L. Nadel and D. Stein, eds.,*SFI 1992 Lectures in Complex Systems*, 163–184, Addison–Wesley 1993

The Emergence of Computational Ecologies

Bernardo A. Huberman and Tad Hogg

Xerox Palo Alto Research Center, Palo Alto, California 94304 USA

ABSTRACT

We describe a form of distributed computation in which agents have incomplete knowledge and imperfect information on the state of the system, and an instantiation of such systems based on market mechanisms. When agents can choose among several resources, the dynamics of the system can be oscillatory and even chaotic. A mechanism is described for achieving global stability through local controls.

1. Introduction

Propelled by advances in software design and increasing connectivity of computer networks, distributed computational systems are starting to spread throughout offices, laboratories, countries and continents. In these systems computational processes consisting of the active execution of programs, can spawn new ones in other machines as they make use of printers, file servers, and other machines of the network as the need arises. In the most complex applications, various processes can collaborate to solve problems while competing for the available computational resources, and may also directly interact with the physical world. This contrasts with the more familiar stand-alone computers, with traditional methods of centralized scheduling for resource allocation and programming methods based on serial processing.

The effective use of distributed computation is a challenging task, since the processes must obtain resources in a dynamically changing environment and be designed to collaborate despite a variety of asynchronous and unpredictable changes. For instance, the lack of global perspectives for determining resource allocation requires a very different approach to system-level programming and the creation of suitable languages. Even implementing reliable methods whereby processes can compute in machines with diverse characteristics is difficult.

As these distributed systems grow, they become a community of concurrent processes, or a *computational ecosystem*[5], which, in their interactions, strategies, and lack of perfect knowledge, are analogous to biological ecosystems and human economies. Since all of these systems consist of a large number of independent actors competing for resources, this analogy can suggest new ways to design and understand the behavior of these emerging computational systems. In particular, these existing systems have methods to deal successfully with coordinating asynchronous operations in the face of imperfect knowledge. These methods allow the system as a whole to adapt to changes in the environment or disturbances to individual members, in marked contrast to the brittle nature of most current computer programs which often fail completely if there is even a small change in their inputs or error in the program itself. To improve the reliability and usefulness of distributed computation, it is therefore of interest to examine the extent to which this analogy can be exploited.

Statistical mechanics, based on the law of large numbers, has taught us that many universal and generic features of large systems can be quantitatively understood as approximations to the average behavior of infinite systems. Although such infinite models can be difficult to solve in detail, their overall qualitative features can be determined with a surprising degree of accuracy. Since these features are universal in character and depend only on a few general properties of the system, they can be expected to apply to a wide range of actual configurations. This is the case when the number of relevant degrees of freedom in the system, as well as the number of interesting parameters, is small. In this situation, it becomes useful to treat the unspecified internal degrees of freedom as if they are given by a probability distribution. This implies assuming a lack of correlations between the unspecified and specified degrees of freedom. This assumption has been extremely successful in statistical mechanics. It implies that although degrees of freedom may change according to purely deterministic algorithms, the fact that they are unspecified makes them appear to an outside observer as effectively random.

Consider, for instance, massively parallel systems which are desired to be robust and adaptable. They should work in the presence of unexpected errors and with changes in the environment in which they are embedded (i.e. fail soft). This implies that many of the system's internal degrees of freedom will be allowed to adjust by taking on a range of possible configurations. Furthermore, their large size will necessarily enforce a perspective which concentrates on a few relevant variables. Although these considerations suggest that the assumptions necessary for a statistical description hold for these systems, experiments will be necessary for deciding their applicability.

While computational and biological ecosystems share a number of features, we should also note there are a number of important differences. For instance, in contrast to biological individuals, computational agents are programed to complete their tasks as soon as possible, which in turn implies a desirability for their earliest death. This task completion may also involve terminating other processes spawned to work on different aspects of the same problem, as in parallel search, where the first process to find a solution terminates the others. This much more rapid turnover of agents can be expected to lead to dynamics at much shorter time scales than seen in biological or economic counterparts.

Another interesting difference between biological and computational ecologies lies in the fact that for the latter the local rules (or programs for the processes) can be arbitrarily defined, whereas in biology those rules are quite fixed. Moreover, in distributed computational systems the interactions are not constrained by a euclidean metric, so that processes separated by large physical distances can strongly affect each other by passing messages of arbitrary complexity between them. And last but not least, in computational ecologies the rationality assumption of game theory can be explicitly imposed on their agents, thereby making these systems amenable to game dynamic analyses, suitably adjusted for their intrinsic characteristics. On the other hand, computational agents are considerably less sophisticated in their decision making capacity than people, which could cause expectations based on observed human performance from being realized.

There are by now a number of distributed computational systems which exhibit many of the above characteristics, and that offer increased performance when compared with traditional operating systems. *Enterprise*[8], is a market-like scheduler where independent processes or agents are allocated at run time among remote idle workstations through a bidding mechanism. A more evolved system, *Spawn*[12], is organized as a market economy composed of interacting buyers and sellers. The commodities in this economy are computer processing resources; specifically, slices of CPU time on various types of computers in a distributed computational environment. The system has been shown to provide substantial improvements over more conventional systems, while providing dynamic response to changes and resource sharing.

From a scientific point of view, the analogy between distributed computation and natural ecologies brings to mind the spontaneous appearance of organized behavior in biological and social systems, where agents can engage in cooperating strategies while working on the solution of particular problems. In some cases, the strategy mix used by these agents evolves towards an asymptotic ratio which is constant in time and stable against perturbations. This phenomenon sometimes goes under the name of evolutionarily stable strategy (ESS). Recently, it has been shown that spontaneous organization can also exist in open computational systems when agents can choose among many possible strategies while collaborating in the solution of computational tasks. In this case however, imperfect knowledge and delays in information introduce asymptotic oscillatory and chaotic states which exclude the existence of simple ESS's. This is an important finding in light of studies which resort to notions of evolutionarily stable strategies in the design and prediction of open system's performance.

In what follows we will describe a market based computational ecosystem and a theory of distributed computation. The theory describes the collective dynamics of computational agents, while incorporating many of the features endemic to such systems, including distributed control, asynchrony, resource contention, and extensive communication among agents. When processes can choose among many possible strategies while collaborating in the solution of computational tasks, the dynamics leads to asymptotic regimes characterized by complex attractors. Detailed experiments have confirmed many of the theoretical predictions, while uncovering new phenomena, such as chaos induced by overly clever decision-making procedures.

Next, we will deal with the problem of controlling chaos in such systems, for we have discovered ways of achieving global stability through local controls inspired by fitness mechanisms found in nature. Furthermore, we will show how diversity enters into the picture, along with the minimal amount of such diversity that is required to achieve stable behavior in a distributed computational system.

2. Computational Markets for Resource Allocation

Allocating resources to competing tasks is one of the key issues for making effective use of computer networks. Examples include deciding whether to run a task in parallel on many machines or serially on one; and whether to save intermediate results or recompute them as needed. The similarity of this problem to resource allocation in market economies, has prompted considerable interest in using analogous techniques to schedule tasks in a network environment. In effect, a coordinated solution to the allocation problem is obtained using Adam Smith's "invisible hand"[10]. Although unlikely to produce the optimal allocation that would be made by an omniscient controller with unlimited computational capability, it can perform well compared to other feasible alternatives[1, 7]. As in economics[3], the use of prices provides a flexible mechanism for allocating resources, with relatively low information requirements: a single price summarizes the current demand for each resource, whether processor time, memory, communication bandwidth, use of a database or control of a particular sensor. This flexibility is especially desirable when resource preferences and performance measures differ among tasks. For instance an intensive numerical simulation's need for fast floating point hardware is quite different from an interactive text editor's requirement for rapid response to user commands or a database search's requirement for rapid access to the data and fast query matching.

As a conceptual example of how this could work in a computational setting, suppose that a number of database search tasks are using networked computers to find items of interest to various users. Furthermore, suppose that some of the machines have fast floating point hardware but all are otherwise identical. Assuming the search tasks make little use of floating point operations, their performance will not depend on whether they run on a machine with fast floating point hardware. In a market based system, these programs will tend to value each machine based on how many other tasks it is running, leading to a uniform load on the machines. Now suppose some floating point intensive tasks arrive in the system. These will definitely prefer the specialized machines and consequently bid up the price of those particular resources. The database tasks, observing that the price for some machines has gone up, will then tend to migrate toward those machines without the fast floating point hardware. Importantly, because of the high cost of modifying large existing programs, the database tasks will not need to be rewritten to adjust for the presence of the new tasks. Similarly, there is no need to reprogram the scheduling method of a traditional central controller, which is often very time consuming.

This example illustrates how a reasonable allocation of resources could be brought about by simply having the tasks be sensitive to current resource price. Moreover, adjustments can take place continually as new uses are found for particular network resources (which could include specialized databases or proprietary algorithms as well as the more obvious hardware resources), and do not require all users to agree on, or even know about, these new uses, thus encouraging an incremental and experimental approach to resource allocation.

While this example motivates the use of market based resource allocation, a study of actual implementations is required to see how large the system must be for its benefits to appear and whether any of the differences between simple computer programs and human agents pose additional problems. In particular, a successful use of markets requires a number of changes to traditional computer systems. First the system must provide an easily accessible, reliable market so that buyers and sellers can quickly find each other. Second, individual programs must be price sensitive so they will respond to changes in relative prices among resources. This implies that the programs must, in some sense at least, be able to make choices among various resources based on how well suited they are for the task at hand.

A number of market-like systems have been implemented over the years[8, 11, 12]. Most instances focus on finding an appropriate machine for running a single task. While this is important, further flexibility is provided by systems which use market mechanisms to also manage a collection of parallel processes contributing to the solution of a single task. In this latter case, prices give a flexible method for allocating resources among multiple competing heuristics for the same problem based on their perceived progress. It thus greatly simplifies the development of programs that adjust to unpredictable changes in resource demand or availability. Thus we have a second reason to consider markets: not only may they be useful for flexible allocation of computational resources among competing tasks, but also the simplicity of the price mechanism could provide help with designing cooperative parallel programs.

One such system is Spawn[12], in which each task, starting with a certain amount of money corresponding to its relative priority, bids for the use of machines on the network. In this way, each task can allocate its budget toward those resources most important for it. In addition, when prices are low enough, some tasks can split into several parts which run in parallel, as shown in Fig. 1, thereby adjusting the number of machines devoted to each task based on the demand from other users. From a user's point of view, starting a task with the Spawn system amounts to giving a command to execute it and the necessary funding for it to buy resources. The Spawn system manages auctions on each of the participating machines, the use of resources by each participating task, and provides communication paths among the spawned processes. It remains for the programmer to determine the specific algorithms to be used and the meaningful subtasks into which to partition the problem. That is, the Spawn system provides the price information and a market, but the individual programs must be written to make their own price decisions to effectively participate in the market. To allow existing, non-price sensitive, programs to run within the Spawn system without modification, we provided a simple default manager that simply attempted to buy time on a single machine for that task. Users could then gradually modify this manager for their particular task, if desired, to spawn subtasks or use market strategies more appropriate for the particular task.

Studies with this system show that an equilibrium price can be meaningfully defined with even a few machines participating. A specific instance is shown in Fig. 2. Despite the continuing fluctuations, this small network reaches a rough price equilibrium. Moreover, the ratio of prices between the two machines closely matches their relative speeds, which was the only important difference between the two types of machine for these tasks. An additional experiment studied a network with some lengthy, low priority tasks to which was added a short, high priority task. The new task rapidly expands throughout the network by outbidding the existing tasks and driving the price of CPU time up, as shown in Fig. 3. It is therefore able to briefly utilize a large number of networked machines and illustrates the inherent flexibility of market based resource allocation. Although the very small networks used in these experiments could be adequately managed centrally, these results do show that expected market behavior can emerge even in small cases.

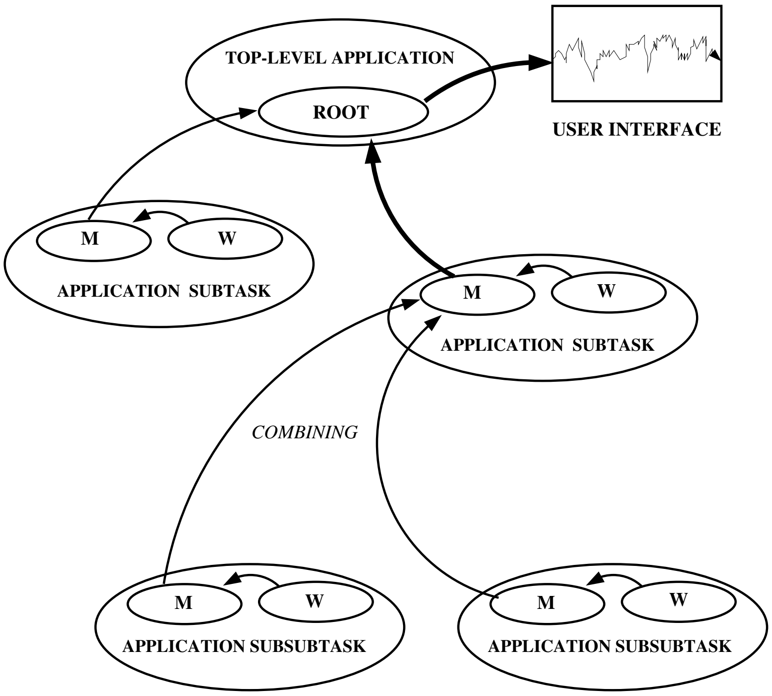

Fig. 1. Managing parallel execution of subtasks in Spawn. Worker processes (W) report progress to their local managers (M) who in turn make reports to the next higher level of management. Upper management combines data into aggregate reports. Finally, the root manager presents results to the user. Managers also bid for the use of additional machines and, if successful, spawn additional subtasks on them.

Computer market systems can be used to experimentally address a number of additional issues. For instance, understanding what happens when more sophisticated programs begin to use the network, e.g., processes that attempt to anticipate future loads so as to maximize their own resource usage. Such behavior can destabilize the overall system. Another area of interest is the emergence of diversity or specialization from a group of initially similar machines. For example, a machine might cache some of the routines or data commonly used by its processes, giving it a comparative advantage in bids for similar tasks in the future. Ultimately this could result in complex organizational

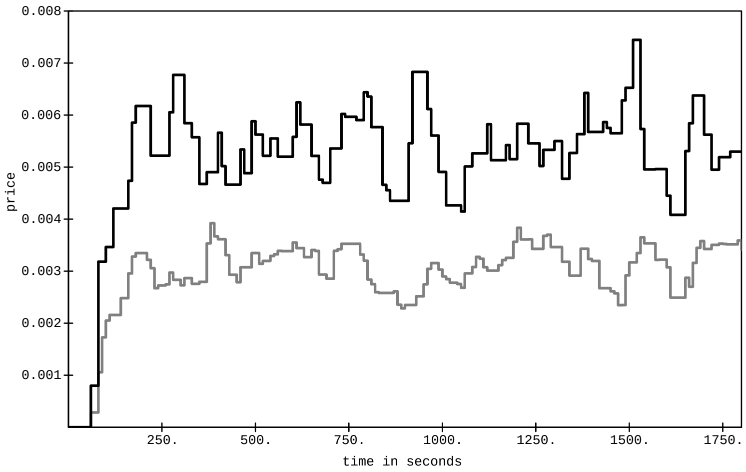

Fig. 2. Price as a function of time (in seconds) in an inhomogeneous Spawn network consisting of three Sun 4/260's and six Sun 4/110's running four independent tasks. The average price of the 260's is in black, the less powerful 110's in gray.

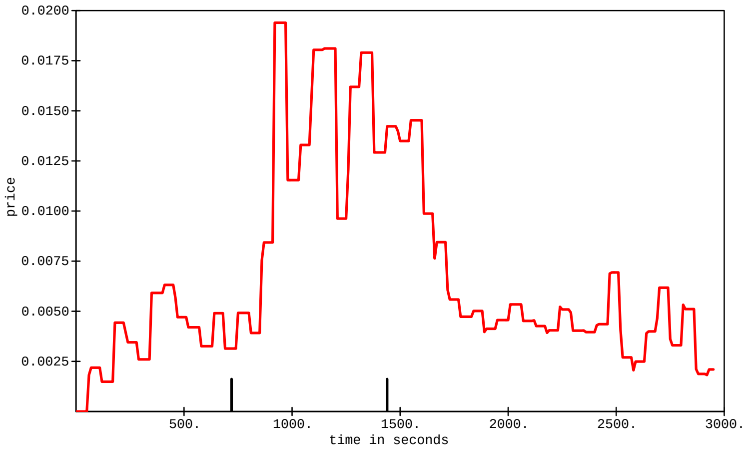

Fig. 3. Price as a function of time (in seconds) when a high-priority task is introduced into a Spawn network running low-priority jobs. The first vertical line segment on the time axis marks the introduction of the high-priority task, and the second one the termination of its funding.

structures embedded within a larger market framework[9]. Within these groups, some machines could keep track of the kinds of problems for which others perform best and use this information to guide new tasks to appropriate machines. In this way the system could gradually learn to perform common tasks more effectively.

These experiments also highlighted a number of more immediate practical issues. In setting up Spawn, it was necessary to find individuals willing to allow their machines to be part of the market. While it would seem simple enough to do so, in practice a number of incentives were needed to overcome the natural reluctance of people to have other tasks running on their machines. This reluctance is partly based on perceived limitations on the security of the network and the individual operating systems; for it was possible that a remote procedure could crash an individual machine or consume more resources than anticipated. In particular, users with little need for compute-intensive tasks saw little benefit from participating since they had no use for the money collected by their machines. This indicates the need to use real money in such situations so that these users could use their revenues for their own needs. This in turn, brings the issue of computer security to the forefront so users will feel confident that no counterfeiting of money takes place and tasks will in fact be limited to only use resources they have paid for.

Similarly, for those users participating in the system as buyers, they need to have some idea of what amount of money is appropriate to give a task. In a fully developed market, there could easily be tools to monitor the results of various auctions and hence give a current market price for resources. However, when using a newly created market with only a few users, tools are not always available to give easy access to prices, and even if they are, the prices have large fluctuations. Effective use of such a system also requires users to have some idea of what resources are required for their programs, or, better yet, to encode that information in the program itself so it will be able to respond to available resources, e.g., by spawning subtasks, more rapidly than the users can. Conversely, there must be a mechanism whereby sellers can make available information about the characteristics of their resources (e.g., clock speed, available disk space or special hardware). This can eventually allow for more complex market mechanisms, such as auctions that attempt to sell simultaneous use of different resources (e.g., CPU time and fast memory) or future use of currently unavailable resources to give tasks a more predictable use of resources. Developing and evaluating a variety of auction and price mechanisms that are particularly well suited to these computational tasks is an interesting open problem.

Finally, these experimental systems help clarify the differences between human and computer markets. For instance, computational processes can respond to events much more rapidly than people, but are far less sophisticated. Moreover, unlike the situation with people, particular incentive structures, rationality assumptions, etc. can be explicitly built into computational processes allowing for the possibility of designing particular market mechanisms. This could lead to the ironic situation in which economic theory has greater predictability for the behavior of computational markets than for that of the larger, and more complex, human economy.

3. Chaos in Computational Ecosystems

The systems we have been discussing are basically made up of simple agents with fast response times, compared to human in economic settings, which are complex and slower. This implies that an understanding of the behavior of computational ecosystems requires focusing on the dynamics of collections of agents capable of a set of simple decisions.

Since decisions in a computational ecosystem aren't centrally controlled, agents independently and asynchronously select among the available choices based on their perceived payoff. These payoffs are actual computational measures of performance, such as the time required to complete a task, accuracy of the solution, amount of memory required, etc. In general, the payoff G_r for using resource *r* depends on the number of agents already using it. In a purely competitive environment, the payoff for using a particular resource tends to decrease as more agents make use of it. Alternatively, the agents using a resource could assist one another in their computations, as might be the case if the overall task could be decomposed into a number of subtasks. If these subtasks communicate extensively to share partial results, the agents will be better off using the same computer rather than running more rapidly on separate machines and then being limited by slow communications. As another example, agents using a particular database could leave index links that are useful to others. In such cooperative situations, the payoff of a resource would then increase as more agents use it, until it became sufficiently crowded.

Imperfect information about the state of the system causes each agent's perceived payoff to differ from the actual value, with the difference increasing when there is more uncertainty in the information available to the agents. This type of uncertainty concisely captures the effect of many sources of errors such as some program bugs, heuristics incorrectly evaluating choices, errors in communicating the load on various machines and mistakes in interpreting sensory data. Specifically, the perceived payoffs are taken to be normally distributed, with standard deviation σ, around their correct values. In addition, information delays cause each agent's knowledge of the state of the system to be somewhat out of date. Although for simplicity we will consider the case in which all agents have the same effective delay, uncertainty, and preferences for resource use, we should mention that the same range of behaviors is also found in more general situations[4].

As a specific illustration of this approach, we consider the case of two resources, so the system can be described by the fraction *f* of agents which are using resource 1 at any given time. Its dynamics is then governed by[5]

$$\frac{df}{dt} = \alpha(\rho - f) \tag{1}$$

where α is the rate at which agents reevaluate their resource choice and ρ is the probability that an agent will prefer resource 1 over 2 when it makes a choice. Generally, ρ is a function of f through the density dependent payoffs. In terms of the payoffs and uncertainty, we have

$$\rho = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{G_1(f) - G_2(f)}{2\sigma}\right)\right) \tag{2}$$

where σ quantifies the uncertainty. Notice that this definition captures the simple requirement that an agent is more likely to prefer a resource when its payoff is relatively large. Finally, delays in information are modeled by supposing that the payoffs that enter into ρ at time t are the values they had at a delayed time $t - \tau$.

For a typical system of many agents with a mixture of cooperative and competitive payoffs, the kinds of dynamical behaviors exhibited by the model are shown in Fig. 4. When the delays and uncertainty are fairly small, the system converges to an equilibrium point close to the optimal obtainable by an omniscient, central controller. As the information available to the agents becomes more corrupted, the equilibrium point moves further from the optimal value. With increasing delays, the equilibrium eventually becomes unstable, leading to the oscillatory and chaotic behavior shown in the figure. In these cases, the number of agents using particular resources continues to vary so that the system spends relatively little time near the optimal value, with a consequent drop in its overall performance. This can be due to the fact that chaotic systems are unpredictable, hence making it difficult for individual agents to automatically select the best resources at any given time.

4. The Uses of Fitness

We will now describe an effective procedure for controlling chaos in distributed systems[4]. It is based on a mechanism that rewards agents according to their actual performance. As we shall see, such an algorithm leads to the emergence of a diverse community of agents out of an essentially homogenous one. This diversity in turn eliminates chaotic behavior through a series of dynamical bifurcations which render chaos a transient phenomenon.

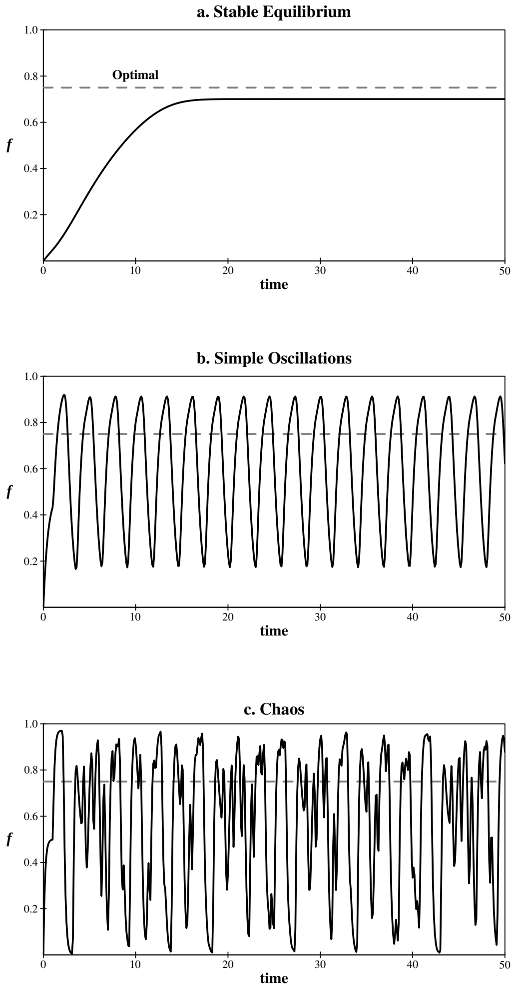

Fig. 4. Typical behaviors for the fraction *f* of agents using resource 1 as a function of time for successively longer delays: a) relaxation toward stable equilibrium, b) simple persistent oscillations, and c) chaotic oscillations. The payoffs are $G_1 = 4 + 7f - 5.333f^2$ for resource 1 and $G_2 = 4 + 3f$ for resource 2. The time scale is in units of the delay time τ, $\sigma = 1/4$ and the dashed line shows the optimal allocation for these payoffs.

The actual performance of computational processes can be rewarded in a number of ways. A particularly appealing one is to mimic the mechanism found in biological evolution, where fitness determines the number of survivors of a given species in a changing environment. This mechanism is used in computation under the name of *genetic algorithms*[2]. Another example is provided by computational systems modelled on ideal economic markets[9, 12], which reward good performance in terms of profits. In this case, agents pay for the use of resources, and they in turn are paid for completing their tasks. Those making the best choices collect the most currency and are able to outbid others for the use of resources. Consequently they come to dominate the system.

While there is a range of possible reward mechanisms, their net effect is to increase the proportion of agents that are performing successfully, thereby decreasing the number of those who do not do as well. It is with this insight in mind that we now developed a general theory of effective reward mechanisms without resorting to the details of their implementations. Since this change in agent mix will in turn change the choices made by every agent and their payoffs, those that were initially most successful need not be so in the future. This leads to an evolving diversity whose eventual stability is by no means obvious.

Before proceeding with the theory we point out that the resource payoffs that we will consider are instantaneous ones (i.e. shorter than the delays in the system), e.g. work actually done by a machine, currency actually received, etc. Other reward mechanisms, such as those based on averaged past performance, could lead to very different behavior from the one exhibited in this paper.

In order to investigate the effects of rewarding actual performance we generalize the previous model of computational ecosystems by allowing agents to be of different types, a fact which gives them different performance characteristics. Recall that the agents need to estimate the current state of the system based on imperfect and delayed information in order to make good choices. This can be done in a number of ways, ranging from extremely simple extrapolations from previous data to complex forecasting techniques. The different types of agents then correspond to the various ways in which they can make these extrapolations.

Within this context, a computational ecosystem can be described by specifying the fraction of agents, f_{rs} of a given type *s* using a given resource *r* at a particular time. We will also define the total fraction of agents using a resource of a particular type as

$$\begin{aligned} f_r^{res} &= \sum_s f_{rs} \\ f_s^{type} &= \sum_r f_{rs} \end{aligned} \tag{3}$$

respectively.

As mentioned previously, the net effect of rewarding performance is to increase the fraction of highly performing agents. If γ is the rate at which performance is rewarded, then Eq. 1 is enhanced with an extra term which corresponds to this reward mechanism. This gives

$$\frac{df_{rs}}{dt} = \alpha\big(f_s^{type}\rho_{rs} - f_{rs}\big) + \gamma(f_r^{res}\eta_s - f_{rs}) \tag{4}$$

where the first term is analogous to that of the previous theory, and the second term incorporates the effect of rewards on the population. In this equation ρ_{rs} is the probability that an agent of type s will prefer resource r when it makes a choice, and η_s is the probability that new agents will be of type s, which we take to be proportional to the actual payoff associated with agents of type s. As before, α denotes the rate at which agents make resource choices and the detailed interpretation of γ depends on the particular reward mechanism involved. For example, if they are replaced on the basis of their fitness it is the rate at which this happens. In a market system, on the other hand, γ corresponds to the rate at which agents are paid. Notice that in this case, the fraction of each type is proportional to the wealth of agents of that type.

Since the total fraction of agents of all types must be one, a simple form of the normalization condition can be obtained if one considers the relative payoff, which is given by

$$\eta_s = \frac{\sum_r f_{rs} G_r}{\sum_r f_r^{res} G_r} \tag{5}$$

Note that the numerator is the actual payoff received by agents of type s given their current resource usage and the denominator is the total payoff for all agents in the system, both normalized to the total number of agents in the system. This form assumes positive payoffs, e.g. they could be growth rates. If the payoffs can be negative (e.g. they are currency changes in an economic system), one can use instead the difference between the actual payoffs and their minimum value m. Since the η_s must sum to 1, this will give

$$\eta_s = \frac{\sum_r f_{rs} G_r - m}{\sum_r f_r^{res} G_r - Sm} \tag{6}$$

which reduces to the previous case when $m = 0$.

Summing Eq. 4 over all resources and types gives

$$\begin{aligned} \frac{df_r^{res}}{dt} &= \alpha\left(\sum_s f_s^{type} \rho_{rs} - f_r^{res}\right) \\ \frac{df_s^{type}}{dt} &= \gamma\left(\eta_s - f_s^{type}\right) \end{aligned} \tag{7}$$

which describe the dynamics of overall resource use and the distribution of agent types, respectively. Note that this implies that those agent types which receive greater than average payoff (i.e. types for which $\eta_s > f_s^{type}$) will increase in the system at the expense of the low performing types.

Note that the actual payoffs can only reward existing types of agents. Thus in order to introduce new variations into the population an additional mechanism is needed (e.g. corresponding to mutation in genetic algorithms or learning).

5. Results

In order to illustrate the effectiveness of rewarding actual payoffs in controlling chaos, we examine the dynamics generated by Eq. 4 for the case in which agents choose among two resources with cooperative payoffs, a case which we have shown to generate chaotic behavior in the absence of rewards[5, 6]. As in the particular example of Fig. 4c, we use $\tau = 10$; $G_1 = 4 + 7f_1 - 5.333f_1^2$; $G_2 = 7 - 3f_2$; $\sigma = 1/4$, and an initial condition in which all agents start by using resource 2.

One kind of diversity among agents is motivated by the simple case in which the system oscillates with a fixed period. In this case, those agents that are able to discover the period of the oscillation can then use this knowledge to reliably estimate the current system state in spite of delays in information. Notice that this estimate does not necessarily guarantee that they will keep performing well in the future, for their choice can change the basic frequency of oscillation of the system.

In what follows, we take the diversity of agent types to correspond to the different past horizons, or extra delays, that they use to extrapolate to the current state of the system. These differences in estimation could be due to having a variety of procedures for analyzing the system's behavior. Specifically, we identify different agent types with the different assumed periods which range over a given interval. Thus, we take agents of type s to use an effective delay of $\tau + s$ while evaluating their choices.

The resulting behavior is shown in Fig. 5 which should be contrasted with Fig. 4c. We used an interval of extra delays ranging from 0 to 40. As shown, the introduction of actual payoffs induces a chaotic transient which, after a series of dynamical bifurcations, settles into a fixed point that signals stable behavior. Furthermore, this fixed point is exactly that obtained in the case of no delays. That this equilibrium is stable against perturbations can seen by the fact that if the system were perturbed again (as shown in Fig 6), it rapidly returns to its previous value. In additional experiments, with a smaller

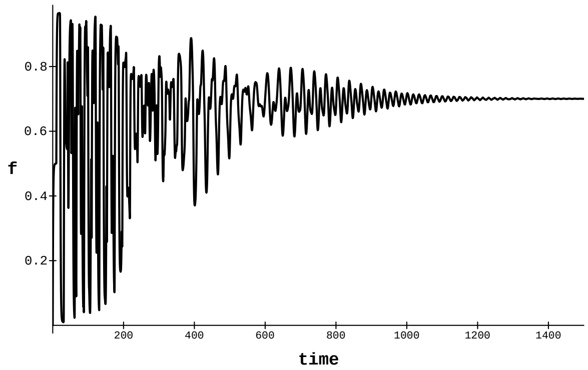

Fig. 5. Fraction of agents using resource 1 as a function of time with adjustment based on actual payoff. These parameters correspond to Fig. 4c so without the adjustment the system would remain chaotic.

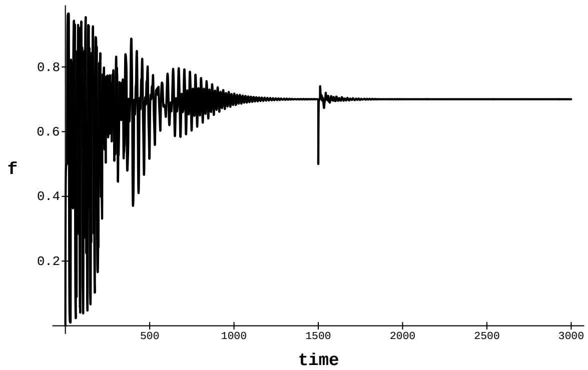

Fig. 6. Behavior of the system shown in Fig. 5 with a perturbation introduced at time 1500.

range of delays, we found that the system continued to oscillate without achieving the fixed point.

This transient chaos and its eventual stability can be understood from the distribution of agents with extra delays as a function of time. As can be seen in Fig. 7 actual payoffs lead to a highly heterogeneous system, characterized by a diverse population of agents of different types. It also shows that the fraction of agents with certain extra delays increases greatly. These delays correspond to the major periodicities in the system.

6. Stability and Minimal Diversity

As we showed in the previous section, rewarding the performance of large collections of agents engaging in resource choices leads to a highly diverse mix of agents that stabilize the system. This suggests that the real cause of stability in a distributed system is that provided by sufficient diversity, and that the reward mechanism is an efficient way of

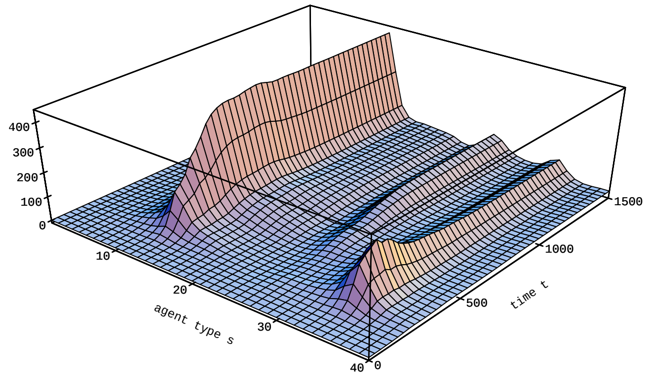

Fig. 7. Ratio $f_s^{type}(t)/f_s^{type}(0)$ of the fraction of agents of each type, normalized to their initial values, as a function of time. Note there are several peaks, which correspond to agents with extra delays of 12, 26 and 34 time units. Since $\tau = 10$, these match periods of length 22, 36 and 44 respectively.

automatically finding a good mix. This raises the interesting question of the minimal amount of diversity needed in order to have a stable system.

The stability of a system is determined by the behavior of a perturbation around equilibrium, which can be found from the linearized version of Eq. 4. In our case, the diversity is related to the range of different delays that agents can have. For a continuous distribution of extra delays, the characteristic equation is obtained by assuming a solution of the type $e^{\lambda t}$ in the linearized equation, giving

$$\lambda + \alpha - \alpha\rho' \int ds\ f(s)\, e^{-\lambda(s+\tau)} = 0 \tag{8}$$

Stability requires that all the values of λ have negative real parts, so that perturbations will relax back to equilibrium. As an example, suppose agent types are uniformly distributed in (0, S). Then $f(s) = 1/S$, and the characteristic equation becomes

$$\lambda + \alpha - \alpha\rho' \frac{1 - e^{-\lambda S}}{\lambda S} e^{-\lambda\tau} = 0 \tag{9}$$

Defining a normalized measure of the diversity of the system for this case by $\eta \equiv S/\tau$, introducing the new variable $z \equiv \lambda\tau(1+\eta)$, and multiplying Eq. 9 by $\tau(1+\eta)ze^{z}$ introduces an extra root at $z = 0$ and gives

$$\left(z^2 + az\right)e^{z} - b + be^{rz} = 0 \tag{10}$$

where

$$
\begin{aligned}
a &= \alpha\tau(1+\eta) > 0 \\
b &= -\rho' \frac{\alpha\tau(1+\eta)^2}{\eta} > 0 \\
r &= \frac{\eta}{1+\eta} \in (0,1)
\end{aligned}
\tag{11}
$$

The stability of the system with uniform distribution of agents with extra delays thus reduces to finding the condition under which all roots of Eq. 10, other than $z = 0$, have negative real parts. This equation is a particular instance of an *exponential polynomial*, whose terms consist of powers multiplied by exponentials. Unlike regular polynomials, these objects generally have an infinite number of roots, and are important in the study of the stability properties of differential-delay equations. Established methods can then be used to determine when they have roots with positive real parts. This in turn defines the stability boundary of the equation. The result for the particular case in which $\rho' = -3.41044$, corresponding to the parameters used in Sec. 5, is shown in the left half of Fig. 8.

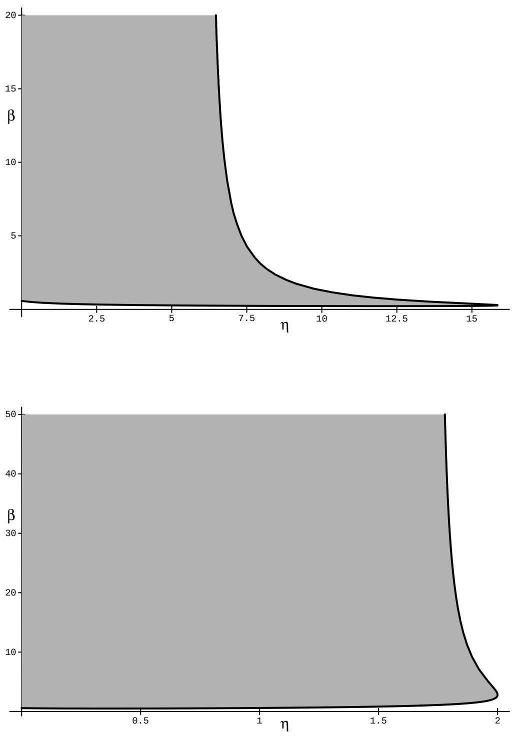

Fig. 8. Stability as a function of $\beta = \alpha\tau$ and $\eta = S/\tau$ for two possible distributions of agent types: a) $f(s) = \frac{1}{S}$ in $(0, S)$, and b) $f(s) = \frac{1}{S}e^{-s/S}$. The system is unstable in the shaded regions and stable to the right and below the curves.

Similarly, if we choose an exponential distribution of delays, i.e. $f(s) = \frac{1}{S}e^{-s/S}$ with positive S, the characteristic equation acquires the form

$$(z^2 + pz + q)e^z + r = 0 \tag{12}$$

where

$$\begin{aligned} p &= \alpha\tau + 1/\eta > 0 \\ q &= \alpha\tau/\eta > 0 \\ r &= -\alpha\tau\rho'/\eta > 0 \end{aligned} \tag{13}$$

and $z \equiv \lambda\tau$. An analysis similar to that for the uniform distribution case leads to the stability diagram shown in the right hand side of the figure.

Although the actual distributions of agent types can differ from these two cases, the similarity between the stability diagrams suggests that regardless of the magnitude of β one can always find an appropriate mix that will make the system stable. This property follows from the vertical asymptote of the stability boundary. It also illustrates the need for a minimum diversity in the system in order to make it stable when the delays aren't too small.

Having established the right mix that produces stability one may wonder whether a static assignment of agent types at an initial time would not constitute a simpler and more direct procedure to stabilize the system without resorting to a dynamic reward mechanism. While this is indeed the case in a non-fluctuating environment, such a static mechanism cannot cope with changes in both the nature of the system (e.g., machines crashing) and the arrival of new tasks or fluctuating loads. It is precisely to avoid this vulnerability by keeping the system adaptive that a dynamic procedure is needed.

Having seen how sufficient diversity stabilizes a distributed system, we now turn to the mechanisms that can generate such heterogeneity, as well as the time that it takes for the system to stabilize. In particular, the details of the reward procedures determine whether the system can even find a stable mix of agents. In the cases describe above, reward was proportional to actual performance, as measured by the payoffs associated with the resources used. One might also wonder whether stability would be achieved more rapidly by giving greater (than their fair share) increases to the top performers.

We have examined two such cases: a) rewards proportional to the square of their actual performance, and b) giving all the rewards to top performers (e.g., those performing at the 90th percentile or better in the population). In the former case we observed stability with a shorter transient, whereas in the latter case the mix of changes continued to change through time, thus preventing stable behavior. This can be understood in terms of our earlier observation that whereas a small percentage agents can identify oscillation periods and thereby reduce their amplitude, a large number of them can no longer perform well.

Note that the time to reach equilibrium is determined by two parameters of the system. The first is the time that it takes to find a stable mix of agent types, which is governed by γ, and the second the rate at which perturbations relax, given the stable mix. The latter is determined by the largest real part of any of the roots, λ, of the characteristic equation.

7. Discussion

In this paper we have presented a case for treating distributed computation as an ecosystem, an analogy that turns out to be quite fruitful in the analysis, design, and control of such systems. In spite of the many differences between computational processes and organisms, resource contention, complex dynamics and reward mechanisms seem to be ubiquitous in distributed computation, making it also a tool for the study of natural ecosystems.

Since chaotic behavior seems to be the natural resultant of interacting processes with imperfect and delayed information, the problem of controlling such systems is of paramount importance. We discovered that rewards based on the actual performance of agents in a distributed computational system can stabilize an otherwise chaotic or oscillatory system. This leads in turn to greatly improved system performance.

In all these cases, stability is achieved by making chaos a transient phenomena. In the case of distributed systems, the addition of the reward mechanism has the effect of dynamically changing the control parameters of the resource allocation dynamics in such a way that a global fixed point of the system is achieved. This brings the issue of the length of the chaotic transient as compared to the time needed for most agents to complete their tasks. Even when the transients are long, the results of this study show that the range gradually decreases, thereby improving performance even before the fixed point is achieved.

A particularly relevant question for distributed systems is the extent to which these results generalize beyond the mechanism that we studied. Since we only considered the specific situation of a collection of agents with different delays in their appraisal of the system evolution, it is of interest to inquire whether using rewards to increase diversity works more generally than in the case of extra delays.

Since we only considered agents choosing between only two resources, it is important to understand what happens when there are many resources the agents can choose from. One may argue that since diversity is the key to stability, a plurality of resources provides enough channels to develop the necessary heterogeneity, which is what we observed in situations with three resources. Another note of caution has to do with the effect of fluctuations on a finite population of agent types. While we have shown that sufficient diversity can, on average, stabilize the system, in practice a fluctuation could wipe out those agent types that would otherwise be successful in stabilizing the system. Thus, we need either a large number of each kind of agent or a mechanism, such as mutation, to create new kinds of agents.

A final issue concerns the time scales over which rewards are assigned to agents. In our treatment, we assumed the rewards were always based on the performance at the time they were given. Since in many cases this procedure is delayed, there is the question of the extent to which rewards based on past performance are also able to stabilize chaotic distributed systems.

The fact that these simple resource allocation mechanisms work and produce a stable environment provides a basis for developing more complex software systems that can be used for a wide range of computational problems.

References

[1] Donald Ferguson, Yechiam Yemini, and Christos Nikolaou. Microeconomic algorithms for load balancing in distributed computer systems. In *International Conference on Distributed Computer Systems*, pages 491–499. IEEE, 1988.

[2] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, NY, 1989.

[3] Friedrich A. Hayek. Competition as a discovery procedure. In *New Studies in Philosophy, Politics, Economics and the History of Ideas*, pages 179–190. University of Chicago Press, Chicago, 1978.

[4] Tad Hogg and Bernardo A. Huberman. Controlling chaos in distributed systems. *IEEE Trans. on Systems, Man and Cybernetics*, 21(6):1325–1332, November/December 1991.

[5] Bernardo A. Huberman and Tad Hogg. The behavior of computational ecologies. In B. A. Huberman, editor, *The Ecology of Computation*, pages 77–115. North-Holland, Amsterdam, 1988.

[6] J. O. Kephart, T. Hogg, and B. A. Huberman. Dynamics of computational ecosystems. *Physical Review A*, 40:404–421, 1989.

[7] James F. Kurose and Rahul Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Transactions on Computers*, 38(5):705–717, 1989.

[8] T.W. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard. Enterprise: A market-like task scheduler for distributed computing environments. In B. A. Huberman, editor, *The Ecology of Computation*, pages 177–205. North-Holland, Amsterdam, 1988.

[9] Mark S. Miller and K. Eric Drexler. Markets and computation: Agoric open systems. In B. A. Huberman, editor, *The Ecology of Computation*, pages 133–176. North-Holland, Amsterdam, 1988.

[10] Adam Smith. *An Inquiry into the Nature and Causes of the Wealth of Nations*. Univ. of Chicago Press, Chicago, 1976. Reprint of the 1776 edition.

[11] I. E. Sutherland. A futures market in computer time. *Communications of the ACM*, 11(6):449–451, June 1968.

[12] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffery O. Kephart, and W. Scott Stornetta. Spawn: A distributed computational economy. *IEEE Trans. on Software Engineering*, 18(2):103–117, February 1992.