Management Without (Detailed) Models

Alva L. Couch[1], Mark Burgess[2], and Marc Chiarini[1]

[1]Computer Science Department
Tufts University
Medford, MA, USA
couch@cs.tufts.edu,marc.chiarini@tufts.edu

[2]Faculty of Engineering
Oslo University College
Oslo, Norway
mark@iu.hio.no

Abstract. We present a resource management algorithm based upon guided "walks" within a system state space. Walks are guided via simple predictions of optimum behavior whose accuracy increases as system state approaches a predicted optimum. Optimum behavior is defined as maximizing payoff, which is the difference between value of provided service and cost of providing the service. Feedback between prediction, movement in the state space, and direct observation of behavior allows the algorithm to track optimum payoff, even though there is no detailed model of system behavior. Efficiency of the algorithm is defined as the ratio between observed and optimum payoffs, and can be estimated without reference to a detailed model. We demonstrate by simulation that, under commonly encountered conditions, our algorithm can achieve near-optimal behavior. Our strategy is thus a potentially viable alternative to management based upon closed control loops in many practical situations.

Keywords: autonomic computing, convergent operators, computer immunology, emergent properties, Cfengine

1 Introduction

Most management paradigms are based upon the possession of a detailed model of assumed behavior and the ability to micro-manage configuration change[1–3]. At Hot Autonomic Computing 2008 (HotAC2008), three grand-challenge problems were identified for autonomic computing[4]:

1. Developing more accurate models of system behavior.
2. Ability to compose autonomic control systems and predict behavior of composed systems.
3. Increasing user trust of autonomic systems.

The consensus of the group was that with better models, autonomic control loops become more predictable; with the ability to compose loops, multi-vendor solutions become possible; and that these challenges are linked with the fundamental problem of trust and expectation: the user needs to know what to expect and what to trust.

In this paper we dispute the assumed correlation in prior work between knowledge and outcome, and show that one can (literally) profit by embracing emergent results of highly adaptive management processes.

We approach the challenges above from a different point of view. Before autonomics was proposed, Burgess described a notion of computer immunology[5], based upon the idea that systems can be managed by guiding system states towards desired behavior using what might be considered a thermodynamic (or economic) approach. Autonomic computing can be "approximated" by application of immunological actions[6]; this is the theoretical framework for the management tool Cfengine[7, 8]. In an immunological approach, a model of behavior is only indirectly linked with the function of the autonomic system, which can be described as an *emergent property* of an underlying set of processes. In spite of much interest in emergent properties[9, 10], few engineers truly believe that emergence can deliver reliable results; this is a part of the trust challenge of autonomic computing. Our experience is that emergent properties can be guaranteed, even within relatively tight tolerances.

We began this work by asking about the kind of autonomic control that would be appropriate to implement in Cfengine. In turn, we asked ourselves how much of a model of system behavior needs to be understood by such an autonomic control system. The answer to this question has turned out to be surprising and counter-intuitive. Rather than "learning" the model of system behavior, one can base a management strategy on remaining flexible and highly reactive to observed changes. This approach is simpler, more adaptive, and more composable than approaches based upon learning models.

2 Cost, Value, and Payoff

Inspired by current work in Business-Driven IT Management (BDIM)[11, 12], we took a very basic, high-level view of autonomic control, based upon the objective of balancing concepts of business *cost* and *value*. *Cost* refers to the cost of operating and managing a system, while *value* refers to the value of operating the system. The *payoff* function for operating a system is its value minus its cost, which varies over time. Given some estimate of best payoff, the *efficiency* of a management system is the quotient of the achieved payoff, divided by the best achievable payoff.

Our model of autonomic control is shown in Figure 1. A system consumes resources (R) and exhibits performance (P) subject to outside forces L (load). A *parameter closure* Q[13–16] controls resource levels based upon interaction with multiple external agents (A) that relay value (V) information to Q. Q knows about the cost $C(R)$ of resources, while the agents know about and describe value

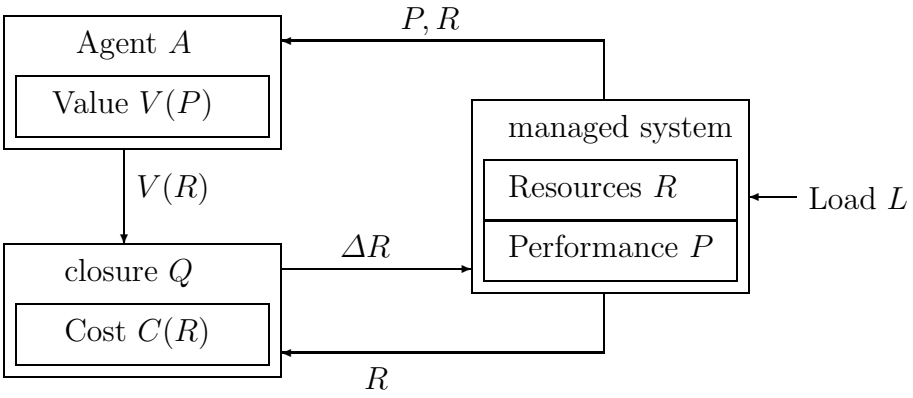

Fig. 1. Our model of autonomic resource control. Q is a resource closure.

$V(P)$ to Q[1]. Q's job is to manage R to balance value and cost and maximize the net payoff $\pi(P, R) = V(P) - C(R)$.

We incorporate as little information about behavior as possible. In particular, we have no precise knowledge of external load drivers L, other than what can be gleaned from direct observations of P and R. We do not assume that the function $P(R, L)$ (relating resources to performance) is known, nor do we assume any entity in the system can even *measure* L directly other than through its effects on $P(R, L)$. We do not even assume that the transfer function P remains the same function over time; it may vary due to unforeseen forces, e.g., a human being unexpectedly upgrading server hardware.

3 Prior Work

A theoretical formulation of maintaining state was proposed initially by Burgess[5, 17] and refined via several iterations, eventually leading to the concept of promises[18, 19] to represent statements about the intent of autonomous components. The assumptions of this work include that:

1. All agents within a system are autonomous.
2. All cooperation between agents is voluntary.
3. Management occurs in an open world in which unforeseen changes can occur without notice.

Our management paradigm is a natural outgrowth of the assumptions of Cfengine and promise theory, e.g.,communications between agents and Q are promises.

In contrast to this, much of autonomic computing is based upon feedback loop controllers[1] which represent so different a management strategy that the two strategies may well be incommensurate:

[1] To paraphrase an old programming aphorism, Q knows the cost of everything but the value of nothing (like C), while A knows the value of everything but the cost of nothing (like LISP).

1. Agents within a system are interdependent.
2. Cooperation is compulsory.
3. Management occurs in a *closed world* in which all (or at least most) achievable system states and management alternatives are known in advance.

One reason it is difficult to compare our approach with what is normally called "autonomic computing" is that the two approaches actually solve different problems. Autonomic computing studies how to manage a controlled environment, such as a data center, while our approach studies emergent properties of management mechanisms in an open world, which includes data centers but also such disparate examples as ad-hoc wireless networks and networks of intelligent appliances. Our mechanisms are best conceptualized as influencing rather than controlling the managed system. While autonomic computing controls the managed system, our approach instead "nudges" the system state toward a more appropriate state over time. Most important, we do not assume that our approach is the sole influence acting on the system.

4 Initial Experiments

Initial simulations of this management paradigm exposed a simple algorithm for maximizing π in the case where C and V are monotonic and invertible functions of R and P, respectively, i.e., $C(R+\Delta R) > C(R)$ for $\Delta R > 0$, and $V(P+\Delta P) > V(P)$ for $\Delta P > 0$. We also assume P is a monotonic and invertible function of R for any constant level L_0 of L, i.e., $P(R + \Delta R, L_0) > P(R, L_0)$ for $\Delta R > 0$. The design and behavior of this algorithm are described in [20].

In the algorithm, agents A send estimates of $\Delta V/\Delta R$ to Q, which combines those estimates with $\Delta C/\Delta R$ to obtain an estimate of $\Delta\pi/\Delta R$. Q then uses this estimate to decide whether to increase or decrease R by some constant increment ΔR. Feedback in the system is expressed in terms of the value estimates $\Delta V/\Delta R$ in response to changes in R.

This algorithm exhibits some surprising behaviors when exercised in the context of a periodically varying (sine-wave) test load L. Very naive estimates of $\Delta V/\Delta R$(based upon just a few pairs of measurements of V and R) suffice, and more information *degrades* the efficiency of the total algorithm, because errors in estimating $V(P(R, L))$ without knowing about changes in L dominate errors in estimating the function precisely. Thus a very naive estimate of the function $V(R)$ leads to a usable estimate of $\Delta V(P(R))/\Delta R$, and exhibits efficiencies in the high 90's, even though the estimates of $\Delta V/\Delta R$ appear to be chaotic due to lack of knowledge of L. The algorithm does *not* rely upon any form of machine learning, but instead, relies upon agility in responding to changes in input data, as well as controlled experimentation to determine dynamically evolving alternatives.

It is just as important to note what failed to be useful. Using estimates to determine anything more than the direction in which to change R led to chaos in the managed system, while larger changes in R prevented the agents A from accurately estimating $\Delta\pi/\Delta R$.

5 Analogy with Thermodynamics

The management of low level resources by high level constraints is directly analogous to the study of heat-energy in a hydrodynamic (adaptive) system. Burgess has discussed how computer systems exhibit basic thermodynamic behavior in a number of ways, as they exchange information with their environments[21].

Our management approach is analogous to a 'heat-pump' (or refrigerator) for performance. As a system changes (e.g., becomes "cold") due to the external demand for its energy (performance), we must replenish by supplying the required energy deficit (adding resources). Our aim is to make this process efficient, by pumping just the right amount of resources to meet demand, and not over-allocating resources[2].

Consider a data center or computer system whose purpose is to deliver a service S at a given rate. S might be measured in transactions per second, for instance[3]. The computer system is composed of many resources that change at different rates, but the net result is to deliver the service S at a given rate (i.e. a speed of transport). This is analogous to the physics of a hydrodynamic process, in which one has a fluid, composed of atoms flying around at a variety of speeds, and collectively resulting in the transport of fluid at some collective rate v.

At the microscopic level, physics is about collisions and transfer of momentum through interactions, but at the hydrodynamic level, we abstract away these microscopic details and talk about the high level (bulk) properties. We do not need to know about the microscopic details to understand macroscopic behavior, because the two scales are only *weakly coupled*.

"Energy" is a collective scale for quantifying (present or potential) activity. Energy is converted and moved by a transfer of "heat" or "work", including:

- Q: Heat - an exchange of activity. Heat is modeled as a fluid that can be injected or extracted from a container, just as a service can be delivered. Heat is the service provided by a *thermal reservoir* (e.g. a server array). A flow of heat is a redistribution of service workload.
- U: Internal energy - a store or reservoir of internal capacity. This is a function of the state of a system.
- W: Work - the change incurred by allowing macroscopic motion to take place. This is analogous to computation.

Energy or resources are conserved in a closed system, so

$$U = Q + W$$

i.e., the total internal activity is a result of the heat injected and the work done on the system. Since only differences matter[4], we often write the closure rule as

$$dU = đQ + đW$$

[2] Though our algorithms can be configured to "over-provision" resources to handle peak load by appropriately modifying concepts of cost and value.

[3] In our simulations, we actually measure performance as frequency = 1/rate to make a few calculations more straightforward.

[4] There is no absolute concept of value, without grounding.

where the slashed-d's indicate imperfect differentials that cannot be integrated to give estimates of Q or W. Only U is a function of the resources of the system; Q and W are not necessarily functions of resources, but rather, behave like extra, uncontrolled resources themselves.

In our analogy, we must be careful in comparing energy to service. A word like "performance" describes a rate or velocity as a potentially chaotic and dynamic low-level concept, and is not a static, high-level concept like heat. To reason about performance at a high level, we must define how we value that performance or, alternatively, define its energy content. As our high-level energy concept, we define a payoff Lagrangian function $\pi(P, R) = V(P) - C(R)$, where $P = P(R, L)$. Equality in the above is an equilibrium or *quasi-static* property.

To complete the analogy with the hydrodynamic problem:

- Resources R correspond to internal state. So ΔU corresponds to $\Delta C(R)$ – a change in energy state for internal resources.
- Performance $P(R, L)$ contains external effects, driven by changes in R, so ΔP represents work ΔW.
- The payoff π corresponds to heat exchange. We would like to be able to generate or remove heat (depending on the sign), like a pump/refrigerator. So this is our product.

Thus, if C, π, and V were continuous, we might write:

$$dC = đV - đ\pi$$

where only C is a true function of R. Since these are in our case step functions, we write $\Delta C = \Delta V - \Delta\pi$, which corresponds to our definition of π.

Note that we are describing changes to a steady state, not the steady exchange of service transactions. $C(R)$ is a function of state, but $V(P)$ is not, since we do not have a closed expression for it. To make a complete deterministic model one would need to relate L to π.

Thermodynamics shows us that we do not need detailed information about this relationship in most cases. In thermodynamics, it is usually sufficient to describe scenarios of constant entropy, which are projections of the external system into our state-space. We thus employ an *equilibrium* approach to bypass the need for information we do not have[22].

6 Discretizing Payoff

The continuum algorithm described in Section 4 requires monotonic and invertible V, C, and P. It cannot be applied when V or C are not invertible, e.g., when V and C are monotonic *step functions* that map multiple performance values and resource levels to the same value and cost, in like manner to many common Service-Level Agreements(SLAs). Viewed as a function, the simplest possible SLA is a function $V(P)$ that is 0 unless requirements (lower bounds) for P are met, and has some constant payoff v_0 if those requirements are met.

In like manner, the cost of a particular physical system configuration is often a step function, e.g., one might allocate a whole rack at a time to a task in a cloud. Of the functions in the original scheme, only $P(R, L)$ remains invertible in R for fixed L: adding resources always improves performance (at least somewhat).

Again, algorithms for managing this system emerged via simulation of several possible options. The new algorithms have little resemblance to the algorithms for the invertible case, but also expose a bit more of the quandaries involved in this form of management.

7 Our Algorithm

The goal of the new algorithm is to estimate which way to change R based upon a limited amount of performance data (expressed as ordered pairs (R_i, P_i)). The goal of changing R is to maximize a payoff function $\pi = V(P) - C(R)$ where $C(R)$ is a cost step-function and $V(P)$ is a value step-function. The main problem in combining these functions into a meaningful estimate of payoff π is that P and R are not *commensurate*, in the sense that there is no function $P(R)$ such that $V(P(R)) = V(R)$ represents value. P is not just a function of R, but also a function of L. If there were such a function $P(R)$, then $\pi(R) = V(R) - C(R)$ would be an estimate of the total payoff function, which we can maximize. This is only true, however, if L is held constant, and then represents an *equilibrium state* for the managed system.

In the new algorithm, each agent A makes an estimate $P_{\text{est}}(R)$ of $P(R)$ from a small number of pairs of samples (R_i, P_i) using linear least-squares approximation. In this paper, the number of samples used for this approximation is $w = 10$. The agent uses this estimate to compute an estimate of the value function $V_{\text{est}}(R) = V(P_{\text{est}}(R))$. This is a matter of transforming only the input axis to the step function V, and the output axis does not change; converting $V(P)$ to $V_{\text{est}}(R)$ only moves the transition boundaries at which the step function V changes value in its domain, and does not affect its range. As P is monotonic and invertible in R, the axis transformation always leaves the transition boundaries in V in the same *order*, so that $V_{\text{est}}(R)$ remains simply increasing like $V(P)$.

Each agent A then forwards its estimate of the function $V(R)$ to Q, which subtracts the function $C(R)$ to estimate the function $\pi(R)$. Relative to the current value of R, Q computes the nearest value R_{target} of R at which the estimate of $\pi(R)$ is maximum. Q increments R in the direction of R_{target}, and then the whole process repeats with new agent estimates of V.

Note that the algorithm behaves predictably even though its estimates are often inaccurate and its decisions are often wrong, so much so that we might characterize the estimates as chaotic[5]. Each individual estimate is a micro-scale event, which can be far from accurate, while overall behavior at the macro scale remains near-optimal. As in the invertible case, the estimates of P (and thus

[5] But this is a misconception due to lack of information. It is not truly "chaotic" in a dynamical systems sense, because L is deterministic and determines P, but the fact that we cannot *observe* L makes P seem to behave randomly *to an observer*.

of V) are highly error-prone, in the sense that changes in the hidden variable L are not accounted for in the model. But, for the algorithm to work properly, all that is required is that the estimates are *statistically correct*, or "correct on average". When L stops changing as rapidly, estimates become more accurate and the management system corrects its prior errors in judgment.

The algorithm also actively compensates for estimation errors. A discards duplicate, older measurements of P for the same R when estimating $P(R)$. Q employs several heuristics, including:

1. Limiting the impact of an erroneous estimate by limiting the resulting change in R to a constant ΔR per time step.
2. Keeping R relatively constant whenever its estimate indicates that R is optimal.
3. Periodically checking (every w steps) that a seemingly optimal resource setting R remains optimal, by perturbing R by one unit ΔR *inside* an optimal zone. This perturbation results in a revised estimate of V.

8 Time-State Diagrams

A crucial difference between the invertible and step-function cases is that "time-state-diagrams" become important to understanding the behavior of the algorithm. The step functions in effect at a given time t_0 (for a given unknown value L_0 of L) determine a global payoff function $\pi(R) = V(P(R, L_0)) - C(R)$ that separates the values of R into a state diagram with specific payoffs. Our estimates $\pi_{\text{est}} = V_{\text{est}}(R) - C(R)$ are also estimates of time-state diagram structure. The best possible payoff for π is approximated by the region with maximum payoff in π_{est}.

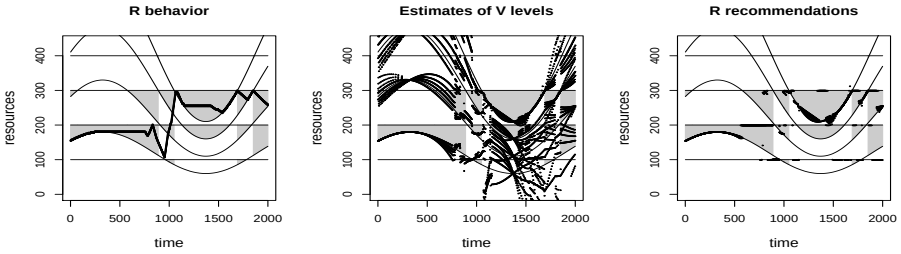

Fig. 2. Attempting to estimate all of the cutoff points for the step-function V leads to seemingly chaotic predictions but generally acceptable tracking.

One way to exhibit the behavior of the algorithm is via a "time-state diagram" as depicted in Figure 2. Each diagram of the figure has a background that is a theoretical time-state plot for R, based upon a known L that is not measurable by agents A or closure Q. The x-axis of each diagram represents

time while the y-axis represents values of R. The horizontal and curved lines on the diagram represent theoretical cutoffs at which step-functions change in value. The horizontal lines represent increases in the cost function C, where cost increases from below to above. The curved lines represent theoretical increases in the value function $V(P(R, L))$ due to changes in L, where value increases from below to above. The curved lines are based upon complete (theoretical) information for L that agents A do not possess and must estimate. The curved lines vary in a periodic pattern based upon a sine-wave form for L.

It is best to think of V and C not as functions, but as *sets of constraints* on value and cost. In each of these diagrams, since the horizontal and curved lines represent all transitions in payoff, each delimited region bounded by lines is a zone of constant payoff; a value of R anywhere in a region achieves the same payoff. The theoretical regions of maximum payoff are shaded in gray; these are the target regions for R. In this figure there are several target regions for each point in time, corresponding to several different solutions to obtaining the same overall payoff.

In the figure, the left-hand diagram contains the trajectory for R over time. $\pi = V - C$ is optimal when R lies in a gray region, and non-optimal otherwise. The middle diagram depicts the algorithm's estimates for value level curves, where each circle represents an inference of some cutoff in the value function. These are only approximations and are often in error; the seemingly chaotic changes in these estimates will be useful later. The right-hand diagram depicts the algorithm's estimates of optimal R, one per cycle.

9 Performance Evaluation

In evaluating this and similar algorithms, one cannot rely on traditional techniques for performance evaluation that compare performance to a theoretical best case. While in our simulations the theoretical best case is known, in realistic situations, the theoretical best case is not merely unknown. Given our framework of assumptions, it is also *unknowable*, in the sense that there is no *mechanism* for observing and learning about load L, and thus no mechanism for fully understanding performance $P(R, L)$.

There are subtle benefits to limiting knowledge. Consider a scenario in which a hacker gains access to a computer and starts utilizing resources. In a traditional autonomic control environment, the performance model would be invalidated by this hidden resource drain, while in our case, there is no model to invalidate. Instead, our algorithm would adapt to the new behavioral conditions without even recognizing that there has been an overall change in state, and try to guarantee performance in the *presence* of the intrusion. Our strategy is not an intrusion-detection tool: it reacts to anomalies *without detecting them.*

Another positive attribute of lack of knowledge is that multiple management algorithms of this kind can be composed without knowledge of one another, e.g., to manage multiple resources. Each algorithm instance will influence L, which will indirectly influence other instances, but the overall effect will be

that each algorithm optimizes its own parameters in isolation from the others. Coordination is achieved by mutual observation, and not by communication.

9.1 Observable Performance Problems

While comparing the algorithm to best case behavior is infeasible, several kinds of performance problems are readily observable in a trace of the algorithm's behavior:

1. *Inaccuracy* resulting from inaccurate estimates of V and inappropriate recommendations for changing R.
2. *Lag* between observing a performance problem and reacting via a change in R.
3. *Undershoot* in which R takes time to approach an optimal value (either from above or below).
4. *Overshoot* in which the increment ΔR chosen for R is too large to allow R to remain in a small optimal region.

It is somewhat ironic that an algorithm that does not explicitly consider time can only be analyzed by considering time. The best way to understand the algorithm's behavior is to remember that all of our parameters are functions of time, so that $R = R(t)$ and $L = L(t)$, and thus $P = P(t) = P(R(t), L(t))$. All of the observable performance problems can be characterized in terms of the behavior of P over time, compared with the behavior of R.

Lag in the algorithm is a direct result of any change whatever in L. By the time information is available, performance has been substandard for a measurement cycle already. This kind of error is always present any time the resource level becomes non-optimal.

Undershoot occurs when a new recommendation for R is distant from a current value. In this case, we see a sequence of linear changes in R that may or may not map to actual changes in payoff. When we observe an undershoot in the trace, all we can do is to estimate the *worst-case* scenario in which the undershoot happened for justifiable reasons. It may also be that the current value of R is optimal and the undershoot does not affect the payoff function, or that the undershoot is due to pursuing an erroneous recommendation.

Overshoot describes a condition in which the increment ΔR of R is too large to allow R to settle into a region of maximum payoff. The symptoms of overshoot include rapid oscillation between two adjacent values of R with no settling down. This is due to the algorithm recommending a downturn for the high value and an upturn for the low value.

9.2 Efficiency

To quantify the algorithm's behavior, we must develop a high-level concept of efficiency. The efficiency of our management paradigm during one time step might be characterized as

$$\pi^{\text{observed}} / \pi^{\text{best}}$$

where π^{observed} represents one measurement of observed payoff and π^{best} represents a corresponding (theoretical) maximum payoff. The efficiency of our management paradigm over time might be defined as the ratio of observed payoff to best payoff, e.g., the ratio

$$E^{\text{theoretical}} = (\Sigma\pi^{\text{observed}})/(\Sigma\pi^{\text{best}})$$

where sums are over time. In both cases, the maximum efficiency is 100 percent, which corresponds to a situation in which observed and theoretically best payoffs always agree. This is impossible due to observation lag, and a certain amount of efficiency is thus lost "by design" due to each change in L, even if the management system is functioning perfectly.

In our simulations, we can compute this notion of efficiency, but in practical situations, the theoretical quantity π^{best} is both unknown and unknowable. Efficiency thus cannot be computed directly, because there is no predictive model of payoff, and thus no concept of best performance. One can, however, bound efficiency by estimating best-case behavior and comparing that estimate to observed behavior.

In bounding best-case behavior, the apparent chaos in our estimates V_{est} of V actually helps. On average, the maximum payoff predicted by $\pi^{\text{est}}(R) = V^{\text{est}}(R) - C(R)$ is an *upper bound* for the theoretical maximum payoff $\pi(R) = V(R) - C(R)$, because statistically, *some* of the estimates of V^{est} are accurate. Thus, provided a large-enough sample of successive estimates π_i^{est} $(i = 1, \ldots, n)$ is considered, the maximum of the upper bounds of the set of estimates is an upper bound on the best-case payoff in the middle of the window:

$$\pi_{n/2}^{\text{best}} \leq \pi^{\text{bound}} = \max_{i=1}^{n}(\max(\pi_i^{\text{est}}))$$

where the inner max represents the maximum payoff predicted by one estimate function π_i^{est}, while the outer max represents the maximum over the set of estimate functions.

Thus a more practical notion of efficiency is an "observed efficiency bound", defined as

$$E^{\text{bound}} = (\Sigma_i \pi_i^{\text{observed}})/(\Sigma_i \pi_i^{\text{bound}})$$

Since empirically $\pi_i^{\text{bound}} \geq \pi_i^{\text{best}}$, $E^{\text{bound}} \leq E^{\text{theoretical}}$, so it is always a lower bound.

10 Simulation Results

To characterize the behavior of this algorithm, we proceed via examples. We simulated the algorithm using a custom simulator written in the R statistics language[23]. In our simulation, the hidden theoretical model of performance is $P(R, L) = R/L$, so that P represents frequency response with perfect scalability of resources, e.g., for a web farm. L is a sine wave varying between (unitless) loads with magnitudes 0.5 and 1.5, corresponding to frequency responses between $R/0.5$ and $R/1.5$. We vary period (and thus frequency) between 100-800 time

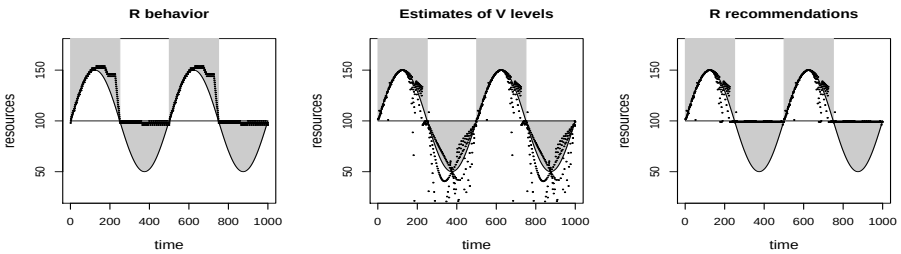

Fig. 3. The simplest management problem consists of one cost change and one value change. In this example, the period of the load function is 500 and ΔR is 2. Theoretical efficiency is 83%, while observed efficiency is 75%.

period	$\Delta R = 1$		$\Delta R = 2$		$\Delta R = 3$		$\Delta R = 4$		$\Delta R = 5$		$\Delta R = 6$		$\Delta R = 7$		$\Delta R = 8$	
100	33.6	34.5	28.9	29.3	45.6	50.7	49.7	43.3	82.6	65.1	73.2	59.6	77.2	61.8	78.5	62.9
200	30.8	34.6	81.9	69.0	74.6	61.6	76.6	62.2	82.6	71.6	86.0	73.2	86.3	74.8	91.3	79.6
300	66.8	77.7	71.9	61.4	81.5	70.5	85.7	73.9	89.5	80.1	92.2	83.0	93.1	83.4	93.8	85.1
400	67.0	75.2	79.9	70.9	86.4	76.8	92.8	83.9	90.6	82.1	95.0	87.1	95.1	87.1	95.3	88.6
500	68.7	60.9	**83.1**	**75.6**	90.8	82.2	94.0	85.8	93.8	86.1	95.3	89.1	96.0	89.6	95.9	90.3
600	74.6	64.0	85.1	75.7	93.2	85.1	95.1	88.1	93.2	87.5	95.8	90.4	96.9	91.4	96.3	90.9
700	77.3	71.0	90.8	82.2	95.0	87.3	95.8	89.9	93.9	89.6	96.3	91.4	97.1	92.2	96.6	92.1
800	79.9	73.9	92.6	84.7	94.6	87.7	96.1	90.5	93.8	89.5	96.4	92.3	97.1	93.0	96.8	92.7

Table 1. Performance of the two-constraint payoff model. The increment ΔR can be tuned by comparing observed efficiencies of values. The boldface entry corresponds to Figure 3. When observed efficiency is greater than theoretical efficiency, this is due to *lack* of seemingly chaotic behavior.

steps per cycle, and ΔR between 1 and 8. This gives us an idea of the interaction between changes in hidden variables and efficiency of the management system.

Inputs to the algorithm include step-functions that we will notate using characteristic ($\chi_{[a,b)}$) notation, where the characteristic function $\chi_{[a,b)}$ is defined by

$$\chi_{[a,b)}(X) = \begin{cases} 0 \text{ if } X \notin [a,b) \\ 1 \text{ if } X \in [a,b) \end{cases}$$

and $[a,b)$ represents the half-open interval from the number a to the number b, not including b.

Our first example is the simplest non-trivial management problem, consisting of one cost transition and one value transition. The cost function is $C(R) = 100\chi_{[100,\infty)}(R)$ while the value function is $V(P) = 200\chi_{[100,\infty)}(P)$. One sample run is shown in Figure 3, whose interpretation is identical to that of Figure 2. Even in the simplest example, there is a relationship between rate of change of the hidden variable L, and choice of increment ΔR. The relationship between rate of change, choice of increment, and efficiency is shown in Table 1, and demonstrates that one can tune ΔR to ΔL by experimentation.

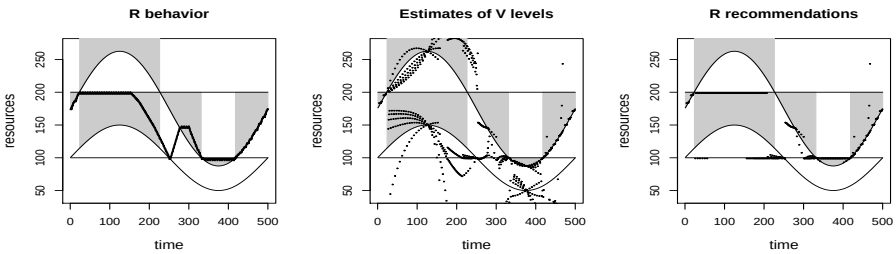

Fig. 4. A slightly more complex management problem with two cost constraints and two value constraints. In this example, the period of the load function is 500 and ΔR is 2. Theoretical efficiency is 74%, while observed efficiency is 65%.

period	$\Delta R = 1$	$\Delta R = 2$	$\Delta R = 3$	$\Delta R = 4$	$\Delta R = 5$	$\Delta R = 6$	$\Delta R = 7$	$\Delta R = 8$
100	48.5 52.8	87.7 -	62.1 -	80.0 61.8	68.1 54.8	50.2 45.6	51.9 40.8	80.4 56.8
200	77.1 78.1	72.8 61.4	65.1 -	56.5 45.4	74.1 62.2	85.4 65.0	87.8 69.8	88.0 70.9
300	85.3 71.3	57.6 45.3	71.8 60.8	75.1 64.8	86.5 70.9	90.3 76.0	91.7 77.9	92.3 79.6
400	84.1 -	60.4 52.8	87.3 71.8	89.6 76.1	79.1 68.5	91.9 80.5	94.3 83.0	93.6 83.5
500	64.5 -	**74.0 64.5**	89.3 75.9	82.7 72.9	90.9 80.6	93.1 83.4	95.3 85.7	93.9 86.0
600	61.3 49.1	87.3 73.6	90.8 79.0	85.8 76.1	91.7 82.8	93.8 85.2	95.7 87.8	93.9 86.9
700	51.8 45.0	88.9 75.7	91.7 80.7	94.6 84.6	92.7 84.6	94.7 87.1	96.2 88.6	94.6 88.2
800	63.9 56.8	89.7 78.1	93.0 83.2	89.4 80.7	93.1 85.4	94.5 87.7	96.1 90.2	94.5 88.9

Table 2. Performance of the four-constraint payoff model. The boldface entry corresponds to Figure 4. Observed efficiency is sometimes not commensurate with theoretical efficiency due to periods of non-prediction; those observed efficiencies are omitted.

Our second example is a slightly less trivial management problem with two levels for each of cost and value:

$$C(R) = 100\chi_{[100,200)}(R) + 300\chi_{[200,\infty)}(R)$$
$$V(P) = 100\chi_{[200,400)}(P) + 175\chi_{[400,\infty)}(P)$$

An example run is shown in Figure 4 and performance data comparing periods and increment values is shown in Table 2. Efficiencies are higher because the travel time between desirable states is smaller, but sometimes, the predictor does not predict a best value, due to rapid changes in R that lead to inappropriate predictions for V. If this happens for an extended period, the observed efficiency cannot be compared with theoretical efficiency, because there is no prediction of best-case behavior for the time when predictions are withheld. In that case(which the algorithm detects), observed efficiencies are not comparable with theoretical efficiencies and are thus omitted from the table.

11 Conclusions

We have demonstrated that order can arise from apparent chaos, with a bit of help. Seemingly chaotic underlying approximations of the high-level state diagram inform resource changes that are observably stable; apparent chaos serves as an effective search aid in planning changes that lead to increased payoff. One key to the emergence of order is that estimates of state are employed only to choose the direction of resource change, while recommended changes serve as experiments that – in turn – refine state estimates near the current resource level. This feedback between experiment and recommendation leads to high-level predictability even though low-level estimates remain unpredictable. At the same time, the nature of the low-level estimates enumerates possibilities in a way that allows estimation of a lower bound on overall efficiency. Because estimates are of the payoff *function*, rather than the payoff *value*, global optimization is possible, which was not possible in prior formulations.

Many open issues remain. The multi-dimensional case has yet to be studied, though the approach here has promise by use of multi-dimensional linear-least squares approximations of $P(R)$ and $V(R)$. Another basic issue is that while the value of performance ($V(P)$) is rather well understood, most sites do not have a clear idea of the cost ($C(R)$) of resources, without which a balance cannot be reached. One possible avenue of study is to link this cost with power awareness. Several design issues also require further study. There are many options for implementing various kinds of policies in this setting, and the best options remain unclear. For example, what is the best way to implement an "over-provisioning" policy?

In the usual concept of autonomics, measuring is easy but modeling is hard. We have constructed a situation in which the reverse is true: modeling is easy but precisely computing efficiency is not possible, though it stays near optimum. Our approach allows us to do without detailed modeling, with benefit in a large number of practical situations.

References

1. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. John Wiley & Sons (2004)
2. Horn, P.: Autonomic computing: Ibm's perspective on the state of information technology. `http://researchweb.watson.ibm.com/autonomic/manifesto/autonomic_computing.pdf` (October 2001) [cited 16 April 2009].
3. IBM: An architectural blueprint for autonomic computing. `http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf` (June 2006) [cited 16 April 2009].
4. Couch, A.: Summary of the third workshop on hot topics in autonomic computing (hotac iii). ;login: Magazine **33**(5) (2008) 112–113
5. Burgess, M.: Computer immunology. Proceedings of the Twelth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA) (1998) 283

6. Burgess, M., Couch, A.: Autonomic computing approximated by fixed-point promises. In: Proceedings of the First IEEE International Workshop on Modeling Autonomic Communication Environments (MACE), Multicon Verlag (2006) 197–222
7. Burgess, M.: Configurable immunity for evolving human-computer systems. Science of Computer Programming **51** (2004) 197
8. Burgess, M.: A site configuration engine. Computing Systems **8**(2) (1995) 309–337
9. Holland, J.H.: Emergence: From Chaos to Order. Oxford Univ Pr (Sd) (March 2000)
10. Johnson, S.: Emergence: The Connected Lives of Ants, Brains, Cities, and Software. Scribner (September 2002)
11. Sauve, J., Moura, A., Sampaio, M., Jornada, J., Radziuk, E.: An introductory overview and survey of Business-Driven IT management. In: The First IEEE/IFIP International Workshop on Business-Driven IT Management (BDIM). (2006) 1–10
12. Moura, A., Sauve, J., Bartolini, C.: Research challenges of Business-Driven IT management. In: The Second IEEE/IFIP International Workshop on Business-Driven IT Management (BDIM). (2007) 19–28
13. Couch, A., Hart, J., Idhaw, E.G., Kallas, D.: Seeking closure in an open world: A behavioral agent approach to configuration management. In: LISA '03: Proceedings of the 17th USENIX conference on System administration, Berkeley, CA, USA, USENIX (2003) 125–148
14. Schwartzberg, S., Couch, A.: Experience implementing a web service closure. In: LISA '04: Proceedings of the 18th USENIX conference on System administration, Berkeley, CA, USA, USENIX (2004) 213–230
15. Wu, N., Couch, A.: Experience implementing an ip address closure. In: LISA '06: Proceedings of the 20th USENIX conference on System administration, Berkeley, CA, USA, USENIX (2006) 119–130
16. Couch, A.L., Chiarini, M.: A theory of closure operators. In: AIMS '08: Proceedings of the 2nd international conference on Autonomous Infrastructure, Management and Security, Berlin, Heidelberg, Springer-Verlag (2008) 162–174
17. Burgess, M.: On the theory of system administration. Science of Computer Programming **49** (2003) 1
18. Burgess, M.: An approach to understanding policy based on autonomy and voluntary cooperation. In: IFIP/IEEE 16th international workshop on distributed systems operations and management (DSOM), in LNCS 3775. (2005) 97–108
19. Bergstra, J., Burgess, M.: A static theory of promises. Technical report, arXiv:0810.3294v1 (2008)
20. Couch, A.L., Chiarini, M.: Dynamics of resource closure operators. In: to appear in AIMS '09: Proceedings of the 3rd international conference on Autonomous Infrastructure, Management and Security, Springer-Verlag (2009)
21. Burgess, M.: Thermal, non-equilibrium phase space for networked computers. Physical Review E **62** (2000) 1738
22. Burgess, M.: Keynote: The promise of self-adapting equilibrium. In: Proceedings of the Fifth IEEE International Conference on Autonomic Computing (ICAC). (June 2008)
23. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. (2008) ISBN 3-900051-07-0.