

;login:

THE MAGAZINE OF USENIX & SAGE

October 2001 • Volume 26 • Number 6

inside:

COMPUTING

Needles in the Craystack:
When Machines Get Sick, Part 7

By Mark Burgess

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

needles in the craystack: when machines get sick

by Mark Burgess

Mark is an associate professor at Oslo College and is the program chair for LISA 2001.



Mark.Burgess@iu.hio.no

Part 7: Diagnosis – A Projection of LISA to Come?

*And now remains
That we find out the cause of this effect
Or rather say, the cause of this defect,
For this effect defective comes by cause.*
(Hamlet, 2.2.100–4)

Earlier in the series, I talked about how computer systems can be understood in a framework which befits any complex, dynamical system, by viewing changes as signals (i.e., processes) which compete for dominance in complex environments of many players. I talked about how order has a price and how disorder or uncertainty inevitably grows, unless it can be held in check by an idealistic, ordering “potential.” I discussed how human attitudes complicate matters by fixing expectations, demanding policy, over-simplifying evidence and thus losing important information, by complacency, and even by irrational psycho-social instinct.

One might get the impression from all of this that the situation for understanding and stabilizing computer systems is rather hopeless, that system administration is really a “soft” subject with no hope of rational analysis. My reason for embarking upon this series is that I believe that this is too pessimistic a view. Looking around at the world we live in, there is astonishing order, in spite of the odds. It is my suspicion that the main limitation in our understanding, is not the world of computers, but rather our vision of them.

How then can we go beyond bemoaning our troubles and come to firm conclusions about improving that understanding? Aiming to do science, rather than guesswork, we need to formalize our methods and investigations and erect a framework for study which is both criticizable and refinable – in which it is possible to *know*, within quantifiable tolerances. Fortunately, the ideas in the previous chapters of this series hold the answers.

The most fundamental and profound of all principles in science is the principle of causality:

For every effect, there is a cause which precedes it.

(See my book *Principles of Network and System Administration*, published by J. Wiley & Sons, for a further discussion of this.) Causality, framed as information theory, was the thrust of Part 5. It might seem trivial, even obvious, but this foundation of all change is quickly forgotten, even by scientists and engineers, when the going gets tough. As I noted in part 5, causal influence is a mapping from events which occurred in the past to events which are occurring now. It is an N:M mapping, i.e., a many-to-many mapping. Each observable phenomenon stems, in general, from several causes, and, conversely, each causal factor leads to many consequences. This is what makes matters

Every security issue essentially boils down to a problem of whom or what we are willing to trust

hard to unravel. When we make observations of the present, it is impossible to say with certainty what the cause was. The best we can do is to see whether statistical evidence supports a model or hypothesis. So the central problem becomes: how do we formulate such a model?

We can study systems empirically and obtain clues, but empirical studies have many shortcomings. What is needed is a simplification. The aim of science, after all, is to provide *suitably idealized descriptions* of phenomena, so that they may be analyzed and verified to within the limits of their assumptions. Science is not about complete descriptions, with every detail pinned down. The latter would be impossible, since the level of detail in the perturbing environment is essentially infinite.

What about the human aspect? Sometimes colleagues suggest to me that one cannot apply science to problems like human management. I find this astonishing. Management is nothing more than the problem of scheduling of resources in space and time, given a somewhat fickle environment. Although science will not have the exact answers, because it is *always* about simplification, the idea that one would rather revert to witchcraft than surrender a problem for analysis is rather frightening. Either such colleagues have no faith in science (in which case they are just bureaucrats going through some learned motions, and will never find anything new), or they are blinkered into believing that knowledge is devoid of principles which can be applied beyond an immediate context.

Clearly, computer science has a lot to say about how data structures and scheduling algorithms can be applied. If they can be applied to computer programs, they can be applied to humans. The results will not be exactly the same, nor exactly predictable, but it is possible to make the study and learn something.

Security is an excellent demonstration of causal trees. Every security issue essentially boils down to a problem of whom or what we are willing to trust. Every security problem can be drawn as a causal tree. At the top is the thing we want to secure, it splits into everything that thing depends on, then in turn the dependencies of each of those elements, and when we decide to stop this (at some arbitrary level of recursion) we end up with a number of possible sources of security breaches. Those are the things we are placing our trust in. If we don't like some of them, they can be replaced by other things, by putting some technology in the way, making another link, and moving the trust. But, however we look at it, we cannot escape this causal dependency. Security hangs on the threads of trust.

Cause-Signal-Effect and Projective Digitization

To sum up the series so far, science can be understood as a causal analysis. Such an analysis needs a motivation, or a direction which can be used to trace the tangled skein from cause to each effect. This is the role of a model. Without it, one is immediately confounded by multiplicity: many causes have many consequences. We have to be able to separate the interesting signals from the background noise.

If you have been following the series, you will now realize that we know something about this problem. It is just information theory: the theory of signals. All causal phenomena can be discussed in terms of the theory of communication, because the arrow of causal development can always be mapped onto the basic idea of a signal from past to present, or cause to effect. Some signals are strong and obvious, while others are down there amidst the noise.

The human computer interaction is fraught with much error

Causal analysis of a system's behavior is also the skill of diagnostics: it is a systematic and logical imitation of the evolutionary probing which complex environments exert on systems. While the involuntary complexity of environment alone will get you sick, a doctor has to simulate complexity systematically by prodding and asking: tell me when it hurts. Tracing backwards from effect to cause is only possible if the mapping is one-to-one, and the information about changes is preserved. One-to-one mappings only occur in strictly isolated systems, with stringent, reversible protocols. Such things are rare, as it happens, and usually only possible for infinitesimal changes, because larger changes inevitably convolute with the environment.

Perhaps you are still of the belief that the relationship between cause and effect is a simple one, that we can just decide how systems should be, introduce "management," and bingo! If so, it is already clear that you are not a perfect manager, but I ask: are you a perfect typist? Consider your interaction with the keyboard as an input device. The human computer interaction is fraught with much error. The interface itself is digital. When we hit the space bar, we do so with information about exactly where we hit it, how hard, how fast, and so on. That might be affected by muscle spasms, distractions, or (in the case of my laptop) random electromagnetic spikes. The computer digitizes this into the coarse classification space or no space. It is a many-to-one map. Information is lost and cannot be recovered.

Now suppose we try to hit the "M" key: now there is a finite chance that we might actually hit the space bar, or the "N" key. Again, the reason for this is lost to the computer, but the result is not: it is neatly classified and recorded, giving a precise yet wrong outcome. The effect is said to be projected into the space of outcomes, which is digital. Determining the cause of a bad key hit is so difficult that most would call it a waste of time to try, but it happens quite regularly, because between the brain and the CPU, there is a bunch of environmental contamination: what Shannon would have called a noisy channel.

An almost identical case of projective causality is found in the hierarchical form of evolution. Phylogenetic trees are branchings of species, which record the causal influence of an environment, projected onto digital genes. Although the tree provides a simple relationship between previous and current, it is a projective description, like a string of typed characters, full of errors. It has forgotten all of the environmental information which led to the changes. It cannot be "rolled back."

Digitization leads to a *projective* representation, like the shadow of an object on a wall, the impression left on the keyboard, or on system resources. Part of the information is dropped, and only an impression of the truth is left as a clue to what really happened.

Causal Trees with Imperfect Information

Computer scientists have acyclic, directed graphs growing in their gardens. That is the graph-theoretical name for a tree. Tree structures abound in science of all kinds, because they are direct representations of causality. In an ideal *microscopic* description of a system, we would know every detail of every change and be able to trace each one from cause to effect in a huge complicated tree. In order to draw such a tree, we would have to have *perfect information* about the changes in the system.

Because data are projected onto a finite, digital map of resources, some of the causal branches which should be there are missing, lumped together. This means that there is hidden information in the projective paths. If temperature of the machine room could

Human issues like customer satisfaction can play a role in system administration.

Ideally, it would be possible to eliminate such subjectivities, but users have an irrational insistence on their own subjective wishes

affect the results of transactions, then that information would also have to be measured and recorded, to get the full picture; if stray cosmic rays could affect input, the results of transactions (as they do on my laptop's electrostatic mouse), then they would also have to be catalogued. Since these things are not recorded in the workings of the machine, a probabilistic element enters into the projective result. Perfect information is stifled by projection. Some administrators try to achieve it with auditing, but even the molasses of information in full system accounting are never complete, because nothing on the system can record what motivates users to do what they do.

When addressing complexity, one does not normally pretend that exact results are possible; rather, one tries to model probable outcomes of the system. Such an analysis must have "hidden variables," which represent what is not known about the system. The best one can then do is to look for likely or possible outcomes using a causal tree analysis.

One kind of analysis is "risk analysis." Risk analyses are common in a variety of disciplines and go by many names (see Rob Apthorpe's paper at LISA 2001 for an application of the method to system administration). Such analyses usually attempt to quantify the different causal pathways in terms of some idealized reward called "pay-off." Risk can be minimized, profits can be maximized, "uptime" can be maximized, and so forth. How the payoff is defined depends on what one is interested in achieving. It is essentially a matter of policy.

Framed as a principle for minimizing risk or maximizing payoff, the optimization problem is one of extremizing a parameterized function. This is something which is well known in the sciences: the principle of minimum risk, the principle of least action, Fermat's principle, the minimax principle. All of these are variational methods looking to optimize some criterion. It is essentially a search-algorithm for probing the parameter space of possibilities for a desirable property.

The properties one might hope to maximize or minimize represent desirable or undesirable pathways from cause to effect. Risk, productivity, user satisfaction, return on investment, etc., are all abstract qualities which are baked into the causal pathways with probabilities that arise from the hidden variables. In contrast to many other areas of analysis, such as pure economics, artificial intelligence decision-making, human issues like customer satisfaction can play a role in system administration. Ideally, it would be possible to eliminate such subjectivities, but users have an irrational insistence on their own subjective wishes. Nothing new there: we are basically concerned with ourselves, not the abstract vagaries of "the system."

States and Models of Change

Our aim is to model how computer systems change by traversing the pathways of a projective causal tree, i.e., we are looking for their dynamical properties, projected onto the set of variables and resources which pertain to the interaction with users. Changes can occur in a machine at several levels; the smallest, most primitive changes (executed instructions, read/write operations, etc.) are often called *microscopic* and happen all the time, over very short intervals. Long-term changes (amount of free memory, level of activity) are called *macroscopic*, because they represent the cumulative effect of many microscopic transactions. Their changes are average changes, and these happen more gradually since there is some reinforcement and some cancellation of the microscopic changes over time.

A characteristic of complexity in a system is that there is no unique way of describing it

What variables characterize the system? Are they functions of time, continuous (smooth) averages or discrete (digital) measurements? Software metrics, such as numbers of processes, numbers of conversations, rate of packets per second, amount of free memory all characterize the resource usage of the system, and many more. These reflect changes taking place, but clearly they do not record why, so there must be hidden variables.

One can choose to examine these over intervals of time (micro or milliseconds) during which they change only slightly, or over longer periods (minutes to weeks) which more closely reflect the activity of external influences such as user behavior. In order to build a model, and find answers, we need to compare values at different times. Sometimes it makes more sense to compare changes to the system with a corresponding value measured a few moments before, and other times it will make more sense to compare to a value from a similar time one or more days or weeks ago. As we shall see below, the working week plays an important role in modeling.

A useful, if somewhat overused notion is that of *state*. A *microstate* is a set of values which characterizes the system at some moment. For instance, the simplest dynamical systems, studied in physics are particles which fly around. Particles are characterized by variables such as their mass, their charge, their position and their velocity. This set forms a state of a microscopic element of the system, or microstate. Once we put together more complex, composite systems, we can talk about emergent properties also as describing *macrostate*: for instance, temperature, pressure, roughness, viscosity, etc.

Computer systems are a bit like this; they have primitive things going on, such as atomic operations: read, write, add, locate. At a higher level, we also combine these actions into programs, processes and other structures, which have emergent properties like “busy,” “idle,” “thrashing,” and so on. At the microscopic level, the state of a system can be thought of as the values of a long line of bits and bit operations. At a higher level, one can talk about numbers of processes, user sessions, protocol states, which are coded into the bits at a higher level.

A characteristic of complexity in a system is that there is no unique way of describing it. Any convenient modeling projection will do, but there is a trade-off. The more detailed one gets, the more information one sees; but information is noise, and meaning is difficult to find. Alternatively, one can step back and perform the half-closed-eye test: there is less information, but the structure is seen more clearly.

What pays, in general, is an approach in terms of the most convenient measurable parameters over the time-scale which is germane to the problem. For system administrators, the variables and time-scales generally occupy the level of the operating system’s interaction with users (processes, files, over minutes or weeks). What we are looking for, then, is a description which captures changes of state variables at some arbitrary level.

Markov Chain

The essence of describing such changes in state, is the Markov chain, or Nth-order Markov model, and its more realistic extension, the “hidden Markov model.”

Put succinctly, a Markov model is a model in which the state of the system after the next time step depends only on the state of the system now. It is literally a sequence of links in a chain. For example, a traffic light has this property: when the light is red, you

When a system has fairly regular behavior and is affected by hidden variables, it is not completely predictable

know the next state will be green (in the US; red and amber in parts of Europe). When it is green, you know the next state will be amber. When it is amber, you know the next state will be red, and so on. One does not have to remember the entire history of what happened to the traffic light in order to understand what it is going to do next. Markov models are the simplest kinds of model, but surprisingly they describe many situations fairly well. One finds simple Markov models in computer science, but, in this form, they are usually trivial.

Traffic lights and other Markov processes are sometimes said to be in a steady-state, because their behavior is predictable for all time. It doesn't vary. Either it is constant, or it goes 'round and 'round in a *limit cycle*. Alas, not many problems are really quite so simple. Nth-order Markov models are models where the next transition to a new state is governed by the last N states of the system. Such models are sometimes useful for parsing simple grammars but are not very useful for understanding anything as complex as a computer system. A Markov model can be represented simply as a *transition function*, which is a list of now-states and next-states.

Real-world problems are too difficult to solve with this kind of approach. Why? Because the level at which Markov models could be applied is usually so low that the amount of detail would be overwhelming, and therefore simply noise. Instead, one purposely hides some of the data, using the half-closed-eye method, and by making the fundamental separation into system plus environment.

Billiards is a game which is often used to illustrate problems in dynamics. It cannot easily be represented as a Markov model, because the positions of the other balls in relationship to the environment of the table influence the outcome of the next move. In other words, the billiards "system" has a memory of what went on before, and the shape of the table play a role in determining what can or will transpire next. Moreover, there is an external entity in the game: the player. The player brings additional information to bear, which is not on display on the table. Chess is another example, which is digital, like a computer. The state of a game of chess is the position of all of the pieces on the board at a given time. The next move is determined not only by the positions of all of the pieces on the board, but also by the choice of the player. The next move has not one but several possibilities, and the extra information which decides which possible it has chosen is hidden from view. The transition diagram for chess is not one-to-one; there are many possible moves at each stage. The game eventually converges to a checkmate when the game runs into a part of its state-space which is a dead end (checkmate) or a limit cycle (stalemate).

When a system has fairly regular behavior and is affected by hidden variables, it is not completely predictable. A useful approach to understanding it is to look at its average or expected behavior. The average behavior is defined by the mean value of the state of the system over an ensemble of equivalent situations. Each equivalent observation of the system brings new values but, over time, these yield approximately the same result, up to smaller corrections. One says that the system exhibits microscopic fluctuations about its macroscopic average value. The separation

$$\text{signal} = \text{average} + \text{fluctuation}$$

is deeply connected to the fundamental split:

$$\text{world} = \text{system} + \text{environment}$$

This is not so much a fundamental property of nature, as it is a management view-point. This is the way the human cognitive apparatus analyzes: what we expect versus what we see.

Models which describe state transitions with imperfect information are called *hidden Markov models*. There are two ways to handle these models. One is to actually model the external information; the other is to create a *stochastic model*, i.e., one which only predicts the probability that a transition between states will be made. These models will form the substance of models of computers as dynamical systems (see papers by Apthorpe and Haugerud at LISA 2001), since computers have external players called *users*. Hidden Markov models are characterized by probabilistic transition functions, with hidden variables H, e.g.,

$$(s_1|s_2) = P_{12}(H)$$

denoting a transition from a microstate s_1 to a microstate s_2 , with probability $P_{12}(H)$, which depends on the hidden variables. The approach has been used to build quite convincing simulations and mathematical models of the behavior of computers in projective representations (numbers of processes, numbers of users, etc). Given such a model, with predictive power, one knows enough about the system in order to characterize its long-term behavior in terms of what is predictable and what is unpredictable. This leads to great simplification and time saving when looking for anomalous behavior.

Boundary Conditions

Every manageable dynamical system makes contact with its environment at some time or place, either at the outset of its evolution or during the act of measurement, at an edge, or at some boundary or interface. The environment leaves its projected imprint on the system: incomplete information about its state. The effect of the environment is usually strong, because the environment is bigger and more pervasive than most systems.

Computers touch base with users via the keyboard and via the network. These channels link computers to a reservoir of thoughts and activity which have a direct impact on what computers do. It would be bizarre indeed if it were not possible to see these effects reflected in the state of the system. Indeed, the working week is easily identified in the patterns of resource usage. It shows a fundamental periodicity in computer behavior, which has its origin in the approximately cyclic behavior of “the average user.”

One way to take account of the approximate periodicity is to formulate computer activity as a process on a circular topology (see Figure 1). By winding the time parameter around a cylinder of one-week circumference, and then squashing the resulting spiral into a circle, one ends up with many recorded values for the time-series variables at each point, a bit like old recording weather barometers. By averaging the many values at each time of the week, one then sees average behavior in relation to the working week. This is a more useful description than an average

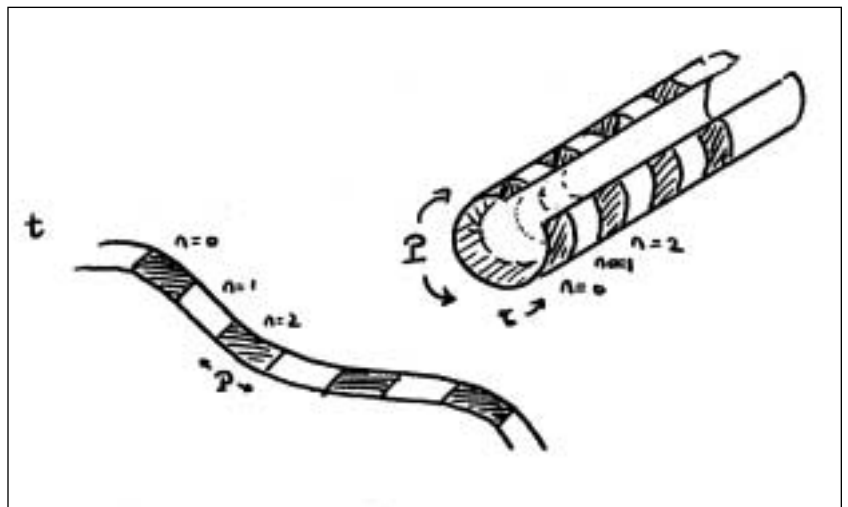


Figure 1

A type I model is a description of the state of a computer, over time, subject to the external behavior of users

over all times, since it contains real information about the changes going on in the environment.

By using stochastic methods or Monte Carlo simulation techniques, on a pseudo-periodic background, it is possible to put together a simple model which reproduces the main features of computer behavior (see the work at <http://www.iu.hio.no/SystemAdmin/scisa.html>). This allows one to say with quantifiable (calculable) certainty when a computer is behaving in one way or another. Any anomalies which are then observed must be due to effects which were not considered by the model and can be singled out as worthy of attention.

We can say two things, at the macroscopic level, about users' effect on computers:

- It is pseudo-periodic (driven by the working week).
- It is stochastic.

At the simplest level of approximation, one could say that the behavior of users was a sinusoidal, diurnal rhythm. This is not a very good approximation, but it is better than assuming that user behavior is constant, as many intrusion detection systems do. With further refinement, one can reproduce the graphs of user behavior displayed (see Haugerud and Straumsnes' paper at LISA 2001, based on our earlier studies), in order to discover how the actual pattern arises, whether it is coincidence or predictable. These patterns are the most basic "laws of nature" in system administration.

I refer to this kind of model as a type I model of a computing system. A type I model is a description of the state of a computer, over time, subject to the external behavior of users. Such a model might have many interesting features: steady-state behavior, dead ends (crashes, deadlocks etc.), and even chaos. I have spent some years working on the separation of system and environment in approximately steady systems (in physics as well as computers) and have a detailed stochastic model in the limit of large numbers of data, which identifies the important scaling properties of the system. To the trained eye, the model is very simple, but it fits the data surprisingly well.

The importance of such a model is in understanding how the structure of *cause* relates to the shape of *effect* in real, observed behavior. While we have barely scratched the surface in our work at Oslo, the results promise to explain many features of observed behavior and can be fed back into actual methodologies and tools such as cfengine. Only when armed with such knowledge does it make sense to speak of anomaly detection.

Policy Constraints: Type II Models

Type I models are likely to be important as a general guide to understanding how cause and effect are related in computers, but the success of that approach is dependant on how well one can represent the behavior of users, who represent the largest perturbation. It is not just about projecting the world onto a model, it is also about how many nuances the model should cover. Type I models treat users as a relatively formless gas of influence in which no overriding, strong signals dominate. This is a beginning, but it will not be sufficient to deal with real systems, in which a single user can make his or her influence felt by all the rest.

So how shall we know the shape of users' behavior? What happens when users do not obey simple rules, i.e., when they are not a formless gas, but an obelisk: a needle in the Craystack? Is it still possible to gauge their effect on the system? The answer is yes,

though the difficulty of doing so steps up an order of magnitude. The reason is that crowds of users behave in simpler average patterns than individuals, just as a view from a distance looks simpler than a view in close-up. Crowds have a natural inertia in number: the averages are augmented only by small fluctuations. However, in smaller group sizes, fluctuations can dominate over the average part, leaving a view of disorder.

The success of science is largely based on the idea that the laws of nature are constant, and that one therefore stands a fighting chance of unraveling them. If the rules are changing too fast, one cannot find meaning in the variation, and the good goes from bad to ugly. One thing one can do then is to look at the long-term variations only, by averaging, and find laws for those. As I said at my LISA 2000 talk on “Theoretical System Administration,” there are no “Newton’s laws” of system administration. There is no single set of rules which governs right from wrong, likely from unlikely. Why not? Because each site has its own environment and its own policy for dealing with it. Strong individuals will shine against this background.

Policy can be used to evaluate user behavior numerically, by defining a scale of value. The value is “payoff” once again, only now one must also say, payoff from whose perspective? The scale is not necessarily unique. It is only required to be consistent in all comparisons. The idea of scales of values determining social behavior is a fascinating problem which has plagued the social sciences for many years. What is new and interesting about computers is that we actually have a chance of quantifying the behavior stringently, because the machine can see everything that is going on, within an automatically limited arena. Also, formalized value systems can be evaluated impartially.

Our quest, then, is to evaluate the likely mixture of behaviors in a mass of users according to some criterion. The scale of measurement will be related to system policy in the sense that users will tend to aggregate around behaviors which are provoked by what they are allowed or supposed to do. Some users are law-abiding or altruistic; others are contrary and selfish. Mixed up in here, is the somewhat fluid notion of “security”; it is rather hard to pin down, but it is clearly related to the extent to which the system and its users work within the boundaries of policy, and the idea that an unfortunate mixture of user behavior might drive the system into an undesirable state.

A model which evaluates a profile of user behavior in relation to policy is what I call a type II model of a computer system. Such a model is not completely independent of type I models. Rather, the two feed off one another.

Policy and State: Paths through a Lattice

At the most primitive level, the resources of a computer can be thought of as a string of bits, subject to external change: disks and memory are represented by the bits, and the external change comes from I/O with users and the network, mediated by the CPU. The structure that we build on top of this bit string, including the file system, the operating system, the structure of data, and so on, is multidimensional, and discrete, i.e., it forms a lattice. As we look at changes in the system, we can classify those changes on this lattice. The contention is that, when one decides policy, the effect is to select a preferred region of this lattice. In other words, policy is a projective action, which effectively selects one or more acceptable regions of the state space.

As far as a computer is concerned, the effect of a system policy is to do the following:

The success of science is largely based on the idea that the laws of nature are constant, and that one therefore stands a fighting chance of unraveling them

- System: specify machine configuration in terms of allowed behavior, access controls etc.
- Environment: encourage users to obey limitations and work patterns.

The initial configuration of the system, places it within a region of the lattice which is chosen by policy. This is controllable and verifiable. Asking users to obey rules is politely asking them not to try to drive the system away from this policy region. This is not controllable, but it is verifiable. Because of the environment, we cannot expect a policy specified to be completely upheld, because we cannot control the minds of users. What we can say, however, is that a stable solution to the problem of policy versus users will lead to a situation where the system remains in the acceptable regions of the lattice for most of the time (on average). A counterforce (police force, or immune system) can correct the minor transgressions which must inevitably occur.

But who says the policy will be stable, that transgressions will only be minor? It is, of course, possible to write system policies for a given mass of users which will provoke them into such rebellion that the policy will immediately fail. I claim that this is a good criterion for an unrealistic policy (governments sometimes make such mistakes) and that such a catastrophic failure is a pathology of the initial assumptions. The aim of system administration is never to build such unstable systems, so sufficient stability is just a basic requirement, a starting point.

This model of the user-machine interaction, constrained by policy, can be written in a more formal way, in order to map it onto well-known methods of stochastic dynamics. Suppose we examine any variable of the system, as a function of time. Suppose also that we collect the data over many periods (weeks) and examine the averages, calculated for all corresponding intervals. This provides us with an average picture of what the system is doing, in addition to an actual picture of what the system is doing. Now we define:

$$\text{Actual value} = \text{average value} + \text{fluctuation}$$

This split is significant for two reasons. The first is that the average value categorizes the approximate behavior of the system at any given moment, while the fluctuation tells us essentially about the variation of the environment. The second is that it separates microscopic from macroscopic, i.e., fast changes, or what happens over short times (fluctuation), from slow changes, or what happens over long times (changing average). We have thus formalized the idea that the environment is a complex changing signal which pokes and prods with much higher resolution than the stable part of the system.

In the lattice of changes, policy can only be associated with the stable part of the configuration. Acceptable levels of deviation from “perfect” can be used to define a distance from acceptable policy, or an average policy, but not an exactly enforceable one. The problem thus becomes, how can one keep the system as close as possible to an ideal policy-abiding state?

Can we curve the lattice, like a gravity well, so that the system rolls back into its point of lowest “energy,” or most “policy correct” configuration (see Part 5)? This is the idea behind computer immunology. By building an immune system, or a mobilizable counterforce which regulates policy, one effectively builds such a gravity well. Unlike a gravity well, where all particles respond equally to the force, an immune system has a harder time of this job, because the lattice is multidimensional and the changes respond differently in each direction. This means that signatures and distinctions have

to be made. Work of this kind has been done at the University of New Mexico using a method of classifying sequences of system calls inspired by the human immune system (see <http://www.cs.unm.edu/~immsec/>).

Let the Games Begin

In the future one can imagine feedback to users which indicates the state of the system. If users see a machine which is not “feeling well,” this would be a signal to avoid that particular machine. This alone might be sufficient relief to allow the machine to correct itself (heal itself). This kind of bilateral feedback has been experimented with in artificial intelligence (e.g., the MIT Kismet robot; see <http://www.ai.mit.edu/projects/kismet>). I think it could be essential to the development of truly robust systems which interact with humans.

What happens when environment meets machines? The unpredictable meets the specified. If the machine is capable of adapting, there ensues a game of competition for the integrity of its design policy. If the machine cannot adapt, the specification ends up being ruined.

In a game theoretical model of system administration, it makes sense to divide users into those who obey policy and those who do not. Users who obey policy are irrelevant to the evaluation of policy because they can be absorbed into the background activity, i.e., the way in which the value of the “payoff” changes normally in time. On the other hand, if users do not obey policy, they might choose any number of strategies to try to confound it. A model of system administration is interested in evaluating how likely it is that such a strategy would succeed against policy.

Thus, at the simplest level, we think of the actors as motivated individuals who are in competition to maximize their gain or minimize their risk. They might work cooperatively, in an altruistic way, or non-cooperatively in a purely selfish way. There might be any number of players in a game, but the simplest case (also the first approximation) is to think of system behavior as a two-person game, in which the users of the system compete with the system itself for possession of valuables.

A game is characterized by a matrix (see Figure 2) in which the rows and columns are labeled by the strategies and counter-strategies of the players, and the body of the matrix contains the payoff to one of the players of interest. By using minimum/maximum techniques, one can seek the most effective mixture of strategies (represented by the histogram distributions), which leads to optimal results. In traditional games, the valuables of the game are easily identifiable game pieces or token rewards. In economics the reward is money; in natural sciences the reward is energy. In Part 6 of the series, I argued that rewards in a social setting are not only tangible assets, but can also be the vagaries of emotional reward: peer respect, personal satisfaction, aesthetics, and any number of others from our complicated emotional psyche. The relative importance of these pieces of the puzzle is also, in a sense, a matter of policy. Little is known in our field about the value-scales for the variability of human traits, but it would be surprising if such research had never been done

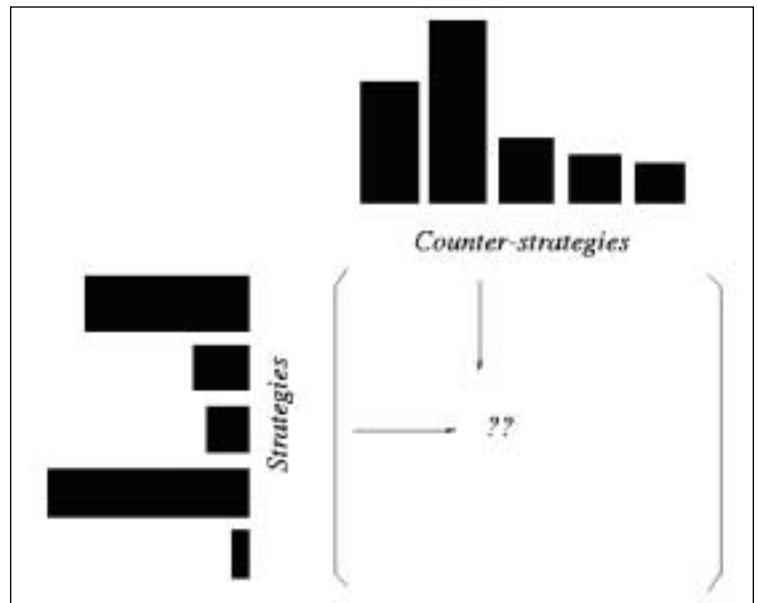


Figure 2

. . . system administration is neither a once-and-for-all solvable problem, nor a problem in which humans have to watch endlessly over their sheep

by psychologists in other contexts. With this viewpoint, there is a considerable simplification of the problem.

A game may be set up of hostile users versus a system policy agent (counterforce) and used to evaluate the optimal mixture of counterforce strategies, given that users do their worst. The game also doubles as a formal framework for finding out what the users' worst actually is. The solution of the game is one or more distributions and counter-distributions of strategies for each of the players, which the players can adopt in order to maximize their interest.

At the simplest level, one can assume that users do not cooperate with the system, but clearly one can extend the sophistication of the game in many ways to explore more refined possibilities. Evaluation of payoff is complex, and game playing is iterative. I can foresee that, in the future, simulation tools such as Petri Nets will play a role in simulating these complexities. It is not certain how much would be gained by this, but it is an avenue for further research.

Conclusion

In this series, I have tried to emphasize the dynamical, competitive aspect of complexity and the central importance of concentration (centralization) versus distribution: the management of entropy. Low entropy can be poison, high entropy dilapidation, but these are the extreme polarities of the scale. The issue is not a simple question of right or wrong; rather, it is one of seeking appropriate balance in the face of prevailing conditions. This theme recurs in many guises: Cray or workstation; central server or distributed database; uniqueness or redundancy; first-come, first-served (FCFS); or time-sharing, law-abiding users or disruptive users; knowledgeable users or ignorant users; automatic regulation or human intervention? These problems are all, at some level, about entropy management. The message, which applies in every case, is that the environment seeks out a balance between these strategies. We have the means to address these problems in quantitative terms.

I hope that I have drawn attention, in this series, to the idea that system administration is neither a once-and-for-all solvable problem, nor a problem in which humans have to watch endlessly over their sheep; rather, it is a process of continual regulation, a constant war against sickness, in which human or someday artificial ingenuity will occasionally be called upon to exceed the boundaries of simplistic programming. I have focused on what happens on computers, not on what happens in between computers (the network). The latter is another story altogether, far more complex, but building on what I have discussed here. At LISA this year, we will begin to see the results of our early probings into this challenging field.