sense that inter-machine transmissions are delayed as a function of the distance, i.e., the number of hops between machines. Routing, link traffic, and congestion, are not simulated, but queueing of messages at the source and destination machines (not at the intermediate nodes) is.

6.2.1. Processor Simulation Model

Each machine is individually modeled as a set of five servers with queues: a source, a CPU, an I/O device, a network interface, and a sink. The servers, queues, and their connections are illustrated in Figure 6.2.

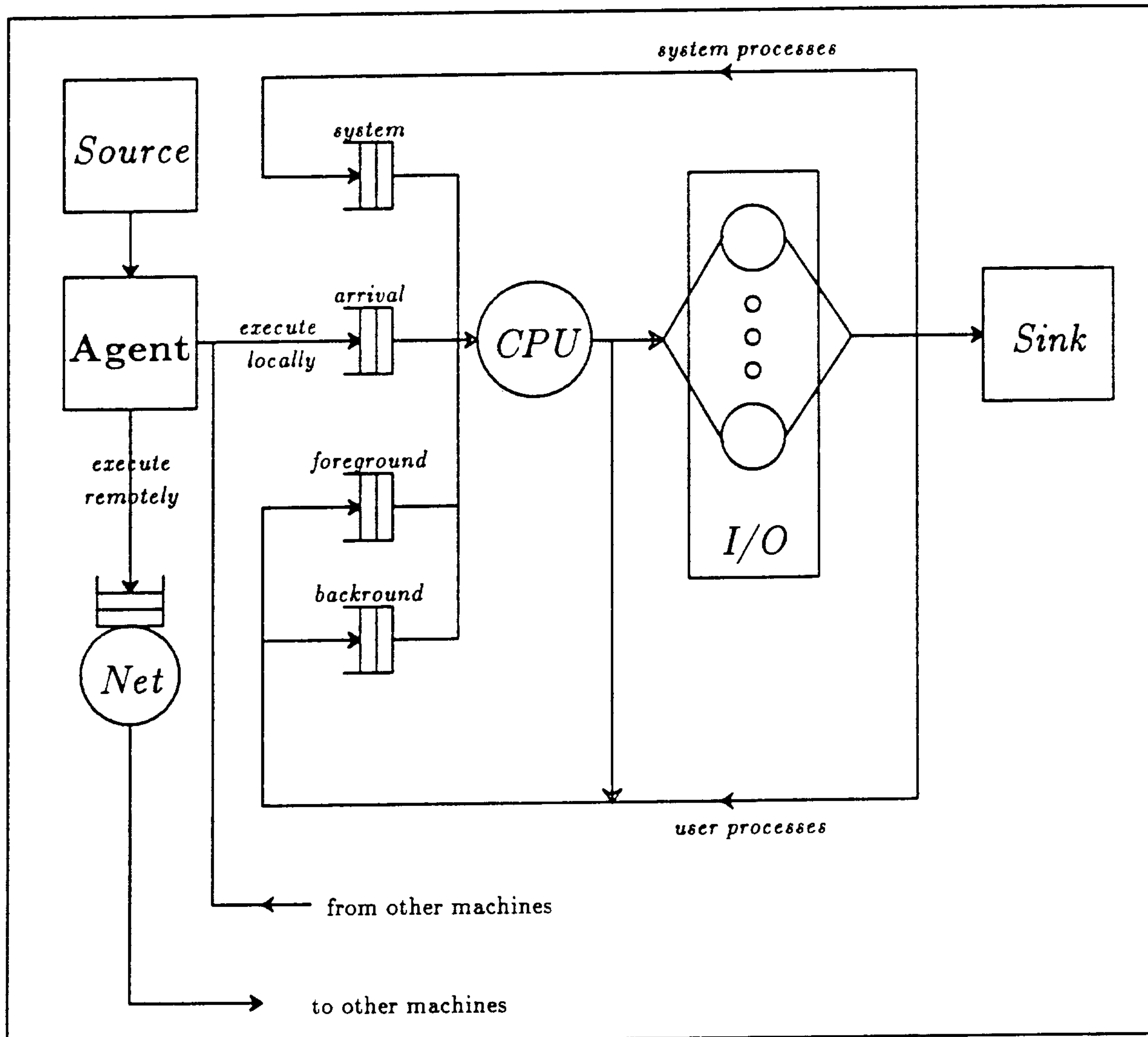


Figure 6.2. Simulation model of a single machine.

These servers and queues are occupied by *jobs,* which are the basic units of work. There are two types of jobs, *user* and *system*. User jobs generally represent work initiated by human users, and are the objects which may get offloaded for load balancing. System jobs represent work done on behalf of the system (e.g., scheduling).

The source server produces user jobs at specific points in time. Characteristics of these user jobs, such as their arrival time, their total execution time, and their total elapsed time, are determined by reading a *trace* of job accounting records derived from the real workload of a Berkeley Unix system. Consequently, the simulation is *trace-driven.*

The CPU server simulates the time-shared execution of jobs. Jobs are served or are *executed* by being delayed in the CPU server for a short fixed period of time q called a *quantum*. They will repeatedly visit the CPU server, each time for one quantum, until they have executed for a time equal to their total execution time defined in the trace.

The CPU server has four queues: arrival, foreground, background, and system. When a user job arrives at a busy CPU, it enters the arrival queue if it is a *new* job, the foreground queue if it is a *young* job, and the background queue if it is an *old* job. A new job is one which has not accumulated any CPU time yet. A young job is one which has accumulated less than $T_{f/b}$ seconds of time, and an old job is one which has accumulated at least $T_{f/b}$ seconds of time. $T_{f/b}$ is a tunable parameter of the simulator. Only user jobs are classified as new, young or old. The system queue is used only for system jobs, which go there regardless of their accumulated CPU time.

The queueing policy is as follows. When the CPU server releases a job (the job has executed for one quantum) it removes a job from the system queue, if one exists, and executes it. If the system queue is empty, it looks at the arrival queue for a job. If the arrival queue is empty, it looks at the foreground queue for a job. If the foreground queue is empty, it finally looks at the background queue for a job. Thus, the queues implement a single virtual *priority* queue, with system, new, young, and old being the job priority order. Jobs of the same type are handled first-come-first-served.

The I/O server is an infinite server with no queueing. It models a job's I/O (e.g., terminal and disk) time as a fixed delay between visits to the CPU. The total I/O time, summed over all visits, is derived from trace file data. (Unfortunately, job arrival times at each I/O device were not available from the traces, and therefore they had to be estimated.) Note that we do not model the queueing that might actually occur in a real system. This was done for two reasons:

(1) this simplifies the simulator;

(2) the CPU is by far the bottleneck for jobs in the systems we have observed, with very little queueing occurring at I/O devices.

The network server is used to transfer jobs from one machine to another, to support load balancing. It simulates the delays that would be incurred in packet transmission in a real system. The delay times are determined by the size of what is being transmitted, and by the distance in hops between the machines. A message is a single packet, the smallest unit of size, and jobs are multiple packets comprising their code and data files (the number of packets for the transfer of a job is inferred from the trace data). The network topology is randomly generated, constraining each machine to have three neighbors (i.e., links to other machines) on the average.

Finally, the sink server is the final destination of a job. Its function is to record job statistics, and to release resources owned by the job.

6.2.2. Job Activity

So far, we have described the operation of each server in isolation. We now consider the possible paths a job will take about the servers, which will illustrate server interactions. When a job is created, a decision by an *agent* must be made whether to execute it locally or remotely. Once this decision is made, our simulated machine guides a job through each server until it has completed execution. Note that every machine has its own single agent which makes load balancing decisions. For now, we will defer the discussion of agents and load balancing, and focus on a job's simulated activity within a machine.

When a job arrives at a machine to begin execution, it first enters the CPU server's *arrival* queue. The arrival queue is only used for *new* jobs; once the job has received some service from the CPU, it will use the foreground queue exclusively, until its execution time surpasses a threshold $T_{f/b}$, after which it uses the background queue exclusively.

When serviced by the CPU, the job executes for one time quantum q. After this, assuming the job needs more execution time, it will cycle about the CPU (and its queues), until it needs to do I/O.

Unfortunately, the trace accounting file created by Berkeley Unix did not provide job arrival times at I/O devices; therefore, they had to be estimated. We simply used a fixed quantity, N_q, which represents the maximum number of times a job can cycle about the CPU before needing I/O. (Values for q, N_q, and $T_{f/b}$ were determined by experimentation. The optimal values which provided the minimal error in validation tests, described later, were $q = 1/64$ seconds, $N_q = 8$, $T_{f/b} = .75$ seconds.)

After a job has visited the CPU N_q times, assuming it has not completed execution, it goes to the I/O server. There, it receives a variable amount of service time, with the constraint that after *all* its visits to the I/O server, the total I/O time equals the value for total I/O time provided by the trace file. After I/O service completes, the job returns to the CPU server.

The cycling between the CPU and I/O servers continues until the job has completed execution. Upon completion, the job enters the sink server, where job statistics, including the mean and variance of the job's queueing and service times for each server, are recorded. The job is then destroyed.

The job's simulated elapsed time is the time interval which begins when the job is generated by the source server, and ends when it leaves the system at the sink server. Note that this simulated elapsed time is the sum of the job's execution and I/O times, which are given values from the trace file, and the CPU queueing time, which is a function of the simulated system's dynamic behavior. (It also includes network queueing and transmission delay time if load balancing is in effect, and the job has been transferred from one machine to another.) If the simulator works well, the simulated elapsed time (with no load balancing) will be close to the real elapsed time, which is also obtained from the trace file. This was one of the measures we used for

the simulator's validation.

6.2.3. Job Movement Between Machines

When load balancing is activated, an agent considers a new job for machine *placement* after leaving the source server. If the job is to execute locally (on the same machine where it was generated), it goes to the local CPU server. If it is to execute on a remote machine, it goes to the local network server, and from there goes to the remote machine's CPU server. Once the placement decision is made, the job resides for its entire lifetime on the selected machine. This is in contrast to *job migration* (also referred to as *process migration*), where a job can be moved at any time during its lifetime. Although job migration is more difficult from an operating system design point of view, it might produce better load balancing results since redistribution of load occurs on a finer granularity. As discussed in Section 5.6.1, the consequences of bad job migration decisions are less severe than those of bad job placement decisions since they can be "corrected." Since our goal is to test the decisionmaking capabilities of a new decentralized control system, and not necessarily to determine what the best load balancing method is, we have chosen job placement load balancing for our experiments.

6.2.4. Operating System Overhead

In a real system, not all CPU time is devoted to running jobs. Some time is lost to overhead incurred by the operating system to carry out such operations as context switching, priority calculation, queue manipulations, and job table lookups, to name a few. In our simulator, this overhead is represented by a system job which runs periodically (each second) and uses up some CPU time.

In Chapter 5, we proposed a simple model for the fraction of time spent due to CPU overhead, based on the simple idea that the more jobs there are for the operating system to consider, the more time it spends in overhead. Assuming that the number of ready jobs β is known (the details about how β can be obtained are discussed later), this fraction was approximated by the linear function of β given in (5.12), which is

$$f_{overhead} = \frac{\beta}{\beta_{\max}} < 1.$$

Using this, we determined in (5.14) the ratio of a compute-bound job's expected elapsed time to its execution time, for a given β:

$$\frac{elapsed\ time}{execution\ time} = \frac{\beta}{1 - \beta/\beta_{\max}}.$$

In fact, this is the measure used to define the utility of an agent in state θ, where $M(\theta) = \beta$.

To obtain $\beta_{\max}$, and, more importantly, to check if the model reflects reality, experiments on a real system were performed. We created a purely CPU-bound job which used 10 seconds of CPU time, and ran it every 10 minutes. Each time it ran, we recorded the elapsed time and the number of ready jobs averaged over the job's elapsed time. The results are summarized in Figure 6.3, showing the elapsed time to execution time ratio, as a function of the average number of ready jobs.

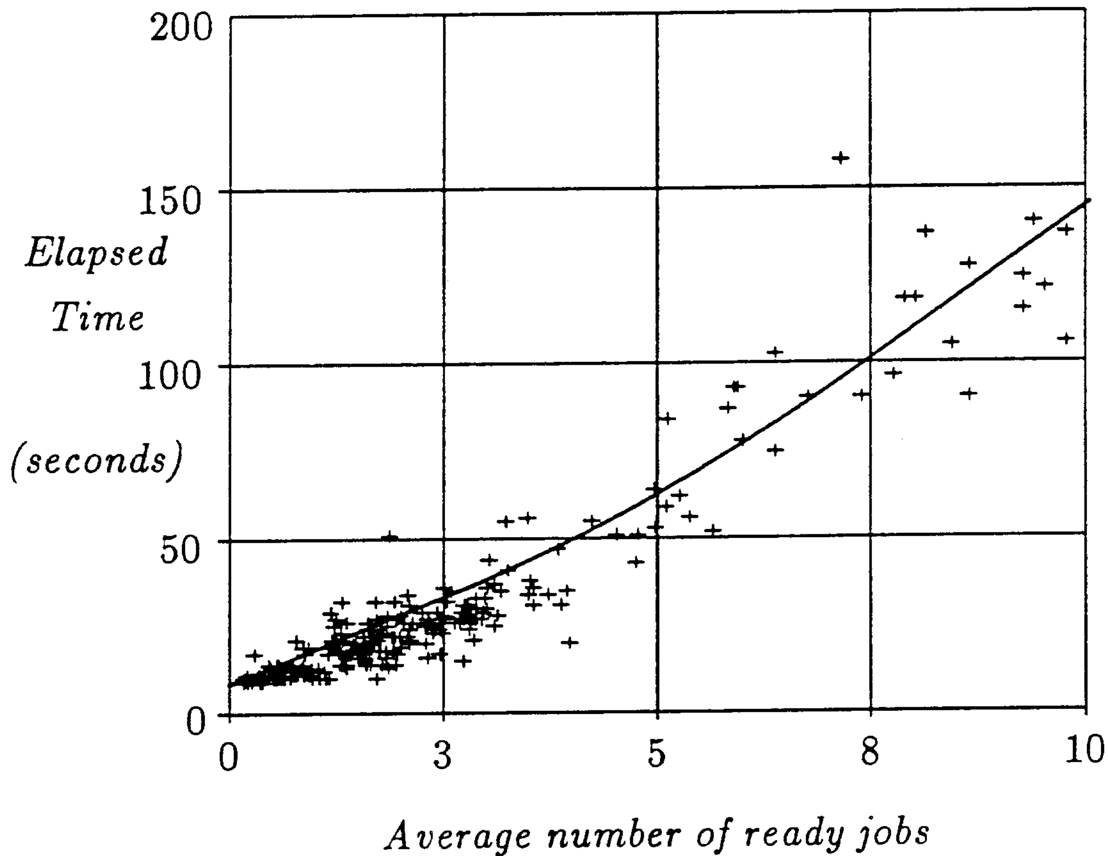


Figure 6.3. Ten-seconds execution of a CPU-bound process.

Note the general shape of the curve, fitted to the data points. Our model of $\frac{\beta}{1-\beta/\beta_{\max}}$ seems to fit. We can then choose the value of $\beta_{\max}$ which minimizes the mean square error between the points of the curve $\frac{\beta}{1-\beta/\beta_{\max}}$ and the experimentally measured points. The best value for $\beta_{\max}$ was found to be 32.258. For efficient implementation of the function, we simply used $\beta_{\max} = 32$.

6.2.5. Input Trace Description

The trace files used to generate input to the simulator are derived from the *real* workloads of systems running under the Berkeley Unix operating system. We believe that the driving of the simulator with genuinely real workloads was one of the most important decisions we made when designing the experiments. This is because load balancing is concerned with the *dynamic* behavior of workloads. A probabilistic workload generator, which makes stationary assumptions about the job interarrival time distributions, or the service time distributions, might not capture this behavior correctly. Also, we wanted the results of our experiments to reflect how a real system would behave; using real inputs was one important step in this direction. A robust simulator is another step, to be described shortly.

The traces reflect workloads from two types of environments: a computer science research one, and a staff-support one. The computer science research environment workload is primarily influenced by text formatting, program compiling, and CPU-intensive simulation jobs. The main components of the staff-support environment workload are text editing and formatting, and mail jobs. Each trace represents one full day of job activity. The traces were recorded at three different sites, U.C. Berkeley, AT&T Bell Laboratories, and Bell Communications Research, from at least two machines at each site. The traces from U.C. Berkeley were gathered at different times of the year. The traces from the other sites were gathered during summer months.

We divided each trace into smaller traces of 2-hour periods, thus obtaining a very large pool of trace files representing a variety of different workloads, and capable of creating a variety of different load levels (different for each machine, and varying differently over the time of each machine's activity). In fact, each trace was used to drive a single-machine simulation experiment to determine how the number of ready jobs varied over the 2-hour time period. We also computed an overall time-averaged value of the number of ready jobs for each 2-hour time period (we call it the *average load* for the trace), giving us an idea of whether the trace produced an overall low or high load. This was useful in the load balancing experiments for constructing non-uniform workload distributions over machines (e.g., high average load traces on some machines, low average load traces on others) to see the effects, if any, of load balancing in those conditions. For each experiment, a random assignment of traces to machines was made, keeping constant the sum of the average loads of all the traces.

The relevant per-job data contained in the traces are the job name, birth time, total CPU usage (user and system time), total elapsed time, time-averaged memory usage, and number of disk I/O's (each taking a known, roughly constant, amount of time). The times are recorded in discrete units of 1/64 second, except for birth times, which are in units of 1 second (due to record packing limitations in the trace file). To avoid discretization effects of arrivals occurring only at 1 second intervals, a different random value between 0 and 63/64 second, in units of 1/64 second, was added to each birth time.

Although the environment was simulated and trace-driven, that aspect of each machine which makes decisions about whether to offload jobs or not, which we have referred to simply as *the agent,* was real. From the perspective of the agent, what we performed were trace-driven *simulation-driven* experiments. What was simulated was the *environment* which provided inputs to each agent, and which each agent could affect. Each agent could be used in a real system without modification (except, of course, for the interfaces).

All experiments lasted for two hours of *simulated* time. Collection of statistics began after the first five minutes of simulated time to minimize the effects of system startup transients. Depending on the number of machines in the distributed system, the experiments took anywhere between 20 minutes (1 machine) to 6 hours (30 machines) of real time to execute on a DEC VAX 8600.

6.2.6. Validation of the Simulator

Simulation is a desirable experimental method because it allows one to observe and test a system which may not be physically realizable or accessible. For our experiments, we would need a large number of machines to construct a distributed system. In fact, our techniques for agent-based decentralized control have greater significance when there are large numbers of machines, since global state uncertainty grows as the number of machines increases. Simulation experiments can be repeated many times at relatively low cost, and the environment can be changed in each experiment in a controlled manner.

Of course, a simulation only makes sense when the simulator does in fact provide a true model of the real environment; this is why validation is not only an important part of any modeling study, it is a necessary part. Only after a complete and careful validation can the experimenter have faith in the results produced by the simulation. Equally important, any description of a simulation experiment must include the validation procedure and its results.

We chose to follow two separate validation procedures. In both procedures the goal was to simulate the activities of a single machine. Since we could actually acquire the same type of results from the real machine, the simulated and real results could be compared and evaluated. Assuming we could rely on the simulated single machine, then we could replicate it and create a large simulated distributed system (which we do not have a real version of such a system that can be used for such experiments as these, and therefore could not measure and compare). Then, using measurements of real point-to-point network transmission delays, we could model the network interconnecting the simulated single machines as a delay, dependent on the number of packets and the number of hops.

In the first validation procedure, we compared the simulated and real elapsed times of jobs. The job elapsed time can be considered as simply the sum of its CPU execution time, its I/O time, and its CPU queueing delay. Since the CPU and I/O

times are fixed by the trace, the only variable which depends on the dynamics of the simulation is the CPU queueing delay. Thus, the relative differences of the simulated and real elapsed times gives us a measure of how faithfully queueing delays have been modeled. This is important since CPU queueing delay is a major component of the cumulative system-dependent delay a job experiences; it is this cumulative delay, averaged over all jobs, that load balancing attempts to minimize. (The other *major* component of cumulative system-dependent delay is network transmission time of offloaded jobs. Note that it is important to minimize the *sum* of CPU queueing delay and network transmission delay. This may mean increasing network transmission delay due to offloading a job in order to obtain a more substantial decrease in CPU queueing delay, and consequently an overall decrease in the cumulative delay. This is the point of load balancing.)

Define the percentage relative error between simulated and real elapsed times as follows:

$$\textit{percentage relative error} = 100\% \times \frac{| R - S |}{R}$$

where R = real elapsed time, and S = simulated elapsed time. Every time a job completed, the percentage relative error was computed and stored. Figure 6.4 displays the cumulative distribution of percentage relative error.

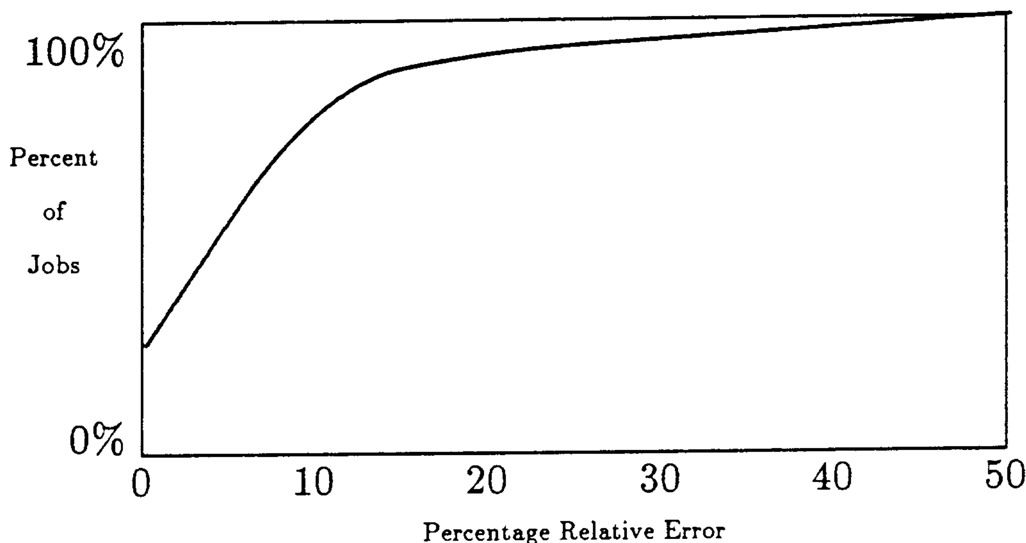


Figure 6.4. Distribution of relative error.

The simulated elapsed times of 24% of all jobs were perfect, i.e., they matched the real elapsed times exactly. 84% of all jobs were simulated to within 10% of their

real elapsed times, and 99% were simulated to within 50% of their real elapsed times. The average relative percentage error per job was 7.69%.

One problem with the relative error measurement is that, if a job's real elapsed time is a very small value, say 30 milliseconds, and its simulated elapsed time is, say, 15 milliseconds, the relative error is 50%, which would be the same relative error as that for a job whose real elapsed time is 4 hours and its simulated time 2 hours. Therefore, we also measured the absolute error, defined as follows:

$$\textit{absolute error} = | \ R - S \ |$$

where, again, R = real elapsed time and S = simulated elapsed time. The average absolute error per job was 157 milliseconds, and the average job elapsed time was 23.89 seconds.

In the second validation procedure, we constructed a controlled experiment where a purely CPU-bound job, called the *test* job, was executed periodically. Recall that this was already done on the real system to obtain constants for modeling system overhead. We also "ran" the test job on the simulated system. The test job had a predetermined and selectable execution time of either 10 seconds or 60 seconds, and the period between runs was 10 minutes, so that a test job was never started while a previously started test job was still running. In both real and simulated systems, each time the job ran, its elapsed time and the number of ready jobs averaged over the elapsed time were recorded. Note that the elapsed time is the sum of the execution time, a fixed known value, and of the time due to system overhead, which includes CPU queueing time. Our hope was that the elapsed times as functions of the number of ready jobs in both systems would turn out to be close to each other. This was indeed the case. Graphs for the simulated elapsed-to-execution-time ratios and the real elapsed-to-execution-time ratios are displayed in Figure 6.5.

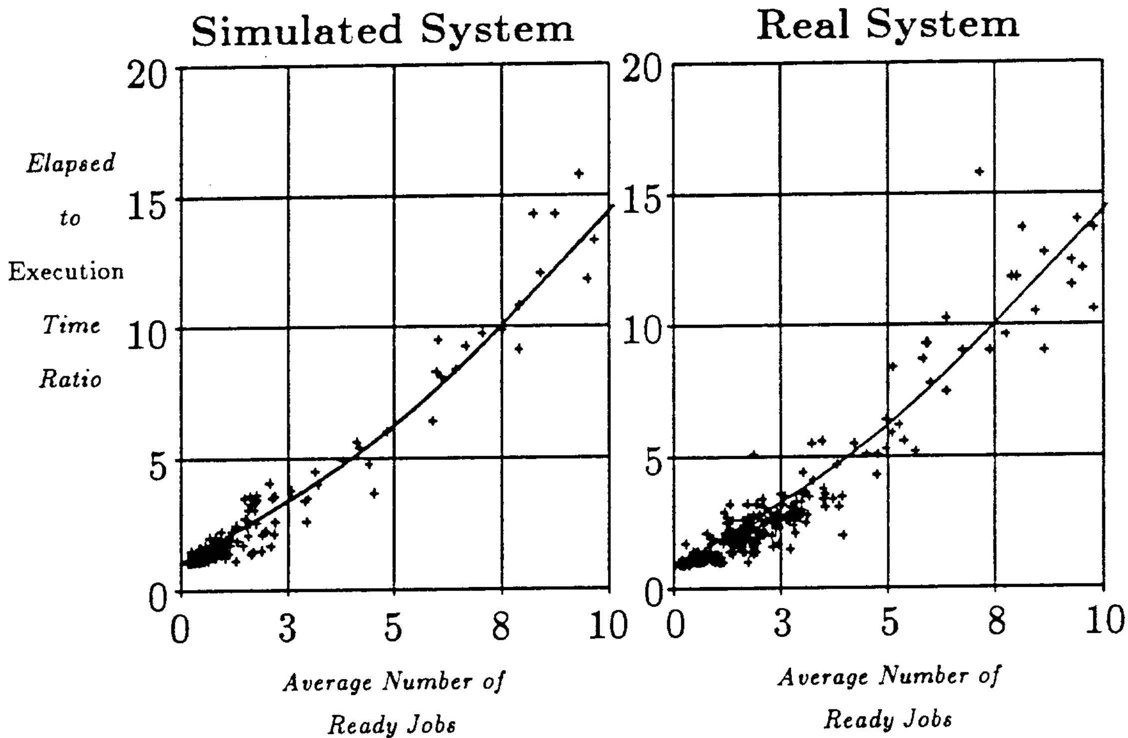


Figure 6.5. **Simulated and real elapsed times.**

In summary, we validated the simulator in two ways, one showing the differences in elapsed times on a per-job basis, the other showing the differences of the elapsed times of a single test job running over a range of different load levels.

6.3. Constants for State Transition and Utility Models

In Chapter 5, we developed a number of parameterized models, constituting an agent's knowledge for load balancing. These models addressed the *general* problem of load balancing; given our experimental system, we now provide the values for the constants appearing in the models.

6.3.1. Abstract State Space

An agent A_i's abstract state indicator $I(x_i)$ was defined as the instantaneous number R_i of jobs ready for execution. This is simply the total number of jobs located in the CPU server, waiting in any of the queues, and including the job currently being executed by the CPU. As this value can change with each clock tick, an agent samples this value every T_s time units to obtain the time series $R_i(n)$, $R_i(n-1)$, $R_i(n-2)$,... A sampling period of

$$T_s = 1/64 \textit{ second},$$

was felt to be sufficient to retain the base frequency component of the time series, and yet impose little overhead due to the sampling itself.

To remove high frequency components, the autoregressive model (5.1), which we repeat here, was used:

$$\overline{R}_i(n) = \omega\cdot\overline{R}_i(n-1) + (1-\omega)\cdot R_i(n).$$

The constant ω was set to

$$\omega = 0.96466162.$$

Choosing this value for ω, it can be shown that any single sample $R_i(n)$ will contribute less than 4% to $\overline{R}_i(n)$. Yet, if $\overline{R}_i(n)$ equals some value r_0, and $R_i(n+k)$ equals a constant value r_1 for 64 periods (one second), $\overline{R}_i(n+64)$ will cover more than 90% of the distance between r_0 and r_1.

These two properties mean that

(1) transient values of $R_i(n)$ have a small effect on $\overline{R}_i(n)$;

(2) $\overline{R}_i(n)$ *tracks* fundamental components of the time series $R_i(n)$ well over relatively short periods of time (i.e., one second).

We use $\overline{R}_i(n)$ to compute $B_i(n)$, the number of ready jobs, as given by (5.2):

$$B_i(n) = ROUND(r_i(n)).$$

This also represents the agent's local abstract state

$$y_i(t) = B_i(n), \quad nT \leqslant t < (n+1)T,$$

and makes up the agent's abstract state space

$$\mathbf{Y}_i = \{0, 1, 2, \ldots, B_{\max}\}.$$

The maximum number of ready jobs observed in our experiments was

$$B_{\max} = 25.$$

The only constraint on $B_{\max}$ is

$$B_{\max} < \beta_{\max},$$

where $\beta_{\max}$ is the hypothesized number of ready jobs which would cause a machine to spend all its time in overhead. We saw in Section 6.2.4 that $\beta_{\max} = 32$; thus, the constraint is satisfied.

The upper half of Figure 6.6 illustrates the variation of the number of ready jobs over an interval of 6.5 minutes for a simulation driven by one of our traces. The variation is representative of that of a highly loaded machine. A less loaded machine would typically show much less activity during any six minutes. The lower half of Figure 6.6 shows the fundamental component of the frequency of this variation, which has a period of approximately one minute. It also shows the *load level* $L_i(n)$, about which the number of ready jobs varies, and the *degree of variability* $V_i(n)$.

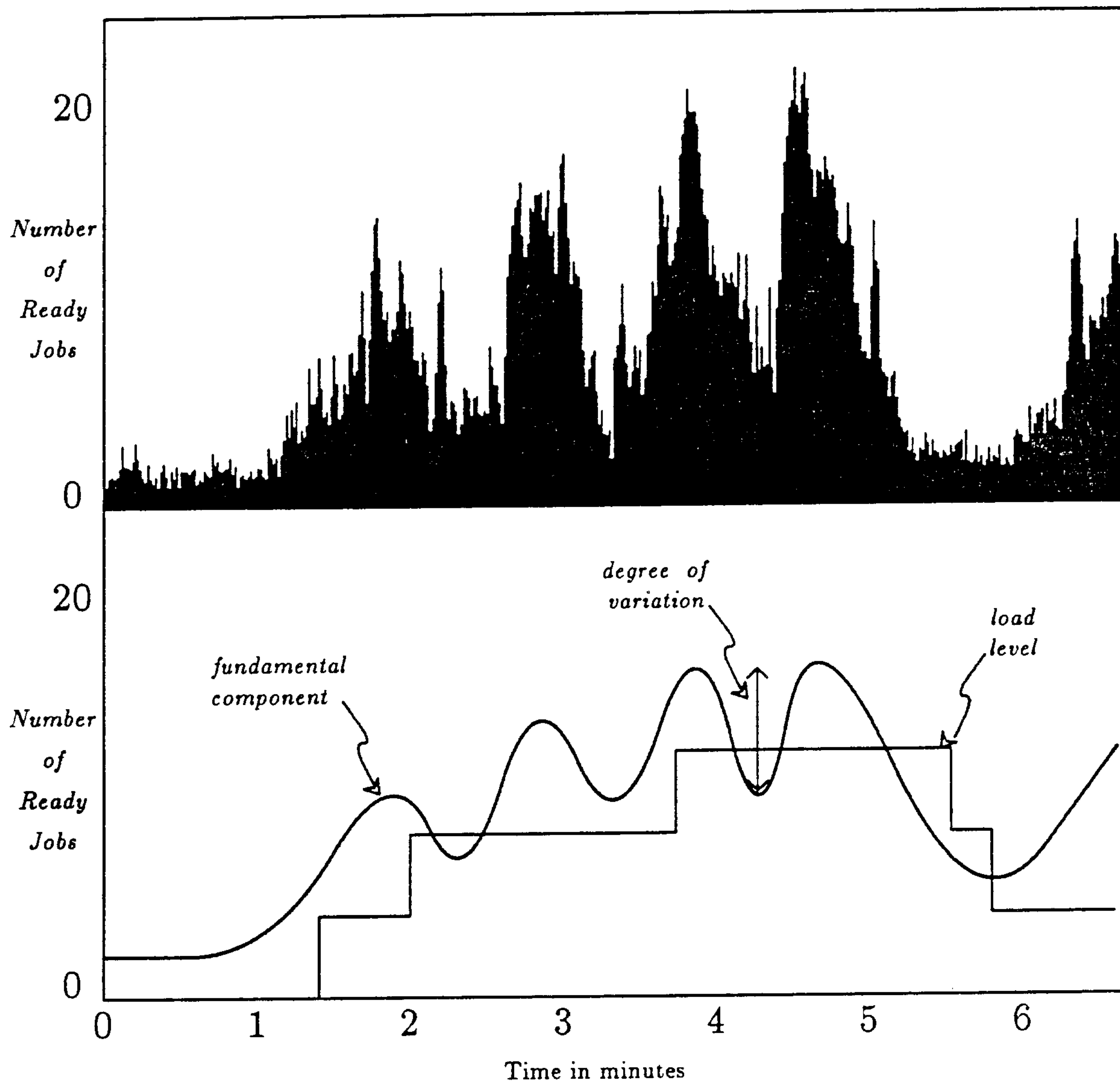


Figure 6.6. Variation in the number of ready jobs.

For the example in Figure 6.6, a sampling period T_s of 1/64 second, to produce samples which are then time-averaged using the autoregressive model (5.2) to yield values of the number of ready jobs spaced one second apart, is sufficient for capturing the fundamental frequency component of R_i.

6.3.2. Load Level and Degree of Variability

To obtain the load level $L_i(n)$, we use a moving average of the number of ready jobs, $B_i(n)$, as given by (5.4):

$$MA_L(n) = \sum_{k=0}^{N_L} \omega_k B_i(n-k),$$

with, in our case,

$$N_L = 60.$$

Thus, the number of ready jobs is averaged over the past one-minute period. The weights ω_k were all set to 1/60, thereby uniformly accounting for each sample over the past minute:

$$\omega_k = \frac{1}{60}, \quad 0 \leqslant k < 60.$$

Using $MA_L(n)$, the load level $L_i(n)$ is given by (5.6):

$$L_i(n) = \begin{cases} ROUND(MA_L(n),\ H_L) & \text{if } MA_L(n) > L_i(n-1) + H_L/2 + h \\ ROUND(MA_L(n),\ H_L) & \text{if } MA_L(n) < L_i(n-1) - H_L/2 - h \\ L_i(n-1) & \textit{otherwise.} \end{cases}$$

with the number of ready jobs considered a significant change in load being

$$H_L = 4.$$

Notice in Figure 6.6 that, in fact, the load level changes with an average period of just more than one minute. And yet, it does represent a long-term average of the number of ready jobs.

The moving average of the absolute differences between the number of ready jobs and the load level, $MA_V(n)$, is given by (5.5):

$$MA_V(n) = \sum_{k=0}^{N_V} \omega_k{}' \,|\, B_i(n-k) - L_i(n-k) \,| \,.$$

with, in our case,

$$N_V = 60.$$

The weights $\omega_k{}'$ were all set to 1/60, as we did for weights ω_k:

$$\omega_k{}' = \frac{1}{60}, \quad 0 \leqslant k < 60.$$

Using $MA_V(n)$, the degree of variability $V_i(n)$ is given by (5.5):

$$V_i(n) = \begin{cases} ROUND(MA_V(n),\ H_V) & \text{if } MA_V(n) > V_i(n-1) + H_V/2 + h \\ ROUND(MA_V(n),\ H_V) & \text{if } MA_V(n) < V_i(n-1) - H_V/2 - h \\ V_i(n-1) & \textit{otherwise.} \end{cases}$$

with H_V, the distance between the differences considered a significant change in

variation about the load level being

$$H_V = 1.$$

6.3.3. State Transition Probability Matrix

We saw in Section 5.4.2 that the parametric model for one-step state transitions, P_v, is a $B_{\max} \times B_{\max}$ matrix given by (5.9):

$$P_v = \begin{bmatrix} \frac{(1+\rho_v)}{2} & \frac{(1-\rho_v)}{2} & 0 & 0 & \cdot & 0 & 0 \\ \frac{(1-\rho_v)}{2} & \rho_v & \frac{(1-\rho_v)}{2} & 0 & \cdot & 0 & 0 \\ 0 & \frac{(1-\rho_v)}{2} & \rho_v & \frac{(1-\rho_v)}{2} & \cdot & 0 & 0 \\ 0 & 0 & \frac{(1-\rho_v)}{2} & \rho_v & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 & \cdot & \rho_v & \frac{(1-\rho_v)}{2} \\ 0 & 0 & 0 & 0 & \cdot & \frac{(1-\rho_v)}{2} & \frac{(1+\rho_v)}{2} \end{bmatrix}.$$

Recall that ρ_v is a decreasing function of V_i as given by (5.10):

$$\rho_v = f(V_i), \quad \frac{d\rho_v}{dV_i} < 0.$$

We now explicitly define this relationship, on the basis of empirical evidence. We conducted approximately 300 single-machine experiments, each driven by a different two-hour trace file. For each experiment, three time-series were generated:

(1) $\tilde{B}_i(n)$, the time-series for the number of ready jobs for trace i;

(2) $\tilde{L}_i(n)$, the time-series for the load level for trace i;

(3) $\tilde{V}_i(n)$, the time-series for the degree of variability for trace i.

We then divided $\tilde{B}_i(n)$, for each trace i, into a number of smaller time-series

$$[\tilde{B}_i(m,n)]_{\lambda v},$$

where m and n define an interval for a sequence of numbers of ready jobs,

$$B_i(m),\ B_i(m+1),\ \cdots,\ B_i(n),$$

such that $L_i(k) = \lambda$, and $V_i(k) = v$, for $m \leqslant k \leqslant n$. Thus, over the interval [m,n], the

load level and the degree of variability remain constant.

The reason for doing this is the assumption that the conditional distribution $p(B_i(n)|\ B_i(n-m))$ is stationary if $L_i(k)$ and $V_i(k)$ are constant for $k \in [m,n]$. (In fact, under these conditions the distribution is considered *second-order or weakly stationary* [Chat85] by definition, since second-order stationarity implies that the mean and the variance of $(B_i(n)|\ B_i(n-m))$ are constant. $L_i(k)$ and $V_i(k)$ are measures of the mean and variance, respectively).

For each time-series $[\tilde{B}_i(m,n)]_{\lambda v}$, we generated a three-dimensional frequency table $f_{\lambda vi}(x,y,z)$, where the (x,y,z) entry indicates the number of times $B_i(k)=x$, given $B_i(k-z)=y$, for $x,y \in \{0,1,...,B_{\max}\}$, and $m \leqslant z \leqslant k \leqslant n$. Tables generated from time-series which had the same load level and degree of variability were then combined (additively) to produce a set of frequency tables

$$F_{\lambda v}(x,y,z) = \sum_i f_{\lambda vi}(x,y,z)$$

From these frequency tables, we verified that the *form* of the state transition probability matrix P_v had a close correspondence; the only remaining task was to determine $\rho_v = f(V_i)$. In our experiments, we observed that V_i took on only five possible values:

$$V_i \in \{0,\ 1,\ 2,\ 3,\ 4\}.$$

For each of these values, we then found a value for ρ_v which provided a best fit (using minimal mean-square error) to the distributions defined by the frequency tables. These were found to be:

$$\rho_0 = 0.993,\ \rho_1 = 0.980,\ \rho_2 = 0.961,\ \rho_3 = 0.934,\ \rho_4 = 0.901.$$

With this, the one-step state transition matrix P_v is completely defined, assuming that $\tilde{V}_i(n)$ for any single machine in the load balancing experiments would not contain values greater than 4. (This was indeed the case.)

Figures 6.7, 6.8, and 6.9, illustrate the effect of the different degrees of variability and of information aging, on the state transition probabilities

$$p(B_j(n) = \beta \mid L_j(n-a_{ji})=\lambda,\ V_j(n-a_{ji})=v),\ 0 \leqslant \beta \leqslant 25.$$

Each figure shows a *family* of state transition distributions, for a given load level λ and degree of variability v. For each figure, the load level λ was set to 4. The degree of variability is different for each figure: in Figure 6.7, $v=0$, $\rho_0=.993$; in Figure 6.8, $v=2$, $\rho_2=.961$; for Figure 6.9, $v=4$, $\rho_4=.901$. Finally, within each figure, distributions are shown for a number of ages of information ranging between 0 and 120 seconds, specifically, $a \in \{0,\ 10,\ 30,\ 60,\ 120\}$.

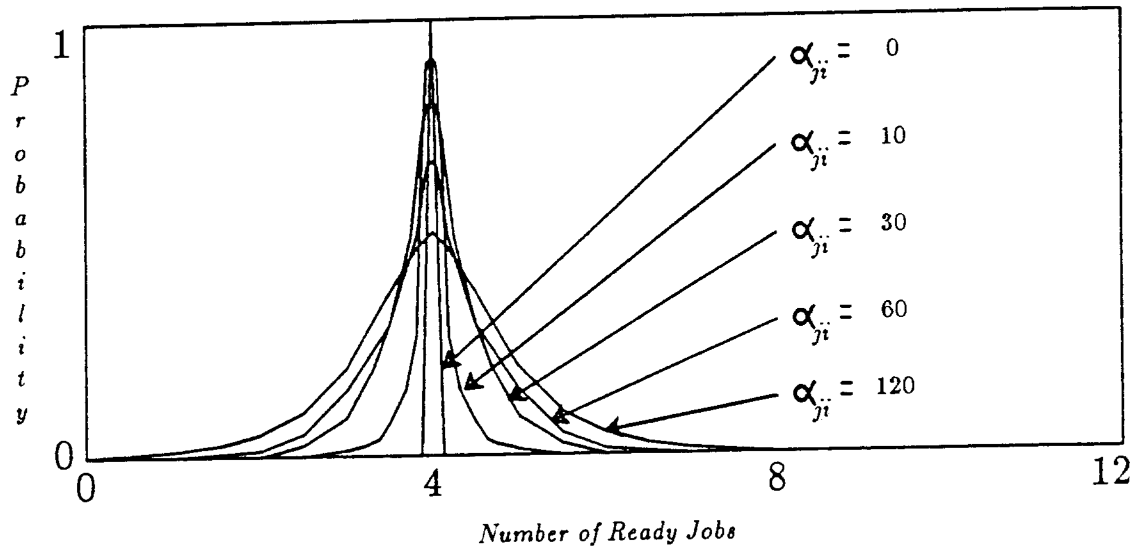


Figure 6.7. Distributions with $\lambda = 4$, $v = 0$.

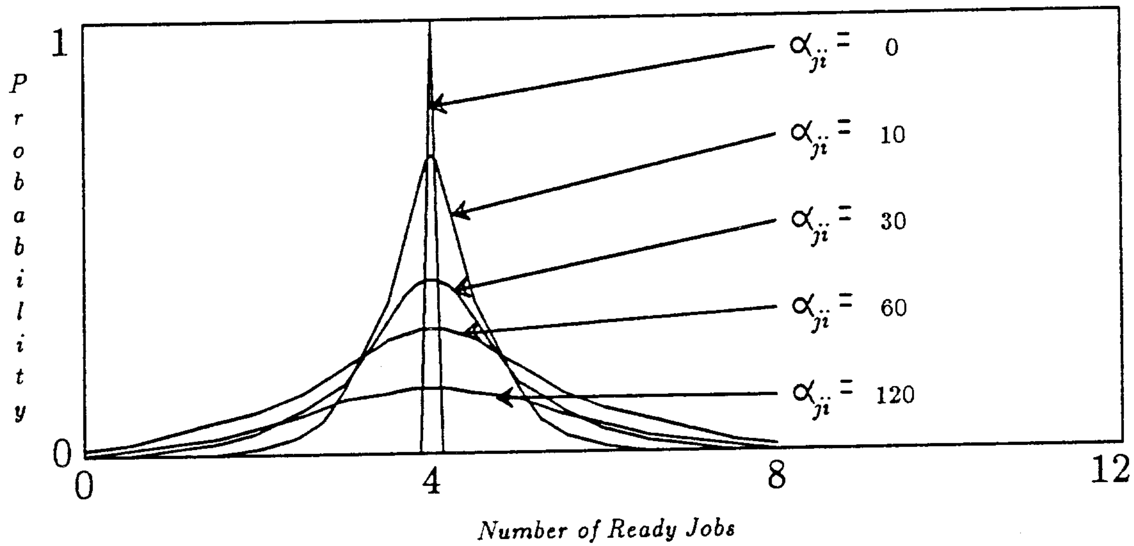


Figure 6.8. Distributions with $\lambda = 4$, $v = 2$.

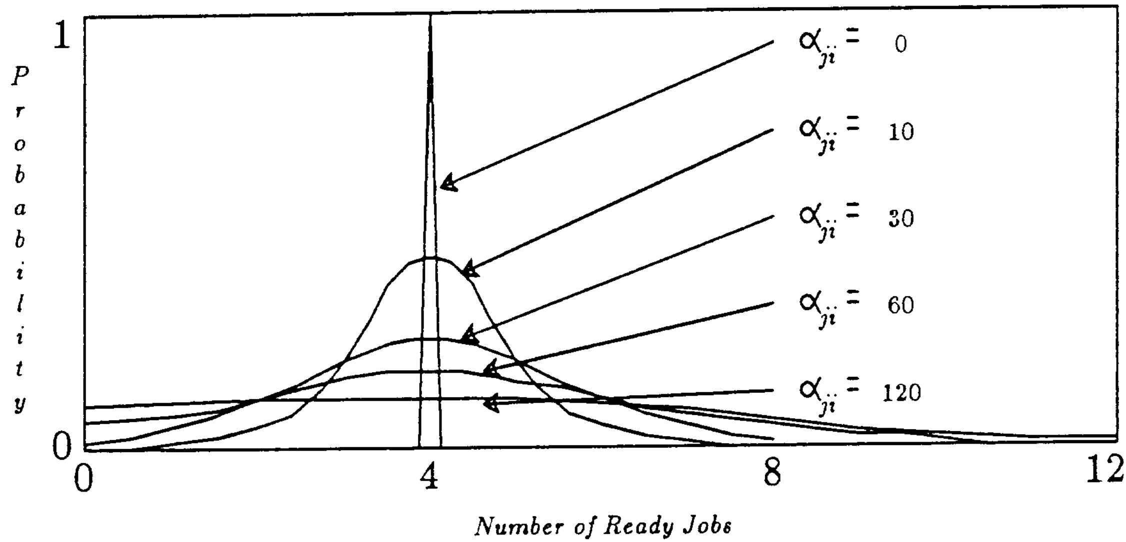


Figure 6.9. Distributions with λ = 4, v = 4.

There are two specific observations to be made about these state transition distribution families. The first observation is that, within one family of distributions, the *width* of the spread about the load level increases as information age increases. This is particularly noticeable when one compares the distribution for age 0 with the distribution for age 120 in any family; the distribution for age 0 has all its mass at the load level, whereas the distribution for age 120 has its mass spread widely about the load level.

The second observation is that, across families of distributions, the *rate* at which the spread widens about the load level increases with the degree of variability v. This can be seen most evidently by comparing the family of distributions with $v = 0$ to the family of distributions with $v = 4$: the width of the spread increases with information age much more rapidly for the latter.

6.3.4. Utility Models

The local state utility of an agent (see Section 5.5.1) is given by (5.15):

$$u_i(\theta) = \mu(\beta) = \frac{-\beta}{1 - \beta/\beta_{\max}}, \quad M(\theta) = \beta.$$

Since we have determined that $\beta_{\max} = 32$, we know the relationship between the number of ready jobs β, and the utility. Note that the domain of $\mu(\beta)$ is the set of integers $\{0, 1, 2,..., B_{\max}\}$, where, as we have also determined, $B_{\max} = 25$.

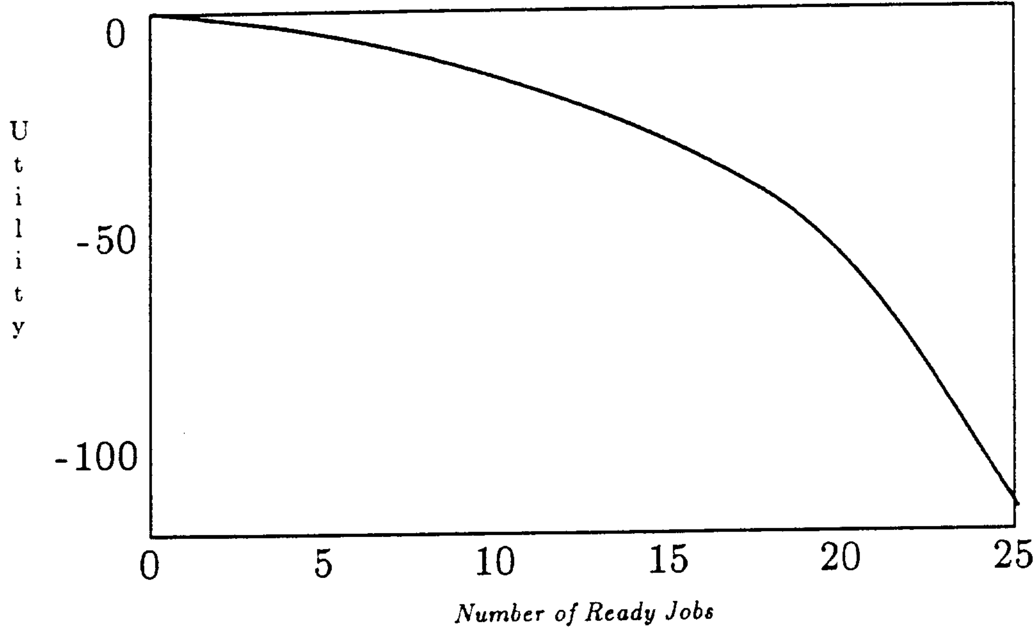


Figure 6.10. Local state utility function.

An agent also needs to determine the future state utility of a remote agent if a job w is offloaded to it. This *conditional* utility of agent A_j, from A_i's viewpoint, given that job w will be offloaded form A_i to A_j, is given by (5.19):

$$u_{ji}(\theta, w) = \begin{cases} \eta_{ij}(w) + \mu(\beta+1), & \beta < B_{max}, \quad M(\theta) = \beta \\ -\infty & \beta = B_{\max} \end{cases}$$

To compute $u_{ji}(\theta, w)$, we need to specify how A_i can compute

$$\eta_{ij}(w) = -\frac{netdelay_{ij}(w)}{elap(w, 0)}.$$

To compute $netdelay_{ij}(w)$, agent A_i needs to know:

(1) the average packet transmission rate between itself and A_j;

(2) the size of job w.

As mentioned earlier, the network topology was randomly generated for each experiment, with the constraint that the average number of neighbors of a machine would be 3. To get a rough idea of the distribution of the number of machines within a given distance from a given agent, we consider a planar graph with an infinite number of nodes, where each node represents a machine, and is adjacent to three other nodes, and analyze the number of nodes within a small neighborhood of a given node.

Let

$$f(d) = \textit{the number of nodes within distance } d \textit{ of the given node.}$$

$f(d)$ must be at least as large as $f(d-1)$, the number of nodes within distance $d-1$. And, each node at distance $d-1$, of which there are $f(d-1) - f(d-2)$, is adjacent to 2 nodes which are at distance d from the given node. Thus, we have the following recurrence relation:

$$f(d) = f(d-1) + 2(f(d-1) - f(d-2)), \quad f(0)=0, \quad f(1)=3.$$

Solving it, we get

$$f(d) = 3(2^d - 1), \; d \geqslant 0.$$

Therefore, we see that, with a unit increase in the distance from the given agent, the number of agents it has to communicate with roughly doubles.

Returning to our experiments, each agent A_i was provided with knowledge about the *distance in hops,* d_{ij}, to every other agent A_j, and the *average packet transmission rate between nodes,* which we set at 64 kilobytes per second. (Similar estimates were observed by [Cabr88] for Berkeley Unix systems using ARPANET protocols. Although this is a low bandwidth considering the high-speed networks available today, we chose it to accentuate the consequences of bad decisions.) Therefore, if an agent knows that the size of a job is $sz(w)$ (in kilobytes), it can determine how long it will take to transmit it to any other agent by computing,

$$netdelay_{ij}(w) = \frac{sz(w) \cdot d_{ij}}{64 \; kbyte/s}$$

Although the trace file did not contain the actual job size and the sizes of the data files which would also have to be shipped (assuming files are not replicated across multiple machines), it did contain the number of disk I/O's a job requested, and its average memory usage. From this information, we computed an approximate job transmission size $sz(w)$. In general, there is no reason why an agent could not know exactly the total transmission size of a job, assuming all code and data files are explicitly identified. The unavailability of the actual job size was a problem for us purely because that size happened to not be recorded in the trace.

To compute $\eta_{ij}(w)$, an agent also needs to know job w's execution time, $elap(w, 0)$. In contrast to the job size, this information is generally *not* available prior to the execution of the job, and yet, it is necessary for estimating the conditional utility, a fundamental quantity needed for rational decisionmaking.

We dealt with this problem by recognizing that, when a job w arrives, it is often the case that w has been executed in the past. If this is the case, an estimate can be made about its execution time, based on its past behavior. This is another example of special-purpose knowledge an agent would have for load balancing.

A good indicator for identifying the same job over a number of executions is its *name*, denoted by $n(w)$, which was recorded in the traces. (As the name of a job is simply the name of the file containing the job's executable code, it certainly is possible to have different jobs with the same name, since file names can be modified over time. Jobs which execute frequently, however, generally retain the same name.)

An agent keeps a list of the names of jobs that have executed in the past. As this information is shared periodically between agents, the jobs could have executed on any machine (and this information is valid for all machines since the machines are homogeneous in our experiments). For each job name, an agent keeps track of the number of times it has executed, plus the mean $m(n(w))$ and the coefficient of variation $c(n(w))$ of the past execution times. The mean $m(n(w))$ is used to construct an estimator for $elap(w, 0)$, and the coefficient of variation $c(n(w))$ provides a measure of the reliability of $m(n(w))$. Agents also keep track of the overall mean execution time of all jobs, $m(\overline{w})$. To estimate $elap(w, 0)$, agents use the following formula:

$$estimate(elap(w, 0)) = \omega^{c(n(w))} m(n(w)) + (1 - \omega^{c(n(w))}) m(\overline{w}), \quad 0 < \omega < 1.$$

The estimate of $elap(w, 0)$ is a weighted sum of the mean execution time of jobs with name $n(w)$, and the mean execution time of all jobs. When the coefficient of variation $c(n(w))$ is small, $m(n(w))$ is emphasized; when $c(n(w))$ is large, $m(\overline{w})$ is emphasized. The best value for ω was found by executing a large number of jobs, recording the actual job execution times, and then minimizing the sum of squared errors

$$\sum_{w} [real(elap(w, 0)) - estimate(elap(w, 0))]^2 .$$

The number of entries (one per job name) in such a list can become very large for each agent. Assuming that availability of memory is not a problem, agents are still faced with the potential problem of highly time-consuming lookup times. To counteract this, agents, in actuality, use *two* data structures for maintaining job information: a balanced binary tree for *information lookup*, and a simple unordered list for *information recording*. The binary tree allows for rapid lookup (requiring $O(\log n)$ comparisons, where n is the number of job name records), and the unordered list allows for rapid recording of job information. When an agent recognizes that the machine on which it resides has no jobs to execute (i.e., it has spare time), it removes a job information record from the unordered list, places it into the binary tree, and does the balancing. (Note that if a job record with the same name already exists in the tree, only a recomputation of the mean and coefficient of variation is necessary.) This repeats until there are no more entries in the unordered list, or until a new job arrives. When a new job arrives, the binary tree data structure must be left in a stable state; thus, the job record currently being placed into the tree is completed, the tree is balanced, and the new job can then begin execution.

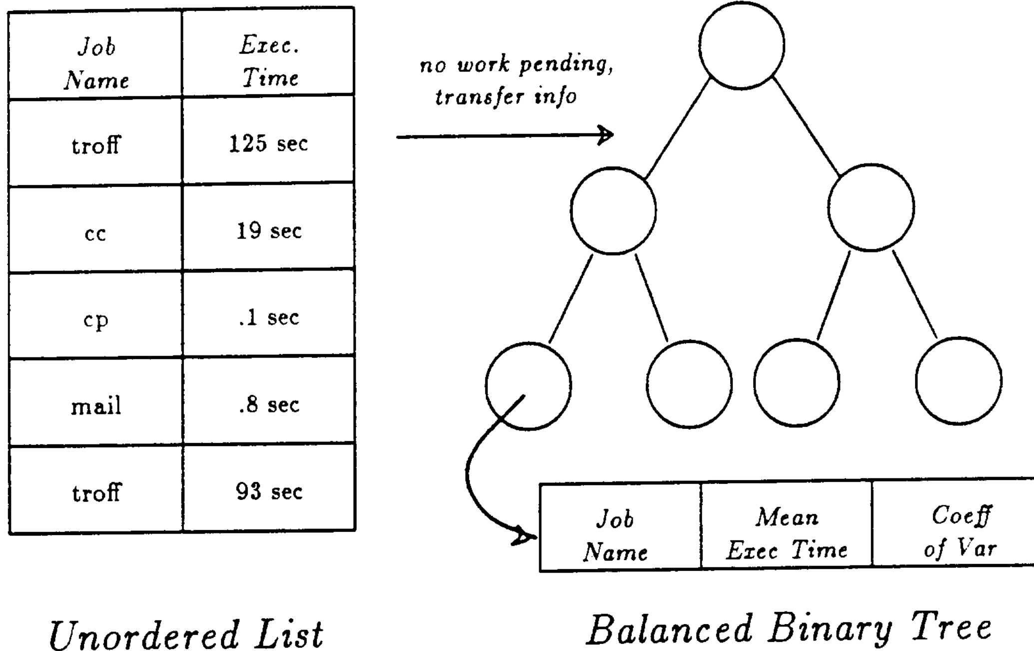


Figure 6.11. Job information data structures.

We make some final observations about an agent's management of job information. Job information can be given to an agent when the agent is created. The agent can add more information dynamically by observing jobs executing on its machine. Agents can learn from each other by sharing information generated on their respective machines. Finally, a human can, at any time, provide additional job information to any agent, which treats this like any other observation, first placing it into its unordered list, and eventually into the binary tree.

As for selecting the optimal time for transferring job information from an unordered list to a binary tree, agents may take advantage of special-case knowledge about how the load varies over time, namely that there will be (often predictable) periods of time (e.g., between 3:00 AM and 6:00 AM) when little or no work is expected, and thus time can be spent reorganizing information so that performance during future periods of high load will be improved. (An agent's objective function could be modified so that the expected future utility is increased by reserving a period of time during which no work is accepted, and information reorganization can take place. Perhaps humans have some similar mechanism for inducing *sleep*.)

In summary, we have shown how an agent obtains estimates for $netdelay_{ij}(w)$ and $elap(w, 0)$, so that it can compute $\eta_{ij}(w)$. This, along with knowledge of the function $\mu(\beta)$, is necessary in computing a remote agent's conditional state utility.

Since the state transition probability matrix P_v has also been explicitly specified, an agent A_i can compute the conditional expected utility (see Section 5.5.4) of remote agent A_j using (5.31).

Let

$$U_1(\lambda, v, a) = \sum_{\beta=0}^{B_{\max}-1} \mu(\beta+1) \cdot [P_v^a]_{\lambda\beta} + \mu(B_{\max}) \cdot [P_v^a]_{\lambda B_{\max}}.$$

$U_1(\lambda, v, a)$ is simply the conditional expected state utility of a remote agent assuming that a job is offloaded there, *ignoring* the network delay, and that the local agent (which is computing the utility) knows the remote agent's load level λ, its degree of variability v, and the age a of the information itself. Since our agents are homogeneous, $U_1(\lambda, v, a)$ is a valid utility model for all of them. ($U_1(\lambda, v, a)$ seems expensive to compute in real time, since it requires a large number of matrix multiplications; we will address this problem shortly.)

We saw earlier how the state transition probabilities varied with the age of information (this generated a family of distributions). We now consider how the conditional expected utility varies with aging information. It is only necessary to analyze $U_1(\lambda, v, a)$, as $\eta_{ij}(w)$ is independent of information age.

Figures 6.12 and 6.13 each show a family of curves for $U_1(\lambda, v, a)$, where in Figure 6.12, the load level is fixed at $\lambda = 0$, while in Figure 6.13 the load level is fixed at $\lambda = 4$. Within each family, each curve corresponds to a different value of $v \in \{0,2,4\}$, and $U_1(\lambda, v, a)$ is varied with respect to information age a. Notice how the utility decreases as information ages for these relatively low load levels.

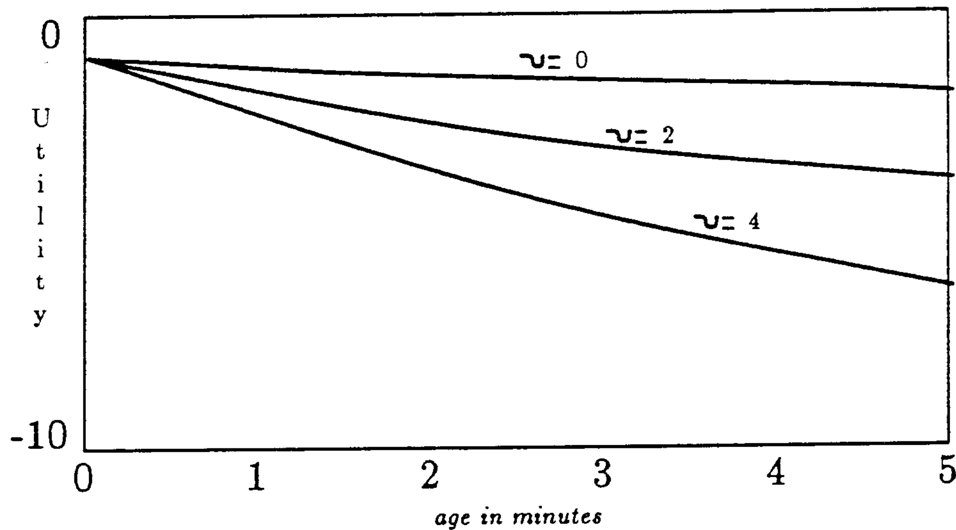


Figure 6.12. Utility curves for λ = 0.

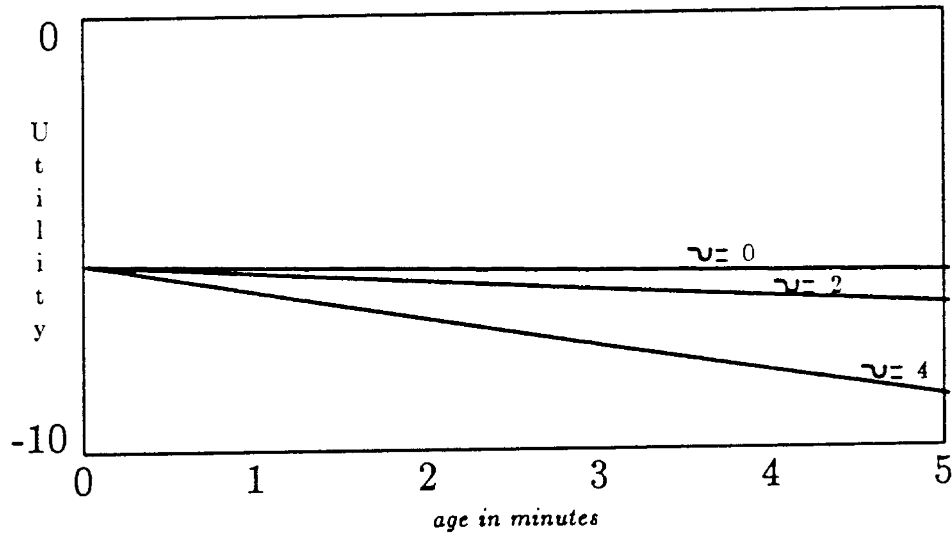


Figure 6.13. Utility curves for λ = 4.

Consider the situation where an agent A_i has information about two remote agents A_j and A_k. A_i's information about A_j is that A_j's load level is 0, and that the degree of variability is 4. A_i's information about A_k is that A_k's load level is 4 and the degree of variability is 0. Information about each has the same age a. The question is: which of the remote agents has the highest expected state utility if a job is offloaded there, ignoring network delays?

Since the A_j's load level is lower than that of A_k, it would seem that A_j should have a better expected state utility. But the information concerning A_j has a higher degree of variability than that of A_k; therefore, as the information ages, the decision-making agent A_i becomes more certain about A_k's state than it is about A_j's state. This is illustrated in Figure 6.14.

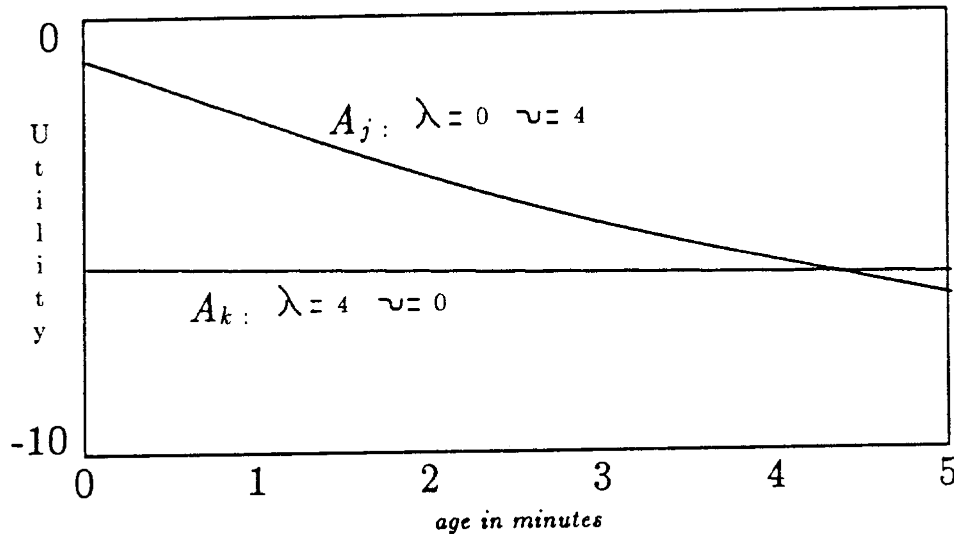


Figure 6.14. Utility curve comparison.

According to this figure, A_j has a higher expected state utility when the information is less than approximately 4.5 minutes old; after that, A_k has a higher expected state utility. This illustrates why it is necessary to base decisions on a comparison of expected utilities, which takes uncertainty of information into account, rather than on a simple comparison of agent states (which in this case would be information about load levels), which ignores information uncertainty.

These examples have ignored network delay, whose effect on state utility, for a given job w, is the quantity $\eta_{ij}(w)$. Based on (5.31), conditional expected utility is given by

$$E[u_{ji}(B_j(n),w) \mid \lambda_j, v_j, a_{ji}] = \eta_{ij}(w) + U_1(\lambda_j, v_j, a_{ji})$$

The effect of $\eta_{ij}(w)$ on the curves reported in Figures 6.12, 6.13, and 6.14 is to translate them downward (since $\eta_{ij}(w)$ is always negative). We shall illustrate this by extending our previous example about the decisionmaking agent A_i and the remote agents A_j and A_k. Suppose that the distance from A_i to A_j is 10 hops (a very large distance), and the distance to A_k is 1 hop. Assume $estimate(elap(w,0)) = 20$ seconds, where w is the job to be possibly offloaded by A_i, and $sz(w) = 700$ kbytes. Since

$$\eta_{ij}(w) = -\frac{sz(w) \cdot d_{ij}/64 \ kbyte/sec}{estimate(elap(w,0))},$$

then, for agent A_j, we have

$$\eta_{ij}(w) = -\frac{700 \cdot 10/64}{20} = -5.47$$

and, for agent A_k,

$$\eta_{ik}(w) = -\frac{700 \cdot 1/64}{20} = -.55$$

The utility curves, assuming that A_j's reported load level is 0 and its degree of variability is 4, and that A_k's reported load level is 4 and its degree of variability is 0, are shown in Figure 6.15.

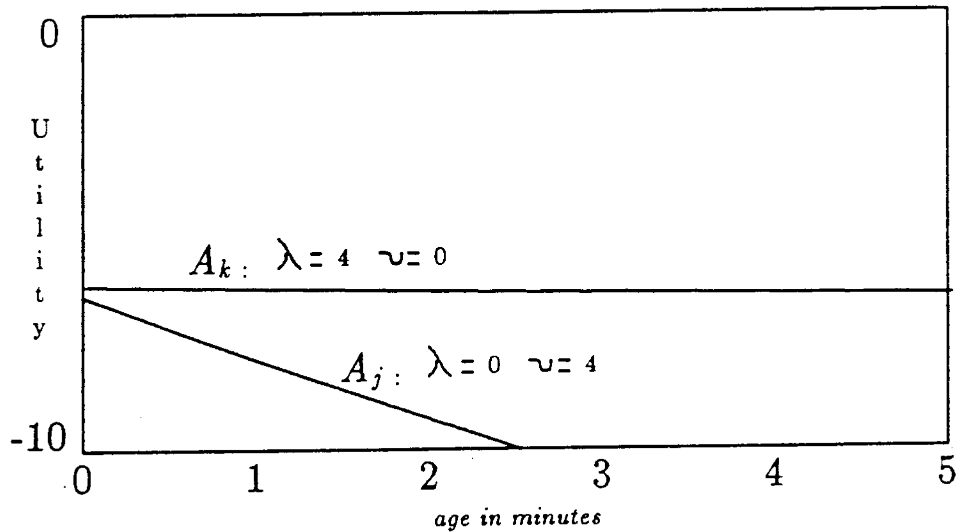


Figure 6.15. Effect of network delay on utility.

A_j's utility curve is translated downward by 5.47 units, and A_k's utility curve is translated downward by .55 units. Thus, when A_i accounts for network delay, A_k's expected state utility is *always* better than A_j's, regardless of the age of information.

6.3.5. Efficient Utility Computations

The function $U_1(\lambda, v, a)$ requires a number of matrix multiplications, a sum of scalar multiplications, and other operations, which can be time-consuming. To avoid this, agents are given a three dimensional table $U_1(\lambda, v, a)$ of *precomputed* values.

How large is this table? For our experiments, we said that the load level can take on values from

$$\lambda \in \{0, 4, 8, 12, 16, 20, 24\}.$$

The degree of variability can take on values from

$$v \in \{0, 1, 2, 3, 4\}.$$

If we measure a, the age of the state information, in seconds for up to five minutes

(this is considered a long time), then

$$a \in \{1, 2, 3, \cdots, 300\}.$$

Therefore, the size of the $U_1(\lambda, v, a)$ table is $7 \times 5 \times 300 = 10{,}500$ entries. Each entry must represent a value for utility, which ranges from -115 to 0 (this comes from $\mu(\beta)$, $0 \leqslant \beta \leqslant 25$), and therefore can consist of a single byte. Thus, our table uses approximately ten kilobytes of memory.

An agent can efficiently compute a conditional expected utility by simply determining $\eta_{ij}(w)$, looking up $U_1(\lambda, v, a)$, and adding them together. The most time consuming part is computing $\eta_{ij}(w)$, which requires a search through a balanced binary tree to obtain $elap(w, 0)$. However, even if the tree contains 1000 job information records (representing the most *common* jobs), the search will require at most 10 comparisons. In fact, an agent A_i will typically compute $\eta_{ij}(w)$ for a number of agents A_j, but they will all use the same value for $elap(w, 0)$; thus, the 10 comparisons are really amortized over multiple conditional expected utility computations.

6.3.6. Efficient Payoff Computations

In Section 5.6.1, we defined the *payoff* $\tilde{\Delta}(\delta_j, w)$ of the decision δ_j (which represented the "offload to A_j" decision) as (see (5.34))

$$E[u_{ji}(y_j(t), w) \mid K_{ji}(t)] + E[u_{ii}(y_i(t), w) \mid K_{ji}(t)] + \sum_{k \neq i, j} E[u_k(y_k(t)) \mid K_{ki}(t)].$$

Theoretically, an agent A_i would compute $\tilde{\Delta}(\delta_j, w)$ for all j, select those of its values which are positive, and make a final selection using the space/time randomization technique described in Section 5.6.2. Unfortunately, this is an extremely time-consuming procedure as just described.

Let us carefully consider what computations must take place. We already know that $E[u_{ji}(y_j(t), w) \mid K_{ji}(t)]$ can be computed quickly by adding $U_1(\lambda_j, v_j, a_{ji})$ and $\eta_{ij}(w)$.

$E[u_{ii}(y_i(t), w) \mid K_{ji}(t)]$ is given by (5.33), which is equivalent to

$$(1 - [P_{v_j}^{a_{ji}}]_{\lambda_j \beta})\mu(\beta_i) + [P_{v_j}^{a_{ji}}]_{\lambda_j \beta}\mu(\beta_i + 1), \quad M(y_i(t)) = \beta_i.$$

In general, an agent will only consider offloading a job to a remote agent whose load level is low. Given an agent A_j with small λ_j, the probability that A_j will be in state $B_{\max}$, $[P_{v_j}^{a_{ji}}]_{\lambda_j \beta}$, will be very small. Therefore, $E[u_{ii}(y_i(t), w) \mid K_{ji}(t)]$ can be approximated by $\mu(\beta_i)$.

We are then left with the sum of expectations

$$\sum_{k \neq i, j} E[u_k(y_k(t)) \mid K_{ki}(t)].$$

This is simply a sum of expected utilities, given by (see (5.24))

$$E[u_k(B_k(n)) \mid \lambda_k, v_k, a_{ki}] = \sum_{\beta=0}^{\beta_{max}} \mu(\beta) \cdot [P_{v_k}^{a_{ki}}]_{\lambda_k \beta}.$$

Rather than computing this expectation, we can create a table of precomputed values similar to that for $U_1(\lambda, v, a)$. Therefore, let

$$U_0(\lambda, v, a) = \sum_{\beta=0}^{\beta_{max}} \mu(\beta) \cdot [P_v^a]_{\lambda\beta}$$

be the table of expected state utilities of an agent whose load level is λ, whose degree of variability is v, and whose information's age is a.

Let

$$U_\Sigma = \sum_{k=1, k \neq i}^{N} U_0(\lambda_k, v_k, a_{ki}).$$

Then,

$$\tilde{\Delta}(\delta_j, w) \approx U_1(\lambda_j, v_j, a_{ji}) + \mu(\beta_i) + (\ U_\Sigma - U_0(\lambda_j, v_j, a_{ji})\ ).$$

This formula is much better in terms of efficiency of computation than the original formula, except that U_Σ still is a potentially very large sum. If we consider that the purpose of computing $\Delta(\delta_j, w)$ for all j is eventually to compare them and select the best ones (i.e., those that are better than the payoff of making the null decision), then since U_Σ is a constant factor added to each payoff, it can be dropped out of the calculation! Consequently, we define the *relative payoff* as

$$\hat{\Delta}(\delta_j, w) = U_1(\lambda_j, v_j, a_{ji}) + \mu(\beta_i) - U_0(\lambda_j, v_j, a_{ji}).$$

This requires three table lookups, one addition, and one subtraction.

Finally, how many payoff computations must an agent make in order to arrive at a load balancing decision? The agent is only interested in remote agents which are lightly loaded. Furthermore, it need not necessarily find the least loaded agent, since it will randomize its decision over a number of underloaded agents anyway. Thus, by grouping agents into underloaded and not-underloaded groups, and then randomly selecting a small number of the underloaded ones (we used 8 in our experiments), the payoffs for offloading jobs to these agents can be determined, and a randomized decision can then be made. Thus, only a small constant number of payoff computations, which are themselves very simple, need to be made (and *not* one payoff computation per remote agent).

6.4. Experimental Results

To evaluate our methods, we conducted a number of experiments for a comparative study of job placement strategies, including our own agent-based strategy. The strategies differ in their costs for state information communication, in their costs for decision procedure computations, in their job transfer costs, and in the degree to

which they minimize average job *elapsed* time, which is the performance index to be optimized.

6.4.1. Types of Job Placement Strategies

The five job placement strategies compared in this study are:

(1) **Perfect Information** [P]: load balancing decisions are made with perfect knowledge of the global system state. When a new job arrives at a machine, a job placement decision is made which takes into account the load on every remote machine, the job transfer time, and the job's expected execution time. In a sense, this is the best decision which can be made given complete certainty about the global system state, but not about concurrent decisions being made by other machines. Therefore, this strategy is immune to the first problem of decentralized control (see Section 3.4), but not to the second.

(2) **Periodic Update** [U]: load balancing decisions are made on the basis of aging information about the global system state. Machines *broadcast* their state to all other machines on a fixed periodic basis. When a new job arrives at a machine, a job placement decision is made which takes into account the load on every remote machine (based on imperfect information), the job transfer time, and the job's expected execution time. The update period is the same for every machine; a number of experiments were carried out with periods varying from 1 second to 5 minutes. The actual periods used were 1, 2, 4, 8, 15, 30, 60, 120, 180, and 300 seconds. This strategy is subject to the two fundamental problems of decentralized control, and makes no attempt to combat them except by periodic refreshing of information.

(3) **Intelligent Agents** [I]: load balancing decisions are based on the principles and techniques described in this dissertation. State information is updated on the basis of our analysis of the tradeoffs between communication overhead and degradation in decision quality due to aging information. Decisions account for uncertainty of information, and expected payoffs based on conditional expected utility. Space/time randomization is used to avoid resonances. This strategy is subject to the two fundamental problems of decentralized control, and combats them in the best ways we believe are possible.

(4) **No Load Balancing** [B]: this strategy does not do any load balancing; i.e., jobs always execute locally. This represents the *baseline* strategy; comparing the other methods to the baseline strategy tells us whether they are better than essentially *doing nothing*.

(5) **Random Decisions** [R]: the random decision strategy is to offload a job to a *randomly* selected remote machine if the local number of ready jobs is above a fixed threshold. The threshold we used was 3, and was determined experimentally to be optimal (i.e., to minimize average job response time) under random decisionmaking in our experimental context. Comparing other methods to this

strategy tells us whether more complex decisionmaking schemes are better than making purely random decisions.

6.4.2. Accounting for Costs

Each of the strategies have different costs. With respect to state information communication costs: [B], [P], and [R] are subject to none; [U] is subject to these costs, and they can be very significant for small update periods; [I] is subject to these costs, but explicitly tries to keep them low.

With respect to job transfer costs, only strategy [B] is not subject to them since it does not do any load balancing. For the other strategies ([P], [U], [I], and [R]), the job transfer costs are the same.

With respect to decision procedure computation costs, strategy [B] is not subject to them because it does not do any load balancing, and strategy [R] is not subject to them because of the simplicity of the random selection decision. Strategies [P] and [U] are subject to these costs, and have the same costs because they use essentially the same decision procedure; their procedures only differ in the quality of the information on which they base their decisions. Finally, strategy [I] has the most complex, and consequently the most costly, decision procedure, even though we will show that the cost is not really significant. Figure 6.16 displays a 3-dimensional cost space, with points for each strategy qualitatively representing their approximate relative costs.

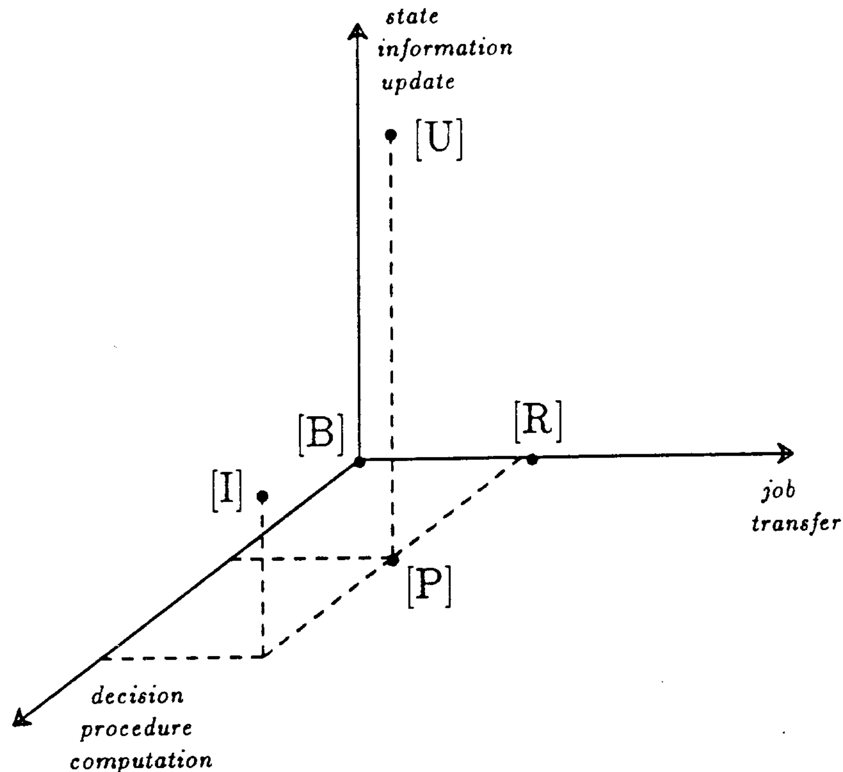


Figure 6.16. **Location of strategies in cost space.**

These costs are explicitly accounted for in our experiments by simulating communication, job transfer, and decision procedure computation times. As already described in Section 6.1.4, communication and job transfer times are based on the amount of data to be sent, the distance between the sender and the receiver, and the communication bandwidth. More interestingly, since all decision procedures correspond to the actual execution of code *within the simulated experiment* (i.e., decisionmaking procedures are *not* simulated), the decision procedure computation time is based on the *real* amount of time it takes to compute the decision. Consequently, simulated time is delayed by the appropriate amounts every time a decision procedure computation is made.

6.4.3. Number and Types of Experiments

We conducted 120 experiments. Each experiment simulated a distributed system comprised of 30 machines, and used a different combination of 30 traces, selected from a total set of 300. Each experiment consisted of a set of simulations, one for each strategy [P], [U] (which included a separate simulation for each update period), [I], [B], and [R]. To limit variations in load distributions between experiments, traces were selected such that the sum of their average loads (see Section 6.1.5), and the

distribution of average loads across machines, were virtually the same. (Within an experiment, the *same* combination of traces was used for each simulation of a different strategy.) On the average, approximately 65000 jobs were executed per experiment. All the results to be presented consist of statistics of performance measures based on all the experiments.

6.4.4. Results

The first experimental result we present addresses how each strategy performs in terms of the average job CPU queueing delay time. Job CPU queueing delay is the total time a job is delayed due to CPU contention with other jobs. If the strategy makes good load balancing decisions, the average CPU queue should be kept low because jobs will be spread more evenly over the various machines.

Figure 6.17 contains a graph showing the improvement in average job delay versus the information update time for each strategy. The other graphs we will present will be of this type. In general, they will show some performance index on the vertical axis (typically expressed as a percentage of the optimal value; the baseline value will correspond to 0%), and the information update period on the horizontal axis, which is scaled logarithmically. *Points* (with lines connecting them) for strategy [U] will be plotted for each update period. Horizontal lines for strategies [P], [I], [B], and [R], all of which are independent of the horizontal axis of the graphs, will be shown for comparison. Each point (for [U]) or horizontal line (for [P], [I], [B], and [R]) represents the *mean value statistic* of the performance index, averaged over 120 runs.

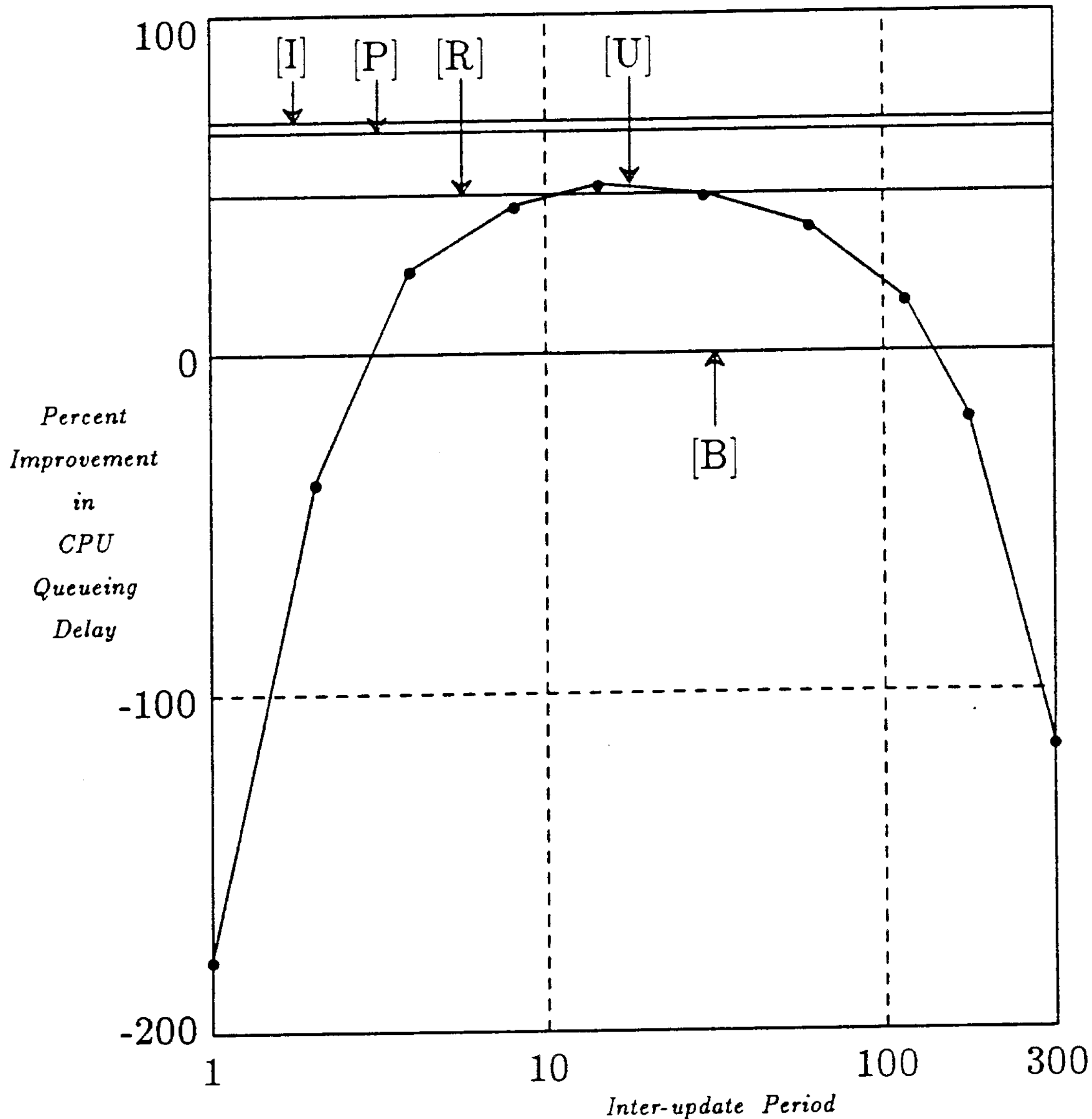


Figure 6.17. Improvement in Average CPU Queueing Delay.

For the graph in Figure 6.17, a 100% improvement in delay corresponds to *no delay*, 0% corresponds to the average job delay for jobs in the baseline experiments using strategy [B]. In absolute terms, each percentage point corresponds to an approximate reduction of 17.5 milliseconds in CPU queueing delay *for every job*.

Strategy [U] has poor performance for very small and very large update periods. For very small update periods, communication overhead costs are very high; for very large update periods, degradation due to decisions based on stale state information is high. For the optimal update period, which is 15 seconds, [U] provides for

approximately a 49.1% improvement in delay.

Percentage Improvement in Average CPU Queueing Delay			
Strategy	% Improvement		
	.05 quantile	mean	.95 quantile
[I] Intelligent Agents	67.6	67.7	67.8
[P] Perfect Information	55.1	63.8	68.1
[U] Periodic Update	38.3	49.1	53.2
[R] Random Placement	46.8	47.5	47.9
[B] No Load Balancing	0.0	0.0	0.0

Table 6.1.

Table 6.1 summarizes the results for each strategy. Strategy [P], which is based on perfect state information, yields a 63.8% improvement in delay, while strategy [I] is slightly better at 67.7%. What is noticeably different between [P] and [I] is the difference in the *variation* of the improvement, indicated by the width of the interval between the .05 and .95 quantiles. [P] has a much wider variation than [I]. In particular, [P]'s .05 quantile is significantly lower than the mean, where [I] does not exhibit this problem. We believe that this is due to [I]'s mechanism for avoiding resonances, which is not present in [P], as this is the only advantage of [I] over [P]. Strategy [R] does quite well relative to [U], with a 47.5 mean improvement in delay, although this is significantly lower than those of [P] or [I]. Notice that the statistical variation for [R] is small, like that for [I], suggesting again that randomization goes far in avoiding resonances.

Although the average job CPU queueing delay is an important performance index, minimizing it does not necessarily insure that the average job elapsed time is minimized. This manifests itself in load balancing systems where job transfer time is a significant cost. Consequently, although offloading of jobs to balance the load causes CPU queues to be shorter on the average, the delay a job experiences during its transfer to other machines will increase its overall elapsed time.

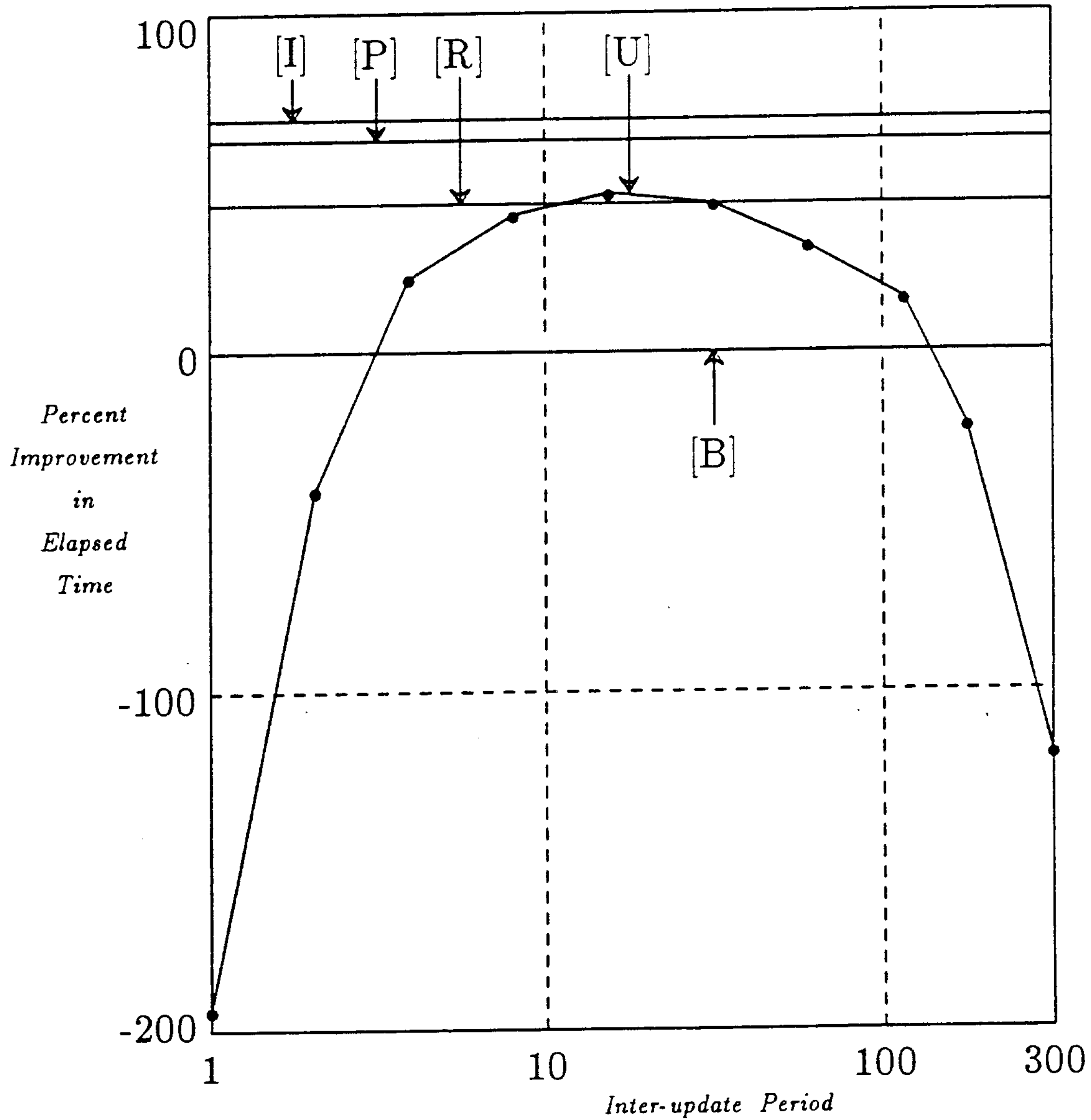


Figure 6.18. Improvement in Average Elapsed Time.

Figure 6.18 is a graph of the improvement in average elapsed time versus the information update period, for each strategy. 100% improvement in average elapsed time means that there are no delays due to CPU queueing, network queueing, or network transmission. Thus, a 100% improvement corresponds to the minimum elapsed times jobs could experience. 0% corresponds to the average job elapsed time in the baseline experiments using strategy [B]. In absolute terms, each percentage point corresponds to an approximate reduction of 228 milliseconds in elapsed time *for every job*. Table 6.2 summarizes the results.

Percentage Improvement in Average Elapsed Time			
Strategy	% Improvement		
	.05 quantile	mean	.95 quantile
[I] Intelligent Agents	67.0	67.1	67.2
[P] Perfect Information	52.8	62.7	67.6
[U] Periodic Update	37.3	48.1	51.7
[R] Random Placement	43.9	44.3	44.8
[B] No Load Balancing	0.0	0.0	0.0

Table 6.2.

As one might expect, the results are very similar to those for the average CPU queueing delay improvement except for the degradation under [R], which is more significant. We see again that, under [U], the performance is poor for very small and very large update periods, with the best period of 15 seconds offering a 48.1% improvement. This is 1.9% less than the CPU queueing delay improvement, suggesting that [U] is less sensitive to the cost of network transmissions and delays than it is to CPU queueing delays (which is true). Under [R], there is a 44.3% improvement, which is 3.2% less than the CPU queueing delay improvement. This is expected since, under [R], machines are not selected on the basis of distance, or on any other criterion; they are simply selected randomly.

Again, the best mean improvements occur under [I] (67.1%), and under [P] (62.7%). In particular, the difference between the improvement under [I] and the CPU queueing delay improvement under [I] is 0.5%; thus, [I] seems to take better into account network transmission delays than the other strategies.

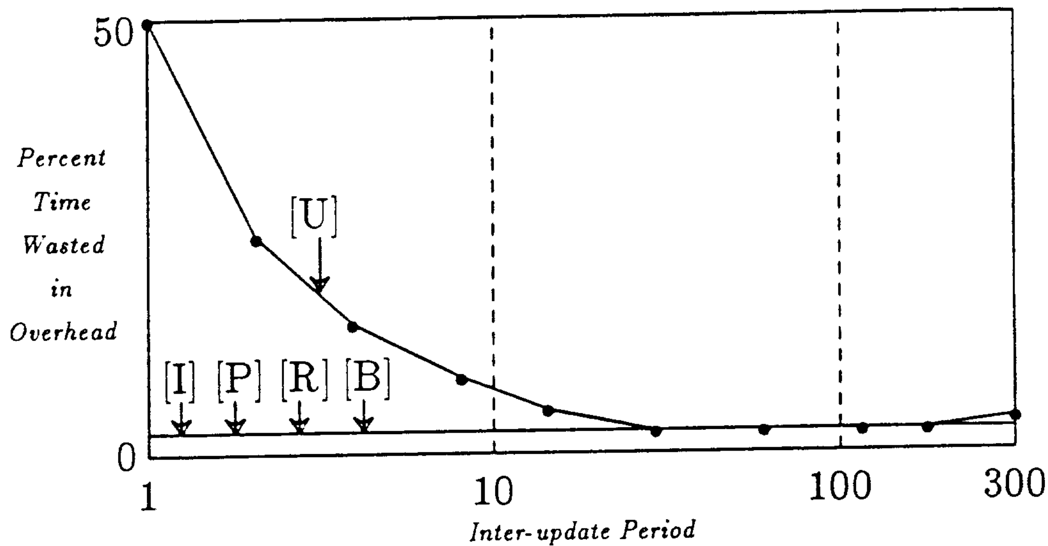


Figure 6.19. Fraction of Time for Overhead.

Figure 6.19 exhibits a graph showing the percentage of time a CPU spends communicating state information and computing decision procedures. Table 6.3 summarizes the results. Under strategy [U] with an update period of 1 second, every machine spends an average of 49.6% of its time for overhead, mostly due to communication. As the update period increases, the overhead is reduced, as expected. For an update period of 60 seconds, the overhead is about 4.0%. For larger update periods, overhead rises a bit due to unbalanced load distributions, which cause some machines to have high loads and consequently to spend more time for local job scheduling (see Section 6.1.4).

Percentage Time Spent for Overhead			
Strategy	% Time Spent		
	.05 quantile	Mean	.95 quantile
[I] Intelligent Agents	3.97	3.98	3.98
[P] Perfect Information	2.85	2.88	3.01
[U] Periodic Update	4.00	4.04	4.31
[R] Random Placement	3.12	3.14	3.17
[B] No Load Balancing	3.70	3.70	3.71

Table 6.3.

For strategies [R], [P], [I], and [B], overhead is less than [U] in all cases. This is not surprising for [B] and [R], which only have local job scheduling overhead, and for [P], which also has to compute its not very complex decision procedure. What is surprising is that, although the overhead for [I] is higher than for all the other strategies except [U], it is still very low in absolute terms. Empirical evidence seems to suggest that intelligent decentralized control is feasible at low cost.

Why does our strategy [I] do so well, and yet impose so little overhead? One of the hypotheses we made at the outset was that, if decisions could be based on information whose reliability could be quantified, and if communication costs could be kept low by updating information only when necessary, we would achieve our goal. It is clear that good decisions are being made under [I], as it surpasses all the other strategies in optimizing the performance index. What can be said about the frequency with which state information is updated?

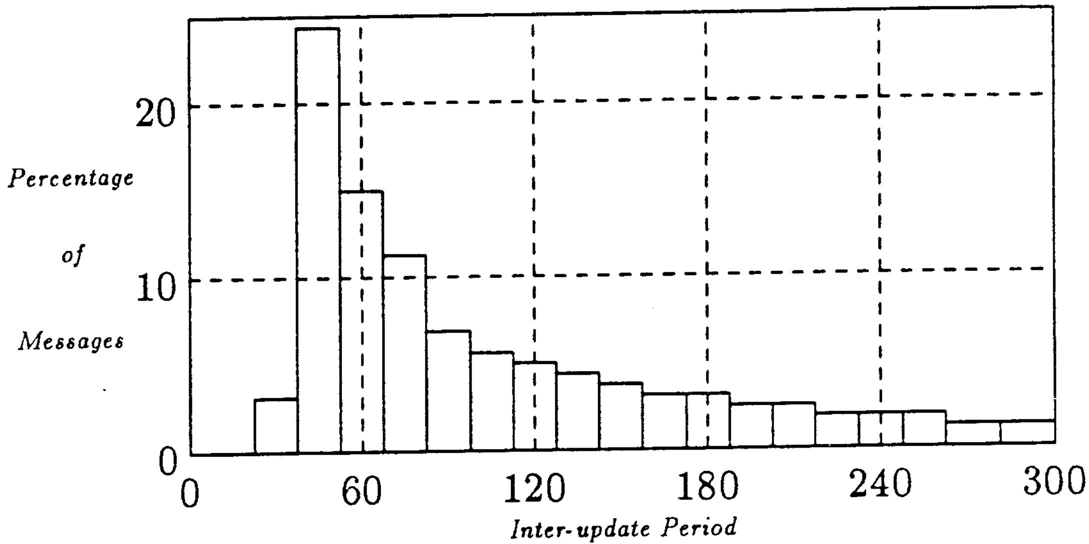


Figure 6.20. Distribution of Update Periods.

Figure 6.20 shows the histogram of update periods between agents under strategy [I]. It shows that update periods under approximately 30 seconds rarely take place. Since, the optimal communication period under [U] was approximately 15 seconds, this suggests that the quantification of information uncertainty and its integration into decisionmaking using conditional expected utility (all of which are lacking under [U]) can dramatically improve decisions. The average update period given by the histogram in Figure 6.20 is 80.16 seconds! This is a dramatic illustration that the principle of frugal state information communication, presented in Section 4.6, is of critical importance in reducing communication overhead, which can be a significant cost in distributed systems. In particular, as the systems get larger, these effects are magnified.

CHAPTER 7

SUMMARY AND CONCLUSIONS

In this chapter, we summarize the main points of this dissertation and we present the major conclusions.

7.1. Summary

In Chapter 3, we presented a formal model for decentralized control, and showed that there are two fundamental problems:

1. **No agent can know with certainty the current global state.**
2. **No agent can know with certainty the current actions of remote agents.**

In Chapter 4, we presented a set of principles for constructing approximate solutions.

- **Adopt a knowledge-based solution**: incorporate all special-case knowledge about the problem as an integral part of the decisionmaking process.
- **Apply knowledge abstraction**: summarize information into a form which can be utilized and communicated more efficiently.
- **Quantify uncertainty**: explicitly account for information uncertainty in decisionmaking.
- **Use directional heuristics**: select decisions based on their tendencies to increase utility.
- **Integrate information aging in decisionmaking**: condition expected state utility on the age of information.
- **Communicate frugally**: communicate only when the cost of the consequences of using out-of-date information in decisionmaking exceeds cost of communication overhead.
- **Avoid resonances using SPACE/TIME randomization**: randomize over the space of good decisions and over the time during which these decisions can be made to avoid mutually conflicting decisions between agents.

Our goal has been to show that, despite the formidable nature of the two fundamental problems of decentralized control, the techniques described in this dissertation can provide acceptable approximate solutions to them. This was demonstrated in Chapter 5 by the effective application of the techniques to the general problem of decentralized load balancing. The main results, presented in Chapter 6, were that

agents can make good decisions (measured by a marked increase in system performance) which do not mutually conflict even though they use uncertain state information. In particular, frequent communication was found to be unnecessary.

7.2. Conclusions

The major conclusions we draw from this research are the following ones.

Correct abstract state design is crucial to efficient decentralized control. The design of the abstract state space has an underlying effect on all aspects of distributed decisionmaking. The abstract state space should have a strong correspondence to the low-level state space partition imposed by the decision space so that decisions can be selected correctly and quickly. It should be small, to minimize the storage required for global state information and reduce the number of terms in the expected utility and payoff computations. If the abstract space is a good one, a simple model for prediction can be constructed (e.g., in the form of a Markov state transition model). Such a model will have slow state transition rates to minimize the need for remote agents to receive state information updates.

Note that it is important to separate measures of states and the utility of states (i.e., the state's value and the state's utility should not generally be one and the same). As the state space design is influenced by the construction of convenient prediction models (e.g., state transition models), and the utility function is influenced by the performance measure to be optimized, combining these influences may not be possible. Forcing an equivalence between state value and state utility will detract from the effectiveness of the state prediction model, or the effectiveness of the performance optimization, or both. Furthermore, the variation of the expected future state and the expected future state utility as a function of aging information will generally be different.

Qualifying state information by quantifying its uncertainty improves decisionmaking.
In general, an agent will regard each item of information about the state of remote agents with varying degrees of uncertainty. Basing decisions not only on what each item of information says, but also on how reliable it is, can have a dramatic effect on improving the quality of decisionmaking. This is due to the general sensitivity of the decisions made by a *large* number of agents over a small interval of time to a relatively *small* number of items of information, namely, the states of the most desirable agents (e.g., agents which have a large capacity for work). The added dimension of reliability of information allows better discrimination of agent utilities.

Formulating expected state utility as a function of aging information is valuable for correctly evaluating alternatives.
Defining the utility of states enables us to use decision theory, which provides a formalism for how a decisionmaker can evaluate alternatives in a statistically optimal manner when the underlying information is uncertain. Decisionmaking is significantly

improved when decisions based on this state information are sensitive to the information's age. The effects of aging information can be incorporated as an integral part of decisionmaking by formulating the expected state utility as a function of the state information's age. This is important for distributed decisionmaking in very large distributed systems where communication costs and delays are significant, and therefore, different items of information will have varying and potentially large ages.

State information communication can often be replaced with inferencing. By making inferences based on past information and predictive models, the need for communication can be reduced significantly. In effect, communication is replaced with local computation, which is a desirable goal in large distributed systems of cooperative agents. This is greatly dependent on the rate at which information becomes stale, and how well this is accounted for by the decisionmaking process.

Space/time randomization is an effective way of avoiding resonances. The advantage of space/time randomization is that it is a cheap decision selection procedure which dramatically reduces the possibility of mutually conflicting decisions. Therefore, it is an effective solution to the second fundamental problem of decentralized control. In particular, as distributed systems become larger and larger, space/time randomization becomes more and more valuable as it avoids reliance on explicit communication.

BIBLIOGRAPHY

[**Abra80**]

S.M. Abraham and Y.K. Dalal, "Techniques for decentralized management of distributed systems," *Proc. 20th COMPCON,* February 1980.

[**Agra82**]

A.K. Agrawala, S.K. Tripathi, and G. Ricart, "Adaptive routing using a virtual waiting time technique," *IEEE Trans. Software Engineering,* vol. SE-8, January 1982.

[**Ande87**]

D.P. Anderson, D. Ferrari, P.V. Rangan, S.-Y. Tzou, "The DASH Project: Issues in the design of very large distributed systems," Computer Science Division, Univ. of Calif., Berkeley, Tech. Report UCB/CSD 87/338, January 1987.

[**Axel84**]

R. Axelrod, *The Evolution of Cooperation.* New York: Basic Books, Inc., 1984.

[**Bash83**]

A.F. Bashir, V. Susarla, K. Vairavan, "A statistical study of the performance of a task scheduling algorithm," *IEEE Trans. Computers,* vol. C-32, no. 8, August 1983.

[**Birr82**]

A. Birrell, R. Levin, R. Needham, and M. Schroeder, "Grapevine: An exercise in distributed computing," *Comm. ACM,* vol. 25, April 1982.

[**Blac54**]
D. Blackwell and M.A. Girshick, *Theory of Games and Statistical Decisions.* New York: Dover, 1954.

[**Boor81**]
R.R. Boorstyn, and A. Livne, "A technique for adaptive routing in networks," *IEEE Trans. Communication,* April 1981.

[**Bore21**]
E. Borel, "La theorie du jeu et les equations integrales a noyau symetrique," *C.R. Acad. Sci. Paris,* vol. 173, 1921. Translated by L.J. Savage in *Econometrica,* vol. 21, 1953.

[**Brya81**]
R.M. Bryant, and R.A. Finkel, "A stable distributed scheduling algorithm," *Proc. 2nd International Conference on Distributed Computing Systems,* April 1981.

[**Cabr86**]
L.-F. Cabrera, "The influence of workload on load balancing strategies," *IBM Technical Report RJ5271 (54311),* August 1986, also appears in *Proc. 1986 Summer Usenix Conference,* June 1986.

[**Cabr88**]
L.-F. Cabrera, E. Hunter, M.J. Karels, D.A. Mosher, "User process communication performance in networks of computers," *IEEE Trans. Software Engineering,* vol. 14, no. 1, January 1988.

[**Camm83**]
S. Cammarata, D. McArthur, and R. Steeb, "Strategies of cooperation in distributed problem solving," *Proc. 8th International Joint Conference on Artificial Intelligence,* Karlsruhe, West Germany, August 1983.

[Casa86]
T.L. Casavant, and J.G. Kuhl, "A formal model of distributed decision-making and its application to distributed load balancing," *Proc. 6th International Conference on Distributed Computing Systems,* May 1986.

[Casa88]
T.L. Casavant and J.G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. Software Engineering,* vol. 14, no. 2, February 1988.

[Chat85]
C. Chatfield, *The Analysis of Time Series (Third Edition)* London: Chapman and Holl, 1985.

[Chu69]
W.W. Chu, "Optimal file allocation in a multiple computing system," *IEEE Trans. Computers,* vol. C-18, October 1969.

[Chu80]
W.W. Chu, L.J. Holoway, W. Lan, and K. Efe, "Task allocation in distributed data processing," *IEEE Computer,* vol. 13, November 1980.

[Chou82]
T.C.K. Chou, and J.A. Abraham, "Load balancing in distributed systems," *IEEE Trans. Software Engineering,* vol. SE-8, no. 4, July 1982.

[Chur61]
C.W. Churchman, *Prediction and Optimal Decisions: Philosophical Issues of a Science of Values.* Englewood Cliffs, NJ: Prentice Hall, 1961.

[Clar80]
D.D. Clark and L. Svobodova, "Design of distributed systems supporting local autonomy," *Proc. 20th COMPCON,* February 1980.

[Cohe78]

G. Cohen, "Optimization by decomposition and coordination: A unified approach," *IEEE Trans. Automatic Control*, vol. AC-23, no. 2, April 1978.

[Cork83]

D.D. Corkill, and V.R. Lesser, "The use of meta-level control for coordination in a distributed problem solving network," *Proc. 8th International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, 1983.

[Cruz87]

J.B. Cruz and A.R. Stubberud, "Knowledge-based approach to multiple control coordination in complex systems," *Proc. IEEE International Symposium on Intelligent Control*, Philadelphia, PA, January, 1987.

[Davi81]

D.W. Davies, E. Holler, E.D. Jensen, S.R. Kimbleton, B.W. Lampson, G. Lelann, K.J. Thurber, and R.W. Watson, *Distributed Systems-Architecture and Implementation, Vol. 105, Lecture Notes in Computer Science.* Berlin: Springer-Verlag, 1981.

[Davi83]

R. Davis, and R.G. Smith, "Negotiation as a metaphor for distributed problem solving," *Artificial Intelligence*, vol. 20, 1983.

[Draz78]

R.J. Drazovich and S. Brooks, "Surveillance integration automation project (SIAP)," *Distributed Sensor Nets Workshop*, Pittsburgh, PA, December 1978.

[Duda79]

R.O. Duda, P.E. Hart, N.J. Nilsson, *Subjective Bayesian Methods for Rule-Based Inference Systems.* Technical Note 124, Artificial Intelligence Center, SRI International, 1976.

[Eage86]

D.L. Eager, E.D. Lazowska, and J. Zahorjan, "A comparison of receiver-initiated and sender-initiated adaptive load sharing," *Performance Evaluation,* vol. 6, no. 1, March 1986.

[Efe82]

K. Efe, "Heuristic models of task assignment scheduling in distributed systems," *IEEE Computer,* vol. 15, June 1982.

[Ensl78]

P.H. Enslow Jr., "What is a distributed data processing system," *IEEE Computer,* vol. 11, no. 1, January 1978.

[Erma80]

L.D. Erman, F. Hayes-Roth, V.R. Lesser, and D.R. Reddy, "The Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty," *Computing Surveys,* vol. 12, no. 2, June 1980.

[Feig63]

E.A. Feigenbaum, and J.A. Feldman, Eds. *Computers and Thought.* New York: McGraw-Hill, 1963.

[Ferr78]

D. Ferrari, *Computer Systems Performance Evaluation.* Englewood Cliffs, NJ: Prentice-Hall, 1978.

[Ferr86]

D. Ferrari and S. Zhou, "A load index for dynamic load balancing," *Proc. 1986 Fall Joint Computer Conference,* Dallas, TX, November 1986.

[Ferr88]

D. Ferrari and S. Zhou, "An empirical investigation of load indices for load balancing applications," in *Performance '87.* Amsterdam: North-Holland, 1988.

[**Gain78**]
B.R. Gaines, "Fuzzy and probability uncertainty logics," *Information and Control,* vol. 38, 1978.

[**Gall77**]
R. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Trans. Communication,* vol. COM-25, January 1977.

[**Gao84**]
C. Gao, J.W.S. Liu, and M. Railey, "Load Balancing Algorithms in Homogeneous Distributed Systems," *1984 International Conference on Parallel Processing,* August 1984.

[**Gene85**]
M.R. Genesereth, M.L. Ginsberg, and J.S. Rosenschein, "Cooperation without communication," *1985 Workshop on Distributed Artificial Intelligence,* Sea Ranch, California, 1985.

[**Giff79**]
D. Gifford, "Violet: An experimental decentralized system," *Operating Systems Review,* vol. 13, no. 5, December 1979.

[**Gins87**]
M.L. Ginsburg, "Decision Procedures," in *Distributed Artificial Intelligence.* London: Pitman, 1986.

[**Gopa87a**]
P.M. Gopal, B.K. Kadaba, and G. Wieber, "Load distribution in packet-switched networks," *Proc. ICC,* June 1987.

[**Gopa87b**]
P.M. Gopal and B.K. Kadaba, "Selective load redistribution in packet-switched networks," *IBM Technical Report RC12707 (57196),* April 1987.

[**Halp84**]

J. Halpern, and Y. Moses, "Knowledge and common knowledge in a distributed environment," *Proc. 3rd Annual ACM Conference on Principles of Distributed Computing,* Vancouver, British Columbia, Canada, 1984.

[**Haye83**]

F. Hayes-Roth, D.A. Waterman, D.B. Lenat, Eds., *Building Expert Systems.* Reading, MA: Addison-Wesley, 1983.

[**Ho80**]

Y.-C. Ho, "Team decision theory and information structures," *Proc. IEEE,* vol. 68, no. 6, June 1980.

[**Howa71**]

R. Howard, *Dynamic Probabilistic Systems.* New York: Wiley, 1971.

[**Huhn87**]

M. Huhns, Ed., *Distributed Artificial Intelligence,* Los Altos, CA: Morgan Kaufmann Publishers, and London: Pitman, 1987.

[**Ishi81**]

M. Ishizuka and J.T.P. Yao, "Inexact Inference for Rule-Based Damage Assessment of Existing Structures," *Proc. 7th International Joint Conference on Artificial Intelligence,* Vancouver, 1981.

[**Jarv75**]

R.A. Jarvis, "Optimization strategies in adaptive control: A selective survey," *IEEE Trans. Systems, Man, and Cybernetics,* vol. SMC-5, January 1975.

[**Jens78**]

D.E. Jensen, "The Honeywell experimental distributed processor: An overview," *IEEE Computer,* vol. 11, no. 1, January 1978.

[Kalm69]

R.E. Kalman, M. Falb, and M. Arbib, *Mathematical System Theory.* New York: McGraw-Hill, 1969.

[Klei80]

L. Kleinrock, and M. Gerla, "Flow control: A comparative survey," *IEEE Trans. Communication,* vol. COM-28, April 1980.

[Kuma87]

A. Kumar, M. Singhal, and M. Liu, "A model for distributed decision-making: An expert system for load balancing in distributed systems," *Proc. 11th Annual International Computer Software and Applications Conference,* Tokyo, Japan, October 1987.

[Lars79]

R.E. Larsen, *Tutorial: Distributed Control,* IEEE Catalog No. EHO 153-7, New York: IEEE Press, 1979.

[Laza87]

A.A. Lazar, J.T. Amenyo and S. Mazumdar, "WIENER: A distributed expert system for dynamic resource allocation in integrated networks," *Proc. IEEE International Symposium on Intelligent Control,* Philadelphia, PA, January, 1987.

[LeLa77]

G. LeLann, "Distributed systems - Towards a formal approach," *Proc. IFIP Congress,* Toronto, Ontario, Canada, August 1977.

[Less78]

V.R. Lesser, and L.D. Erman, "Cooperative distributed problem solving: A new approach for structuring distributed systems," Department of Computing and Information Sciences, Univ. Massachusetts, Amherst, Tech. Report 78-7, May 1978.

[Less80]

V.R. Lesser, and L.D. Erman, "Distributed interpretation: A model and experiment," *IEEE Trans. Computers*, vol. C-29, no. 12, December 1980.

[Less81]

V.R. Lesser, and D.D. Corkill, "Functionally accurate, cooperative distributed systems," *IEEE Trans. Systems, Man, and Cybernetics*, vol. SMC-11, no. 1, January 1981.

[Less83]

V.R. Lesser and D.D. Corkill, "The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks," *AI Magazine*, vol. 4, no. 3, Fall 1983.

[Luce57]

R.D. Luce, and H. Raiffa, *Games and Decisions, Introduction and Critical Survey*. New York: John Wiley and Sons, 1957.

[Malo84]

T.W. Malone, R.E. Fikes, and M.T. Howard, "Enterprise: A market-like task scheduler for distributed computing environments," *Working paper CISR WP 111 (Sloan WP 1537-84)*, Center for Information Systems Research, MIT, Cambridge, MA, 1983.

[Marc59]

J.G. March, and H.A. Simon, *Organizations*. New York: Wiley, 1959.

[Mars55]

J. Marschak, "Elements for a theory of teams," *Management Science*, vol. 1, 1955.

[McQu77]

J.M. McQuillan, and D.C. Walden, "The ARPA network design decisions," *Computer Networks*, vol. 1, August 1977.

[McQu80]
J.M. McQuillan, I. Richer, and E.C. Rosen, "The new routing algorithm for the ARPANET," *IEEE Trans. Communication,* vol. COM-28, May 1980.

[Mesr70]
M.D. Mesrovic, D. Macko, and Y. Takahara, *Theory of Hierarchical Multilevel Systems.* New York: Academic, 1970.

[Metc76]
R.M. Metcalf, and D. Boggs, "Ethernet: Distributed packet-switching for local computer networks," *Comm. ACM,* vol. 19, July 1976.

[Neym33]
J. Neyman and E.S. Pearson, "The testing of statistical hypothesis in relation to probability a priori," *Proc. Cambridge Philosophical Society,* vol. 29, 1933.

[Neum28]
J. von Neumann, "Zur Theorie der Gesellschaftspiele," *Math. Annalen,* vol. 100, 1928.

[Neum47]
J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior.* Princeton, NJ: Princeton University Press, 1947.

[Padu74]
L. Padulo and M.R. Arbib, *System Theory.* Philadelphia, PA: W.B. Saunders Co., 1974.

[Patt73]
H.H. Pattee, Ed. *Hierarchy Theory.* New York: Braziller, 1973.

[**Pear86**]

J. Pearl, "On Evidential Reasoning in a Hierarchy of Hypothesis," *Artificial Intelligence Journal,* vol. 28, no. 1, February 1986.

[**Powe83**]

M.L. Powell and B.P. Miller, "Process migration in DEMOS/MP," *Proc. 9th Symposium on Operating Systems Principles* (OS Review), vol. 17, no. 5, October 1983.

[**Radn62**]

R. Radner, "Team decision problems," *Ann. Math. Stat.,* vol. 33, 1962.

[**Rama84**]

K. Ramamritham, and J.A. Stankovic, "Dynamic task scheduling in distributed real-time systems," *IEEE Software,* vol. 1, no. 3, July 1984.

[**Reit81**]

J. Reiter, *AL/X: An Inference System for Probabilistic Reasoning,* M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1981.

[**Salt78**]

J.H. Saltzer, "Research problems of decentralized systems with largely autonomous nodes," *Operating Systems Review,* vol. 12, no. 1, January 1978.

[**Sand78**]

N.R. Sandell, P. Varaiya, M. Athans, and M. Safonov, "Survey of decentralized control methods for large scale systems," *IEEE Trans. Automatic Control,* vol. AC-23, no. 2, April 1978.

[**Schw80**]

M. Schwartz, and T.E. Stern, "Routing techniques used in computer communication networks," *IEEE Trans. Communication,* vol. COM-28, April 1980.

[**Sega76**]

A. Segall, "Dynamic file assignment in a computer network," *IEEE Trans. Automatic Control*, vol. AC-21, April 1976.

[**Sega77**]

A. Segall, "The modelling of adaptive routing in data-communication networks," *IEEE Trans. Communication*, vol. COM-25, no. 1, January 1977.

[**Sega79**]

A. Segall and N.R. Sandell, "Dynamic file assignment in a computer network-Part II: Decentralized control," *IEEE Trans. Automatic Control*, vol. AC-24, no. 5, October 1979.

[**Shel64**]

W. Shelly II, and G.L. Bryan, Ed. *Human Judgements and Optimality.* New York: Wiley, 1964.

[**Shaf76**]

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

[**Shor75**]

E.H. Shortliffe and B.G. Buchanan, "A model of inexact reasoning in medicine," *Mathematical Biosciences,* vo. 23, 1975.

[**Shor76**]

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

[**Simo57**]

H.A. Simon, *Models of Man.* New York: Wiley, 1957.

[Sing87]

M. Singhal, "On the application of AI in decentralized control: an illustration by mutual exclusion," *Proc. 7th International Conference on Distributed Computing Systems*, September 1987.

[Smit80]

G.R. Smith, "The contract net protocol: High level communication and control in a distributed problem solver," *IEEE Trans. Computers*, vol. C-29, no. 12, December 1980.

[Smit81]

G.R. Smith, and R. Davis, "Frameworks for cooperation in distributed problem solving," *IEEE Trans. Systems, Man, Cybernetics*, vol. SMC-11, no. 1, January 1981.

[Stan82]

J.A. Stankovic, N. Chowdhury, R. Mirchandaney, and I. Sindhu, "An evaluation of the applicability of different mathematical approaches to the analysis of decentralized control algorithms," *Proc. COMPSAC*, November 1982.

[Stan84a]

J.A. Stankovic, "A Perspective on Distributed Computer Systems," *IEEE Trans. Computers*, vol. C-33, No. 12, December 1984.

[Stan84b]

J.A. Stankovic, "Simulations of three adaptive decentralized, job scheduling algorithms," *Computer Networks*, vol. 8, no. 3, June 1984.

[Stan84c]

J.A. Stankovic, and I.S. Sidhu, "An adaptive bidding algorithm for processes, clusters and distributed groups," *Proc. 4th International Conference on Distributed Computing Systems*, May 1984.

[**Stan85**]

J.A. Stankovic, "An applicaton of Bayesian decision theory to decentralized control of job scheduling," *IEEE Trans. Computers,* February, 1985.

[**Stee86**]

R. Steeb, D.J. McArthur, S.J. Cammarata, S. Narain, and W.D. Giarla, "Distributed Problem Solving for Air Fleet Control: Framework and Implementation," in *Expert Systems: Techniques, Tools and Applications,* P. Klahr and D.A. Waterman, eds., Reading, MA: Addison-Wesley, 1986.

[**Ston78a**]

H.S. Stone, "Critical load factors in distributed computer systems," *IEEE Trans. Software Engineering,* vol. SE-4, May 1978.

[**Ston78b**]

H.S. Stone, and S.H. Bokhari, "Control of distributed processes," *IEEE Computer,* vol. 11, July 1978.

[**Suri80**]

R. Suri and Y.-C. Ho, "Resource management for large systems: concepts, algorithms, and an application," *IEEE Trans. Automatic Control,* vol. AC-25, no. 4, August 1980.

[**Tane81**]

A.S. Tanenbaum, *Computer Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

[**Tenn81a**]

R.R. Tenney, and N.R. Sandell, "Structures for distributed decisionmaking," *IEEE Trans. Syst., Man, Cybern.,* vol. SMC-11, No. 8, August 1981.

[**Tenn81b**]

R.R. Tenney, and N.R. Sandell, "Strategies for distributed decisionmaking," *IEEE Trans. Systems, Man, and Cybernetics,* vol. SMC-11, No. 8, August 1981.

[**Tenn81c**]

R.R. Tenney, and N.R. Sandell, "Detection with distributed sensors," *IEEE Trans. Aerospace and Electronic Systems,* vol. AES-17, no. 4, June 1981.

[**VanT68**]

H.L. Van Trees, *Detection, Estimation, and Modulation Theory, vol. 1.* New York: Wiley, 1968.

[**Wald50**]

A. Wald, *Statistical Decision Functions,* New York: Wiley, 1950.

[**Wins84**]

P.H. Winston, *Artificial Intelligence (2nd Edition).* Reading, MA: Addison-Wesley, 1984.

[**Wits68**]

H.S. Witsenhausen, "A counterexample in stochastic optimum control," *SIAM,* vol. 6, no. 1, 1968.

[**Zade65**]

L.A. Zadeh, "Fuzzy sets," *Information and Control,* vol. 8, 1965.

[**Zade77**]

L.A. Zadeh, "Theory of fuzzy reasoning and probability theory vs. possibility theory in decision-making," *Proc. of the Symposium on Fuzzy Set Theory and Applications, IEEE Conference on Decision and Control,* New Orleans, 1977.

[**Zade79**]

L.A. Zadeh, "A Theory of Approximate Reasoning," in *Machine Intelligence.* New York: Wiley, 1979.

[**Zade83**]

L.A. Zadeh, "The role of fuzzy logic in the management of uncertainty in expert systems," *Fuzzy Sets and Systems,* vol. 11, 1983.

[**Zhou87**]

S. Zhou and D. Ferrari, "An experimental study of load balancing performance," *Proc. International Conference on Distributed Systems Principles,* Berlin, September, 1987.

[**Zhou87**]

S. Zhou, "An experimental assessment of resource queue length as load indices," *Proc. Winter USENIX Conference,* Washington, D.C., 1987.