

# Problems of Decentralized Control: Using Randomized Coordination to Deal with Uncertainty and Avoid Conflicts<sup>\*</sup>

*Joseph Pasquale*

pasquale@cs.ucsd.edu

Computer Systems Laboratory  
Department of Computer Science and Engineering  
University of California, San Diego  
La Jolla, CA 92093-0114

## Abstract

We explore certain problems that inhibit coordination in distributed systems based on decentralized control. The most fundamental of these problems are those due to state-information uncertainty and mutually conflicting decisions. We present an integrated view of a series of experiments we carried out in previous studies, showing the common thread of how randomization can be used to directly attack the fundamental problems of decentralized control, and we show the resultant performance gains that are achieved.

## 1 Introduction

Distributed systems based on decentralized control are becoming increasingly important; a good example is the Internet. A well-designed decentralized control system offers the potential of high performance, reliability, and scalability, to a much greater degree than distributed systems with centralized control. However, there are certain significant problems that arise in these systems that inhibit coordination. These problems are fundamental: they are an inherent part of any system where there are multiple decision-makers and communication between them takes significant time. These problems can never be eliminated completely; they can only be dealt with by minimizing the damage that results from them, or by lowering the probability that their negative effects will occur. Indeed, it is a tribute to the power of distributed systems that, despite these problems, it is still worth building systems based on decentralized control.

We will explore the nature of these problems and the value of using *randomization* to attack them. Randomization can support coordination in a number of ways, such as preventing “mutually conflicting” decisions. It can also be used for coordination in cases where it is

---

<sup>\*</sup> This work was supported by the National Science Foundation as part of a Presidential Young Investigator

not known what is the best thing to do, and so you try something (at random), see what happens, and learn so that you do better next time. Finally, randomization is useful when you do not want others to know what you will do, so that they may not take advantage of your decisions for malicious reasons.

As an illustration of the type of problems under study, consider an everyday activity that most of us are familiar with: driving to work. You and many others are driving downtown on a highway during rush hour, on your way to a very important meeting for which you cannot afford to be late. Over the radio, you hear a traffic report that states that there is a major traffic jam a few miles ahead of you. You know of an exit that will allow you to take an alternative route. You have several options:

- you can take the alternate route which, while longer in distance, may take less time because you may be able to travel at a faster speed
- you can stick with your current route, whose traffic may lessen if many other drivers decide to get off the highway and take alternative routes

Unfortunately, you must decide quickly what to do, and then stick with whatever route you think is best, since you will not have time to correct a wrong decision later. How can the situation be prevented where all drivers heading to the same destination decide to take, say, the same route, creating congestion? A possible solution is to somehow spread the traffic more or less evenly over all routes. If congestion persists because there are more cars than available capacity over all routes, an additional solution is for some drivers to voluntarily remove their cars from the traffic. This way, at least some fraction of drivers will successfully reach their destinations in time, rather than all drivers being late. If only we could be so cooperative!

What is needed is a method of *coordination*, “the act of managing interdependencies between activities performed to achieve a goal” [1]. Coordination can be achieved by having each driver randomly decide whether or not to get off the highway (to remove their cars from the traffic). If a driver chooses to remain in the traffic, the driver then selects a route at random (to spread traffic over different routes). This use of randomization allows autonomous decision-makers (the drivers) which have limited shared information (the level and distribution of traffic) to independently allocate from a shared set of resources (the routes) in a minimally conflicting manner.

The problem described above is very common in real distributed systems with decentralized control, where there are many resources and many users of resources, and conflicts between users all trying to simultaneously use the same resources are detrimental to overall performance. Examples include:

- A broadcast communication channel, where a busy station attempts to send when it senses that the channel is free. However, if more than one busy station exists, their transmissions will conflict, resulting in no messages getting through
- A processor load-sharing system, where the goal is to assign jobs to processors such that no processor is heavily loaded while some are lightly loaded. Thus, a processor may decide to move a job to the least-loaded processor. However, if many processors select the same processor to move jobs, it will become overloaded, and the load will remain or become even more unbalanced
- A communication network routing system, where it is desirable to distribute traffic across different routes to avoid congestion. Again, many traffic-generating sources may decide on using the same least-loaded route, resulting in congestion

In the rest of this paper, we first discuss the nature of distributed systems and one of their most important features, their control structure which may be centralized or decentralized (Section 2). We analyze the fundamental problems of decentralized control (Section 3), and how they affect coordination, the central organizing activity of a distributed system that promotes order over chaos (Section 4). We then analyze the positive role that randomization can play in the coordination process (Section 5), and we present examples of using randomization in various experiments on decentralized control (Section 6). Finally, we present our conclusions (Section 7).

## 2 Distributed Systems

Let us begin with basics: what is a distributed system? Very simply (and informally), a *distributed system* is a set of multiple computing nodes connected by a communication network. A *user* can program a subset of the nodes to solve a computational problem. Each node does some computation, nodes communicate intermediate results between each other, and the final result is communicated to the user.

Distributed systems have become popular because of advances in both hardware and software technology. Hardware components such as processors, memories, storage media,

and network transmission media, have become smaller, cheaper, faster, and more reliable, at an ever-increasing rate. Economy of scale favors the construction of computing systems by aggregating commodity parts. The system software infrastructure for taking advantage of the distributed power of these systems has reached a critical level of maturity. In addition to the underlying communication protocols, this includes higher-level control/communication programming abstractions such as the remote procedure call, and organizational structures for distributed software such as the client/server model.

In actuality, all systems are “distributed systems”. This is because, at some low-enough level, one has an organization of components that operate and communicate in a structured manner. What really matters, especially in this paper, is how *control* is distributed in the system. This is generally a function of the speed of communication between the components, relative to the speed at which their states are changing. If communication speed is relatively high, one can have a central state repository that can be kept very current, and a central decision-maker operating based on its inputs and the state as recorded in the repository. This is *centralized control*.

Alternatively, it may be best to distribute control over multiple decision-makers that can work cooperatively, without necessarily giving up autonomy (at least not completely). This is *decentralized control*. With decentralized control, there is no shared state that is simultaneously available to the decision-makers. The rate of state change is comparable to communication time; in particular, by the time you receive a message, the state may have changed.

Decentralized control has numerous advantages, including the following:

- *Reliability*: control can be made redundant so that if any decision-maker fails, others can continue to operate and perform its function. This is the principle of avoiding dependence of the system’s operation on any single point of failure.
- *Performance*: the parallelism derived from multiple decision-making nodes can increase throughput. This is the principle of avoiding bottlenecks by not tying the system’s performance to any particular node, e.g., the slowest node. Furthermore, response is improved by having decision-makers located close to where their decisions are needed. This is the principle of co-locating control and data.
- *Scalability*: because the system is already designed to support multiple decision-making nodes, adding more should not warrant a change in structure. The degree of scalability

especially depends on how flexible the organizational structure is, and whether control relationships are symmetrical and adaptable, rather than hierarchical and fixed.

- *Autonomy*: since control is not constrained to be centralized, a node need not abandon control of itself to some “master” node. Rather, in systems where control relationships are flexible (part of what the organizational structure defines), a node can maintain control over itself when autonomy is important, or decide to share control when group decision-making is most advantageous.

As one might expect, these advantages come at a price.

### 3 The Fundamental Problems of Decentralized Control

There are two fundamental problems that arise in decentralized control systems:

1. **The state-information uncertainty problem**: there is no consistently observable global state describing the entire system since state information is distributed. The state information takes time to collect; the global system state may have changed by the time this happens. Consequently, decision-makers must operate under uncertainty, and make decisions based on possibly incorrect, out-of-date, and conflicting, state information

2. **The mutually conflicting decision problem**: as there are multiple decision-makers, many decisions are made in parallel and may conflict. This is due to a number of reasons:

- A decision-maker may not know the decision procedure of the other decision-makers (a decision procedure is an algorithm that, given a set of inputs and a state, generates a decision, i.e., what action to take given the current observations and circumstances)
- The locally optimal decision may not be globally optimal
- Decision-makers may have differing views of the system state
- Decision-makers may have the same view of a system state which happens to be incorrect

One can try to solve the state-information uncertainty problem by increasing communication between decision-makers. They can exchange their views, and then take part in an agreement protocol to converge on a single system state. However, by the time all this communication happens, the agreed-upon system state may be an outdated view.

Furthermore, communication incurs overhead which can itself negatively affect the system and increase response time, which inhibits effective control.

One can try to solve the mutually conflicting decision problem by endowing each decision-maker with either the same decision procedure, or different ones appropriate to each one but making all of them known to all decision-makers. The decision procedures can be assumed to be common knowledge [2], meaning that everyone (the set of all the decision-makers) knows  $X$  (the set of all the decision procedures), and everyone knows that everyone knows  $X$ , and everyone knows that everyone knows that everyone knows  $X$ , *ad infinitum*.

However, since they are all working with potentially different information, common knowledge does not ensure the optimal group decision. One can try to resolve this problem by using an agreement protocol to “coordinate” decisions (along with views of the system state), with the same problem of overhead and delayed response as mentioned above.

In a real system with autonomous decision-makers, the common knowledge assumption and the use of agreement protocols are often not applicable. Real decision-makers will be limited in their computational capacity, they may have to operate under real-time constraints, and furthermore, one must consider the possibility that they may act maliciously, for selfish or other reasons.

Despite all this, even if the latter-mentioned problems did not exist, one can see that one can never hope to build decentralized systems where all decision-makers make “optimal” decisions. This is because of the two fundamental problems of decentralized control. However, one does not have to give up completely when optimal solutions do not exist. The subject of this paper is to show the value of randomization for efficient coordination in real systems.

## **4 The Necessity of Coordination**

Coordination is the process of decision-makers making choices that are harmonious, that “fit together” so that a common goal is achievable, or is better achieved (e.g., with higher efficiency, in less time, at less cost, etc.). Coordination necessarily requires that the two fundamental problems of decentralized control be addressed.

Let us consider each problem separately. Regarding the state-information uncertainty problem, coordination of decision-makers requires mechanisms for directly observing that portion of the system state that is observable, and mechanisms for communicating with others who know that portion of the system state that is not directly observable. A decision-

maker must be able to decide whether it is best to directly observe or whether to communicate. The manner of communication is important, as indiscriminate communication is bad: some know more than others, and some communication methods are more effective than others.

State information will not be perfectly accurate; it may be incorrect, incomplete, not timely, etc. Consequently, coordination requires methods for determining the *value* of state information. Given state information, how reliable is it? If it describes a past state, what can be said about the current state? If information is missing, can it be inferred from other information?

No single decision-maker will know what is going on exactly, i.e., what the complete current system state is. To estimate of the complete system state, coordination requires mechanisms for the composition of various sources of state information.

Regarding the mutually conflicting decision problem, coordination requires methods for finding a good “group decision”. This may be done dynamically, using a protocol that allows decision-makers to agree on a particular group decision. Alternatively, it may be done statically, by having a prearranged agreement on what decisions to make given various circumstances. This requires an assumption that the decision-making algorithms used by all decision-makers are common knowledge. Unfortunately, even with this assumption, different decision-makers can have different views of the system state (or, they might even have the same, but incorrect view), which can result in a bad group decision.

Because decision-makers are autonomous and have limited capabilities, all that can be expected of them is that they “do the best they can” given difficult conditions; any illusions of achieving “optimal” behavior should be dismissed. In a real system, a decision-maker must base decisions on multiple pieces of state information, from various sources, and of varying quality. Furthermore, these decisions take into account what others may or may not do.

## **5 The Usefulness of Randomization**

Randomization is a powerful concept that can be used to improve coordination. According to Webster’s dictionary [3], randomization is an “arrangement (of samples ...) so as to simulate a chance distribution, reduce interference by irrelevant variables, and yield unbiased statistical data.” As indicated by this definition, the concept of randomization is typically applied to the design of experiments. We use randomization for three reasons:

- to dynamically break symmetry in distributed systems without requiring a statically-asymmetric control structure
- to reduce the need for complex interactions between decision-makers
- to easily incorporate feedback mechanisms

However, the dictionary definition is worth analyzing in the context of coordination and decentralized decision-making. Coordination can itself be considered a series of “experiments”. The goal is to make good decisions, or perhaps more important, at least avoid very bad ones. The procedure is to sample the group decision space in an efficient way, and see what works best. One generates the “hypothesis” that a particular group decision is a good one, and evaluates whether it was indeed good or not.

Let us consider each part of the definition separately.

*“arrangement so as to simulate a chance distribution”*

Consider multiple decision-makers, where each has a set of possible decisions from which to choose. Each must have some method to make this choice. For the moment, assume that there was some virtual master decision-maker, that actually made the choices for all the real decision-makers. This master generates an “arrangement” of choices by selecting a choice from each real decision-maker’s decision set in such a way that this selection process “simulates a chance distribution”.

If the randomization process can be distributed, it can be done by multiple separate and independent processes. In this case, the master decision-maker is indeed only a virtual requirement; the randomization process can be distributed over all the decision-makers, and so no master is really required. The probability distribution defined over all decisions of all decision-makers can be partitioned into separate distributions over each decision-maker’s set of possible decisions. Each decision-maker chooses from its own decision set according to its own probability distribution.

After the decisions are made (i.e., selected and carried out), the experiment continues by observing the results of the decisions. Some metric must be used to determine whether the group decision was good or bad. Based on the result, the individual probability distributions may be updated to improve the future likelihood of good decisions and lessen that of bad decisions. That each decision-maker agrees to execute this algorithm is an important part of coordination that must not be overlooked.

*“reduce interference by irrelevant variables”*

Distributed systems are generally so complicated that it is impossible for the decision procedure(s) to take everything into account that is relevant. More importantly, it is not generally known what is and is not relevant. Fortunately, the randomization process essentially filters out the irrelevant information because it does not rely on it. Furthermore, it introduces “noise” in the decision selection process, consequently lessening the negative effects of relevant bad information, e.g., incorrect views of the system state and bad predictions of what others will do.

There is an assumption here that there are relatively few combinations of decisions that will produce disastrous results. However, these particular combinations often arise exactly in situations where everyone is relying on, say, the same bad information, or that everyone is making the same bad predictions of what others will do. Introducing a little “confusion” can actually help in these situations!

*“yield unbiased statistical data”*

The randomization process basically provides a way of searching the decision space in an unbiased way, or biased toward finding good decisions as the experiments are repeated and one learns from the results. In this sense, the procedure is robust. If we lower our expectations, and focus on simply avoiding very bad decisions rather than trying to find the best decision, this general search procedure works generally very well.

Using a distributed randomization process in a decentralized control system also minimizes the biases of any single decision-maker. (Compare this to the defining bias of a deterministic master decision-maker in a centralized control system.) This is important, not only for the problems discussed above, which are the result of the limitations of decision-makers due to multiplicity and distribution, but also for problems arising from “malicious” behavior. A malicious decision-maker can decide to not play by the rules, and do what is best for itself even if it means that the rest of the system suffers. However, if one does not know exactly what others will do, it is difficult to take advantage of their decisions. This is actually a benefit derived from decentralized control, that one’s decision is a small part of the group decision and so its effect is potentially limited, and from randomization, which introduces a non-determinism that realizes this potential.

## **6 Avoiding Conflicts and Working Together**

We now discuss a series of problems that illustrate the value of using randomization to coordinate autonomous decentralized decision-making agents. All have elements of the general problem of avoiding mutually conflicting decisions, as was illustrated in the traffic example described in the introduction. The problems all consider coordination for resource allocation where the following basic system model, which we will call *the multi-user multi-resource system*, is used. Given  $n$  users and  $r$  resources, each of the users must allocate one of the resources in order to get its work done. The goal is to avoid the situation where many users try to allocate the same resource. The optimal situation occurs when no resource is allocated by more than one user and all the resources are allocated (assuming  $n > r$ ).

In a real system, the mutually conflicting decisions modeled here correspond to access decisions made at nearly the same time (i.e., less than the time interval to communicate a message) by a set of users regarding the same resource. These decisions may conflict because each user is unaware that it is one of many users trying to access the same resource. The experiments consider various scenarios where communication between users may or may not be possible, and where the users may or may not work together in groups.

## **6.1 Using Randomization to Select Resources**

In this first group of experiments, we focus on how randomization is effectively used to avoid the mutually conflicting decisions of many users selecting the same resource. We first discuss its value when there is no communicated information about the dynamic state of the system, and then we discuss the possible improvements when state information is available.

### **6.1.1 Resource Allocation without Communication**

In some systems, decision-makers do not communicate, either because it is too costly or because it simply may not be possible to do so. Yet, to avoid conflicts, they must find a way of coordinating their decisions. Consider a multi-user multi-resource system with no communication between users. All they know are the values of  $n$  and  $r$ , and they must independently make decisions that do not conflict. A simple approach is to use randomization, in particular, *Time-Space Randomization* (TSR) which works as follows. Each user makes a *time* decision: should I try to allocate one of the resources (yes or no)? If the answer is yes, then it makes a *space* decision: which resource shall I select (1, 2, ...,  $r$ )? Both decisions are decided randomly: the time decision is answered in the affirmative with probability  $\alpha$  (called the *access probability*), and each resource is selected with probability  $1/r$  (equal likelihood).

After all users make their decisions, a resource will be either wasted (no one allocated it), congested (more than one user tried to allocate it), or utilized (exactly one user allocated it). Performance is measured by the following formula:  $G = U - \beta C$ , where the gain  $G$  is a measure of performance goodness,  $U$  is the number of utilized resources,  $C$  is the number of congested resources, and  $\beta$  is a parameter used to weigh the importance of congestion relative to utilization. The simple interpretation of this formula is that performance increases linearly with the number of utilized resources, and decreases linearly with the number of congested resources, and the ratio of the rate of decrease with congestion to the rate of increase with utilization is  $\beta$ .

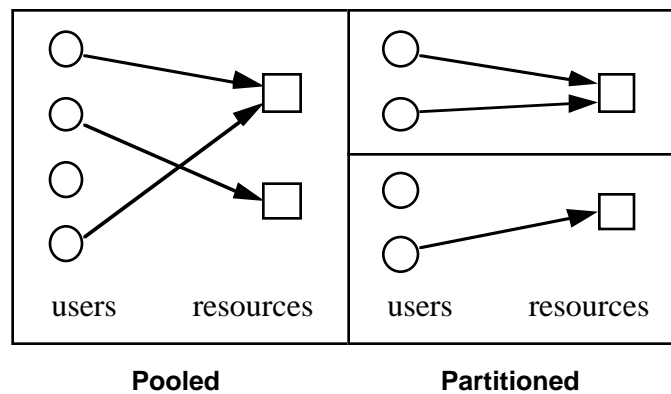
The relevant questions are: what should the value of  $\alpha$  be to maximize performance, and, how good is the performance? This problem was analyzed in [4], which we summarize here. It illustrates the value of using randomization to coordinate decision-makers to improve performance without the aid of communication, and also indicates the limitations of the improvement.

The complete formulas (and proofs showing their validity) for the optimal access probability and the optimal expected gain are contained in [4]. Here, we present certain simple cases that adequately illustrate the points to be made. Consider a system where performance gain was simply a function of the number of utilized resources, i.e.,  $\beta = 0$ , and that resources are scarce relative to users, i.e.,  $n > r$ . The optimal access probability reduces to the simple formula of  $\alpha^* = r/n$ ; in words, decide to access a resource with a probability based on the ratio of the number of resources to the number of users. For only one resource, the formula is very intuitive: access that resource  $1/n$  of the time, as there are  $n-1$  other users trying to access it.

Regarding expected gain, consider a system where there is an extremely large number of users and again with  $\beta = 0$ . Assuming each user limits its access according to the optimal access probability, how good is the performance? First, note that the maximum possible gain is  $r$  (i.e., all the resources are utilized), and the worst is  $0$  (i.e., all are either wasted or congested). This can be determined by evaluating  $EG^*$  as  $n$  goes to infinity. The result is surprisingly simple and elegant:  $EG^* = r/e$ , where  $e$  is the base of natural logarithms. This tells us that, by using randomization to coordinate the users, the system achieves approximately 37% of the maximum possible gain. Is this good or bad? It is good in that one can cheaply (i.e., without communication) achieve a positive level of performance, whereas without any coordination the performance can easily degrade to zero utilization. On

the other hand, 63% of the potential performance is lost, and so while randomization does provide value, its value is certainly limited. While this result is based on an infinite number of users, it turns out that  $EG^*$  quickly approaches the value of  $r/e$  for  $n \gg r$ .

A final interesting question is that of pooling versus partitioning: given  $r > 1$  resources and  $n = kr$  users for some integer  $k \geq 1$ , is it better for all  $n$  users to jointly compete for all  $r$  resources by pooling them, or is it better to partition the system *a priori* into  $r$  subsystems operating in parallel, each subsystem containing only  $k$  users that jointly compete for a single resource? An example is illustrated in Figure 1.



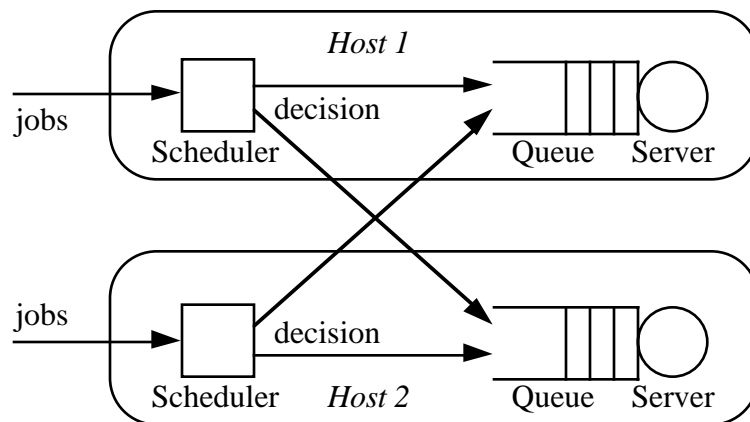
**FIGURE 1. Is it better to pool or to partition?**

The answer is that if coordination is based on TSR, it is always better to *partition*. One might imagine that pooling resources is always better because there is more opportunity for users finding resources to allocate. However, the potential for congestion dominates. By partitioning the system, an implicit coordination is imposed on the users; they are limited to working in small groups where the potential for mutually conflicting decisions is reduced. This raises the question as to whether there are situations where it is better to work in larger groups. We will see that, if there is sufficient communication, there is value to working in larger groups, whereas if communication is insufficient, it is better to limit the potential interactions (explicit and implicit) of decision-makers as much as possible.

### 6.1.2 Resource Allocation with Communication

TSR establishes the baseline performance improvement when there is no communicated state information. As one might expect, the availability of state information can improve performance; but, one must take its quality and the overhead of communication into account to determine whether, in the end, it is worth it.

To illustrate the value of communicated state information, consider a distributed system of  $n$  host computers connected by a network, where the application is load balancing. Each host is composed of a scheduler and a server with a queue. Jobs arrive at each host. When a job arrives, the host's scheduler determines which server should execute the job; this may be the local server, or a remote server. The decision may be based on state information such as the loads, i.e., the number of jobs queued or executing, of the servers, if such information is available. Selecting the server is the key decision here; we evaluate several algorithms below. The job is then sent to the server and is immediately executed if the server is not already executing another job, or is queued for later execution. In the terminology of the multi-user multi-resource model, the "users" are the schedulers and the "resources" are the queue-server pairs. See Figure 2.



**FIGURE 2. Load Balancing**

Note that, unlike the system model for resource access described in the TSR example above, congestion is not fatal. If more than one job is assigned to the same server, the additional jobs are simply queued. However, it is still the case that it is better to balance the load across the servers, and in particular, avoid situations where one server has queued jobs while the other server is idle.

We constructed a simulation model of this distributed system. The study is fully described in [5]. Jobs arrive at each host stochastically, with job arrival and service times exponentially distributed. Most important to this example, there is communicated state information: the loads of the servers are communicated to all the schedulers, but with a delay. The delay is a control parameter of the experiment to evaluate the sensitivity of the various scheduling algorithms to information quality as affected by different amounts of delay.

We evaluated three different scheduling algorithms:

- Simple Randomization (SR): select a server randomly, all with equal likelihood. This is like TSR, except that a scheduler always selects some server (rather than possibly abstaining from accessing, as in TSR).
- Weighted Randomization (WR): select a server randomly, with the likelihood of selecting a server optimally weighted according to relative loads of the servers. A server that is heavily loaded will have a lower weight than one that is lightly loaded.
- Deterministic (DET): select the server that has the least load.

These are all decentralized control algorithms, and each scheduler executes a copy of the same algorithm. WR and DET make use of the communicated state information, while SR does not. Consequently, one would expect system performance, defined by how quickly jobs are able to be serviced, to be better under WR and DET than under SR since they can take advantage of the additional information. However, WR and DET can also lead to worse performance if the information is unreliable (e.g., because it is not timely).

The results are that when the state information is perfect (i.e., no communication delays), DET is best, followed by WR, followed by SR. This is because DET-based schedulers make the best decisions: the least-loaded server is selected based on completely up-to-date information. WR is better than SR because it effectively incorporates the load information in its decision procedure, whereas SR does not. How much better is WR over SR? On a scale of 0-100%, where 0% and 100% are the respective performances of systems under SR and DET, the performance under WR comes in at 68%. This gives us a quantitative measure of the significant improvement one can obtain when enhancing a randomization-based decision algorithm with communicated state information.

However, if the information is not perfect, the relative performance changes. As communication delay increases, performance under DET degrades rapidly. What was the

least-loaded server in the past may be very highly loaded now because all the schedulers are sending jobs to it, all based on out-of-date information. This is a classic case of the mutually conflicting decision problem. For a wide range of non-zero delays, WR does best, followed by SR, followed by DET. Because of randomization, WR and SR avoid always sending to the same server. Finally, when delays are very high, SR does better than WR because it is insensitive to bad (or good) information.

## 6.2 Using Randomization to Search for Coordinated Decisions

In the previous examples, randomization is used as a defensive coordination mechanism, i.e., to avoid mutually conflicting decisions. We now discuss how randomization can be used as an offensive mechanism, to pro-actively search for good collective decisions that support coordination.

We describe a series of experiments where decision-makers randomly search the joint decision space and, based on feedback they receive, learn over time. The basic decision-making model is as follows:

- each decision-maker has a set of choices to select from, and a probability associated with each choice
- each decision is made by randomly selecting one of the choices, weighted according to their respective probabilities
- after decisions are made, each decision-maker is given feedback on whether their decision's overall effect was good or bad
- each decision-maker adjusts the probabilities of the choices, raising the probability of the selected choice (and lowering those of the others) if its effect was good, or lowering its probability (and raising those of other others) if its effect was bad

The feedback may be provided by an external source, such as a server in a load-balancing application, or by something internal to the decision-maker, such as a model of the system that is kept up-to-date via communications. These feedbacks are not necessarily perfect, as they may be delayed or based on information that is not accurate. We explore how these uncertainties affect coordination below.

The scheme for adjusting the choice probabilities is based on the well-developed theory of learning automata [6]. The probability  $p$  of a selected choice is increased to  $p+a(1-p)$  if the

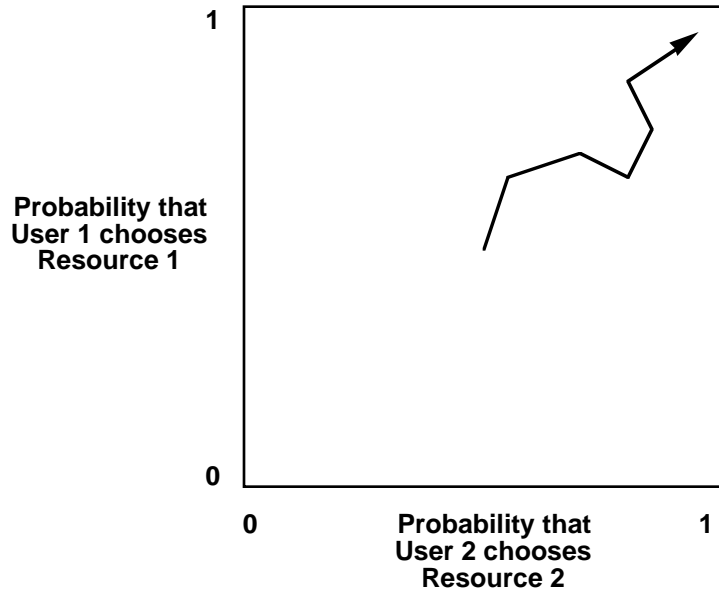
choice was determined to be good, or decreased to  $p(1-b)$  if bad, where  $a$  and  $b$  are learning constants (parameters of the “reward-epsilon-penalty” learning automata scheme). The probabilities of the other choices are proportionally decreased or increased.

It is worth noting that randomization is now being used not only to coordinate the selection of resources, but also to improve the coordination by taking feedback into account. This raises an important point regarding the value of using randomization: since probabilities are used and can be incrementally modified, randomization methods can conveniently incorporate adaptation.

### **6.2.1 Learning Optimal Group Decisions**

Consider a multi-user multi-resource system of two users and two resources, where the users are learning-automata-based decision-makers as described above. The optimal decision for each user is to select a different resource. If both users are initialized such that there is an equal probability of selecting either resource, and if they are then allowed to repeatedly make their decisions, rewarding them when they choose different resources and penalizing them when they choose the same ones, we find that they quickly learn to select different resources.

In effect, the users are searching a two-dimensional decision space, where one axis is the probability that user 1 will choose resource 1 (which also implies that it will choose resource 2 with the complementary probability), and the other axis is the probability that user 2 will choose resource 2 (or that it will choose resource 1 with the complementary probability). See Figure 3. Our studies show that, over time, the search converges on the extreme points of  $(0, 0)$ , i.e., user 1 should select resource 2 and user 2 should select resource 1, or  $(1, 1)$ , i.e., user 1 should select resource 1 and user 2 should select resource 2. By randomizing their decisions, the users are able to probe the decision space and in effect coordinate their decisions. These results are not surprising, given that the feedback provided is based on perfect state information. However, what happens if the feedback is not perfect?



**FIGURE 3. Searching the Joint Decision Space**

We devised a simulation model of this two-user two-resource system where we carried out an experiment similar to that just described; however, we incorporated a delay for communicated state information. This study is described in [7], and expands on the results we present here. The state information comprised the current probabilities of a user's choices. If there is no delay for communication, then each user can know the current probabilities of the other user's choices with complete certainty, and can determine whether its choice, made randomly, was the best one given probabilistic expectations. When this experiment is carried out, the results are as described above: both users learn to make the best joint decisions.

However, while communication is used to improve a decision-maker's knowledge of the system state, there is generally a cost in that the more communication there is, the more overhead there is. On the other hand, the more communication there is, the more up-to-date the received state information is, and the better the decisions will be that are based on this information. So, there is tradeoff between decision quality and overhead that affects performance. The result is that one will never have perfect state information because one cannot communicate constantly, and even if one could, communication is not instantaneous.

In our simulation model, each user communicates its state with the other every  $P$  time units, where  $P$  is the period of communication. Consequently, state information can be as old as  $P$  time units (and actually more, since we also incorporated a transmission delay). By varying  $P$ , we could determine its effect on decision quality, and most important, whether or not the users could learn the optimal decisions as they do under the perfect information scenario.

The main result is that there is a threshold for the communication period whereby, below the threshold the users learn to make optimal decisions (coordination is achieved), and above it coordination breaks down. This threshold, for the simple system described, can be determined analytically (although, despite the simplicity of the system, the analysis is not simple given the complexity of the learning automata probability update scheme), and the analytical results we derived matched the results of the experiments with the simulation model.

Consequently, this study tells us that, even though state information may not be perfect (e.g., because of a non-zero communication period that leads to out-dated state information), randomized decisions over the space of resources are still able to not only cope, but eventually produce near-optimal coordinated behavior by learning over time. However, there is a point beyond which the information becomes so bad that such learning is not possible.

We carried out a similar study, described in [8], of a load-balancing application where the distributed system consisted of two hosts, each composed of a scheduler and a server with a queue. The schedulers make randomized decisions and are based on learning automata, as described above. If a scheduler chooses the server with the shorter queue, it is rewarded, otherwise it is penalized. This feedback is provided by “the system”, and is not delayed. The result of this study is that the schedulers are able to “co-adapt” to eventually make good decisions. The more one scheduler favors a particular server, the more the other scheduler favors the other server.

However, this system is significantly more complicated than the simple two-user two-resource system described above, since job arrival times and service times in the load-balancing system are stochastic. Consequently, while the decisions the schedulers learn to make are generally good, they are not always optimal, due to these uncertainties. On a scale of 0-100%, where 0% is the performance of a system where no off-loading was possible (i.e., when a job arrives at a host, it is executed on its own server) and 100% is that of a system where the least-loaded server was always selected, the co-adaptive system achieves a performance level of 44%. This shows the significant degree to which uncertainty, which is

directly traceable to the two fundamental problems of decentralized control, will degrade performance, despite the randomization-based coordination mechanisms we apply.

### **6.2.2 Learning to Work Together or Work Alone**

In the previous examples, it was determined that with sufficient communication a set of decision-makers can learn to make optimal (or at least good) decisions that achieve coordination. Without sufficient communication, coordination breaks down; in this case, it would be better if the system was partitioned to limit interactions between decision-makers. However, we can incorporate this problem as yet another decision that must be considered: should a decision-maker work in small groups (i.e., partition the system) or should it work as part of one large group? Furthermore, can the optimal choice of working in small groups or in one large group be learned over time?

The smallest possible group is one that includes only a single decision-maker; this corresponds to working alone. Consider the value of working alone: performance is more predictable since one does not depend on the decisions of others. However, by working in a large group, i.e., working together, and accessing all resources from a single shared pool, there is the potential for achieving much higher levels of performance. This is the result of greater opportunities for sharing and better utilization of resources. We saw the value of working alone in the discussion of using randomization without communication: in such a system, despite the implicit coordination achieved by the use of randomization, it is still better to partition resources than to pool them. However, with communication resulting in the availability of timely but not necessarily perfect state information, higher levels of performance are achievable, as shown in the other examples.

To investigate these issues, we constructed a simulation model of a distributed system of two hosts (each composed of a scheduler, and a server with a queue) with the goal of load balancing, where each host decides whether it wishes to work with the other host or not. This study is described in [9]. As in the previous examples, the choices have associated probabilities that are updated according to the learning automata scheme, and the decision is made randomly, weighted according to the probabilities. If both hosts decide to work together, then both servers are accessible to each scheduler, and non-conflicting decisions generate high performance while mutually conflicting decisions result in low performance. If either or both hosts decide not to work together, then the system is partitioned into two separate single-host systems, each having “medium” performance, i.e., between the high and low performance of a two-host system. So, should the hosts “play it safe” by working

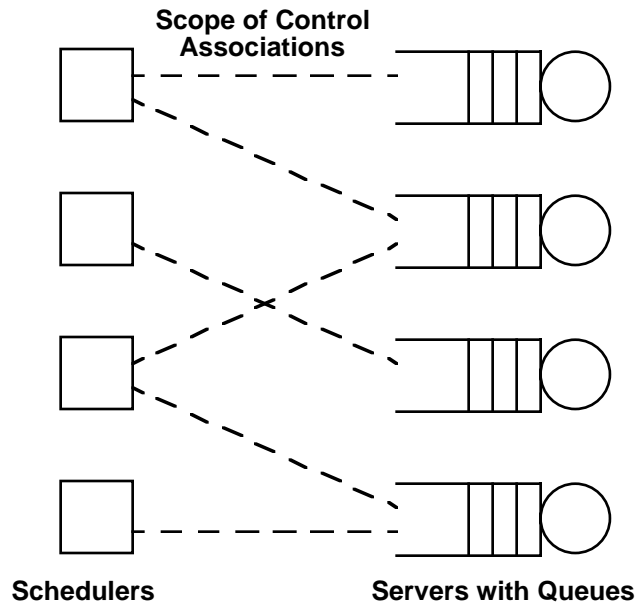
alone, or risk working together to attain the higher potential performance? Uncertainty is introduced by having hosts communicate periodically: the greater the period, the more out-of-date state information can get.

The results of this study are that the hosts learn to do what is best: with sufficient communication, they learn that working together is best and so that is what they do; with insufficient communication, they learn that it is best to work alone. Consequently, the hosts are able to take the value of state information into account. If its value is high, then coordination is possible, and it is attempted; if it is low, coordination is not possible, and so it is not attempted.

### **6.2.3 Learning How Large Working Groups Should Be**

In a follow-up study described in [10], we considered distributed systems of many ( $n > 2$ ) hosts. Now, the decision is not the simple binary one of either working together or working alone. The optimal working group size can range from one (work alone) to  $n$  (work together), or any value in between.

We developed the concept of “scope of control”. In the context of load balancing, a host controls another host if the former’s scheduler is able to send jobs to the latter’s server (with possible queueing). A host’s scope of control is the set of all hosts it controls. We use randomization to determine the control sets: each host  $i$  maintains a set of probabilities  $c_{ij} = \text{Prob}(\text{host } i \text{ controls host } j)$ , for all hosts  $j$ . If  $c_{ij} = 0$  for all  $j$ , we have a partitioned system. Otherwise, we have various degrees of a pooled or shared resource system.



**FIGURE 4. Scope of Control**

Load balancing decisions are made as follows. Each time a job arrives at a host  $i$ , a control set is determined: for each host  $j$ , randomly decide if it is in the control set according to its current probability  $c_{ij}$ . The scheduler then selects the least-loaded host in the control set, and the job is sent to that host's server.

To learn "optimal" control sets, the control set probabilities are updated according to the learning automata scheme. If the host assignment is a good decision, then the probability of including that host in the control set in the future is increased; a bad decision results in a lowering of the probability. A "good" decision is defined as one that results in better-than-average performance, i.e., the job completes execution faster than what is expected in a completely balanced system.

The hosts communicate their state information with each other, i.e., they exchange their control set probabilities. A host uses the received probabilities of other hosts to determine whether the decision it makes is expected to be good; this is what is actually used to reward or penalize decisions. Finally, as in our previous experiments, uncertainty is introduced by only periodically communicating state information, so that it becomes less timely until a new update is received.

The main result is that as the communication period increases (i.e., as more uncertainty is introduced), the average sizes of the control sets decrease. When there is very low uncertainty, the hosts learn that it is best to work together in one large group (any host can off-load jobs to any other host). As the uncertainty increases, it is best to work in smaller, but not necessarily singleton, groups (hosts off-load only to other hosts within their group). Finally, when there is high uncertainty, it is best to work alone (no off-loading of jobs).

Consequently, the scope-of-control randomization mechanism provides a way for decision-makers to take uncertainty of state information into account. They learn by using randomization to search the group (or control set) space, allowing them to act as a self-organizing system. In fact, the mechanism is very robust. Extending the simulated system by having heterogeneous hosts, i.e., some hosts have faster servers than others, the system self-organizes so that the hosts with faster servers appear more frequently in control sets than those with slower servers. In another related study described in [11], where the distance between hosts is taken into account (the greater the distance, the more communication delay, and hence the greater the uncertainty), the system self-organizes so that hosts prefer to work with those hosts that are in closer proximity than other hosts.

## **7 Conclusions**

We have shown that randomization can be used to attack the two fundamental problems of decentralized control, that of state-information uncertainty and that of mutually conflicting decisions. These problems inhibit the coordination required to achieve the potentially high levels of performance offered by distributed systems. Using randomization is very appealing because it simplifies the interactions to achieve coordination between decision-makers. Randomization can be used as part of defensive mechanisms to prevent mutually conflicting decisions by breaking symmetry. Furthermore, by using randomization, one can easily incorporate adaptive feedback mechanisms to allow a decentralized control system to self-organize into a cooperating whole. This allows randomization to be used to pro-actively search for good collective decisions that are coordinated, in contrast to its complementary use in defensive mechanisms.

We reviewed a number of studies where randomization was used to improve the performance of various decentralized control systems given different constraints. With no communication, randomization provides a significant but limited performance improvement. The limitation does not make the pooling of resources worthwhile; rather, it is better to statically partition the system. This changes when communication is possible and where the

communicated information is of sufficient quality. Where communication of state information is periodic, there is generally a threshold for the communication period whereby, below the threshold, decision-makers learn to eventually make optimal decisions (coordination is achieved), and above it coordination breaks down.

Decision-makers can learn whether it is worth working together (which requires coordination) or working alone (which does not), based on the quality of the communicated state information. By working together, higher levels of performance are achievable. By working alone, performance is more predictable because decision-makers do not interact, but the performance gain is significantly more modest. In fact, they can learn the optimal size for coordinated groups; this size is directly proportional to the quality of communicated state information.

Yet, despite the power of using randomization, we see very clearly throughout our studies the significant degree to which uncertainty will degrade performance. The two fundamental problems of decentralized control are the roots of this uncertainty; it is only by gaining a deeper understanding of them that we will be able to develop additional methods to realize the potential power of distributed systems.

## **8 Acknowledgments**

I am grateful to my students and colleagues, especially Ted Billard and Alex Glockner, with whom I explored many of these issues. Most importantly, I thank the late Dr. Larry Rosenberg for his support and friendship. Larry's vision and enthusiasm for a research agenda in coordination and collaboration remain an inspiration to me.

## **9 References**

- [1] T. W. Malone and K. Crowston, "What is coordination theory and how can it help design cooperative work systems," *Proc. ACM CSCW Conf.*, Los Angeles, October 1990.
- [2] J. Halpern and Y. Moses, "Knowledge and common knowledge in a distributed environment," *Journal of the ACM*, Vol. 37, pp. 549-587, July 1990.
- [3] Webster's Ninth New Collegiate Dictionary. Springfield, MA: Merriam Webster Inc., 1987.
- [4] J. Pasquale, "Randomized coordination in an autonomous decentralized system," *Proc. 1st IEEE Intl. Symp. on Autonomous Decentralized Systems (ISADS)*, Kawasaki, Japan, March 93, pp. 77-82.

- [5] E. Billard and J. Pasquale, "Utilizing local and global queueing resources with uncertainty in state and service," *Proc. 2nd IEEE Intl. Symp. on Autonomous Decentralized Systems (ISADS)*, Phoenix, April 1995, pp. 258-265.
- [6] K. Narendra and M. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
- [7] E. Billard and J. Pasquale, "Adaptive coordination in distributed systems with delayed communication," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 25, No. 4, April 1995, pp. 546-554.
- [8] A. Glockner and J. Pasquale, "Coadaptive behavior in a simple distributed job scheduling system," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 3, May/June 93, pp. 902-906.
- [9] E. Billard and J. Pasquale, "Effects of delayed communication in dynamic group formation," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 5, September/October 93, pp. 1265-1275.
- [10] E. Billard and J. Pasquale, "Dynamic scope of control in decentralized job scheduling," *Proc. 1st IEEE Intl. Symp. on Autonomous Decentralized Systems (ISADS)*, Kawasaki, Japan, March 93, pp. 183-189.
- [11] E. Billard and J. Pasquale, "Localized decision making and the value of information in decentralized control," *Proc. 7th Intl. Conf. on Parallel and Distributed Computing Systems (PDCS)*, Las Vegas, October 1994, pp. 417-425.