
ELSEVIER

Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Science of Computer Programming 49 (2003) 1–46

Science of Computer Programming
www.elsevier.com/locate/scico

On the theory of system administration

Mark Burgess

Oslo University College, Cort Adelers gate 30, 0254 Oslo, Norway

Received 23 June 2003; received in revised form 29 July 2003; accepted 8 August 2003

Abstract

This paper describes a mean field approach to defining and implementing policy-based system administration. The concepts of regulation and optimization are used to define the notion of maintenance. These are then used to evaluate stable equilibria of system configuration, that are associated with sustainable policies for system management. Stable policies are thus associated with fixed points of a mapping that describes the evolution of the system. In general, such fixed points are the solutions of strategic games. A consistent system policy is not sufficient to guarantee compliance; the policy must also be implementable and maintainable. The paper proposes two types of model to understand policy driven management of Human-Computer systems: (i) average dynamical descriptions of computer system variables which provide a quantitative basis for decision, and (ii) competitive game theoretical descriptions that select optimal courses of action by generalizing the notion of configuration equilibria. It is shown how models can be formulated and simple examples are given.

Keywords: Computer models; Dynamical system; System administration; Mean field theory

1. Introduction

This paper is about system administration, i.e. the design, running and maintenance of human-computer communities. It explores the viability, the strategies and the some of the limitations of system administration. The principal aim of the paper is to show that it is meaningful to regard system administration as a strategic game, whose short-term aim is to maintain a state of approximate predictability, and whose long-term aim is to maximize productivity. Along the way, one encounters stepping stones such as the concept of a stable policy. This viewpoint forms a well-defined problem which can

E-mail address: mark.burgess@iu.hio.no (M. Burgess).

0167-6423/$ - see front matter
doi:10.1016/j.scico.2003.08.001

be solved, subject to local constraints, by human–computer systems. The conclusion is not trivial; it requires a chain of reasoning which this paper can attempts to summarize and exemplify. The conclusion is important, however, because the administration of systems is still practiced, in many organizations, by allowing systems to reach failure and then by resetting them. This approach is unjustified in most cases. The conclusion has both reliability and security implications.

System administration is currently founded on mainly assertion and anecdotal experience [19], combined with a few ideas from distributed software engineering [28]. To date, few mathematical analyses of system administration have been undertaken, and no formal framework has been constructed for undertaking this task; this has made the expression of objective truths about the field difficult.

1.1. Aims

The aim of this work is to establish a formal view of system administration which bridges the conceptual divide: a way of formulating objective discussions about computer management, that are sustainable, using a mathematical framework based on general assumptions. The notions of convergence and fixed points of mappings (equilibria) will play a central role in this, since they provide a definition of stability that is the basis of system maintenance. Other ideas that feed into this kernel include descriptions of computer systems that have appeared in physics of computing [6,7,14,21,26]. Each of these ideas adds a piece to a jigsaw puzzle that culminates in a definition of dynamical stability and the maintenance processes that make it possible.

What should a theory of system administration be about? The task of elucidating this sounds straightforward, but it is a slippery business. System administration, in reality, is based on mainly qualitative, high level concepts, which mix technical and sociological issues (such as user management) at many levels. Although it is clear to system administrators that there is a body of technical principles involved in the discipline, it remains somewhat intangible from the viewpoint of a scientist. It is hard to find anything of general, reproducible value on which to base a more quantitative theory.

One of the obstacles to formulating such a theory is the complexity of interaction between humans and computers. There are many variables in a computer system, which are controlled at distributed locations. Computer systems are *complex* in the sense of having many embedded causal relationships and controlling parameters. Computer behaviour is strongly affected by human social behaviour, which is largely unpredictable. The task of identifying and completely specifying the ideal state is therefore a non-trivial one, yet certain criteria such as stability guide us in this process. It is nonetheless this task which this paper attempts to address. Can one formulate a quantitative theory of system administration, which is general enough to be widely applicable, but which is specific enough to admit analysis?

The plan for this paper is as follows.

(1) Section 2 develops a self-consistent and sustainable definition of policy for a human-computer system, in terms of functional mappings and system specifications. It is shown that a sustainable policy requires there to be system maintenance, and describes what maintenance means.

(2) In Section 2.8, a theorem is proven, which shows that there must exist a class of policies which lead to sustainable behaviour in the system. This theorem is general and makes no reference to specific semantics of policy. The result is important in motivating the remainder of the paper, since it turns attention away from discussions of possibility to acceptance of possibility and methodology: i.e. to examine the relative value of different routes towards attaining this goal. The theory of games will play this role.
(3) Given the existence proof, that a desirable policy can be created, Section 3 comments on how specific variables can be measured and modelled as a basis for formulating policy in terms of empirical metrics [25]. This is the second prerequisite for the culminating synthesis: it allows one to place a relative value on different approaches or strategies to achieving a stable policy.
(4) In Section 4, the pieces are combined: strategies for maintaining a stable or sustainable policy are formulated in a game theoretical framework, which allows different pathways to stable equilibrium (strategies) to be pitted against one another. An example game illustrates the method for a simple example, reproducing intuitive results.

1.2. Scope of system administration

One of the first obstacles in discussing the theory of system administration is limiting the scope of the discussion. System administrators are called upon to perform all manner of tasks as part of their duties. This battery of skills has no particular cohesion or structure to it, so it often resists formalization. One must improve on this situation, by restricting to core activities, in order to make progress in forming a theoretical framework. These core activities will include insuring availability, efficiency, and security for all users; fault diagnosis of the system is also a natural inclusion. These include issues such as software installation and upgrades, which can be classified under availability and efficiency. It also includes user management at a high level, though it will not be useful to address the issue of creation of user accounts in this context.

Even a limited theory of system administration should cover some key aspects of the problem:

- Policy determination and evaluation,
- Strategic decisions about resource usage,
- Interaction between users and system for resources.
- Productivity considerations (the economics of the system),
- Empirical verification of strategies and policies,
- Efficiency of policy and of policy implementation,
- Efficiency of the system in doing its job.

More pragmatic details, such as the need for software installation and upgrade, will have to be tackled at an abstract level, in terms of productivity, probability of failure, resource usage, risk and so on. Software bugs can be addressed in terms of productivity,

risk or security. Security, in turn can be viewed as a contest for resources at the level of the system.

1.3. On scales and coding

Complex systems are often so disparate in their nature, at different scales, that quite different descriptions are required to capture the full essence. A theory of system behaviour at, say the microscopic level of system calls, need not resemble a theory for the behaviour at a macroscopic scale of larger entities, such as patterns of user behaviour. Both are needed in order to understand the whole hierarchy of things going on. This has implications for maintenance of systems.

Predictability depends on the sustainability of assumptions about a system. In a predictable system there is an approximate separation of scales, so that high and low level details can be described independently [41]. This means that the functioning of a system at a high level is not strongly dependent on the implementation details of the system at lower levels (see the examples below). This idea allows a notion of high level stability in the coming sections. However, scale separability is only an approximate quality. The price one pays for assuming the integrity of lower level details is a finite uncertainty in the behaviour of the high level system, whose magnitude depends (inversely) on the truth of the assumption of separability.

Example 1. A database is a high level system. It may be implemented with paper and filing cabinets, or with electronic storage on Unix or Windows computers—the low level implementation does not alter the information transactions themselves. This is a desirable quality, which promotes several important design principles; however, low level details can affect the high levels in the case of an anomaly. For instance, a paper record might unexpectedly be damaged or lost; an electronic record is vulnerable to power failures. If low level details like these can be corrected at the lower level then the high level assumption of ignoring low level detail can still be maintained, otherwise an unpredictable change will be seen to occur at the high level. Thus, while one would like to keep the high level view for simplicity, this requires a commitment to detail in the lower levels to preserve the property of scale-separability, otherwise the high level view becomes an over simplification, because information about the lower levels has been ignored.

Example 2. The separation of scales in time is also a desirable property, since it leads directly to a notion of stability in the coming sections. Suppose one measures the hourly average behaviour of a variable, like CPU usage, for verifying the behaviour of a critical system that normally varies slowly. This average is representative of the system, provided low level fluctuations are only small. The presence of a sudden anomaly (occurring within minutes or seconds) could break this assumption, unless the anomaly were countered quickly enough to prevent any serious consequences. A fast (low level) repair, can maintain a slow (high level) notion of stability.

In taking a high level view, one conceptually separates an average view from a low level detailed view; this is like the procedure of information hiding in creating

directory structures or use of subroutines in programming. The expression 'averaging over' then applies also to the concealment of dynamical detail: i.e. the suppression of small, short-term changes by choosing to look at average behaviour over a longer time-scale.

To summarize, a description of system behaviour at a high level is, for many purposes, independent of specific details of the lower levels, or shorter time scales, but this assumption can be damaged in an unpredictable environment, e.g. if we assume that a system is static when it is only approximately constant on average, then our assumptions about it are actually violated for many brief intervals of time. This implies a potential uncertainty in the behaviour of each new level. System administration is a problem which is tackled at a high level, and over traditionally long time scales, since it addresses the interaction of users with the system through high level interfaces. This means that there can always be an intrinsic uncertainty in understanding the detailed system, and a corresponding uncertainty in any changes made. A high level change introduced to curb a lower-level problem is intrinsically limited. This problem is particularly important because computer systems are stochastic in nature [11] and the fluctuations of the low level system are generally more detailed than the consequences that human administrators can address through any interface.

The implication here is that computer systems cannot be micro-managed with complete predictability, as long as there are parts of the system which are not completely determined by the system administrator. Systems can only be regulated, because one cannot exert certain control over low-level changes, with only high level interfaces. This limitation must be taken into account in managing the system.

1.4. Generic computer models

In order to elucidate the results and goals of computer configuration and maintenance, it is necessary to identify the main characteristics of the interactions between computer systems and their users, at a suitable level of abstraction. This includes finding:

- relevant variables,
- invariance properties,
- persistent structures,
- sources of information loss (entropy),

which affect the principal goals. Several studies of computer systems have attempted to identify such qualities [6,11,21,26] and it is supposed here that a suitably abstracted description can be built on principles such as those identified by these authors.

The basic model proposed for a computer is that of a dynamical 'community' of processes and resources, coupled to an external environment of users. The environment is represented as a source or sink which generates the stochastic influences of all of the users of the system, and any other computer systems which communicate with hosts within the perimeter of one's own system. As pointed out in Ref. [39], the issue of networking does not increase the complexity of the administration problem, only its localization and perhaps its absolute magnitude. A set of networked hosts, sending

external messages, is no different, for present purposes, from a single virtual host with internal inter-process communication.

There is a need for at least two distinct types of theoretical model for the computer-user interaction in system administration: models for impartial evaluation, and models for strategic planning and adaptation. These may be referred to as passive (type I) and strategic (type II).

- *Type I models* are passive descriptions of resource usage, as stochastic processes; they describe the changing array of variables which characterizes host state. Type I models can be verified by empirical measurement of computer systems and used to predict the passive aspects of computer behaviour. They form the basis for a type II model.

Example 3. Type I models have described the rates of starting and stopping of processes on a computer, the expedition of network service transactions, and the accumulation of disk usage, as functions of time. One could model distributions of the relative level of usage of different computer programs, and their effects on the rest of the system. These are based on random arrival of requests, the multi-periodic work patterns of users. Models such as these lead to an understanding of how resources are used, and the distributions of requests.

- *Type II models* introduce a user-level, semantic interpretation of the system based on a set of values. They evaluate strategies for effectively achieving the goals of the system, i.e. planning and maintaining policy. They apply knowledge learned from type I models, in order to compare and contrast the effect of different policy decisions on the system, and determine optimal strategies for achieving some goal.

Example 4. Type II models have been created to determine the best strategies for deleting temporary files from user-disks, so as to prevent disks from filling. Games for maximizing system availability, and for locating network infrastructure, with the constraint of limited resources have been suggested. Service level agreements (SLA) are included in this. Psychological and technical strategies for encouraging users to follow basic system rules could be modelled, based on type I studies of user attitudes and the relative merits of these in a larger context. Such models would have security implications.

Thus, a type I model provides a substantive basis for discussing system management in concrete terms, whereas a type II model introduces a value system into the analysis: a notion of what is desirable or undesirable. This must be quantified somehow, thus one requires a *currency*, or notion of wealth and poverty of purpose, in relation to system policy. It is in type II models that system policy takes centre stage.

1.5. The role of time

Time plays a central role in the understanding of system administration, because computing systems are dynamical systems. Several important time scales emerge, and

are important in separating the details of the response of the system to different influences. The passage of system time in a computer system occurs through the iteration of the fetch-execute cycle, thus the time-development of a computer is discrete at the microscopic level. Over longer time periods, this fine-grain discreteness is unimportant however, and the system may readily be approximated as a function of continuous (differentiable) time. Furthermore, this approximation can be made at several levels, which will be described in the next section. This allows one to deal with changes of interest, while suppressing changes which have no immediate bearing.

The time scale at which users work, reflects changes in the system's internal resources as a result of human work patterns. This situation has been studied empirically and it is found that the average behaviour can most clearly be seen on a periodogram spanning a working week, with a resolution of approximately one hour [11]. Superimposed on top of this coarse trend, is the more microscopic fluctuation behaviour, which reflects environmental complexity. This can be shown to see changes on the order of seconds up to about 5 minutes. A computer system is an arena in which users and processes compete for limited resources. The situation one aspires to is one of relative stability, with slow long-term changes on a background of rapid fluctuations.

2. The meaning of policy

In order to formulate the purpose of system administration explicitly, one requires a basic 'world view' of the problem: one regards computers as constrained dynamical, but stochastic systems; later they can be developed as purposeful semantic systems. This is a somewhat heretical viewpoint in computer science, but is directly analogous to the view taken by Shannon in developing his theory of communication [35]. The semantic content of a system is not required to discuss many of its aspects; it returns only in the latter part of the paper to discuss strategy.

This major section elaborates on that viewpoint and culminates in the conclusion that one only manages the *average behaviour* of systems. It presents the notion of *convergent behaviour*, and argues that convergence is a desirable (and approximately necessary) condition for stability of systems.

We begin by taking the following heuristic definition of policy, as a description of how a system is configured and used, as an axiom:

Assumption 1. A *policy* is a description of what is intended and allowed of a system and its behaviour. The exact nature of policy remains to be determined.

Example 5. The set of all executable programs installed on a computer, together with configuration decisions, a code of conduct for users and for usage, form a policy. This contributes to determining the activities which will be performed by the computer. Access controls are also a part of policy: some programs can be restricted, by access controls or resource limitations. Rules applied to users are less predictable than rules

applied to machines: some users will misuse the programs, either by accident or by will, thus a policy can never be identified with absolute control.

In simple terms, it is a detailed specification of the human-computer system. The purpose of this section is to refine this concept into a form which approximates the real situation experienced in system administration. Policy must somehow encompass the issues above and be formalized to the point at which it can be submitted for analysis.

The identification of a computer as a dynamical system, obeying clear laws and methods of analysis, is central to the attempt to model its behaviour by a variety of analytical and numerical techniques. The motivation for this is to achieve a level of abstraction which is appropriate to a high-level description of the user-machine-administrator interaction. The study of dynamical systems has a long history and serves as a highly convenient framework for discussion.

Assumption 2. A computer system is a dynamical system whose average behaviour can be modelled by continuous functions of time $q(t)$ for the purpose of discussing rates of change within the system. This continuum approximation is valid for descriptions of computer behaviour over sufficiently long times compared to the discrete time scale of its instruction cycle.

A justification of this assumption follows in the next section. The main reason for adopting this viewpoint is that it allows one to simplify the description of system changes, in the case where there is an approximately regular pattern of discrete changes.

2.1. Administrative configuration space

The basic states of a computer system are coded within its memory, consisting of both primary (RAM) and secondary (disk) storage. The memory takes the form of a pattern of bit values, written on memory that is referred to here as the configuration space of the system. We denote this R^1, in anticipation of higher level partitionings to follow.

Definition 1. Let $x_1 \in R^1$, label the coordinate (address) of a bit in the one-dimensional *configuration space* R^1, where x_1 ranges over its dimension $0 \leqslant x \leqslant D_1$. This is the space of objects onto which a configuration is written.

The time-evolution of a computer system is driven by its most primitive operation: the fetch-execute cycle. This means that time is discrete at its highest resolution. However, for the purpose of describing averages, it is useful to allow t to take on arbitrary real values, denoted by R^t.

Definition 2. Let $t \in R^t$ label the time coordinate at which an event occurs within the system.

Example 6. The fundamental system time is discrete; it belongs to the set of integers multiplied by a basic clock time scale T_c, i.e. $t = 0, T_c, 2T_c, \ldots, , nT_c$, where n is an integer but we might use t to label the mid-point of such an interval, or other arbitrary point when grouping such intervals in coarser grains.

Since T_c yields far greater resolution than one needs to discuss the administration of a system, we shall be looking at times thousands of times longer than this and greater. At these scales, the discrete nature of time can be made irrelevant by locally averaging over small details (see below).

Written onto the configuration space of bits is a pattern of values, zero or one, which changes over time; thus, a configuration may be defined as follows:

Definition 3. A lowest level *configuration* $q_1(x,t)$, on R^1, is a pattern of values associated with each point on the configuration space, for each position $x \in R^1$ at time t:

$$q_1 : R^1 \otimes R^t \to \{0,1\}, \tag{1}$$

where $\otimes$ denotes an outer product of two sets or spaces. The *state space* of q_1 is the discrete set $\{0,1\}$. Note that q is a scalar function of the bounded space R^1, so it has the form of a vector of values, each component labelled by a different coordinate x.

A configuration at $x_1 = z$ for some z, is a *chain*, or *process* which develops in time (see Appendix A for a review of chains). The whole configuration $q_1(x,t)$ can therefore be thought of as a set of D_1 parallel interdependent chains.

Example 7. The configuration space R^1 is the array of bits in a computer memory (including RAM, disk and other registers), which exhibits a bit-pattern $q_1(x_1,t)$ at time t. The bit pattern changes in time as a result of the fetch-execute cycle. The variation with x_1 is a result of the coded instructions for the development of the system. Since a computer approximates a universal turing machine, this can be thought of as the turing tape.

The language of chains is more appropriate here than the language of state machines because it is not the function of the system, rather its consistency with time and its maintenance that one is concerned with in system administration. It is assumed for the remainder of the paper that programming instructions can be coded within the configuration $q_1(x_1,t)$ in order to instruct the system. These instructions will clearly become a part of the system policy.

There are now two points to be made: (i) averaging leads to a different state space with a higher density of values in a given range, (ii) high level coding of information on R^1 allows us to view the basic objects of the system not as bits, but as objects with a greater range of values. Combining coding and averaging, one can view a computer system as a new set of interdependent processes with a high resolution. It becomes reasonable to approximate their development using a continuum model, in which any marked discreteness can be handled explicitly, but the actual resolution of the system can be ignored.

The discrete nature of the time t is not normally perceived by the users or the high level software running on the system, since the time interval is too small to be resolved by them. This is certainly true at the time scales over which system administration takes place. It is therefore convenient to consider only the average behaviour of the system over longer, more appropriate time intervals $\Delta t \gg T_c$. The procedure of averaging over short-term detail is also a strategy by which one can understand the long-term behaviour of processes without attention to irrelevant detail. The process of averaging a discrete function of system time is denoted here by expectation value angle brackets $\langle \cdots \rangle$. The effect of averaging a portion of a function over a number n of discrete points is as follows:

$$\langle f(x_1, \bar{i}) \rangle = \sum_{i=1}^{n} \frac{f(x_1, i)}{n}. \tag{2}$$

Note that there is a division by the number of points. Applying this to discrete time intervals, where $n = \Delta t / T_c$, about the mid-point of the interval $\bar{t}$ one has

$$\langle f(x_1, \bar{t}) \rangle = \sum_{t=\bar{t}-\Delta t/2}^{\Delta t/2} \frac{f(x_1, t)}{n}. \tag{3}$$

We shall make frequent use of this construction. Averaging of f, over n steps, makes the range of values of the configuration take on values from the rational numbers:

$$\langle q_1(x_1, t) \rangle \in m \left\{ \frac{T_c}{\Delta t} \times (\max\, q_1 - \min\, q_1) \right\}, \tag{4}$$

for integer $m = 0, \ldots, n-1$. Since q_1 consists of bits, $(\max\, q_1 - \min\, q_1) = 1$. As a mapping from parameter domain to range:

$$\langle q_1 \rangle : (R^1 \otimes R^t) \to Q_1, \tag{5}$$

where composition operator $\otimes$ denotes the product space and Q_1 is the set

$$Q_1(\Delta t) = \left\{ \frac{m}{n} \right\}, \quad m = 0, 1, \ldots, n-1. \tag{6}$$

As $\Delta t \to \infty$ (equivalently $n \to \infty$), the range of $Q_1(\Delta t)$ approximates that of an open interval of real values between 0 and 1, over longer discrete intervals. Because the basic time scale T_c is so much shorter than the scale of seconds and minutes at which administrative and user events take place, one can locally average over time intervals hundreds of times longer than T_c and still have sufficient time resolution to be able to ignore the discrete nature of the coarse grained time.

Assumption 3. Over time scales much greater than the basic time interval T_c, the average properties of a computer system may be treated in a continuum approximation, as real functions of time. This leads to many notational and descriptive simplifications.

The *state space* of this coarse grained description will be denoted by Q_ℓ, where ℓ will take a value explained in the next section.

2.2. Coded information

Although computer systems are bit-configurations, operated on by low level rules, this viewpoint ignores an important issue, which is the layered coding of information used to represent user-level data. Even the basic ASCII symbols nowadays require a representation of 7-bits; most data and processes require complicated representations that require one to deal with high level objects such as files and directories. It is at these levels that both the usage and the administration of the system take place, so it is this level to which one must address a model of administration. Describing change in such high level objects is subtle because the content might be ordered on R^1, but for the purpose of discussing rates of change, one may ignore the interpretation of the high level objects and view them as closed containers. This is analogous to a local averaging in time, and means that a representative numerical value can be used for each object.

The transition from low level to high level is accomplished by successive levels of grouping smaller objects into larger ones, such as the hierarchy of objects expressed in Table 1. For simplicity, we shall assume that all the objects have the same size. This is not the case in practice, but the assumption serves to avoid unnecessary additional specification and does not alter the argument, only the details.

These items represent effective properties of the system, i.e. new alphabets of non-overlapping objects. At level ℓ, the system may be considered ℓ-dimensional, since there are up to ℓ independent degrees of freedom. A degree of freedom is a capacity for independent change.

Definition 4. Let R^ℓ be the set of D_ℓ objects of level ℓ, and let Q_ℓ be the set of d_ℓ sub-objects within each element of R^ℓ (see Fig. 1). A *high level coding* of the configuration space R^1, is a mapping L, which coarse-grains a set of D_ℓ lower level objects into $D_{\ell+1}$ higher level objects, of size $d_{\ell+1}$:

$$L_\ell : R^\ell \to R^{\ell+1}. \tag{7}$$

The set of high level objects has coordinates $x_{\ell+1} \in \{0, \ldots, D_{\ell+1}\}$.

Table 1
A separation of scales in a computer system. At a certain level, human users begin to interact with the system, and thus form part of it

Level	Example objects
6	Groups, departments, LANs
5	Users, virtual machines, agents
4	Compound objects, files and attributes
3	int, float, char etc
2	bytes, words
1	bits (RAM, ROM, disk)

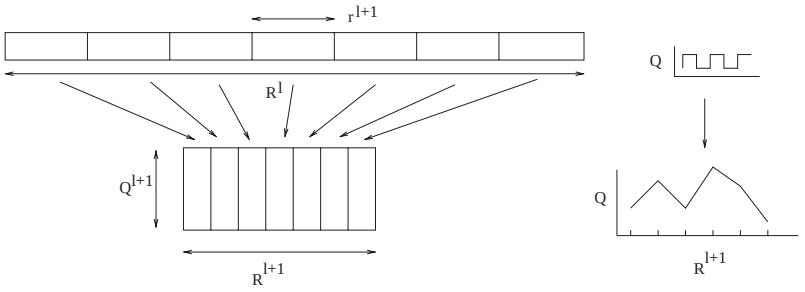

Fig. 1. A partitioning of the configuration space into higher level objects, is a mapping from one to two dimensions. The vertical scale is the range of the coding level; as the level increases so does the range of values taken by objects at that level. For the purpose of discussing rates of change, these values can be represented as numerical values.

A high level configuration is thus no longer defined in terms of bits, but in terms of high level objects such as numbers, symbols or file objects. The range of values represented by such objects increases exponentially in proportion to their size.

Definition 5. A level ℓ configuration $q_\ell(\vec{x}_\ell, t)$ is a homomorphism from the coded configuration space at each discrete time $t \in R^t$, to the object

$$q_\ell : (R^\ell \otimes R^t) \to Q_\ell(\Delta t), \tag{8}$$

where Q_ℓ is the set of states represented by a range of $1 \ldots d_\ell$ rational values.

Example 8. The configuration space R^ℓ is a set of high level objects in a computer memory which contains a set of high level data $q_\ell(x_1, x_2, \ldots, x_\ell, t)$ at time t. This pattern of high level data changes in time as a result of the operation of software, like the operating system. The number of different values in the range of $\langle q_\ell \rangle$ is of the order $2^{d_\ell} \times \Delta t / T_c$.

Definition 6. A *variable* $q(\vec{x}, t)$ at coordinate $\vec{x}$, is a synonym for $q_\ell(x_1, x_2, \ldots, x_\ell, t)$.

$$q(\vec{x}, t) \equiv q_\ell(x_1, \ldots, x_\ell, t). \tag{9}$$

This notation is used for convenience; a subscript ℓ is deliberately suppressed on $q(\vec{x}, t)$, as well as on the coordinate x. q is a scalar function of a vector of coordinate labels; i.e. it can also be interpreted as a vector of vectors of scalar values. If ℓ is chosen at some fixed level which characterizes the lowest level of interaction between users and computer system, then the set $q(\vec{x}, t)$ may be called a level ℓ state of the system at time t.

Influences from outside the system, i.e. from users, are limited by the way the interaction (system interface) is coded at a high level. Users actions are filtered through high level interfaces. The environment of the system (next section) presents itself to

Table 2
The separable time-scales for changes in a computer system interacting with an environment

Stochastic open system	Time scale
Fluctuations, system operations δq environmental changes	$T_c \sim T_e < T$
Cycles of persistent behaviour	$T \equiv 2\pi\omega^{-1}$
A coarse grain of N cycles	$\Delta t = NT \gg T$ i.e. $(N \gg 1)$
User/policy time scale	$T_p \gg T$
Long term behavioural trends	$T_b \gg T_p$

the computer as a interaction which is effectively coded at a level ℓ. Even though the intrinsic information content of the environment is greater than can be represented by level ℓ resolution, the environment's projected image cannot exceed this level of detail in R. There is thus an intrinsic uncertainty in the response of the system to its environment, which occurs because the interaction has a digital granularity. The entropy of the environment therefore limits the maintainability of the system S.

2.3. Systems and dynamics

This section presents a view of computer systems interacting with external agents, such as users and network clients. This interaction is important to system administration because the arrival of information from external sources affects the configuration state. The information from the environment has a potentially very high entropy. A table of the time scales referred to below is given in Table 2.

Definition 7. Let a *closed dynamical system S* be defined to consist of (i) a fixed configuration space R^ℓ, (ii) an initial configuration $q(x,t_i)$ on R^ℓ at time $t=t_i$, and (iii) a rule for subsequent time development, mapping a configuration $q(x,t)$ to a new configuration at $q(x,t')$, where $t'=t+\mathrm{d}t$, for a small increment $\mathrm{d}t>\Delta t$. The mapping is represented by a transition matrix

$$U_t : R^\ell \otimes R^t \otimes Q_\ell \to R^\ell \otimes R^t \otimes Q_\ell, \tag{10}$$

where each iteration yields

$$U_t : q(x,t) \to q(x,t+\mathrm{d}t), \tag{11}$$

so that the nth application of the map to $q(x,t_i)$ yields $q(x,t_i+n\,\mathrm{d}t)$. Represented as a transition matrix operator $\hat{U}_t(t,t')$, this becomes

$$q(x,t+\mathrm{d}t) = \hat{U}_t(t+\mathrm{d}t,t)q(x,t). \tag{12}$$

The functions $q(x,t)$ will be considered at least once differentiable with respect to time, i.e. piecewise continuous. [1]

[1] Note the use of the continuum approximation is only for convenience in what follows. It is not a necessary device, only a notational convenience.

Example 9. A physically isolated computer which receives no input, from any device, is a closed system. The computer has a fixed amount of memory (configuration space). Its initial configuration is the installed software, plus the contents of all files and registers at start-up. The rule for time development is determined by the processor's fetch-execute cycle on the software which is part of the initial configuration. It takes memory locations, operates on them and alters other memory locations. The mapping is not one-to-one; e.g. addition is two-to-one, but all the operations are completely determined by the configuration inside the memory R^1. Another example of a closed system is a physically isolated human being. The dynamical system is thus an abstract definition of a process.

Closed dynamical systems are completely determined by U_t and $q(x,t_0)$, for all times, but this is not the case for true computer systems, since any useful computer must receive input and output. This requires it to interact by coupling to the outside world.

Definition 8. An *interaction* U_I between two dynamical systems S_1 and S_2 is a closed map on the combined systems $(S_1 \cup S_2)$, such that both systems determine the time developments of both. Let the configuration space of S_1, S_2 be R_1, R_2, with coordinates $\vec{x}_1, \vec{x}_2$, respectively, and let these have configuration functions q_1, q_2. The interaction between the systems is a mapping

$$U_I : (R^\ell \otimes Q_\ell)_1 \cup (R^\ell \otimes Q_\ell)_2 \to (R^\ell \otimes Q_\ell)_1 \cup (R^\ell \otimes Q_\ell)_2, \tag{13}$$

such that configurations in $q_1(\vec{x}_1, t + \mathrm{d}t)$ and $q_2(\vec{x}_2, t + \mathrm{d}t)$ depend on the state of both systems at time t, i.e. the probability for the transition $\langle q_1(t')|q_1(t)\rangle$ depends on $q_1(t)$ and $q_2(t)$ for some domain of $t < t'$. In matrix operator form, this can be represented as

$$\begin{pmatrix} q_1(\vec{x}_1, t + \mathrm{d}t) \\ q_2(\vec{x}_2, t + \mathrm{d}t) \end{pmatrix} = \begin{pmatrix} \hat{U}_{11} & \hat{U}_{12} \\ \hat{U}_{21} & \hat{U}_{22} \end{pmatrix} \begin{pmatrix} q_1(\vec{x}_1, t) \\ q_2(\vec{x}_2, t) \end{pmatrix}, \tag{14}$$

where the presence of non-zero $\hat{U}_{12}$ and $\hat{U}_{21}$ implies an interaction.

Example 10. Consider two isolated systems, e.g. two computers, or a computer and a human, or two processes within a computer system, and equip them with a communications channel. The ability to communicate means that values in the configuration of one can be result in an alteration of values in the other, and vice versa. There is thus a generalized mapping of the two systems into one another, according to the rules contained in both.

The effect of an interaction is to modify the behaviour of each system. The resulting dynamics must be viewed in the combined system $S_1 \cup S_2$ to observe the full closure of the system; but usually it is viewed in only one of the sub-systems S_1 or S_2 as the projective image $(S_1 \cup S_2) \cap S_n$, where $n = 1, 2$. The rules governing the other system

S_2 are not available to S_1 directly, thus The meaning of $\cap$, in vector form, for an observer in S_1 is thus

$$q_1(\vec{x},t) = (1,0)\begin{pmatrix} q_1(\vec{x}_1,t) \\ q_2(\vec{x}_2,t) \end{pmatrix} \tag{15}$$

at any time t, hence the influence of S_2 seems non-deterministic.

In the example above, one might consider the effect of input on a local computer, from a remote computer, by looking only at what happens in the local computer. This is not the full picture, and thus the behaviour of the interacting local computer will not be completely predictable based only on the information in the local system. This lack of predictability presents a real issue to be dealt with in the administration systems coupled to users and networks.

The interaction which is of principal interest in the case of computer systems, is that with the users of the system, and with networked peers. This ensemble of influences is the combined effect of an external world of considerable complexity. We can now define the environment of a computer system in a more precise way, using the foregoing definitions.

Definition 9. Let $C_i(q(\vec{x}_i,t))$ be the complete configuration of host i in an ensemble of systems, $i = 1,\ldots,N$. The *environment* of a system S refers to an ensemble of mutually interacting systems, interacting with S. An ensemble E of mutually interacting systems develops by the following rule:

$$\begin{pmatrix} C_1(\vec{x}_1,t+\mathrm{d}t) \\ C_2(\vec{x}_2,t+\mathrm{d}t) \\ \vdots \\ C_N(\vec{x}_N,t+\mathrm{d}t) \end{pmatrix} = \begin{pmatrix} \hat{U}_{11} & \hat{U}_{12} & \cdots & \hat{U}_{1N} \\ \hat{U}_{21} & \hat{U}_{22} & & \\ \vdots & & & \\ \hat{U}_{N1} & & & \hat{U}_{NN} \end{pmatrix} \begin{pmatrix} C_1(\vec{x}_1,t) \\ C_2(\vec{x}_2,t) \\ \vdots \\ C_N(\vec{x}_N,t) \end{pmatrix}. \tag{16}$$

If the system S has configuration $C_1(x,t)$, then the environment of S is the remainder of the configurations $C_2 \cup C_2 \cup \cdots \cup C_N$.

In computers coupled to users and networked hosts, the informational complexity of E is generally assumed to be much greater than that of S.

Example 11. Two computers, with a direct cable between them, interact directly by sharing and exchanging information. These computers are each others' environments. Computers interact indirectly through third parties, or by chains of interactions which lead to multiple, nth order and cyclic dependencies. A user and a computer interact directly via a keyboard, or indirectly by passing through another user or computer etc. An environment can be any of these scenarios, but usually it is a vast, complex web of such inter-relationships with a very high informational complexity. In general, the environment of a system is anything external which can affect it in any way.

The significance of this definition is that it illustrates that the environmental interaction of a local computer system, which leads to unpredictability within the scope of the local system, is nothing more than an ensemble of systems which are essentially comparable to the local system. In the case of a standalone computer, with no network connection, the environment is limited to a single user, but this user interacts with the rest of the world. This leads to great complexity, and thus the interaction is normally viewed as being a stochastic process.

In the study of dynamical systems, the environment is not normally modelled as a detailed entity owing to its complexity; rather one considers the projected image of the environment in the main system of interest. The essence of the definition is that the environment leads to a projected component in S which appears to be partially random (stochastic), because the information about cause and effect is not available. This causes S to behave as an *open dynamical system*, defined below.

Definition 10. An *open dynamical system* is the projection of an ensemble of interacting systems $E = \{S_1, S_2, \ldots, S_N\}$, onto S_1. The time development of the open system, may be considered an endomorphism over a noisy channel, since information from the rest of the ensemble affects $q_1(x,t)$. The closed rule for development is

$$C_1(\vec{x}_1, t + \mathrm{d}t) = (1, 0, \ldots, 0) \begin{pmatrix} \hat{U}_{11} & \hat{U}_{12} & \cdots & \hat{U}_{1N} \\ \hat{U}_{21} & \hat{U}_{22} & & \\ \vdots & & & \\ \hat{U}_{N1} & & & \hat{U}_{NN} \end{pmatrix} \begin{pmatrix} C_1(\vec{x}_1, t) \\ C_2(\vec{x}_2, t) \\ \vdots \\ C_N(\vec{x}_N, t) \end{pmatrix} \tag{17}$$

or in component form:

$$C_1(\vec{x}_1, t) = \hat{P}(i = 1) \sum_j \hat{U}_{ij} \vec{C}(\vec{x}_j, t), \tag{18}$$

where $\hat{P}$ is the projection operator in Eq. (17).

This definition is an admission of unpredictability in a system that is open to outside influence. Indeed, this unpredictability can be stated more precisely:

Lemma 1. *The configuration state of an open system S is unpredictable over any interval $\mathrm{d}t \sim T_e$, the scale of environmental fluctuations.* (*See Table 2.*)

Proof. This follows trivially from Eq. (17).

$$C_1(\vec{x}_1, t + \mathrm{d}t) \neq \hat{C} q_1(\vec{x}_1, t), \tag{19}$$

for any $\hat{C}$ coded within S, since $C_1(\vec{x}_1, t + \mathrm{d}t)$ is determined by information unavailable within S, iff $U_{ij} \neq 0$ for $i \neq j$, which defines the open system. □

We now wish to attempt to describe a class of systems that interact with a real world environment. These systems must be able to perform a useful computing

function, so completely arbitrary systems need not be considered. In such a system, it is convenient to decompose the time-evolution of any $C(\vec{x},t)$ into slowly varying stable parts and rapidly varying noise. This is only possible for systems which exhibit "sufficient stability", but the meaning of sufficient stability can be explained by appealing to self-consistency and the notion of maintainability. This will be done by introducing the statistical notion of *persistent states*.

2.4. Maintainability and fluctuations

The notion of system administration is closely allied with that of maintenance. One of the aims of this work is to discuss the implications of maintenance, without bringing the semantics of maintenance into the discussion, i.e. to think of maintenance as a response to a stochastic process. There is a parallel here to Shannon's discussion of communication theory [35]. To overlay the language of stochastic systems onto the maintenance process, one needs to make a separation into what is normal and what is anomalous. Contrary to what one might expect, this separation can be made self-consistently, without any reference to the semantic content of a policy. To begin with, one requires a basic axiom:

Assumption 4. The short-term stability of a system is a desirable quality that enables it to perform a function predictably. One is not interested in managing systems which cannot achieve this minimum level of stability, since these cannot perform any reliable function.

This assumption of medium-term stability guarantees that it will be possible to make the separation of normal and anomalous. Not all systems admit to such a clean separation (e.g. systems which exhibit approximate statistical self-similarity do not have this behaviour, but they are not of interest here, since one cannot begin by assuming that the system is hopelessly out of control).[2] The meaning of normal and anomalous is not automatically clear, but studies have indicated that it is self-consistent to associate these with slowly and rapidly varying changes, on the time scale of user-behaviour [11].

Example 12. Computational processor operations are the fastest changes which occur in a computer. Rates of human behaviour are millions of times slower than this, and may be called medium term. Long term changes, over months and years, are hundreds or thousands of times longer than this again. In order for humans to perceive computers as useful, the rapid changes must leave general features of the computer approximately constant over medium, human time-scales. A human definition of what is normal thus refers to something regular or constant over a time scale which is greater than that of the medium, human scale of changes.

[2] The stability of a system is a statistical concept that can be made precise by through the Central Limit Theorem and the distributions of Lévy. That discussion is beyond the scope of the present paper. See, for instance, Ref. [3] for an introduction.

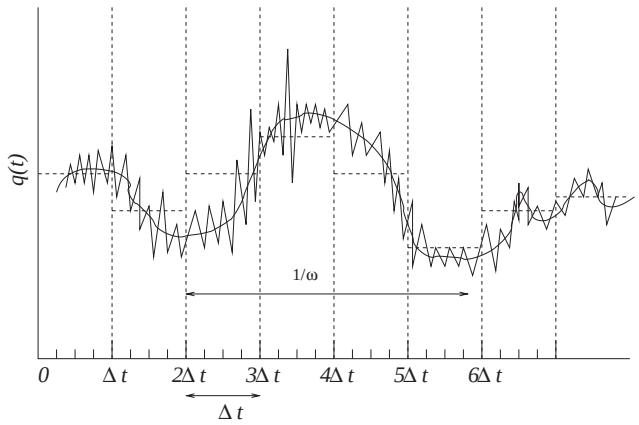

Fig. 2. An schematic picture of the separation of scales in an open dynamical system, which satisfies Eq. (20). The jagged line represents the highest resolution view of what $q(t)$ is doing. The solid curve is a short-interval local average value of this behaviour, and the solid blocks (dotted line) are a much coarser local average. The order of magnitude of the system's approximate oscillations is ω^{-1}.

The separation of slowly and rapidly changing configurations can be made precise by observing the system through a *local averaging procedure*. This is the aim of the next sections. We shall refer to the schematic diagram in Fig. 2.

Suppose that such a definition exists for the separation, as it does in the cases of interest to; one may then write the exact configuration of the system, at any time, as a sum of two parts:

$$q(x,t) \equiv \langle q(x,t)\rangle + \delta q(x,t), \tag{20}$$

where $\langle q\rangle$ refers to a slowly-varying, local average value of q, and δq refers to a rapid fluctuating, stochastic remainder. This decomposition will be used later; its principal advantage is in isolating which parts of the environment of users lead to a stable (smooth) average configuration and which parts tend to be rough and unpredictable. In systems of interest, one expects $|\delta q| \ll |\langle q\rangle|$.

Note also that, by definition, $\langle \delta q\rangle = 0$, thus the fluctuations are evenly (though not necessarily symmetrically) distributed about the local mean value. This means that, if fluctuations tend in one particular direction, they will drag the mean value with them, preserving their zero mean. If one wishes to avoid a change in the mean value, then one must either offer dynamical resistance to this kind of monotonic drift, or respond to it with a counter-change, which balances it on average. This concept of preserving the mean behaviour provides us with a notion of maintenance.

Definition 11. Let a *task* $\tau(t)$ be a system contained within a subset $r \in R^{\ell}$ of a system S:

$$\tau(\vec{x},t) = q(\vec{x},t):\ \vec{x} \in r, \tag{21}$$

where the restricted coordinates x ranges only over the subspace. A task is a representation of an autonomous process executed on related chains (see the appendix), and thus evolves in time according to its own irreducible transition matrix $\hat{U}_t$. A task is closed if it is a closed system, and open if it is an open system.

The concept of a task is needed to discuss a part of a system which operates autonomously for some purpose, such as maintenance.

Example 13. A task is an autonomous sub-part of a system, like a computer program or external changes made by a user. A computer program is a task which consists of a text segment and some workspace, coded and stored within a subset of the high level configuration of the system. If the program is closed, it does not affect anything outside of its own resources; if the program is open it can affect the state of the rest of the system also. In a distributed environment a program on one host can affect the state of a program on another host. The actions of a human interacting with the system can lead to a task.

We now have a representation of programs running on the system as well as processes carried out but external agents (other computers and humans). One can now define maintenance in terms of the effect of sub-systems on the total system, as a fluctuation–dissipation result.

Definition 12. Let $\tau_M(\vec{x},t)$ be a task in a system S with configuration subspace r, and $\tau_{M_c}(\vec{x},t)$ be the complement to the subspace, i.e. the remainder of the configuration of S, with configuration subspace r_c; then $\tau_M(\vec{x},t)$ is said to be a *maintenance* task if $\{\tau_M(x\in r,t)\}$ is an open system and

$$\frac{\mathrm{d}}{\mathrm{d}\bar{t}}\left\langle \sum_{x\in r}\log\,\tau_M(\vec{x},\bar{t})+\sum_{y\in r_c}\log\,\tau_{M_c}(y,\bar{t})\right\rangle < \frac{\mathrm{d}}{\mathrm{d}\bar{t}}\left\langle \sum_{y\in r_c}\log\,\tau_{M_c}(y,\bar{t})\right\rangle. \tag{22}$$

In other words, the presence of a maintenance task τ_M reduces the total rate of change of the average configuration state $C(\vec{x},t)=\{\tau_i\}$ in S; i.e. it exerts a balancing influence on fluctuations δC within any coarse time interval Δt. If the rate of maintenance is less than the rate of fluctuation, it will lead to a window of uncertainty in the value of $\langle C\rangle$, which can result in a real change of average state. Note that the logarithms make the ordering and scale of the changes unimportant (see Appendix A). This is a characterization of the change of information in the configuration, where the spatial ordering is unimportant.

The definition of maintenance allows for gradual evolution of the idealized persistent state (e.g. a slow variation in the average length of a queue), since the average value can be slowly modified by persistent fluctuations. This change of the persistent state is said to be *adiabatic* in statistical mechanics, meaning slow compared to the fluctuations themselves. A summary of time scales is shown in Table 2.

In order to describe and implement a *system policy*, for managing the behaviour of a computer system, it must be possible to relate the notion of policy to rules

and constraints for time-evolution which are programmed into $q(\vec{x},t)$. Such rules and constraints are coded as software in $q(\vec{x},t)$, or are issued verbally to users in the environment of the system. The behaviour of the configuration state is not completely deterministic and is therefore unpredictable. By separating slowly and rapidly varying parts, using a local averaging procedure, we find an average part that is approximately predictable.

We note, as a commentary, that while this shows that the rate of change in the system can be arranged to maintain a particular state over a consistent set of time-scales, it does not specify a unique route to such a state through the state space (including space and time scheduling) of the human–computer system [13,38]. The existence inequivalent different routes must be handled by a framework in which they can be compared in some system of returned value. The theory of games, as presented in the final sections of the paper, is suitable for selecting such a route. The existence of a unique path has been addressed in Ref. [9].

2.5. Symmetries in $q(\vec{x},t)$ and equivalent policies

A high level partitioning of the configuration space, which evolves according to rules for time-development at the same level, leads to the appearance of symmetries, with respect to the dynamical evolution of a computer system. A symmetry may be identified, whenever a change in a configuration does not affect the further evolution of the system except for the order of its elements. The configurations of the system which are symmetrical, in this sense, form a group.

Definition 13. A group $\mathscr{G}$ of transformations $X \to g(X)$, for some $X \subset R^{\ell}$ and time t, that reparameterizes the coordinate labels:

$$g : R^{\ell} \to R^{\ell}. \tag{23}$$

is a *symmetry* of the high level configuration $q(\vec{X},t)$, if the transformation of the configuration domain

$$q(\vec{X},t) = q(g(X),t), \tag{24}$$

is an identity, and $g \in \mathscr{G}$ is a bijection.

Thus the permutation of process address labels is unimportant to the configuration, as is any change in $\hat{U}_t$ which leads to a relabelling in the future. Since the deterministic part of the mapping $\hat{U}_t$ is coded in $q(\vec{x},t)$, this includes changes in the way the system evolves with time.

Definition 14. A group Γ of transformation $Q \to \gamma(Q)$ that reparameterizes the state value labels,

$$\gamma : Q_{\ell} \to Q_{\ell}. \tag{25}$$

is a *symmetry* of the state space Q_ℓ, if

$$q(\vec{x},t) = \gamma(q(\vec{x},t)), \tag{26}$$

is an identity, and $\gamma \in \Gamma$ is a bijection.

Symmetries of state are hard to describe formally (they include issues such as the presence of comments in computer code, irrelevant orderings of objects, and so on), but they have a well-defined meaning in real systems.

Example 14. One possible symmetry transformation on a system would be to rename every reference to a given file: the result would have no effect on the behaviour of the system. Another example would be to intersperse instructions with comments, which have no systemic function. Another an important symmetry of systems is independence of the system to changes in parts of the configuration space R^ℓ which are unused by any of the programs running on the system.

The presence of symmetries is of mainly formal interest here, but their inclusion is necessary for completeness. The notion of equivalence motivates the definition of a factor set of inequivalent configurations

$$P(t) \equiv \frac{q(R^\ell,t)}{\mathscr{G} \otimes \Gamma} = \frac{C(x,t)}{\mathscr{G} \otimes \Gamma}, \tag{27}$$

which signifies one representative configuration from the set of all equivalent configurations. This factored system is now uniquely prescribed by an initial configuration, rules for time development and the environment. It is scarcely practical to construct this factor set, but its existence is clear in a pedantic sense.

Up to stochastic noise, the development of the open system is completely described by this configuration, which includes the programs and data which drive it. Conversely, the behaviour at level ℓ is completely determined by the specification of a $P(t)$. This is therefore a natural object to identify with *system policy*.

In practice, only a part of the configuration will directly impact on the evolution of the system at any time. If a constant part of $P(t)$ can be identified, or if $P(t)$ is sufficiently slowly varying, then this quantity plays the role of a *stable policy* for the system. If no such stability arises, then the policy and configuration must be deemed unstable.

How does this definition of policy fit in with conventional, heuristic notions of policy? A heuristic definition is (i) a system configuration, (ii) rules for behaviour of the system (programmed), (iii) rules for human users (requested), and (iv) a schedule of operations. Of these, (i) and (ii) may be coded into the configuration space without obstacle. (iii) needs to be coded into the environment, however the environment is not a reliable channel, and can only be expected to obey policy partially, thus there will be an unpredictable component. (iv) is also programmed into the computer, but there is also a schedule of random events which belongs to the environment; this also leads to an unpredictability. The resulting 'error' or tendency towards deviation from steady behaviour must be one of two things: a slow drift $\Delta P = P(t) - P(t')$ (systematic error)

or a rapid random error $\delta P(t)$ (noise). In order to use a definition of policy such as that above, one is therefore motivated to identify the systematic part of system change.

2.6. Convergence

The notion of convergence is related to the idea of the fixed point of a mapping [30]. If $q' = U(q)$ is any mapping, then a fixed point q^* is defined by

$$q^* = U(q^*). \tag{28}$$

This definition is too strict in a dynamical system, rather we need a limiting process that allows for some fuzziness:

$$q^* - U(q^*) < \varepsilon. \tag{29}$$

As defined, a policy is neither a force for good nor for evil, neither for stability nor for chaos; it is simply an average specification of equivalent system behaviours. Clearly, only a certain class of policies has a practical value in real systems. This refers to policies that lead to short term stability, thus allowing a stable function or purpose to be identified with the system. A system which modifies itself more rapidly than a characteristic human time scale T_{p}, will not have a stable utility for humans.

The notion of *convergence* is especially useful [5,15,16] for regulating systems. A system which possesses a cycle that persists over a given interval of time can be defined as having predictable behaviour over that interval.

Definition 15. A *convergent policy* $P(t)$, of order n, is one whose chain of time transitions, expressed by the operator $\hat{U}_t$, ends in a fixed point configuration $q(\vec{x}, t_f)$, for all values x and times $t_i > t_f$, $f \leqslant n$. i.e.

$$(\hat{U}_t)^n_{ij} q_j(\vec{x}, t_i) = q_i(\vec{x}, t_f) \quad \text{for some } n \geqslant 0, \quad t_i < t_f, \tag{30}$$

where i, j run over a closed number of systems.

Note first that this result is true at any scale: for a system, or for a subsystem within a closed system. Thus the extent of convergence depends on the extent of the closure under which the operators for time development are constrained to work. Indeed, if one starts from the bottom-up, then convergence can extend through any scale by block decomposition, and the operators are automatically orthogonal (see Ref. [9]).

Next, the fixed configuration on which the time development ends is sometimes said to be 'absorbing', since once the system has entered that state, it does not change again. In the language of system administration, one says that the system has converged. In a stochastic, interacting system, this finality cannot be guaranteed precisely. Within a short time period a change away from the final state can occur at random, thus it is useful to define the notion of average convergence.

Definition 16. A *convergent average policy* $P(t)$, of order n, is one whose average behaviour in time ends in an average state $\langle q(\vec{x},t_f)\rangle$ between any two times t_i and t_f, such that $t_f - t_i > \Delta t$.

$$\langle(\hat{U}_t)^n q(\vec{x},t_i)\rangle = \langle q(\vec{x},t_f)\rangle \quad \text{for some } n \geqslant 0, \quad t_i < t_f, \tag{31}$$

where $\langle\ldots\rangle$ is any local averaging procedure.

This condition is weaker, because it allows the final state of exhibit fluctuations that are balanced within the time of the averaging interval.

A discrete chain interpretation of periodicity may be found in [23]; it is convenient here to use the continuum approximation. Over the time interval, it can thus have the general form:

$$\begin{aligned}\langle q(\vec{x},t)\rangle &= \left\langle Q_0(x) + A(t)\,\mathrm{Re}\,\exp\left(\mathrm{i}\,\frac{\omega}{n}\,t\right)\right\rangle,\\ &= Q_0(x),\end{aligned} \tag{32}$$

i.e. it has an average value and oscillations whose average effect is zero. Since Q is positive, $A < Q_0/2$. Notice that a process that has converged becomes memory-less, i.e. its dependence on previous states becomes irrelevant.

A policy in which the average resource usage is constant over the policy timescale T_p is a convergent average policy; e.g. a policy of deleting all old temporary files, killing old processes and so on, or by adding new resources, so that fraction of used resources is constant on a average of a few cycles.

Another example of convergence would be one in which errors in a configuration file, made by human error, were corrected by an automatic process, within a short time interval, by regular checkups, thus preserving the average condition. This has already become a common practice by administrators [5], so convergence is a commonly used strategy for achieving stability.

2.7. Persistence

Implicit in the foregoing discussion of averages are two notions of stability which now crave definition, at the level of the continuum description. These form the basis for a self-consistent definition of convergent system policy, which show that system administration is a soluble problem, within clear limits.

Definition 17. A *locally averaged state* $\langle q(\vec{x},t)\rangle$ is a local coarse graining procedure, i.e. a classification of the time-variation of $q(\vec{x},t)$ into intervals with some characteristic length $\Delta t \gg 1/\omega$, Several cycles are averaged over, and the entire interval is replaced with a common value.

$$\langle q\rangle(\vec{x},\tilde{t}) \equiv \frac{\int_{\tilde{t}-\Delta t/2}^{\tilde{t}+\Delta t/2} q(\vec{x},\tilde{t})\,\rho(\tilde{t})\,\mathrm{d}\tilde{t}}{\int_{\tilde{t}-\Delta t/2}^{\tilde{t}+\Delta t/2} \rho(\tilde{t})\,\mathrm{d}\tilde{t}}, \tag{33}$$

where $\rho(t)$ is some weighting function, to be specified.

The coarse graining procedure is the analogy of level ℓ coding in the configuration space, only here it is applies to the time dimension. It is a redigitization of the time-line. Local averaging procedures are used to separate structures in the time evolution of systems at different levels. One begins by digitizing a details function of time into coarser blocks (like a pixelized image). As one zooms out, the behaviour of a local average looks smooth and continuous again.

Definition 18. A *persistent state* $\Psi(\vec{x},t)=q(\vec{x},t)$ is a configuration for which the probability of returning to a configuration $\Psi(\vec{x},t_0)$ at a later time $\Psi(\vec{x},t_0+\Delta t)$, for $\Delta t>0$ is 1. In the continuum description, persistence is reflected in the property that the rate of change of the average state $\langle\Psi\rangle$ be much slower than the rate ω of $\delta\Psi$:

$$\left|\frac{1}{\langle\Psi\rangle}\frac{\mathrm{d}\langle\Psi\rangle}{\mathrm{d}\bar{t}}\right|=\left|\frac{\mathrm{d}}{\mathrm{d}\bar{t}}\log\langle\Psi\rangle\right|\ll\omega, \tag{34}$$

i.e. the fast variation extends over several complete cycles, of frequency ω, before any appreciable variation in the average is seen. (See Table 2 for a summary of time-scales.)

Example 15. A system job queue has a fluctuating queue size, whose average length can be determined as a matter of policy, based on observed behaviour, by choice of a scheduling. Since the arrival of jobs in the queue cannot be accurately predicted, the average length will vary slowly, as long as jobs are expedited at only approximately the same rate as they arrive. There is thus a short term cycle; add job, expedite job, that increases then decreases the queue size. A persistent state is much larger than this cycle. It means that the cycle is locally stable. If the system is characterized by a convergent policy (incoming jobs are indeed expedited at an appropriate rate), then any fluctuations occurring at the rate ω will be counteracted at the same rate, leading to a persistent (slowly varying average) state. See Fig. 3.

Thus the meaning of a convergent policy is its resulting persistence. Thus, policy itself must be identified with that average behaviour; this is the only self-consistent,

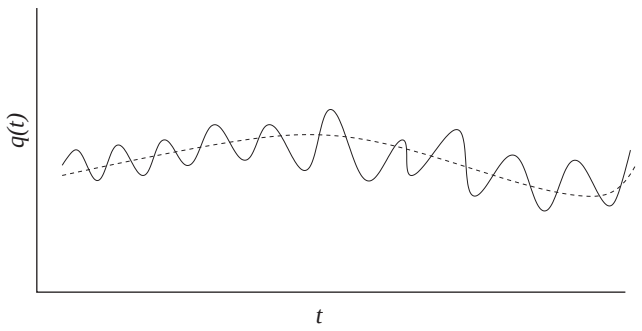

Fig. 3. A persistent state is one in which the cycle does not vary appreciably over many cycles. Here one sees small variations repeated many times, on a slowly varying background.

sustainable definition, as long as there are stochastic variables in the system, due to environmental interaction.

The development of an open system is stochastic and that this naturally motivates a local averaging procedure. The effect of the time development, with random noise, is described by the time-evolution (the so-called density matrix) operator for an open system $\rho_{\mathrm{E}}(t)$, which is equivalent to an effective evolution operator for an open system, i.e.

$$\rho_E(t-t') = U(t-t'), \tag{35}$$

this normal time development is a natural averaging procedure.

Lemma 2. *The aggregate of incremental changes of state during the time development of a system over an arbitrary interval $\tilde{t}$, centred on its midpoint $\bar{t}$ leads to a locally averaged state.*

Proof. The time development is traced over time, by compounding the effect of this operator. This is analogous to repeatedly performing the fetch-execute cycle. This is modelled by integrating the signal over a period of time $T_{\mathrm{e}} \ll \bar{t} \ll T_{\mathrm{p}}$, where T_{e} is a time scale over which the environment changes and T_{p} is a time scale over which policy changes:

$$\begin{aligned} P'(\bar{t}) &= \int_{\bar{t}-\tilde{t}/2}^{\bar{t}+\tilde{t}/2} \mathrm{d}(t-t')P(\vec{x},t-t')U_t(t-t') \\ &= \int_{\bar{t}-\tilde{t}/2}^{\bar{t}+\tilde{t}/2} \mathrm{d}tP(\vec{x},t)\,\rho_{\mathrm{E}}(t). \end{aligned} \tag{36}$$

One now observes that, if one performs this operation over a coarse-graining interval then, up to a normalizing factor, this is the definition of a local average. By explicitly dividing by a normalizing factor, one thus determines that the time-evolution of a stochastic system naturally defines an average over fluctuations. The split one makes in Eq. (20), therefore ensures that the fluctuations are zero on average, distributed about the average behaviour, so by blurring out these fluctuations, one is left with a unique description of the average behaviour. □

This observation is referred to in statistical physics as the *mean field* approximation. The normalized, coarse-grained policy may now be written as

$$\begin{aligned} \langle P(\vec{x},\bar{t})\rangle &= \frac{\int_{\bar{t}-\tilde{t}/2}^{\bar{t}+\tilde{t}/2} \mathrm{d}tP(t)\,\rho_{\mathrm{E}}(t)}{\int_{t-\tilde{t}/2}^{\bar{t}+\tilde{t}/2} \mathrm{d}t\,\rho_{\mathrm{E}}(t)} \\ &= \left\langle \frac{q(\bar{t})}{(\mathscr{G}\otimes\Gamma)} \right\rangle, \end{aligned} \tag{37}$$

where $q(t)$ denotes the entire configuration, for all $\vec{x}$, at any level. In other words, the short term evolution of policy can be identified with a local average configuration in time; i.e. a set of locally average variables, at an appropriate coding level for the system.

2.8. A theorem about maintenance

With the meaning of the local averaged mean-field established, it is now a straightforward step to show that local averaging leads to persistence, and hence that this measure of stability applies only to locally averaged states. We thus approach the end of the lengthy argument of this section, which shows that policy can only be an agent for average system state. The theorem suggests that a strategy for maintaining stability in computer systems is to strive for convergence.

Theorem 1. *In any open system S, a policy $P(\bar{t})$ specifies a class of persistent, locally average states $\langle q(\bar{t})\rangle$ equivalent under symmetry groups $\mathscr{G}$ and Γ, if and only if $P(\bar{t})$ exhibits average convergence.*

Proof. From Lemma 1, in an open system S, a configuration is unpredictable over a time scale $T_e \sim \omega^{-1}$, hence a configuration can only be guaranteed persistent on average. We thus need only to show that a convergent average policy $\langle P(t)\rangle$, of order n, is persistent for a time $\Delta t \gg T$, since, by definition, this implies a set of equivalent persistent average configurations, under the available symmetries. Recalling the time scales, we consider an arbitrary interval

$$\Delta t = NT = \frac{2\pi N}{\omega}, \tag{38}$$

where $N \gg 1$, parameterized by

$$\bar{t} - NT/2 \leqslant \tilde{t} < NT/2. \tag{39}$$

If $P(t)$ exhibits average convergence, it varies no faster than

$$P(t) = P_0 + A(t)\,\mathrm{Re}\,\mathrm{e}^{-\mathrm{i}\omega t/n}, \tag{40}$$

on any part of the interval, for slowly varying amplitude $A(t)$ (where $|\partial_t A/A| \ll \omega$) and P_0 is a constant such that $A(t) < P_0/2$. The local average of this function is

$$\langle P\rangle(\bar{t}) = P_0 + \frac{1}{\Delta t}\,\mathrm{Re}\int_{\bar{t}-\Delta t/2}^{\bar{t}+\Delta t/2} A(\tilde{t})\mathrm{e}^{-\mathrm{i}\omega\tilde{t}/n}\,\mathrm{d}\tilde{t}, \tag{41}$$

where $\Delta t \gg 1/\omega$. The rate of change of this averaged policy may now be constructed, according to the rules of calculus:

$$\begin{aligned}\left|\frac{\mathrm{d}\langle P\rangle}{\mathrm{d}\bar{t}}\right| &= \lim_{\Delta t\rightsquigarrow 0}\left|\frac{\langle P\rangle(\bar{t}+\Delta t) - \langle P\rangle(\bar{t})}{\Delta t}\right| \\ &= \frac{\omega^2 An}{(2\pi N)^2}\left|\left[\frac{\sin(\omega t/n)}{\omega}\right]_{\bar{t}+\Delta t/2}^{\bar{t}+3\Delta t/2} - \left[\frac{\sin(\omega t/n)}{\omega}\right]_{\bar{t}-\Delta t/2}^{\bar{t}+\Delta t/2}\right| + \varepsilon \\ &\leqslant \frac{\omega n}{(2\pi N)^2}\left|4\times\frac{P_0}{2}\right| + \varepsilon.\end{aligned} \tag{42}$$

Here $\rightsquigarrow$ denotes an approximate limit, since a true limit is not possible. The error in evaluating this quantity $\varepsilon \ll P_0/N^2$, which is of the order or less than the error due to the continuum approximation itself, and may thus be neglected. The same result can also be obtained more clumsily without the continuum picture. Now, since $N \gg n \geqslant 1$, it follows trivially that,

$$\left| \frac{\mathrm{d}\langle P \rangle}{\mathrm{d}\bar{t}} \right| \ll \omega |\langle P \rangle(\bar{t})|, \tag{43}$$

hence

$$\left| \frac{1}{\langle P \rangle} \frac{\mathrm{d}\langle P \rangle}{\mathrm{d}\bar{t}} \right| \ll \omega, \tag{44}$$

and $\langle P \rangle$ is persistent. Finally, $P(t)$ is associated with a class of states, equivalent under a symmetry group $\mathscr{G}$, which can vary no faster than policy, since it is a part of the policy, hence a locally average state, resulting from a non-divergent policy specification is persistent. This completes the proof. □

The maintenance theorem provides a self-consistent definition of what a stable state is, and hence what a stable policy is, for a computer interacting with external agents (users, clients, etc). The implication is thus that system administration can be pursed as a regulation technique [17,22,24,27,34], for maintaining the integrity of policy, provided one can find a convergent average policy. It sets limits on what can be expected from a policy in a dynamical environment. Finally, the argument makes no reference to the semantic content of policy; it is based purely on information and timing.

It is interesting to note another theorem which is better known but also applicable (and very similar) to the stochastic and semantic views of policy as a propagating influence: it is simply a transcription of Shannon's channel capacity theorem for a noisy channel [35].

Theorem 2. *There exists a policy $P(t)$ which can evolve in time with arbitrarily few errors, i.e. the system can be forced to obey policy to within arbitrary accuracy.*

Shannon's original theorem stated that "there exists a message coding which can be transmitted with arbitrary few errors"; i.e. by creating a policy which is so strictly enforced as to police the activities of users in every detail, one could prevent users from doing anything which might influence the strict, predictable development of the system. Such a policy is possible if the average configuration of the host that it represents has sufficiently low entropy that it can be compressed into a part of the system dedicated to maintenance (error correction).

In contrast to the transmission of coded data, one is not interested in completely eschewing the environment, but adapting to it in a controlled fashion. Everything users do is formally a part of the environment. As long as the entropy of the system configuration is kept under a threshold level implied by the maintenance theorem, a meaningful regularity can be maintained in the propagation of the system, because the information content of errors will be much less than the total transmitted information.

3. Type I models—dynamical systems

The remaining sections of the paper illustrate how to formulate detailed models for understanding, deciding and optimizing policy, starting with the language of dynamical systems. This warrants two approaches: a stochastic type I approach, and a semantic type II approach. For type I models, some work has been done in this area already [6,7,11,21,26], so the details will not be repeated.

In a type I model of system state, the change in persistent state of the system is described as a stochastic drift (random walk) away from an 'ideal' average state, i.e. that state which is described by policy at an initial time $t = t_i$. The ideal state itself is characterized in terms of incremental changes of averaged system variables. In such a description, the intentions and individual actions which led to the state are washed out by the averaging process, and all that remains is a separation of scales into the slow change of averages and the rapidly changing spectrum of fluctuations. This is the approach used in Refs. [7,21,26] in order to study changing resource usage. This is the simplification afforded by type I models. While much can be learned from this, it is insufficient to describe system administration. The main topic for this paper is the type II model.

4. Type II models: strategic maintenance

By couching system administration as a game of strategy, one can formulate strategies in a framework that aims to achieve the larger goals of maintaining a persistent stable state and doing work.

In the preceding sections, we have approached the problem of stability in terms of single scalar measurable values that undergo stochastic fluctuations. This provides a notion of statistical stability in terms of fluctuation distributions and system integrity. The complexity of human computer systems makes this level of approximation inadequate to address the general case, however. More general notions of stability that include conflicts of interest amongst the users and system. In a type II description, the computer system is viewed as the chequerboard for a game of competition between motivated individuals [8]. A computer system is a community, of limited resources, where many individuals meet with goals, strategies and personalities. What one user does, affects neighbouring users; what one area of a network does affects neighbouring parts of the network [36,43]. The theory of games is, amongst other things, about the identification of stable equilibria or generalized fixed points, in which the mapping is not a smooth function but a piecewise continuous patchwork of distinct choices.

Traugott and Huddleston have pointed out [39] that it is often pertinent to view a local computer community as a single virtual machine, rather than as a conglomeration of individual hosts. In this context, the term computer system will be used to refer to the collective hosts of a local domain, or some appropriate logical unit of networked computers. It is taken for granted that there may be internal competition for resources and even conflict between competing parties.

In order to formulate a strategic theory of system administration we need to establish a set of possible goals, procedures and obstructions and state them in formal terms. The aim is then to postulate or derive strategies which best achieve those goals, given the essential constraints. From Section 2, it follows that a self-consistent goal is to attempt to achieve a persistent state of dynamical stability. There is a number of stages in this programme of study. A key assumption is the following: following.

Assumption 5. The short-term aim of benign users and clients is to produce useful work, with the aid of the system. The short-term aim of malicious clients is to maximize their control over system resources, vandalize the system, or confound system policy.

Similarly the role of the administrator motivates policy:

Assumption 6. The long term aim of system administration is to optimize the policy $P(t)$ for maximum productivity, insofar as this is allowed by local constraints. The short term aim is to keep the system as close the ideal average state $\langle q(t)\rangle$ as possible, by minimizing fluctuations δq.

The long- and short-term goals are compatible, because if the system is kept close to an idealized persistent state, it will survive in order to work for users. However, in a competitive environment, one user can produce work at anothers' expense, so there are non-trivial issues to be resolved. This includes competition between local users and remote users, e.g. users accessing web services.

This formulation of the ideal is particularly appealing from a mathematical point of view, because it is a variational definition, which can be tackled as a problem of regulation. The resulting problem, although a simplification, is a well-defined goal which seeks to sustain the basic infra-structure of the system community.

One arrives at the following question which a theory of system administration must address: is there an optimal, compatible set of strategies for keeping the system as close as possible to its ideal state, and still maximize productivity?

4.1. Level ℓ primitive operations and regulation

At coding level ℓ, the operations which alter system configuration are not simply Boolean algebraic operations on bits. There is a level ℓ set of operations, which effects change in accordance with this level of abstraction. Such a set of operations more closely resembles the actions of a system administrator, or of management software, than does a discussion in terms of bit operations. One therefore wishes to express actions in terms of linear combinations of primitive actions at this higher level. Let O^i be a set of such primitive actions, represented as matrices, where $i = 0, 1, 2, \ldots$. Assume further that the operations are linearly independent, so that no linear combination of the O^i is equivalent to any other linear combination. A linear combination may be written:

$$\delta C = \sum_i c_i O^i, \tag{45}$$

where c_i are constant multiples.

An implementation of primitive operations has been considered in Ref. [4] in developing the software configuration engine cfengine. This is also reminiscent of the transaction model used in Ref. [29]. Although these primitives are not complete in every aspect, they form an approximation to an almost complete set. The basic operations are:

Primitive operation O^i
Create file
Delete file
Rename file
Link file
Unlink file
Edit file
Access control
Request resource
Copy file
Process control
Process priority
Configure device

The operation "edit file" clearly conceals many sub-operations, which do not necessarily commute; these will not be elaborated upon here.

Are these example primitives above sufficient to implement a policy forbidding, say, downloading of pornographic material between the hours of 9:00 and 17:00 at a site? If such a policy is implementable, it must be possible to filter content-specific data, or deny access to data. This requires some kind of software, with a configuration (file or database) which would need to be brought into an appropriate state. The time limits can be handled by a scheduler (more configuration software), also configured in terms of state. These configuration details are thus all implementable with state editing and process control and are thus implementable in terms of a complete set of primitives.

4.2. Game theory: the contest for the ideal state

Since it is the system administrator who sets system policy, 'ideal' refers to the viewpoint of the system administrator. Whether or not this reflects the view of the user, depends on whether the user is cooperative or hostile with respect to the policy. Contests, which are caused by conflicts of interest between system policy and user wishes, unfold in this framework as environmental interactions which tend to oppose convergence and stability.

A framework for analysing conflicts of interest, in a closed system, is the theory of games [18,32]. Each move in a game affords the player a characteristic value, often referred to as the 'payoff'. In system administration, a complete game takes place on the implicit 2ℓ-dimensional board, spanned by the $\vec{d}$ vectors, and each move leads to a payoff for one or other of the players. However, the idea of building a single large game

theoretical model for the entire system is somewhat ambitious, and possibly impractical. Smaller games can also elucidate parts of the whole, in more limited studies, to good effect. An example of such a game is presented in Section 4.6.

Games come in several forms. Some are trivial, one-person games of chance, and are not analysable in terms of strategies, since the actions of the players are irrelevant to the outcome. In a sense, these are related to the stochastic noise referred to previously. More interesting, is the case in which the outcome of the game can be determined by a specific choice of strategy on the part of the players. The most basic model for such a game is that of a two-person zero-sum game, or a game in which there are two players, and where the losses of one player are the gains of the other. This model is simplistic, applied as users versus system, because it seems to say that all users must work contrary to system policy, which is clearly not true. However, experience shows that it is mainly those few users who do attempt to confound policy, who need to be dealt with strategically. Thus, the real 'battle' for the ideal state of the system, is between those factions who are for and those who are against policy. The majority of neutral users play only a background role (as chance noise) and do not need to be modelled explicitly.

Many games can be stated in terms of this basic model: it is, for example, the model taken by current system administration agents such as cfengine [4] and PIKT [33], as well as several commercial products, to good effect.

4.3. Type II models as 'stable' sets

The evolution function U, or operator $\hat{U}$ is a mapping

$$U : \{q\} \to \{q\}. \tag{46}$$

from a set of states into itself. Such a mapping has the mathematical structure of a graph [2,40]. The generalization of the previous discussion is to allow states to be set-valued, i.e. for there to be several admissable solutions to the stability criteria. Multiple solutions need not be equivalent (symmetrical) when policy is allowed to address competing strategies.

Graphs provide the simplest notion of stability for sets, and these correspond to the most basic type of game solution. The concepts of *internal stability* and *external stability* are key here. A subset of states $Q \in \{q\}$ is said to be internally stable if no two states in Q are connected by a transition in U, i.e. $UQ \cap Q \neq \emptyset$, or none of the states are linked together directly by a transition in U. A subset of states Q' is set to be externally stable if the set is only accessible by transitions from states outside the set Q', i.e. if $\bar{q} \notin Q'$, then $U\bar{q} \cap Q' \neq \emptyset$, or the image of every state in Q' lies outside of Q' itself. A set that is both internally and externally stable is said to be a *kernel* or core of the graph. The kernel is free of loops and contains all states for which $U\{q\} = \emptyset$. It is thus a good candidate for the set of possible ideal states that can be chosen as policy. The notion of the kernel was introduced by von Neumann and Morgenstern as a solution to N-person games. Existence theorems for kernels exist, but kernels are not guaranteed [2].

Another type of equilibrium is based on the notion of a fixed point of the graph of all rational strategy preferences (see below). This is not stability of the system's evolution, but stability of the choice of end point in the kernel under different policy decision criteria. This equilibrium is best known as the Nash equilibrium, or Kakutani fixed point [30] of the preference graph. It looks for a subset of states that can be regarded as a limit points of competing decision criteria. In a two-person zero sum game, this corresponds to the minimax solution that is used below. The idea of an ideal configuration Q^* for a system [5] can be defined as a fixed point of the 'response matrix' for mapping non-ideal states onto ideal ones. This matrix is readily defined in terms of convergent mappings [4,12],

$$Q' = R(Q). \tag{47}$$

However, the convergence property is not enough to select a stable base state for a convergent process, because convergence can be applied to any state. In order to prevent configuration loops and find the set of self-consistent fixed points that can be identified with policies, we must solve

$$Q^* = R(Q^*), \tag{48}$$

where Q^* is optimal. This condition is the essence of the Nash equilibrium in game theory. Graph theory and game theory are thus suitable mathematical frameworks for extending the applicability of the maintenance theorem to take into account competing policy preferences.

4.4. The payoff currency

In economics, the game currency is money; in the physical sciences it is energy. In order to apply the idea of a game to system administration, it is to introduce the idea of a currency of reward for the empirically known value-systems in which users and administrators compete. In social games, value systems are often multi-faceted, not simple linear counters. For example, in the popular game of 'civilization', players fight not only for land and money, but for intangibles such as education, services, and 'state of development'. These, in turn, feed back and allow the acquisition of even greater wealth, leading to a compounded gain. The final score is an arbitrary mixture of these qualities, according to a policy, set by the rules of play.

The challenge is to represent user desires and intentions as numerical values or functions which can be evaluated and compared at different times. Let us define payoff to one player as a sum of these different value systems:

$$\pi = \sum_a w_a \pi_a, \tag{49}$$

weighted according to a set of weights w_a, which determines the supposed relative importance of these qualities. The values of π_a are to be based on quantitative measures and empirical evidence, using the predictions found from type I models. For example: use of system resources over time, leads to an accumulation of disk and CPU 'wealth'

which changes in time according to a pattern described by a type I model. The amount of disk-space or CPU resource used by a user are a measure of material pay-off, for following a particular strategy of usage. Similarly, the good-will accrued as a result of cooperating with system policy over time could lead to less tangible payoffs, such as the reward of privileged access to the system, or an increased influence on system policy.

Two convenient categories can be retained:

$$\pi = \pi_{\mathrm{M}} + \pi_{\mathrm{S}}, \tag{50}$$

i.e. material payoff and social payoff, as described above.

4.5. Strategy and tactics

The system administrator's strategies should always bring the system closer to the ideal stable state, while still maintaining productivity.

In reality, not all users are hostile, in this fashion; most users are neutral in the conflict between those for and against system policy, and their effect on the outcome can be absorbed into the generic time-development. Nor does 'hostile' necessarily imply malice of any kind: it only means that the net effect of users' actions goes against policy, and pushes the system away from its ideal condition. In fault tree analysis, any pathway which leads to a possible fault of the system is regarded as being hostile, whether it is intentional or accidental [1]. However, it is such 'hostile' users which one must pay special attention to in a game theoretical model. Thus it is convenient in what follows to assume that all of the notable users are hostile.

The administrator can accumulate payoff currency, either by limiting or reducing the consumption of resources or by extending the resources of the system. A user can 'win', in a pessimistic sense, by moving the actual state so far from the ideal that the system crashes and thus the game ends, or by gaining total control of the resources.

As a zero sum, N-person game one could make a more detailed model, in which users compete against one another in addition to the system administrator. The system administrator's task then becomes to act as a kind of 'Robin Hood' character, preventing any one user's consumption of all resources, trying to distribute resources fairly. Again, the aim of the administrator is to maximize the duration of the game by keeping the system as close to the ideal state as possible.

In a realistic situation, both parties in the two-person game would use mixed strategies. A strategy is:

- a schedule of operations.
- a specification of moves and counter-moves (rules).

In addition to simple short-term strategies (tactics), there can be meta-strategies, or long-term goals. For instance, a nominal community strategy might be to implement the stability criteria discussed earlier:

- maintain the stability of the system.
- maximize total productivity or the generation of work.
- gain the largest feasible share of resources,

but this might be implemented in the short term by a variety of tactics, such as policy cooperation, non-cooperation and so on. An attack strategy might be to

- consume or destroy key resources.
- oppose system policy.
- denial of service.

Tactics for attaining intermediate goals might include covert strategies such as *bluffing* (falsely naming files or other deceptions), taking out an attacker, counter attacking, or evasion (concealment), exploitation, trickery, antagonization, incessant complaint (spam), revenge, etc. Security and privilege, levels of access, integrity and trust must be woven into algebraic measures for the pay-off. Faced with a problem to the system, one may address it either by patching symptoms, or by seeking to route out the fundamental cause. Most successful strategies, including those used by biological life, employ both. A means of expressing all of these devices must be formulated within a model.

4.6. Example game: disk garbage collection

It is helpful to refer to a specific example, which reproduces verifiable and intuitive results in an actual problem. This example has been verified in practice as an example of the maintenance theorem [8]. Consider a multi-user computer system, such as that run by an university or internet service provider. Such a system is used by many users, and they are a wide mixture of different personalities. Disk usage, in such an environment, is often dominated by the creation of large temporary files, such as those generated by web browsers (cached graphics, etc.). A convergent policy therefore requires these files to be cleared away regularly (called tidying), so that the system does not 'choke' on them. The need for forced garbage collection has been argued on several occasions [4,44]. Here we analyze the problem as a simple two-person, zero-sum game in which all users compete with the system administrator. A fully convergent policy, in this context, is not normally possible, since useful work tends to consume disk space in a legitimate fashion. However, this legitimate increase is slow compared to actual increase of temporarily needed resources. According to the central theorem, there is therefore an ideal state in which the system's average disk-space is increasing only slowly.

We form a type II model, with characteristic matrix, or pay-off matrices π_A and π_D, where

$$\pi_A + \pi_D = 0, \tag{51}$$

for attack by users and defence by the system administrator, as a function of strategies σ_A, σ_D. One hopes to find an optimal mixture of convergent strategies, Σ, (a linear combination of pure strategies)

$$\Sigma = \frac{1}{N} \sum_i^N c_i \sigma_i, \tag{52}$$

such that, if the dominant mixture of defensive strategies Σ_D^*, leads to a better pay-off than any other combination,

$$\pi_D(\Sigma_D^*, \Sigma_A) > \pi_D(\Sigma_D, \Sigma_A), \tag{53}$$

it can be considered optimal; similarly from the viewpoint of the attacker.

The first issue is to determine the currency of this game. What payment will be transferred from one player to the other in play? Here, there are two relevant measurements to take into account: (i) the amount of resources consumed by the attacker (or freed by the defender), and sociological rewards: (ii) 'goodwill' or 'privilege' which are conferred to users as a result of sticking to the policy rules, or, conversely, the 'bad-will' incurred on the administrator by introducing oppressive policies.

A satisfaction measure for the players is used in order to model the goodwill aspect. A system administrator could clearly prevent users from creating any new files. This is clearly not a defensible use of the system, since it contravenes the assumption that a larger goal of strategy is to maximize the usage of the system, thus the system administrator's defence strategy should be penalized for restricting users too much. It is convenient to construct the payoff matrix for attackers (users), since the consumption of files follows known patterns. The characteristic matrix, for users (attackers), now has two contributions:

$$\pi_A = \pi_r(\text{resources}) + \pi_s(\text{satisfaction}). \tag{54}$$

It is convenient to define

$$\pi_r \equiv \pi(\text{resources}) = \frac{1}{2}\left(\frac{\text{resources won}}{\text{total resources}}\right). \tag{55}$$

Satisfaction π_s is assigned arbitrarily from values from plus to minus one half, such that,

$$\begin{aligned} -\tfrac{1}{2} &\leqslant \pi_r \leqslant +\tfrac{1}{2} \\ -\tfrac{1}{2} &\leqslant \pi_s \leqslant +\tfrac{1}{2} \\ -1 &\leqslant \pi_A \leqslant +1. \end{aligned} \tag{56}$$

The different convergent strategies can now be regarded as duels, or games of timing in which files are created and removed at competing rates. The payoff-matrix for 'attacking' users has the following form:

Users/system	Ask to tidy	Tidy by date	Tidy above Threshold	Quotas
Tidy when asked	$\pi_A(1,1)$	$\pi_A(1,2)$	$\pi_A(1,3)$	$\pi_A(1,4)$
Never tidy	$\pi_A(2,1)$	$\pi_A(2,2)$	$\pi_A(2,3)$	$\pi_A(2,4)$
Conceal files	$\pi_A(3,1)$	$\pi_A(3,2)$	$\pi_A(3,3)$	$\pi_A(3,4)$
Change time stamps	$\pi_A(4,1)$	$\pi_A(4,2)$	$\pi_A(4,3)$	$\pi_A(4,4)$

The elements of the characteristic matrix must now be modelled by suitable algebraic or constant terms. The rate at which users produce files may be written as

$$r_{\mathrm{u}} = \frac{n_{\mathrm{b}} r_{\mathrm{b}} + n_{\mathrm{g}} r_{\mathrm{g}}}{n_{\mathrm{b}} + n_{\mathrm{g}}}, \tag{57}$$

where r_{b} is the rate for bad users and r_{g} is the rate for good users. The total number of users $n_u = n_{\mathrm{b}} + n_{\mathrm{g}}$. From the authors experience, the ratio $n_{\mathrm{b}}/n_{\mathrm{g}}$ is about one percent. The rate can be expressed as a scaled number between zero and one, for convenience, so that $r_{\mathrm{b}} = 1 - r_{\mathrm{g}}$.

Empirical evidence suggests that, on average, users consume resources at a rate which is periodic and polynomial in time [11]:

$$W(t) \propto \sin(\Omega t) \sum_n c_n t^n. \tag{58}$$

With reference to Fig. 3, one has, daily tidying $T_{\mathrm{p}} = 24$. User numbers are set in the ratio $(n_{\mathrm{g}}, n_{\mathrm{b}}) = (99, 1)$, based on rough empirical ratios, i.e. one percent of users are considered mischievous. The filling rates are in the same ratio: $r_{\mathrm{b}}/R_{\mathrm{tot}} = 0.99, r_{\mathrm{g}}/R_{\mathrm{tot}} = 0.01$, $r_a/R_{\mathrm{tot}} = 0.1$, where $R_{\mathrm{tot}} = r_{\mathrm{b}} + r_{\mathrm{g}}$. The flat dot–slashed line is $|\pi_q|$, the quota pay-off. The lower wavy line is the cumulative pay-off resulting from good users, while the upper line represents the pay-off from bad users. The upper line doubles as the magnitude of the pay-off $|\pi_a| \geqslant |\pi_{\mathrm{u}}|$, if we apply the restriction that an automatic system can never win back more than users have already taken. Without this restriction, $|\pi_a|$ would be steeper.

Simplifying this, the payoff in terms of the consumption of resources by users, to the users themselves, may be written:

$$\pi_{\mathrm{u}} = \frac{1}{2} \int_0^T \mathrm{d}t \, \frac{r_{\mathrm{u}} \, (\sin(2\pi t/24) + 1)}{R_{\mathrm{tot}}}, \tag{59}$$

where the factor of 24 is the human daily rhythm, measured in hours, and R_{tot} is the total amount of resources to be consumed. Note that, by considering only good user or bad users, one has a corresponding expression for π_{g} and π_{b}, with r_{u} replaced by r_{g} or r_{b} respectively. An automatic garbage collection system results in a negative pay-off to users, i.e. a pay-off to the system administrator. This may be written as

$$\pi_a = -\frac{1}{2} \int_0^T \mathrm{d}t \, \frac{r_a(\sin(2\pi t/T_{\mathrm{p}}) + 1)}{R_{\mathrm{tot}}}, \tag{60}$$

where T_{p} is the period of execution for the automatic system, considered earlier. This is typically hourly or more often, so the frequency of the automatic cycle is some twenty times greater than that of the human cycle. The rate of resource-freeing r_a is also greater than r_{u}, since file deletion takes little time compared to file creation, and also an automated system will be faster than a human. The quota payoff yields a fixed allocation of resources, which are assumed to be distributed equally amongst

users and thus each quota slice assumed to be unavailable to other users. The users are nonchalant, so $\pi_s = 0$ here, but the quota yields

$$\pi_q = +\frac{1}{2}\left(\frac{1}{n_b + n_g}\right). \tag{61}$$

The matrix elements are expressed in terms of these.

$\pi(1,1)$: Here $\pi_s = -\frac{1}{2}$ since the system administrator is maximally satisfied by the users' behaviour. π_r is the rate of file creation by good users π_g, i.e. only legal files are produced. Comparing the strategies, it is clear that $\pi(1,1) = \pi(1,2) = \pi(1,3)$.

$\pi(1,4)$: Here $\pi_s = 0$ since the users are dissatisfied by the quotas, but the system administrator must be penalized for restricting the functionality of the system. With fixed quotas, users cannot generate large temporary files. π_q is the fixed quota payoff, a fair slice of the resources. Clearly $\pi(4,1) = \pi(4,2) = \pi(4,3) = \pi(4,4)$. This tells us that quotas put a straight-jacket on the system. The game has a fixed value if this strategy is adopted by system administrators. However, it does not mean that this is the best strategy, according to the rules of the game, since the system administrator loses points for restrictive practices. This is yet to be determined.

$\pi(2,1)$: Here $\pi_s = \frac{1}{2}$ since the system administrator is maximally dissatisfied with users' refusal to tidy their files. The pay-off for users is also maximal in taking control of resources, since the system administrator does nothing to prevent this, thus $\pi_r = \pi_u$. Examining the strategies, one find that $\pi(2,1) = \pi(3,1) = \pi(3,2) = \pi(3,3) = \pi(4,1) = \pi(4,2)$.

$\pi(2,2)$: Here $\pi_s = \frac{1}{2}$ since the system administrator is maximally dissatisfied with users' refusal to tidy their files. The pay-off for users is now mitigated by the action of the automatic system which works in competition, thus $\pi_r = \pi_u - \pi_a$. The automatic system is invalidated by user bluffing (file concealment).

$\pi(2,3)$: Here $\pi_s = \frac{1}{2}$ since the system administrator is maximally dissatisfied with users' refusal to tidy their files. The pay-off for users is mitigated by the automatic system, but this does not activate until some threshold time is reached, i.e. until $t > t_0$. Since changing the date cannot conceal files from the automatic system, when they are tidied above threshold, we have $\pi(2,3) = \pi(4,3)$.

Thus, in summary, the characteristic matrix is given by

$$\pi_A = \begin{pmatrix} (-\frac{1}{2} + \pi_g(t)) & (-\frac{1}{2} + \pi_g(t)) & (-\frac{1}{2} + \pi_g(t)) & \pi_q \\ (\frac{1}{2} + \pi_u(t)) & (\frac{1}{2} + \pi_u(t) + \pi_a(t)) & (\frac{1}{2} + \pi_u(t) + \pi_a(t)\theta(t_0 - t)) & \pi_q \\ (\frac{1}{2} + \pi_u(t)) & (\frac{1}{2} + \pi_u(t)) & (\frac{1}{2} + \pi_u(t)) & \pi_q \\ (\frac{1}{2} + \pi_u(t)) & (\frac{1}{2} + \pi_u(t)) & (\frac{1}{2} + \pi_u(t) + \pi_a(t)\theta(t_0 - t)) & \pi_q \end{pmatrix}, \tag{62}$$

where the step function is defined by

$$\theta(t_0 - t) = \begin{cases} 1 & (t \geqslant t_0), \\ 0 & (t < t_0), \end{cases} \tag{63}$$

and represents the time-delay in starting the automatic tidying system in the case of tidy-above-threshold.

It is possible to make several remarks about the relative sizes of these contributions. The automatic system works at least as fast as any human so, by design, in this simple model we have

$$\tfrac{1}{2} \geqslant |\pi_a| \geqslant |\pi_{\mathrm{u}}| \geqslant |\pi_{\mathrm{g}}| \geqslant 0, \tag{64}$$

for all times. In addition, for short times $\pi_q > \pi_{\mathrm{u}}$, but users can quickly fill their quota and overtake this. In a zero-sum game, the automatic system can never tidy garbage faster than users can create it, so the first inequality is always saturated. From the nature of the cumulative pay-offs, we can also say that

$$(\tfrac{1}{2} + \pi_{\mathrm{u}}) \geqslant (\tfrac{1}{2} + \pi_{\mathrm{u}} + \pi_a \theta(t_0 - t)) \geqslant (\tfrac{1}{2} + \pi_{\mathrm{u}} + \pi_a), \tag{65}$$

and

$$|\tfrac{1}{2} + \pi_{\mathrm{u}}| \geqslant |\pi_{\mathrm{g}} - \tfrac{1}{2}|. \tag{66}$$

One can now apply these results to a modest strategy of automatic tidying, of garbage, once per day, in order to illustrate the utility of the game formulation. The first step is to compute the pay-off rate contributions. Referring to Fig. 4, one sees that the automatic system can always match users' moves. As drawn, the daily ripples of the automatic system are in phase with the users' activity. This is not realistic, since tidying would normally be done at night when user activity is low, however such details need not concern us in this illustrative example.

The policy we have created in setting up the rules of play for the game, penalizes the system administrator for employing strict quota shares. Even so, users do not gain much from this, because quotas are constant for all time. A quota is a severe handicap to users in the game, except for very short times before users reach their quota limits. Quotas could be considered cheating in such a game, since they determine the outcome even before play commences. There is no longer a contest. Moreover, comparing the values in the figure, it is possible to see how resource inefficient quotas are. Users cannot create temporary files which exceed these hard and fast quotas. An immunity type model which allows fluctuations is a considerably more resource efficient strategy, since it allows users to span all the available resources for short periods of time, without consuming them for ever.

Any two-person zero-sum game has a solution, either in terms of a pair of optimal *pure* strategies or as a pair of optimal *mixed* strategies [18,32]. The solution is found as the balance between one player's attempt to maximize his pay-off and the other

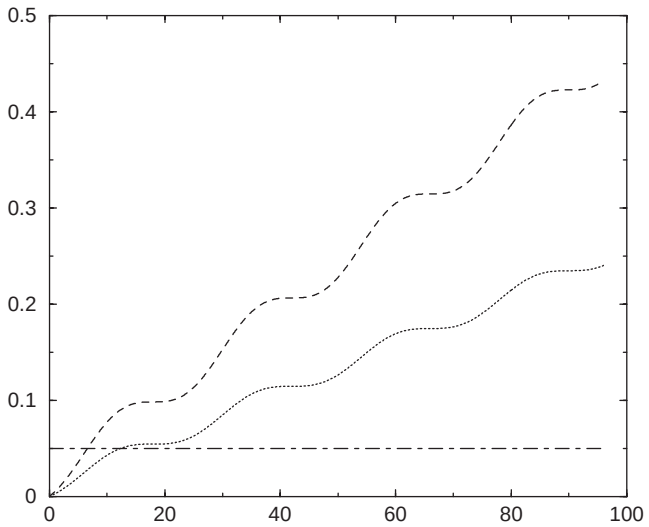

Fig. 4. The absolute values of pay-off contributions as a function of time (in hours), based on a type I model.

player's attempting to minimize the opponent's result. In general one can say of the attacker's pay-off matrix that

$$\max_{\downarrow}\min_{\rightarrow}\pi_{\mathrm{rc}} \leqslant \min_{\rightarrow}\max_{\downarrow}\pi_{\mathrm{rc}}, \tag{67}$$

where the arrows refer to the directions of increasing rows ($\downarrow$) and columns ($\rightarrow$). The left hand side is the least users can hope to win (or conversely the most that the system administrator can hope to keep) and the right is the most users can hope to win (or conversely the least the system administrator can hope to keep). If we have

$$\max_{\downarrow}\max_{\rightarrow}\pi_{\mathrm{rc}} = \min_{\rightarrow}\max_{\downarrow}\pi_{\mathrm{rc}}, \tag{68}$$

it implies the existence of a pair of single, pure strategies in row-column (r^*, c^*) which are optimal for both players, regardless of what the other does. If the equality is not satisfied, then the minimax theorem tells us that there exist optimal mixtures of strategies, where each player selects at random from a number of pure strategies with a certain probability weight.

The situation for our time-dependent example matrix is different for small t and for large t. The distinction depends on whether users have had time to exceed fixed quotas or not; thus 'small t' refers to times when users are not impeded by the imposition of quotas.

For small t, we have

$$\max_{\downarrow}\min_{\rightarrow}\pi_{\mathrm{rc}} = \max_{\downarrow}\begin{pmatrix} \pi_{\mathrm{g}} - \frac{1}{2} \\ \frac{1}{2} + \pi_{\mathrm{u}} + \pi_a \\ \frac{1}{2} + \pi_{\mathrm{u}} \\ \frac{1}{2} + \pi_{\mathrm{u}} + \pi_a\theta(t_0 - t) \end{pmatrix}$$

$$= \tfrac{1}{2} + \pi_{\mathrm{u}}. \tag{69}$$

The ordering of sizes in the above minimum vector is

$$\tfrac{1}{2} + \pi_{\mathrm{u}} \geqslant \tfrac{1}{2} + \pi_{\mathrm{u}} + \pi_a\theta(t_0 - t) \geqslant \pi_{\mathrm{u}} + \pi_a\theta(t_0 - t) \geqslant \pi_{\mathrm{g}} - \tfrac{1}{2}. \tag{70}$$

This is useful to know, if we should examine what happens when certain strategies are eliminated. For the opponent's endeavours we have

$$\min_{\rightarrow}\max_{\downarrow}\pi_{\mathrm{rc}} = \min_{\rightarrow}(\tfrac{1}{2} + \pi_{\mathrm{u}}, \tfrac{1}{2} + \pi_{\mathrm{u}}, \tfrac{1}{2} + \pi_{\mathrm{u}}, \pi_q)$$

$$= \tfrac{1}{2} + \pi_{\mathrm{u}}. \tag{71}$$

This indicates that the equality in Eq. (68) is satisfied and there exists at least one pair of pure strategies which is optimal for both players. In this case, the pair is for users to conceal files, and for the system administrator to tidy by any means (these all contribute the same weight in Eq. (71). Thus for small times, the users are always winning the game if we assume that they are allowed to bluff by concealment. If the possibility of concealment or bluffing is removed (perhaps through an improved technology used by the administrator), then the next best strategy is for users to bluff by changing the date. In that case, the best system administrator strategy is to tidy at threshold.

These results make qualitative sense and tally well with experience. The result also makes a prediction for system administration tools like cfengine [4]. System administration tools must be able to see through attempts at bluffing if they are to be an effective opponent against the worst users.

For large times (when system resources are becoming or have become scarce), then the situation looks different. In this case one finds that

$$\max_{\downarrow}\min_{\rightarrow}\pi_{\mathrm{rc}} = \min_{\rightarrow}\max_{\downarrow}\pi_{\mathrm{rc}} = \pi_{\mathrm{q}}. \tag{72}$$

In other words, the quota solution determines the outcome of the game for any user strategy. As already commented, this might be considered cheating or poor use of resources, at the very least. If one eliminates quotas from the game, then the results for small times hold also at large times.

Two things emerge from the limited analysis here. The first is that purely dumb automatic systems are inadequate to perform every task in system administration today. There can be no zero-maintenance system. Intelligent incursions are required to counter

complex problems, because the environment of users generates intelligent strategies for opposing system policy.

The second, more specific prediction is that the use of quotas is an inefficient way of counteracting the effects of selfish users. A quota strategy cannot approach the same level of productivity as one which is based on competitive counterforce. The optimal strategies for garbage collection are rather found to lie in the realm of the immunity model [11]. However, it is a sobering thought that a persistent user, who is able to bluff a regulatory system into disregarding it (a situation analogous to cancer of the body), will always win against the resource battle. The need for new technologies which can see through bluffs will be an ever present reality in the future. With the ability of encryption and compression systems to obscure file contents, this is a contest which will not be easily won by system administrators.

4.7. Other games

Many more problems can be analysed by game theoretical models. Formulating a problem in game theoretical language can help to relate the chain of cause and effect to relative likelihood of success. For example:

- optimal routing configurations based on game theoretical payoff models have been discussed in Ref. [42].
- service level agreements in competitive environments can be determined by bargaining equilibria [37].
- the configuration agent cfengine [10] uses the idea of gaming strategies to choose actions and schedules for regulatory checks in host-based system administration, in both pure and mixed strategy patterns. This technology was implemented as a direct result of the present paper, and gaming ideas can help to identify and evaluate strategy and policy coding.
- evaluating the implications of new software at a site: e.g. the Java language tools have extremely high memory consumption, but simplify many tasks. Thus social needs place a burden on system stability.
- what are the implications of introducing a firewall, or other inconvenient security measure. Will users obey the security procedures, or find a way around them? What incentives can be provided to obey policy and preserve system integrity?

There are many possibilities. Often, one sees organizations opting for short-term benefits, by introducing technologies to protect themselves from only well-known vulnerabilities. The difficulty with this strategy is that is can lead to an arms-race of tit-for-tat escalation, where users are involved. A better long-term strategy might be to deflect interest from the assets of the system, by specific choice of policy.

As game theoretical methods are, in the computational sense, variational methods, they are well suited for automation schemes, such as those handled by systems of decision-making software agents. The possible pathways of attack and defence can often be analysed using the methods of fault tree analysis [1]. Other games will be considered in future work.

5. Summary

This paper presents a framework for analyzing problems of system administration, at the level of policy. System administration is identified with a search for stable equilibria of the system, through judicious use of policy. The paper defines policy, its extent and limitations, by asking the question: how long can systems predictably be maintained? The schema is:

(1) define policy as a scheme of partially reliable rules and constraints for the evolution of the system, coupled to an unpredictable environment.
(2) identify policies which have desirable average stability properties, i.e. those which counter the random component of the environment.
(3) model the behaviour of the system in general terms, to characterize its response to different stimuli from the environment. This indicates how the system will respond to strategically planned change. (Type I model)
(4) use the knowledge, learned from type I studies, to construct game theoretical models that identify and evaluate strategies for optimizing maintenance. This process must take account of the 'social values' of the players. (Type II model)

The approach taken in this paper can be viewed as a more rigorous definition of the meaning of computer immunology [5,20,31] (i.e. the identification of the ideal state as the 'healthy' state), or as a definition of information physics (i.e. the dynamics of the human-computer system under a given set of boundary conditions). Software systems, using the idea of an ideal state are already in use [4,37].

A mathematical formalism is only a tools for relating assumptions to conclusions, in an impartial way. With a mathematical approach, it becomes easier to see through the personal opinions and vested interests. However, one can only distinguish between those possibilities which are taken into account. That means that every relevant strategy, or alternative, has to be considered, or else one could miss the crucial combination which wins the game. This requires intimate and expert knowledge of system.

Acknowledgements

I am grateful to Trond Reitan for a useful discussion about evolutionary stable strategies and to Lars Kristiansen for a critical readings of the manuscript. Special thanks to the members of USENIX and SAGE who understood my original words back in 2000. I would also like to thank Joe Halpern for patience and encouragement as I clumsily tried to present these ideas to an audience of unwilling computer scientists.

Appendix A. Processes

This appendix summarizes some of the background on which this paper is founded, concerning dynamical systems for discrete and continuous processes. For an excellent reference on this topic, see [23].

A *process* is a chain of events X_n $(n=0,1,\ldots,N)$, where each event X_n takes a value q_i $(i=1,\ldots d)$ in a state-space Q. d is called the dimension of the space. The integers n normally label the development of the process in discrete steps, interpreted as time intervals. In this paper, we describe only the time development of processes, though the method is more general than this.

The *transition matrix* T_{ij}, defined by

$$T : Q \otimes Q \to [0,1], \tag{A.1}$$

describes the possible transitions between states. It is written in a variety of notations in the literature, including the following:

$$\begin{aligned} T_{ji} &= p_{ji} = |\langle q_j | q_i \rangle|^2 \\ &= P(X_{n+1} = q_j | X_n = q_i). \end{aligned} \tag{A.2}$$

It represents the probability that the next event X_{n+1} will be in a state q_j, given that it is currently in the state q_i. By discussing the *probability* for transitions, we leave open the issue of whether such transitions are deterministic or stochastic. There is a number of possibilities. If $T_{ij}=1$ at X_n, for some i,j, the process is *deterministic* and one may write the development of the chain as a rule

$$X_{n+1} = U_t(X_n,\ldots,X_0), \tag{A.3}$$

in the general case. If $T_{ij} < 1$ at X_n, for all i,j,n, the process is *stochastic*.

If T depends only on the current state of the system,

$$P(X_{n+1} = q_i | X_0, X_1, \ldots, X_n) = P(X_{n+1} = q_i | X_n), \quad \forall n \geqslant 1, \tag{A.4}$$

then the chain is said to be a *Markov chain*, or *memoryless*. Markov processes are also called steady state, or equilibrium processes. If T depends on the whole history of $\{X\}$, then it is called a non-equilibrium, non-Markov process.

A state is called *persistent* if

$$P(X_{n+m} = q_i | X_m = q_i) = 1, \quad \text{for some } n \geqslant 1 \tag{A.5}$$

and *transient* if

$$P(X_{n+m} = q_i | X_m = q_i) < 1. \tag{A.6}$$

The terms periodic, aperiodic and ergodic also describe chains in which the processes return to the same state. Readers are referred to [23] for more about this.

A set of states Q is set to be *closed* if

$$T_{ij} = 0, \quad \forall q_i \in Q,\ q_j \notin Q. \tag{A.7}$$

Two or more parallel chains X_n and Y_n *interact* if their transition matrices are not closed with respect to their state spaces:

$$\begin{aligned} P(X_{n+1} = q_x) &= P(X_{n+1} = q_x | X_n, Y_n, \ldots), \\ P(Y_{n+1} = q_y) &= P(Y_{n+1} = q_y | X_n, Y_n, \ldots), \end{aligned} \tag{A.8}$$

i.e. the state q_x is selected in the chain X as a result of the state in both chains, and similarly for Y.

In some cases, the transition matrix is factorizable into two independent processes Y and Z:

$$P(X_n = q_i + \delta q_i) = P(Y_n = q_i)P(Z_n = \delta q_i). \tag{A.9}$$

The two processes thus take place independently, but are superposed. In this paper, we are interested in processes which can be separated, approximately, into a non-Markov process with states q_i and a Markov process δq_i. This is called the adiabatic approximation in statistical mechanics and it applies when it is possible to separate two scales of variation: a slowly varying change and a rapidly very change. This is an important feature of this paper, because it means that a process has an approximately steady behaviour, modulated by a trend.

The transition to continuous processes is straightforward. A discrete chain

$$X_0 = q_i, X_1 = q_j, \ldots, X_n = q_k, \tag{A.10}$$

is replaced by a function q of a continuous parameter t, so that a time interval from an initial time t_i to a final time t_f maps into the state space Q:

$$q(t) : [t_\mathrm{i}, t_\mathrm{f}] \to Q. \tag{A.11}$$

The discrete event notation X_n is now redundant. A set of parallel chains, labelled by a parameter x, and development parameter t is thus written $q(x,t)$.

The transition matrix is now a function of two times:

$$T(t,t') = |\langle q(t')|q(t)\rangle|^2 = T(\tilde{t},\bar{t}), \tag{A.12}$$

where

$$\tilde{t} = t - t',$$

$$\bar{t} = \tfrac{1}{2}(t + t'). \tag{A.13}$$

If there is no dependence on the absolute time $\bar{t}$, the process is said to be *homogeneous* or *translationally invariant*, otherwise it is *inhomogeneous*. The deterministic time development operator U_t has the property:

$$q(t') = \int U_t(t,t')q(t)\,\mathrm{d}t', \tag{A.14}$$

which may be written in a condensed notation as

$$q(t') = \hat{U}_t(t,t')q(t). \tag{A.15}$$

The derivative of a configuration $q(x,t)$, discussed in the text, is defined in the normal way, in terms of the smallest granular steps that can be represented:

$$\frac{\mathrm{d}q(x,t)}{\mathrm{d}t} = \frac{q(x,t+\delta t) - q(x,t)}{\Delta t}. \tag{A.16}$$

In calculus one then assumes that the limit $\Delta t \to 0$ can be taken. Similarly, in the continuum approximation, one assumes that—over time scales relevant to the problem—the error incurred by the finite size of Δt is too small to be of interest. In the text, one is particularly interested in the value of the so-called conformal rate of change

$$\frac{\mathrm{d}}{\mathrm{d}t} \log \frac{q(x,t)}{\alpha} = \frac{1}{q} \frac{\mathrm{d}q}{\mathrm{d}t}. \tag{A.17}$$

This is independent of the scale α, and represents the rate of change of information (e.g. bits per second, for base 2 logarithm) in the variable at x. The ordering of the information is not known or required in order to make this characterization. Hence the existence of a symmetry group $\alpha \in \mathscr{G} \otimes \Gamma$, seeks further to emphasize the unimportance of the ordering for this rate of change. The characterizations of policy rates (e.g., Eq. (44)) are therefore presented in this scale invariant manner.

References

[1] R. Apthorpe, A probabilistic approach to estimating computer system reliability, Proc. 15th Systems Administration Conf. (LISA XV), USENIX Association, Berkeley, CA, 2001, p. 31.

[2] C. Berge, The Theory of Graphs, Dover, New York, 2001.

[3] J.P Bouchard, M. Potters, The Theory of Finanical Risks, Cambridge University Press, Cambridge, 2000.

[4] M. Burgess, A site configuration engine, Computing systems, Vol. 8, MIT Press, Cambridge, MA, 1995, p. 309.

[5] M. Burgess, Computer immunology, Proc. 12th Systems Administration Conf. (LISA XII), USENIX Association, Berkeley, CA, 1998, p. 283.

[6] M. Burgess, Thermal, non-equilibrium phase space for networked computers, Phys. Rev. E 62 (2000) 1738.

[7] M. Burgess, The kinematics of distributed computer transactions, Internat. J. Mod. Phys. C 12 (2000) 759–789.

[8] M. Burgess, Theoretical system administration, Proc. 14th Systems Administration Conf. (LISA XIV), USENIX Association, Berkeley, CA, 2000, p. 1.

[9] M. Burgess, Cfengine's immunity model of evolving configuration management, Sci. Comput. Programming, 2002, submitted for publication.

[10] M. Burgess, Cfengine www site. http://www.iu.hio.no/cfengine.

[11] M. Burgess, H. Haugerud, T. Reitan, S. Straumsnes, Measuring host normality, ACM Trans. Comput. Systems 20 (2001) 125–160.

[12] M. Burgess, R. Ralston, Distributed resource administration using cfengine, Software Practice and Experience 27 (1997) 1083.

[13] S.C. Cheung, J. Kramer, Checking subsystem safety properties in compositional reachability analysis, 18th Internat. Conf. Software Eng. (ICSE'18), Berlin, Germany, 1996.

[14] R. Cohen, K. Erez, D.B. Avraham, S. Havlin, Resilience of the internet to random breakdowns, Phys. Rev. Lett. 85 (2000) 4626.

[15] A. Couch, N. Daniels, The maelstrom: network service debugging via "ineffective procedures", Proc. 15th Systems Administration Conf. (LISA XV), USENIX Association, Berkeley, CA, 2001, p. 63.

[16] A. Couch, M. Gilfix, It's elementary, Dear Watson: applying logic programming to convergent system management processes, Proc. 13th Systems Administration Conf. (LISA XIII), USENIX Association, Berkeley, CA, 1999, p. 123.

[17] Y. Diao, J.L. Hellerstein, S. Parekh, Optimizing quality of service using fuzzy control, IFIP/IEEE 13th Internat. Workshop on Distributed Systems: Oper. Management (DSOM 2002), 2002, p. 42.

[18] M. Dresher, The Mathematics of Games of Strategy, Dover, New York, 1961.

[19] R. Evard, An analysis of unix system configuration, Proc. 11th Systems Administration Conf. (LISA XI), USENIX Association, Berkeley, CA, 1997, p. 179.
[20] S. Forrest, S. Hofmeyr, A. Somayaji, Commun. ACM 40 (1997) 88.
[21] N. Glance, T. Hogg, B.A. Huberman, Computational ecosystems in a changing environment, Internat. J. Mod. Phys. C 2 (1991) 735.
[22] K.M. Goudarzi, J. Kramer, Maintaining node consistency in the face of dynamic change, Proc. of the Third Internat. Conf. Configurable Distributed Systems (CDS '96), IEEE Computer Society Press, Annapolis, MD, USA, 1996, p. 62.
[23] G.R. Grimmett, D.R. Stirzaker, Probability and Random Processes, Oxford Scientific Publications, Oxford, 1982.
[24] J.L. Hellerstein, F. Zhang, P. Shahabuddin, An approach to predictive detection for service management, Proc. IFIP/IEEE IM VI, 1999, p. 309.
[25] C. Hogan, Metrics for management, Proc. Ninth Systems Administration Conf. (LISA IX), USENIX Association, Berkeley, CA, 1995, p. 125.
[26] T. Hogg, B.A. Huberman, Artificial intelligence and large scale computation: a physics perspective, Phys. Rep. 156 (1987) 227.
[27] P. Hoogenboom, J. Lepreau, Computer system performance problem detection using time series models, Proc. USENIX Tech. Conf., USENIX Association, Berkeley, CA, 1993, p. 15.
[28] IFIP/IEEE, Im Conf. Series Proc. (1989–2001), 1989.
[29] J. Kramer, J. Magee, The evolving philosophers problem: dynamic change management, IEEE Trans. Software Eng. 16 (1990) 1293.
[30] R.B. Myerson, Game Theory: Analysis of Conflict, Harvard University Press, Cambridge, MA, 1991.
[31] J.V. Neumann, Probabiltistic logics and the synthesis of reliable organisms from unreliable components, Reprinted in vol 5 of his Collected Works, 1952.
[32] J.V. Neumann, O. Morgenstern, Theory of Games and Economic Behaviour, Princeton University Press, Princeton, 1944.
[33] R. Osterlund, Pikt: Problem informant/killer tool. Proc. 14th Systems Administration Conf. (LISA XIV), USENIX Association, Berkeley, CA, 2000, p. 147.
[34] M.I. Seltzer, C. Small, Self-modifying and self-adapting operating systems, Workshop on Hot Topics in Operating Systems 1997, Proc. Sixth Workshop on Hot Topics in Operating Systems, May 5–6, 1997, Cape Cod, MA, IEEE Computer Society Press, 1997, pp. 124–129.
[35] C.E. Shannon, W. Weaver, The Mathematical Theory of Communication, University of Illinois Press, Urbana, 1949.
[36] T.B. Sheridan, Allocating functions among humans and machines, in: D. Beevis, P. Essens, H. Schuffel (Eds.), Improving Function Allocation for Integrated Systems Design, Wright–Patterson Airforce Base, CSERIAC State-of-the-Art Report, 1996, pp. 179–198.
[37] A. Somayaji, S. Forrest, Automated response using system-call delays, Proc. Ninth USENIX Security Symp., 2000, p. 185.
[38] S. Traugott, Why order matters: turing equivalence in automated systems administration, Proc. 16th Systems Administration Conf. (LISA XVI), USENIX Association, Berkeley, CA, 2002, p. 99.
[39] S. Traugott, J. Huddleston, Bootstrapping an infrastructure, Proc. 12th Systems Administration Conf. (LISA XII), USENIX Association, Berkeley, CA, 1998, p. 181.
[40] D.B. West, Introduction to Graph Theory, 2nd Edition, Prentice-Hall, Upper Saddle River, 2001.
[41] K.G. Wilson, Phys. Rev. B 4 (1971) 1144.
[42] H. Yaïche, R.R. Mazumdar, A game theoretic framework for bandwidth allocation and pricing in broadband networks, IEEE/ACM Trans. Networking 8 (2000) 1063.
[43] L.A. Zadeh, Outline of a new approach to the analysis of complex systems and decision process, IEEE Trans. Systems Man Cybern. 3 (1973) 28–44.
[44] E.D. Zwicky, Disk space management without quotas, Proc. Workshop on Large Installation Systems Administration III, USENIX Association, Berkeley, CA, 1989, 1989, p. 41.