Combining learned and highly-reactive management

Alva L. Couch[1] and Marc Chiarini[1]

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

Abstract. Learned models of behavior have the disadvantage that they must be retrained after any change in system configuration. Autonomic management methods based upon learned models lose effectiveness during the retraining period. We propose a hybrid approach to autonomic resource management that combines management based upon learned models with "highly-reactive" management that does not depend upon learning, history, or complete information. Whenever re-training is necessary, a highly-reactive algorithm serves as a fallback management strategy. This approach mitigates the risks involved in using learned models in the presence of unpredictable effects, including unplanned configuration changes and hidden influences upon performance not considered in the learned model. We use simulation to demonstrate the utility of the hybrid approach in mitigating pitfalls of both learning-based and highly-reactive approaches.

Keywords: autonomic computing, convergent operators, computer immunology, self-organizing systems, emergent properties, Cfengine

1 Introduction

In the literature, the term "autonomic computing" seems to have become synonymous with employing control loops to control a closed system[1–3]. In this paper, we propose an alternate approach to autonomic resource management based upon an *open-world assumption* that at least some influences upon the managed system are not just unknown, but also *unobservable*, *unlearnable*, and *unknowable*. While open-world systems may perchance behave like closed-loop systems, such behavior is not guaranteed, and at any time, open-world behavior may arise.

An open-world assumption is necessary in order to manage many kinds of contemporary computing systems, e.g., virtual instances of servers in clouds. By design, a cloud server instance is not supposed to know about co-location of other services on the same physical server, but such co-location can and does influence performance as a *hidden variable*. Thus any management strategy for optimizing the behavior of cloud server instances must presume the existence of *unobservable influences* upon the managed system.

1.1 Immunology and approximation

We cope with unobservable influences partly by adopting Burgess' immunological approach to autonomic control[4]. In the immunological approach, management is expressed as voluntary cooperation between an ensemble of independently-acting distributed agents; the result of management is an emergent property of that ensemble. This is the philosophy behind the management tool Cfengine[5, 6], which provides one kind of management agent. Burgess demonstrated that the Cfengine paradigm is guaranteed to converge to predictable states[4, 7] and is perhaps best characterized as accomplishing management by seeking states of "thermodynamic equilibrium"[8, 9], where that equilibrium is an emergent property[10, 11] of the system and agents. Burgess and Couch demonstrated that autonomic computing can be "approximated" by the Cfengine paradigm[12]. The decentralized nature of the Cfengine paradigm is perhaps best described using the concept of a *promise*[13, 14]: a non-binding statement of intent from one agent to another. The input to a Cfengine agent is a set of promises, while its output is a sequence of actions intended to seek equilibrium and thus put the system into a desirable state.

We began this work by seeking an appropriate solution for the problem of autonomic resource management via use of Burgess' autonomous agents. Resource management refers to the problem of matching resource availability for a computation to the performance demands for the computation. For example, in an elastic cloud, one must match the number of available servers for an application with the demand for that application; if there is low demand, one server might suffice, while a high-demand situation might require hundreds of servers. While theoretical results implied that resource management is possible via autonomous agents, no solution was obvious from the theory.

We thus sought a formulation of resource management compatible with Burgess' autonomic computing model. This paper is the third in a series on this topic.

1.2 Closures and cost-value tradeoffs

In the first paper[15] in the series, we developed the idea of a resource closure and the basics of open-world resource management. A closure is a predictable (and thus closed) subsystem of an otherwise open system[16–19]. We proposed a resource management paradigm based upon two kinds of agents: a *resource closure agent* that manages the resource, and several *gatekeeper agents* that provide performance feedback to the resource closure. The closure makes resource allocation decisions based upon a *cost-value tradeoff*; the closure knows the *cost* C of resources, while the gatekeepers know the *value* V of performance. Thus the closure attempts to *maximize reward,* which is the difference between value and cost $(V - C)$. In the following, we will refer to $V - C$ as the *objective function* to be maximized. We demonstrated that $V - C$ can be managed to near-optimal levels by simple hill-climbing *without* knowledge of the (perhaps hidden) performance influences, provided that there are sufficient constraints

on the behavior of those influences and the nature of V and C as functions. If those constraints are violated, even catastrophically, the system still converges eventually to a stable, near-optimal state, which stays near-optimal as long as constraints (on hidden variables) continue to be satisfied. Thus the hill-climbing approach is *highly reactive* in the sense that it can react to unforeseen changes efficiently and without an appreciable learning curve.

The key result of the first paper is that *one can trade constraints for model precision* in managing a complex system; constraints on the managed system are as important as model precision in achieving management objectives. Management can meet objectives even if the model does not account for any performance influences at all; hill-climbing still meets management objectives as long as hidden influences obey reasonable constraints. Even if constraints are not met, hill-climbing continues to eventually meet management objectives, although the response time is slower than desired.

1.3 Step-function SLAs and level curve analysis

One weakness of the approach in the first paper is that hill-climbing requires that value V and cost C are simply increasing functions that are never constant on an interval. In realistic cases, cost and value are both step functions: one cannot (usually) allocate part of a server as a resource, while Service-Level Agreements (SLAs) specify value as a step-function depending upon performance.

In the second paper of this series[20], we discussed how to handle step functions for cost C and value V. Value V is a function $V(P)$ of system performance P, while cost C is a function $C(R)$ of resources R. While closed-world management relates performance P and resources R via some (assumed or learned) model of behavior, in an open-world system, performance P and resources R are *incommensurate*, in the sense that no fixed model relates the two concepts to one another. Thus the problem of choosing R to maximize $V(P)-C(R)$ requires first relating P and R in some meaningful way, so that one can estimate value $V(R)$ and cost $C(R)$ both as functions of the same independent variable R.

As time progresses, costs C are fixed while performance P varies with time-varying hidden influences. Thus the step function $V(R) \approx V(P(R))$ varies with time. While the range of V is fixed by the SLA, the resource levels R at which $V(R)$ changes in value vary over time and are predicted by a model of $P(R)$.

Step-functions expose a weakness in the incremental strategy. Incremental hill-climbing – which serves us well when value and cost functions have no steps – now leads to situations in which the incremental search spends too much time with sub-optimal reward. Increments to resources during hill-climbing are too small to keep up with drastic step-changes in $V - C$.

1.4 A hybrid approach

In this paper, we take the final step in proposing an immunological approach to resource management. We combine highly-reactive hill-climbing with simple machine learning to exploit the strengths of both approaches. We first describe

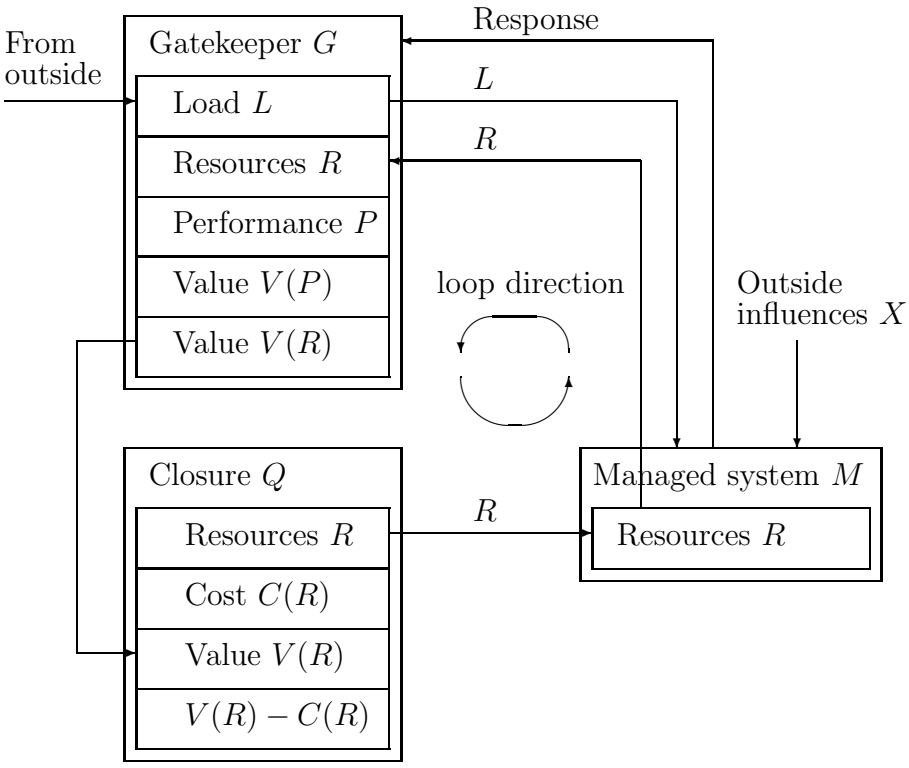

Fig. 1. Our architecture for autonomic resource control. A closure Q controls the managed system M, whose performance P is observed by the gatekeeper G, which informs Q of an observed value function V, thus closing the control loop.

the control architecture and the hybrid model by which we accomplish control. We simulate the model in simple cases to compare behaviors of the hybrid model to behaviors of its components. Our conclusions include limitations of the hybrid approach, as well as a discussion of remaining challenges in open-world resource management.

2 Our control architecture

Our basic control architecture is depicted in Figure 1. A managed system M is controlled through a vector of resource parameters R and influenced by a known load L and unknown factors X. A gatekeeper (e.g., a load balancer) G observes L and system performance P in response to L. G knows the resources R utilized by the managed system. G also knows the value function $V(P)$ of performance P (as specified by an SLA). G uses the history of observed triples (P, R, L) that it observes over time to estimate $P(R, L)$ and thus, to estimate $V(R, L) \approx V(P(R, L))$. Holding L constant at its current value, G communicates an estimate of $V(R)$ to the closure Q. Q combines that estimate with its own

knowledge of $C(R)$ to obtain an estimate of the objective function $V(P(R)) - C(R)$. Q uses this estimate to choose the next value for R, thus closing the control loop. $P(R, L)$ is assumed to be simply increasing in R and simply decreasing in L. $C(R)$ and $V(P)$ are assumed to be increasing step functions in R and P, respectively.

In this paper, the notation $P(R, L)$ refers to a hypothetical functional dependency between P and V; likewise writing $V(P(R, L))$ refers to a hypothetical functional dependency between (R, L) and P, and between P and V. The statement $V(R, L) \approx V(P(R, L))$ means that there is a (hypothetical) functional dependency between (R, L) and V that is a (transitive) result of the (hypothetical) functional dependencies between (R, L) and P, and between P and V. Sometimes, these functional relationships are known for sure, e.g., $V(P)$ or $C(R)$; sometimes they must be estimated, e.g., $P(R, L)$.

This is an *open control loop*, because G and Q have no knowledge whatever of the managed system's input X that might influence P. In theory, $P = P(R, L, X)$, and not $P(R, L)$; estimates $P(R, L)$ do not account for the hidden (and unobservable) variable X, so that the estimates of $V(P(R, L))$ likewise do not account for X. The ideal of observing X and estimating $P(R, L, X)$ and $V(P(R, L, X))$ is considered to be impossible; X represents the *unobservable* and *unknowable* component of influences upon M.

For example, consider a cloud computing environment. M represents the cloud infrastructure, while G represents a load balancer at the border and Q represents the elasticity manager. G can measure the performance of instances, but does not have knowledge of external influences X such as co-location of other services with the current services (on the same physical server). G cannot obtain such information without agreements with other clients of the cloud; such information is *unobservable* and thus *unknowable*.

This architecture differs from those we studied previously in one important aspect: we allow G to utilize knowledge of L in modeling $P(R, L)$; in previous work, G modeled only $P(R)$ and did not utilize L. Adding L to the model allows the new architecture to intelligently deal with step-functions for cost and value. As in prior work, however, X remains as an unknowable load parameter.

3 Two approaches to control

The controller Q in the proposed architecture utilizes a hybrid control strategy incorporating two kinds of control:

1. *Reactive* control based upon localized state and behavior.
2. *Learned* control based upon accumulated history of behavior.

Reactive control incrementally changes R in a small neighborhood of the current R setting, based upon observations and predictions that are local to the current system state in both time and space. This control mechanism makes mistakes due to lack of knowledge of X, but quickly corrects those mistakes via aggressive experimentation[15]. It reacts slowly, however, to major situational

changes requiring large swings in R settings, and fails to determine optimal behavior if the objective function $V - C$ is not convex, i.e., if it has more than one local maximum that is not a global maximum.

Learned control changes R according to some learned model of $P(R, L)$ based upon a long-term history. Learned control bases new R values upon *global* prediction of future state. It reacts quickly to changes in observable requirements, e.g., rapid swings in L, but slowly to changes in hidden factors or system architecture, e.g., replacements of system components that invalidate collected history. Thus learned control is invalidated when current behavior becomes dissimilar to historical data.

These control paradigms are in some sense opposites. Prior work demonstrated that reactive control has an opposite character to that of machine learning[15]: incorporating more history into reactive control *increases* reaction time to mistakes as well as recovery and convergence time.

4 Selecting a control mechanism

Our open-world assumption means that no learned model can ever be considered definitive. Instead, we propose a hybrid architecture in which learned control and highly-reactive control are chosen depending upon conditions. The key to our hybrid strategy is to choose a mechanism by which the validity of the learned model can be tested. If the learned model tests as valid, then learned control is used, while if the learned model fails the validity test, the reactive model is utilized instead.

After testing several validity measures, we settled upon studying the simplest one: a statistical measure commonly referred to as the "coefficient of determination" or r^2[1]. If $\{X_i\}$ is a set of samples, $\hat{X}_i$ represents the prediction for X_i, and $\overline{X}$ represents the mean of $\{X_i\}$, then the coefficient of determination is

$$r^2 = 1 - (\Sigma(X_i - \hat{X}_i)^2))/\Sigma(X_i - \overline{X})^2$$

r^2 is a unitless quantity that varies between 0 and 1; it is 1 if the model exactly predicts behavior, and 0 if the model is useless in predicting behavior. In our case, the history consists of triples (P_i, R_i, L_i) , and the coefficient of determination for the model P is:

$$r^2 = 1 - (\Sigma(P(R_i, L_i) - P_i)^2)/(\Sigma(P_i - \overline{P})^2)$$

where $\overline{P}$ is the mean of the $\{P_i\}$, and $P(R_i, L_i)$ represents the specific model prediction of P_i from R_i and L_i.

Use of the coefficient of determination to test goodness-of-fit is a subtle choice, and not equivalent to using goodness-of-fit tests for the model itself. Our goodness-of-fit test is intentionally *less powerful* than those utilized in model fitting, but also *more general* in the sense that it is independent of the way in which

[1] The name "r^2" is traditional in statistics and has no relationship to our R, which is a measure of resources.

the model was constructed. It can thus be used to compare models constructed in different ways, even with offline models not fitted to the current history.

5 Simulations

Our proposed control system has many variables, and a quantitative study would require specifying too many details. For this paper, we will instead simulate and remark upon the qualitative behavior of the system when exposed to various kinds of events.

We simulate the simplest possible system amenable to the two kinds of control. Our system obeys the linear response law

$$P = \alpha R/(L + X) + \beta + \epsilon(0, \sigma)$$

where α and β are constants, and ϵ represents a normally-distributed measurement error function with mean 0 and standard deviation σ. Recall that the gatekeeper G cannot measure or observe X. Learned control is thus achieved by determining constant coefficients α and β such that

$$P(R, L) \approx \alpha R/L + \beta$$

through linear least-squares estimation. We can then maximize

$$V(P) - C(R) \approx V(\alpha R/L + \beta) - C(R)$$

by choosing an appropriate R. If the r^2 statistic for $P(R, X)$ is sufficiently near to 1.0 (e.g., ≥ 0.9) then this estimate of R is used (the learning strategy); otherwise, a short-term model of P is utilized to estimate local benefit from changing R incrementally (the reactive strategy).

5.1 Simulator details

We experimented with the proposed algorithm using a simulator written in the R statistics system[21]. Although there are now many ways to estimate the response of a managed system via machine learning, several qualitative properties of our proposed architecture arise from even the simplest systems. We simulate the response of a simple system, employing statistical learning, to various kinds of events. The system's (hidden) response function is

$$P(R, L, X) = 1.0 * R/(L + X) + 0.0$$

This is not known to the simulated gatekeeper G, which instead estimates α and β according to:

$$P(R, L) \approx \alpha R/L + \beta$$

by linear least-squares estimation. The learned strategy utilizes 200 time steps of history to estimate P, while the reactive strategy utilizes 10 time steps of history and the same model.

The value function $V(P)$ is known to G while the cost function $C(R)$ is known to Q. The value $V(P)$ is given by:

$$V(P) = \begin{cases} 0 \text{ if } P < 100 \\ 200 \text{ if } 100 \leq P < 175 \\ 400 \text{ if } 175 \leq P \end{cases}$$

while the cost $C(R)$ is given by:

$$C(R) = \begin{cases} 0 \text{ if } R < 100 \\ 100 \text{ if } 100 \leq R < 200 \\ 300 \text{ if } 200 \leq R \end{cases}$$

These functions are arbitrary and were chosen mainly because they have interesting properties, e.g., there is more than one optimal range for R.

Both learned and reactive strategies attempt to maximize $V(P) - C(R)$ by changing R. In the learned strategy, R is set to the recommended value, while in the reactive strategy, R is incremented toward the recommended value by 5 resource units at a time[2]. Using increments rather than settings compensates for errors that inevitably arise from the small number of samples utilized to estimate R in the reactive strategy.

Additionally, regardless of whether learned or reactive strategy is used, the R value is left alone if the current value of R is predicted to be optimal. If R is predicted to be optimal and does not change for 10 steps, it is incremented or decremented temporarily to explore whether the situation has changed, and then returned to its prior (optimal) value if no change is detected. This exploration is necessary to inform the reactive strategy of changes in situation.

In all simulations, Load L varies sinusoidally between 0.5 and 1.5 every 500 time steps. This leads to sinusoidal variation in the cutoffs for optimal $V - C$. Initially, measurement noise σ and unknown load X are set to 0. Each simulation consists of 1500 time steps, of which the first 500 steps of settling time are omitted from the figures.

In the following experimental results, plots consist of several layers as depicted in Figure 2. Acceptable goodness of fit is depicted as a light-gray background (upper left) when the goodness of fit measure r^2 (depicted as a curve) is greater than 0.9. When the background is light gray, learned management is being used; a white background means that reactive management is being used. The regions for R that theoretically maximize $V - C$ are depicted in dark gray (upper right); these always occur between horizontal lines representing where C changes and wavy lines representing where V changes; these are not known to G and G must estimate them. The trajectory of R over time (either recommended or observed) is depicted as a series of black dots (lower left). The composite plot shows the relationships between all four components (lower right).

[2] In prior work we demonstrated that the size of this increment is a critical parameter[15]. In this work we size it near to optimality and leave the increment constant, because we are studying instead how to combine the two strategies.

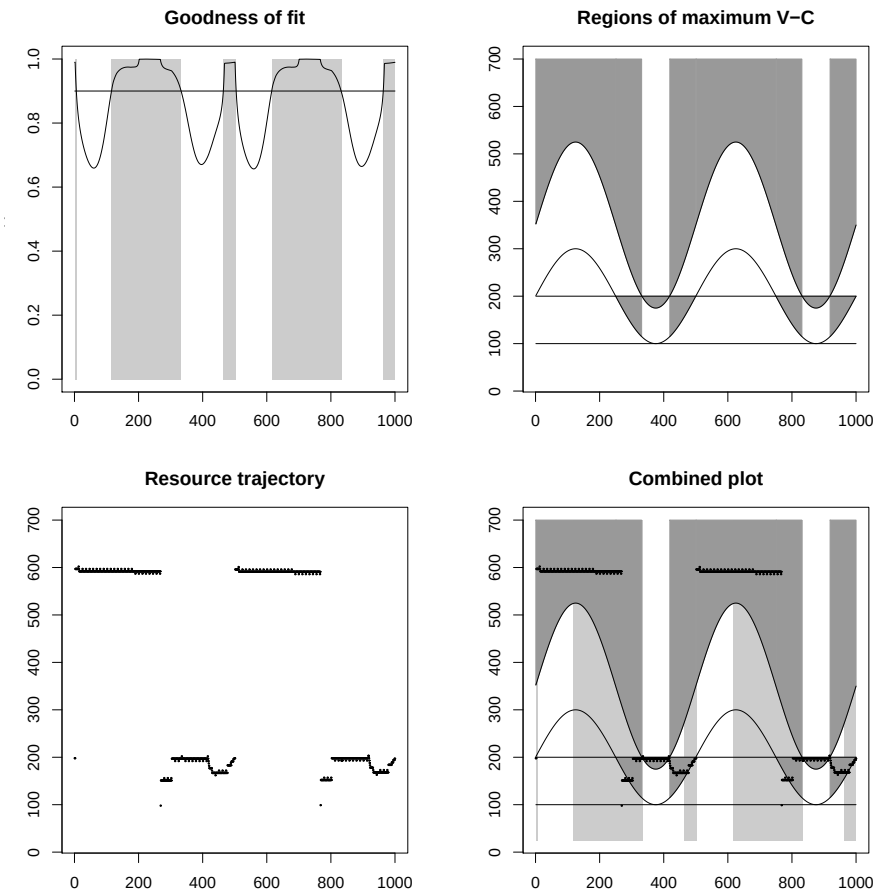

Fig. 2. We depict behavior of the combined strategy via plots that overlay goodness of fit (top left), maximum reward regions(top right), and actual resource trajectories(bottom left) into a single plot (bottom right). All horizontal axes represent time. The vertical axis of the top left plot is the coefficient of determination r^2, while the vertical axes of the other plots depict R.

5.2 Effects of uncertainty

The combined strategy – not surprisingly – does relatively well at handling situations where there are no outside influences or small changes. However, there were some small surprises during simulation. Even a constant hidden influence $X = 0.5$ (50% of the average value of L) poses difficulties for the learned strategy(Figure 3) and the reactive strategy must compensate. In Figure 3, the upper left plot depicts goodness of fit, while the upper and lower right plots show the recommendations of the learned and reactive strategies. When the background becomes white, the reactive model is being used instead of the learned model. The lower left plot shows the actual management response of the system; this

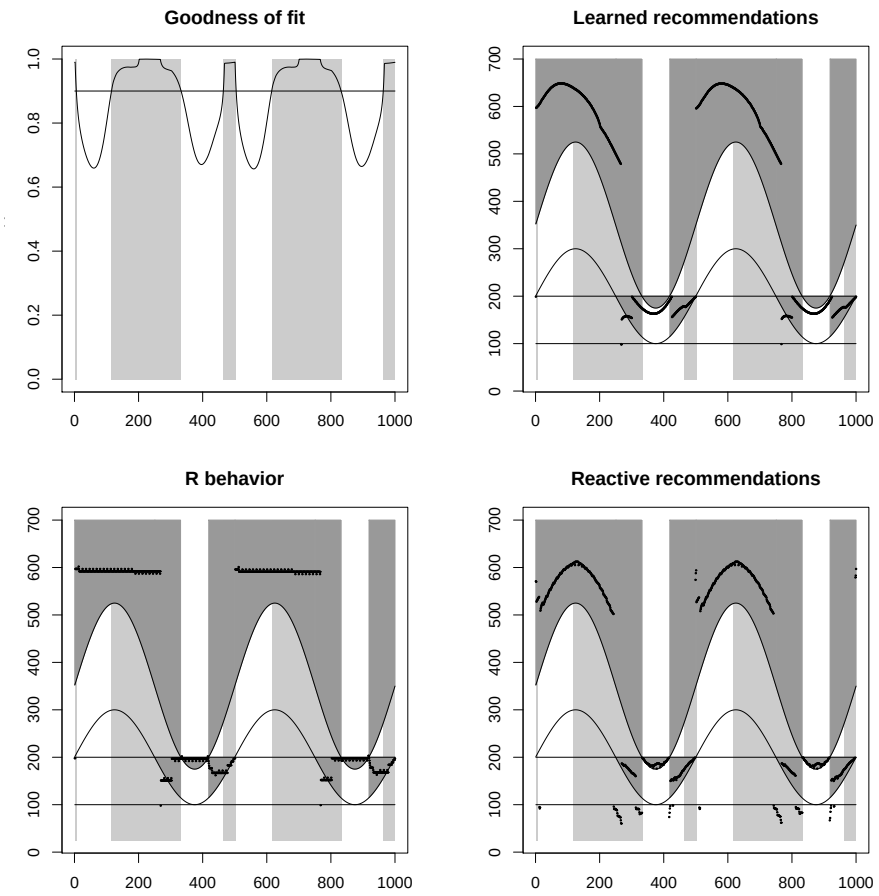

Fig. 3. Even a constant hidden load factor X, where X is 1/2 of the average value of L, periodically invalidates predictions of the learned model (top right, white background) when the hidden influence becomes commensurate with known ones. During this time, predictions of the reactive model (bottom right, white background) control R, resulting in an actual R trajectory depicted on bottom left.

is also the plot whose construction is described in Figure 2. The goodness-of-fit varies with the magnitude of L, in rough proportion to the ratio of signal L to (hidden) noise X.

The behavior in Figure 3 is surprisingly similar to what happens when noise is injected into the measurements of L.[3] Again, goodness-of-fit varies with the magnitude of L and its relative magnitude to the magnitude of the noise.

[3] Results are omitted for brevity.

6 Dealing with discontinuities

The two management strategies differ greatly in how they deal with discontinuities. A *discontinuity* is a relatively large change in some attribute of computation, e.g., load, hidden influences, policy, or model factors.

Discontinuities in input and discontinuous changes in objective function are no problem for a learned model; the model suggests a jump to an optimal resource level because the model is not compromised by the discontinuity. Discontinuities in the model itself (or – equivalently – discontinuities in hidden variables not accounted for by the model) are handled poorly by a learned model (whose accuracy is compromised) and handled better by a reactive strategy.

By contrast, reactive management deals well with situations of model flux but cannot easily cope with discontinuities in the input, such as sudden bursts of load. There is no model; therefore, there is nothing to invalidate in situations of great flux. But the exact same pattern of controlled, cautious experimentation that compensates for errors in judgement also makes it difficult for reactive management to deal with sudden, large changes in *input* and large discontinuities in the objective $(V - C)$ function.

An extreme form of model discontinuity is shown in Figure 4. At time $t = 500$, the underlying performance model changes from $P = 1R/(L + X) + 0$ to $P = 2R/(L+X)+3$, simulating the complete and unexpected replacement of the server with one that is roughly twice as fast. The learned model is temporarily invalidated (and predicts resource values in the white region (upper right plot)), and the reactive model takes over (lower left plot) and predicts near-optimal values of R until the learned model can discover what to do in the new situation. Other forms of model discontinuity behave similarly.

7 Conclusions

In designing this management approach, we utilized several unusual techniques, including:

1. *Trading accuracy of the model for constraints on changes in input.* The reactive strategy works well when changes are small, and requires no model accuracy to work properly.
2. *Exploiting opportunistic accuracy.* The learned strategy is useful even if it does not apply at all times, or even if its model oscillates between validity and invalidity.
3. *Compensating for inaccuracy via use of local data.* The reactive strategy avoids a learning curve by using only data from the immediate past.
4. *Using models to cope with input and objective function discontinuities.* The model, when accurate, can deal with sudden changes in input or objective function.
5. *Using local exploration to cope with hidden variables and sudden model changes and discontinuities.* The reactive strategy deals well with small changes and is not subject to validation limits of the learned model.

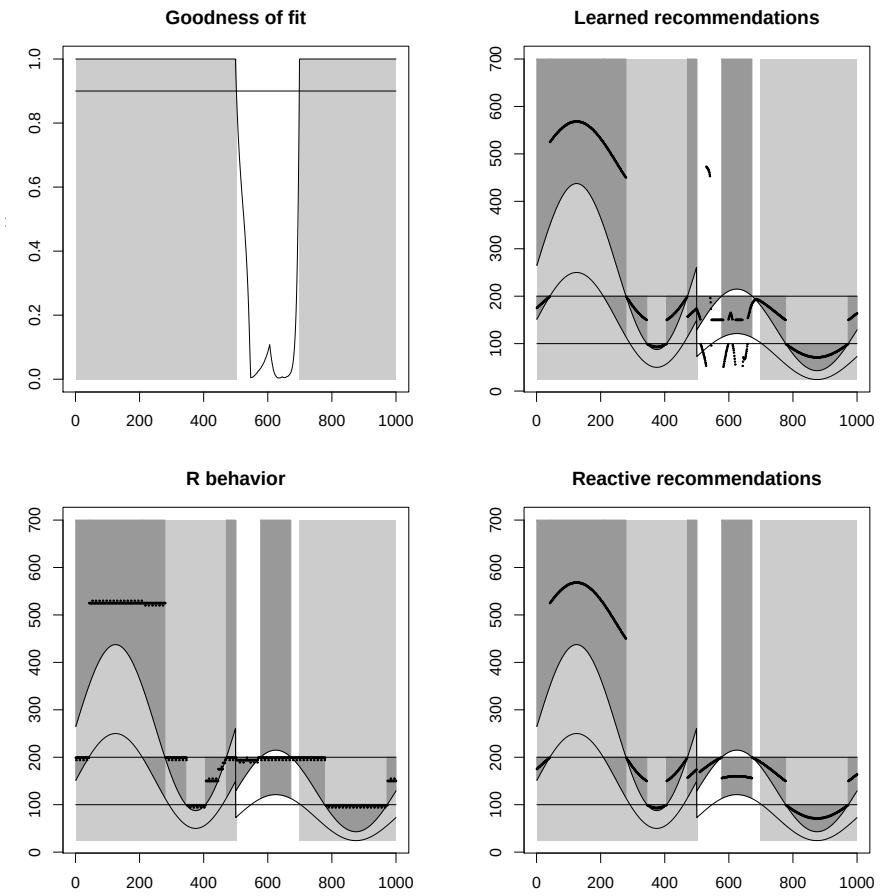

Fig. 4. Discontinuities in P invalidate the learned model (in the region with a white background) and require reactive help.

Several generalizations are relatively straightforward. Asynchrony between actions of G and Q, as well as multiple G's for one Q, is easy to handle and simply slows down response time. L, R, X, and P can be multi-dimensional as long as $P(L, R)$ remains linear and V and C remain single-dimensional step-functions of P and R.

But the door is closed permanently on several options previously believed to be potentially useful. Inferring an unobservable X in this situation turned out to be not just impractical, but impossible; brief experiments with non-linear regression to predict γ in the model

$$P \approx \alpha R/(L + \gamma) + \beta$$

showed that this model has too much freedom and γ is underdetermined. Using gradient-descent non-linear regression, γ could only be inferred when the initial γ was very close to the real one.

Simulating the algorithm exposed many new quandaries.

1. A significant but constant unknown parameter (X) is just as problematic to the validity of the learned model as is significant noise in the input (σ). X does not even have to vary to be problematic. The resulting behaviors seem to be indistinguishable.
2. All kinds of model discontinuity (e.g., changes in X and P) look similar from the point of view of the simulation. There is a recovery time in which the learned model learns the new situation while the incremental model handles management. During this recovery period, the incremental model deals poorly with further discontinuities in either input or model.

Several open questions remain:

1. Can we improve the behavior of the incremental algorithm during long learning periods?
2. Should several learned models with different learning times be utilized?
3. Can static models (e.g., offline learning) be utilized effectively via this mechanism?
4. Can we analyze the transient response of a composite system?

We believe that the answers to all of these questions are "yes", but only further work will tell.

The problem of open-world management is a difficult one, and we have only scratched the surface of the problem. The real underlying issue is one of human expectation. An open world precludes exact guarantees of management behavior. We can only estimate and bound behavior of such systems, and traditional control theory – with its dependence upon exact models – proves less than useful. Worse, in such a system, there is no way to know best-case behavior.

Ours is a story of complexity made simpler. Bounding behavior reduces the complexity of management processes, and dealing with cost and value (rather than more abstract ideas like "performance") simplifies the description of the management problem. A collection of simple management strategies – applied opportunistically – works better than a single strategy. Absolute precision is a thing of the past, and even the most imprecise strategies can contribute positively when things go wrong. This is a fairly good description of human management, as well as that accomplished by autonomic tools.

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. Burgess, M.: Computer immunology. Proceedings of the Twelth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA) (1998) 283
5. Burgess, M.: Configurable immunity for evolving human-computer systems. Science of Computer Programming **51** (2004) 197
6. Burgess, M.: A site configuration engine. Computing Systems **8**(2) (1995) 309–337
7. Burgess, M.: On the theory of system administration. Science of Computer Programming **49** (2003) 1
8. Burgess, M.: Thermal, non-equilibrium phase space for networked computers. Physical Review E **62** (2000) 1738
9. Burgess, M.: Keynote: The promise of self-adapting equilibrium. In: Proceedings of the Fifth IEEE International Conference on Autonomic Computing (ICAC). (June 2008)
10. Holland, J.H.: Emergence: From Chaos to Order. Oxford Univ Pr (Sd) (March 2000)
11. Johnson, S.: Emergence: The Connected Lives of Ants, Brains, Cities, and Software. Scribner (September 2002)
12. 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
13. 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
14. Bergstra, J., Burgess, M.: A static theory of promises. Technical report, arXiv:0810.3294v1 (2008)
15. 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)
16. 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
17. 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
18. 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
19. 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
20. Couch, A.L., Burgess, M., Chiarini, M.: Management without (detailed) models. In: ATC '09: Proceedings of the 2009 Conference on Autonomic and Trusted Computing, Berlin, Heidelberg, Springer-Verlag (2009)
21. 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.